INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Deteção inteligente de golos de futebol

CATARINA FILIPA GONCALVES FERNANDES
Outubro de 2022

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Deteção inteligente de golos de futebol

Catarina Filipa Gonçalves Fernandes

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Engenharia de software

Orientador: Carlos Ramos

Coorientador: Nuno Silva

Porto, outubro 2022

Resumo

Informação e dados estatísticos sobre jogos de futebol são importantes para qualquer equipa, mas nem todas possuem os recursos, monetários e humanos para obter os mesmos. Sendo que atualmente grande parte das soluções requer serviços e dispositivos dispendiosos ou esforço humano para a recolha manual de dados, surgiu a necessidade de desenvolver uma solução simples e sem a necessidade de grandes recursos por parte dessas equipas.

Esta dissertação visa apresentar todo o processo de estudo, experimentação, desenvolvimento e avaliação de uma solução de reconhecimento de momentos de golo em jogos de futebol. Desde a definição de momento de golo em termos de eventos à avaliação de diferentes abordagens e fluxos para a obtenção dos mesmos para a catalogação dos golos. Durante o desenvolvimento foram aplicados conhecimentos obtidos na fase de estudo e técnicas de visão computacional.

Nesta tese pode ser encontrado um estudo e comparação da aplicação das arquiteturas *ConvLSTM* e *LRCN* para classificação de eventos, assim como um estudo sobre o impacto nos modelos das configurações e parâmetros na arquitetura e treino.

Palavras-chave: Visão computacional, Golo, *ConvLSTM*, *LRCN*, *TensorFlow, Python*

Abstract

Information and statistical data about football matches are important for any team. However, not all of them have the resources, monetary and human, to obtain these. Since currently most of the solutions require expensive services and devices or human effort to manually collect data, the need for a simple solution without the usage of large resources arose.

This dissertation aims to present the entire process of study, experimentation, development and evaluation of a solution that recognizes goal moments in football matches, from the definition of the goal moment in terms of events to the evaluation of different approaches and flows to obtain the necessary events. The knowledge obtained in the study phase is considered, as well as the application of computer vision techniques.

A study and comparison of the application of the ConvLSTM and LRCN architectures for event classification can be found in the document, as well as a study of the impact of the different possible configurations and parameters in the models, regarding architecture and training.

Keywords: Computer Vision, Goal, ConvLSTM, LRCN, TensorFlow, Python

Agradecimentos

Primeiramente, gostaria de agradecer à minha família e namorado que sempre me apoiaram ao longo destes anos de estudo e que nesta fase mais complicada mantiveram a paciência e apoio, apesar da minha ausência.

Agradeço também ao Instituto Politécnico de Engenharia do Porto e tudo o que este me trouxe durante estes cinco anos. A todos os docentes deste, em especial ao professor Carlos Ramos e ao professor Nuno Silva pelas sugestões e orientação ao longo deste projeto e que me acompanharam desde a licenciatura. A todos os colegas e amigos que fui tendo pelo caminho e que de uma forma ou outra ajudaram ao longo deste. A todos os momentos e mais alguns passados dentro deste instituto, a todo o conhecimento e a todas as oportunidades que este abriu para mim.

Em especial, agradeço ao João Tomás Rodrigues e Henrique Ribeiro pelo suporte e ajuda incondicional, especialmente durante este projeto, e agradeço ao Rui Machado, Ana Rita Fernandes e Sílvia Silva pela amizade, motivação e apoio ao longo destes anos.

Por último, agradeço à Armis pela oportunidade e disponibilização no embarque deste desafio comigo e todo o apoio dado pelos meus colegas de trabalho, com um especial agradecimento ao engenheiro Joel Carneiro.

Índice

1 Introdução 1

1.1 Enquadramento 1

1.2 Problema 1

1.3 Objetivos 2

1.4 Contributos 2

1.5 Abordagem 3

1.6 Estrutura 4

2 Estado da Arte 5

2.1 Visão computacional 5
2.1.1 Deteção de objetos 6
2.1.2 Deteção de eventos 7

2.2 Aprendizagem automática 7
2.2.1 Aprendizagem não supervisionada 8
2.2.2 Aprendizagem supervisionada 8
2.2.3 Aprendizagem semi-supervisionada 9
2.2.4 Aprendizagem por reforço 10

2.3 Redes neuronais 10

2.4 Aprendizagem profunda 11

2.5 Redes neuronais profundas 12
2.5.1 Rede neuronal convolucional 12
2.5.2 Rede neuronal recorrente 17
2.5.3 Outras redes 19
2.5.4 Aprendizagem por transferência 19

2.6 Modelos de Visão Computacional 20
2.6.1 R-CNN 20
2.6.2 Fast R-CNN 21
2.6.3 Faster R-CNN 21
2.6.4 Mask R-CNN 22
2.6.5 YOLO 23

2.7 Datasets 24
2.7.1 *Datasets* públicos em repositórios 24
2.7.2 SoccerDB 25
2.7.3 SoccerNet 25

2.8 Trabalhos relacionados 25
2.8.1 Proposta de arquitetura de *Jiang et al. de 2017* 26
2.8.2 FootballAnalysis 27
2.8.3 Soluções atuais 28

2.9 Tecnologias 29
2.9.1 Python 29

2.9.2 TensorFlow 31
2.9.3 PyTorch 32
2.9.4 Comparação TensorFlow e PyTorch 32

3 Análise e *Design* 35

3.1 Golo 35

3.2 Arquitetura do Modelo 36
3.2.1 *ConvLSTM* 36
3.2.2 *LRCN* 37
3.2.3 Uso de modelos pré-treinados 38

3.3 Fluxo de deteção de Golos 38

3.4 Seleção da *framework* de desenvolvimento 39

4 Implementação 45

4.1 Instalação 45
4.1.1 Máquina 45
4.1.2 *Python* 46
4.1.3 *TensorFlow* 47
4.1.4 *Notebooks* 47
4.1.5 Ambiente virtual 48

4.2 Experimentação 50
4.2.1 Tutorial base 50
4.2.2 Tutorial de deteção de objetos 50
4.2.3 Tutorial de reconhecimento de ações 51

4.3 *Dataset* 51
4.3.1 Comparação 52
4.3.2 *Download* 54
4.3.3 Processamento dos dados 55
4.3.4 Divisão dos dados 60

4.4 Modelo *ConvLSTM* 61

4.5 Modelo *LRCN* 63

4.6 Treino 65
4.6.1 Funções de regularização 66
4.6.2 Hiperparâmetros 66
4.6.3 Compilação do modelo 67
4.6.4 Treino do modelo 67

4.7 Guardar Modelos 67

4.8 Deteção de golos 68
4.8.1 Criação do ficheiro de *output* 68
4.8.2 Leitura e processamento do vídeo 69
4.8.3 Processamento do resultado do modelo 70
4.8.4 Resultados 72

5 Avaliação 75

5.1 Hipótese 75
5.2 Indicadores e fontes de informação 75
5.2.1 Matriz de confusão 76
5.2.2 Métricas de avaliação 77
5.3 Metodologia de avaliação 77
5.4 Avaliação dos modelos 78
5.5 Avaliação solução final 81
6 Conclusão 83
6.1 Limitações 83
6.2 Melhorias 84
6.3 Trabalho futuro 84
6.4 Apreciação final 84

Lista de Figuras

Figura 1 – Proposta de valor 2
Figura 2 – Diagrama *FAST* 3
Figura 3 – Deteção de objetos (Russakovsky et al., 2015) 6
Figura 4 – Evento de golo de um jogo de futebol (Feng et al., 2020) 7
Figura 5 – Esquema dos conceitos de inteligência artificial (Ramos, 2021b) 7
Figura 6 – Comparação entre componentes de um neurónio e componentes de um neurónio artificial (Gurney, 2018) 10
Figura 7 – Comparação entre aprendizagem automática e profunda (Janiesch et al., 2021a) . 11
Figura 8 – Arquitetura rede neuronal convolucional (Jones, 2017) 13
Figura 9 - Comparação entre rede neuronal e rede neuronal convolucional profunda (CS231n, 2021) 13
Figura 10 – Matriz de pixéis (Yamashita et al., 2018) 14
Figura 11 - Convolução da imagem com um filtro (Pai, 2020) 14
Figura 12 – Agrupamento de CNNs (CS231n, 2021) 15
Figura 13 – Funções de ativação (Yamashita et al., 2018) 16
Figura 14 – *Dropout* (Szeliski, 2021) 17
Figura 15 – Comparação da arquitetura de redes neuronais recorrentes com arquitetura *feed-forward* (Pai, 2020) 18
Figura 16 – Vistas de representações de *RNNs* (IBM Cloud Education, 2020) 18
Figura 17 – Formas de aplicação da técnica de aprendizagem por transferência (Yamashita et al., 2018) 19
Figura 18 – Modelo *R-CNN* (Girshick et al., 2014) 21
Figura 19 – Modelo *Fast R-CNN* (Girshick, 2015) 21
Figura 20 – Modelo *Faster R-CNN* (Ren et al., 2015) 22
Figura 21 – Modelo *Mask R-CNN* (He et al., 2017) 23
Figura 22 – Modelo *YOLO* (Redmon et al., 2015) 23
Figura 23 - Proposta de arquitetura por *Jiang et al.* (H. Jiang et al., 2017) 26
Figura 24 – Serviços *FootballAnalysis* (Akhaee et al., 2021b) 27
Figura 25 – Proposta de arquitetura por *Akhaee et al.* (Karimi et al., 2021) 27
Figura 26 – Deteção e análise de eventos do *Track160* (Track160, 2021) 29
Figura 27 – Estimativa de *SlashData* de desenvolvedores mundiais por cada linguagem de programação (SlashData, 2021) 30
Figura 28 – Interesse a nível mundial das linguagens *Python*, *Java*, *C++, R* e *JavaScript* relativos a aprendizagem automática e inteligência artificial nos últimos cinco anos (Google, 2022b) .. 31
Figura 29 – Percentagem de questões realizadas no *Stack Overflow* referentes à *framework TensorFlow*, à *API Keras* e à *framework PyTorch* (Stack Overflow, 2022) 33
Figura 30 – Abordagem da arquitetura do modelo 36
Figura 31 - Arquitetura LSTM (Olah, 2015) e arquitetura *ConvLSTM* (Alexandre Xavier, 2019) 37
Figura 32 – Arquitetura *LRCN* (Donahue et al., 2017) 37

Figura 33 - Diagrama de fluxo da solução a desenvolver 38
Figura 34 – Árvore hierárquica de decisão para seleção da *framework* 39
Figura 35 – Árvore hierárquica de decisão para seleção da framework 44
Figura 36 – Custo das máquinas virtuais na *Microsoft Azure* 45
Figura 37 – Página inicial *Jupyter Notebooks* 47
Figura 38 - Validação *kernel GPU* dentro do *notebook* 49
Figura 39 – Ativação de crescimento de memória *GPU* (TensorFlow, 2022i) 49
Figura 40 - Experimentação com algoritmo de deteção de objetos 51
Figura 41 - Experimentação com algoritmo de reconhecimento de ações (*LRCN*) 51
Figura 42 – Eventos do *dataset SoccerNet-v2* (Deliege et al., 2021) 53
Figura 43 – Eventos do *dataset SoccerDB* (Y. Jiang et al., 2020) 53
Figura 44 – Tipos e subtipos de eventos do *Wyscout dataset* (Pappalardo et al., 2019) 53
Figura 45 – Função auxiliar para converter minutos para segundos 56
Figura 46 – Extração de *frames* 56
Figura 47 – Validação dos componentes da pasta do jogo 57
Figura 48 – Estrutura do *json* dos ficheiros de anotações dos jogos 58
Figura 49 - Recolha das anotações de cada jogo 58
Figura 50 - Extração das *frames* e armazenamento de cada *feature* 59
Figura 51 – Divisão dos dados para teste 60
Figura 52 - Divisão dos dados para treino e validação 60
Figura 53 – Armazenamento dos subconjuntos de treino, validação e teste em ficheiros 61
Figura 54 – Leitura dos ficheiros com os subconjuntos de treino, validação e teste 61
Figura 55 – Modelo *ConvLSTM* (Experiência B) 62
Figura 56 – Modelo *LRCN* (Experiência B) 64
Figura 57 – Treino 65
Figura 58 – Guardar modelo após treino e avaliação 67
Figura 59 – Obter modelos previamente guardados 68
Figura 60 – Inicio da escrita do ficheiro de *output* 68
Figura 61 – Extração e processamento de *frames* 69
Figura 62- Início de um momento de jogo (golo ou *kick-off*) 70
Figura 63 – Continuação do momento captado 70
Figura 64 – Alteração de momento de jogo 71
Figura 65 – Função auxiliar para converter milissegundos em uma *string* com o formato "HH:MM:SS.sss" 71
Figura 66 – Nenhum momento encontrado 72
Figura 67 – Exemplo do resultado deteção de golos com o Modelo *LRCN* (Experiência H) 72
Figura 68 – Exemplo do resultado deteção de golos com o Modelo *ConvLSTM* (Experiência G) 73
Figura 69 – Matriz de confusão para um classificador binário (Aniruddha Bhandari, 2020) 76
Figura 70 – Técnica *K-Fold Cross-Validation* (Rastogi, 2021) 78
Figura 71 – Avaliação do modelo 79
Figura 72 – Método de desenho das curvas de *accuracy* - *val_accuracy* e *loss* - *val_loss* 79
Figura 73 – Método de desenho da matriz de confusão 79

Lista de Tabelas

Tabela 1 – Funções de ativação da última camada mais utilizadas de acordo com o objetivo final da rede neuronal 16
Tabela 2 – Comparação de critérios 40
Tabela 3 – Normalização da matriz de comparação e prioridade relativa por critério 41
Tabela 4 – Índice aleatório calculado para matrizes quadradas de ordem n 42
Tabela 5 & Tabela 6 – Matriz de comparação, normalização e prioridade relativa do critério de facilidade de aprendizagem e usabilidade 43
Tabela 7 & Tabela 8- Matriz de comparação, normalização e prioridade relativa do critério de bibliotecas e *APIs* 43
Tabela 9 & Tabela 10 - Matriz de comparação, normalização e prioridade relativa do critério de desempenho 43
Tabela 11 & Tabela 12- Matriz de comparação, normalização e prioridade relativa do critério de comunidade 44
Tabela 13 – Comparação de máquina físicas 46
Tabela 14 – Comparação *datasets* 52
Tabela 15 – Experiências modelo *ConvLSTM* 63
Tabela 16 – Experiências modelo *LRCN* 65
Tabela 17 – Avaliação das experiências dos modelos *ConvLSTM* e *LRCN* 80

Lista de Equações

(1) 10
(2) 11
(3) 14
(4) 15
(5) 15
(6) 16
(7) 16
(8) 41
(9) 42
(10) 42
(11) 42
(12) 42
(13) 77
(14) 77
(15) 77
(16) 77

Lista de Excertos de Código

Código 1 – Criação de ambiente virtual *CPU* 48
Código 2 – Criação de ambiente virtual *GPU* 48
Código 3 – *Imports* para *download* do *dataset* 54
Código 4 – *Download dataset* 54

Lista de Acrónimos

AHP	*Analytic Hierarchy Process*
API	*Application Programming Interface*
APP	*Application*
CNN	*Convolutional Neural Network*
ConvLSTM	*Convolutional Long Short-Term Memory*
CPU	*Central Process Uni*
CUDA	*Compute Unified Device Architecture*
cuDNN	*NVIDIA CUDA Deep Neural Network*
DL	*Deep Learning*
DBN	*Deep Belief Network*
DNN	*Deep Neural Network*
DS	*Digital Sports*
FAST	*Function Analysis system Technique*
FCN	*Fully Convolutional Network*
FC-LSTM	*Fully connected Long Short-Term Memory*
FFE	*Fuzzy Front End*
FIFA	*Fédération Internationale de Football Association*
FPF	Federação Portuguesa de Futebol
FPGA	*Field-programmable gate array*
FPN	*Feature Pyramid Network*
FPS	*Frames Per Second*
GAN	*Generative Adversarial Network*
GPS	*Global Positioning System*
GPU	*Graphics Processing Unit*
IA	Inteligência Artificial

ILSVRC *ImageNet Large Scale Visual Recognition Challenge*

ISEP Instituto Superior de Engenharia do Porto

JSON *JavaScript Object Notation*

KNN *K-Nearest Neighbors*

LPS *Local Positioning System*

LRCN *Long-term Recurrent Convolutional Network*

LSTM *Long Short-Term Memory*

ML *Machine learning*

MNIST *Modified National Institute of Standards and Technology*

MRI *Magnetic Resonance Imaging*

NCD *New concept development model*

NDA *Non-disclosure agreement*

NPD *New product development*

OTS Optical Tracking System

OvO *One-vs-One*

OvR *One-vs-Rest*

PIP *Pip Installs Packages/ Pip Installs Python*

RAM *Random Access Memory*

R-CNN *Region Based Convolutional Neural Networks*

ReLU *Rectified Linear Unit*

RNN *Recurrent Neural Network*

RoI *Region of Interest*

RPN *Region Proposal Network*

SEV *Soccer Event Dataset*

Tanh Tangente hiperbólica

TL *Transfer Learning*

TLU	T*hreshold Logic Unit*
TMDEI	Tese do Mestrado de Engenharia Informática
UPN	*Unsupervised pre-trained network*
VAE	*Variational Autoencoder*
VAR	*Video Assistant Referee*
VGG	*Visual Geometry Group*
YOLO	*You Only Look Once*
VRAM	*Video random access memory*

1 Introdução

Neste capítulo é feita uma breve introdução e enquadramento do problema abordado, são enumerados os objetivos e contributos da solução do mesmo e a abordagem adotada.

O último subcapítulo descreve a estrutura do documento para ajudar na leitura do mesmo.

1.1 Enquadramento

Este projeto foi elaborado na *Armis Digital Sports* (*DS*), como dissertação para a unidade curricular de TMDEI presente no mestrado de Engenharia Informática – Engenharia de Software do Instituto Superior de Engenharia do Porto (ISEP).

Sendo a *Armis DS* responsável por soluções na área desportiva, maioritariamente futebol, com ligações à Federação Portuguesa de Futebol (FPF), aperceberam-se que grande parte das equipas não profissionais não tinham recursos para recolher dados e estatísticas sobre o jogo.

Estes dados são importantes não só para parceiros como para fãs, permitindo manter estes informados e com melhor noção do jogo, atrair novos fãs e parceiros e, até mesmo utilizar a informação para análises dentro da equipa (Steiner & Allan, 2021).

1.2 Problema

A análise de jogo para devolver estas informações envolve grande esforço humano ou dependem de serviços de desporto dispendiosos, com *hardware* específico (ex. *multi cam video inputs*) e/ou outras funcionalidades não necessárias. Como é o caso das soluções de deteção de eventos apresentadas na secção de Trabalhos relacionados. No entanto nem todas as equipas, principalmente juvenis e amadoras, dispõem dos recursos necessários.

Desta forma, uma solução de baixo custo capaz de interpretar jogos sem a necessidade de grandes recursos, monetários e humanos, seria o ideal para as equipas mencionadas.

1.3 Objetivos

O principal objetivo é o desenvolvimento da solução para interpretar vídeos de jogos de futebol e catalogar momentos de golo, o evento mais importante num jogo de futebol.

No final da análise de cada jogo deve ser possível perceber se existiram golos e quando é que os mesmos aconteceram (instante do vídeo).

Como as equipas juvenis e amadoras são as principais visadas, deve-se ter em consideração os seus baixos recursos monetários e, consequentemente a falta de *hardware* ótico por exemplo. Sendo assim, a solução deve permitir vídeos de diferentes qualidades e até mesmo de um único ângulo.

1.4 Contributos

Dum ponto de vista da empresa é expectado que esta solução permita o surgimento de novos projetos a partir da adaptação ou adoção desta solução, a obtenção de novas parcerias desportivas e fortificação de parcerias atuais. Para além destes contributos, também as competências adquiridas ao longo do projeto e o estudo são benéficas para a mesma.

No caso das equipas de futebol, principalmente juvenis e amadoras, apesar de terem de fazer um investimento inicial (ex. para gravação dos jogos), os benefícios serão maiores, tal como é demonstrado no modelo da proposta de valor de *Osterwalder*.

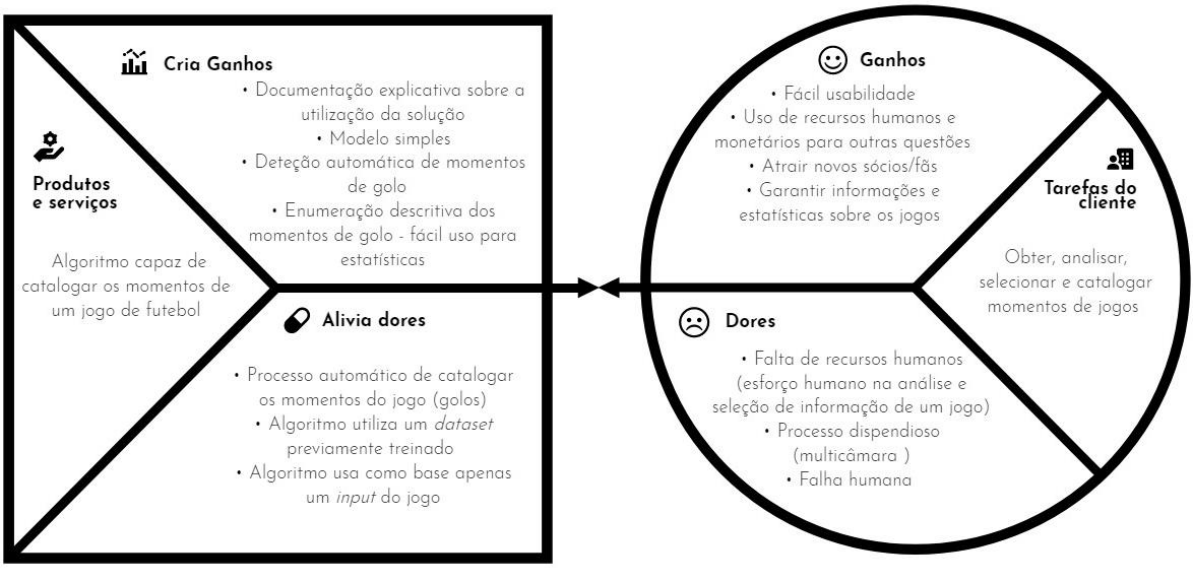


Figura 1 – Proposta de valor

1.5 Abordagem

Com base no objetivo principal temos a questão de investigação "Como automatizar o processo de detetar e catalogar os momentos de golo de um jogo de futebol? ".

Para idealizar uma abordagem com vista a responder à questão anterior foi adotada a Técnica Sistemática de Análise Funcional (*FAST*) que através da representação gráfica das funções de um projeto de acordo com as perguntas "Porquê?" e "Como?" permite realizar uma análise do problema e ter uma ideia da abordagem a seguir na sua resolução (Value Analysis Canada, 2017).

No diagrama seguinte, Figura 2, é feita essa representação gráfica para o projeto.

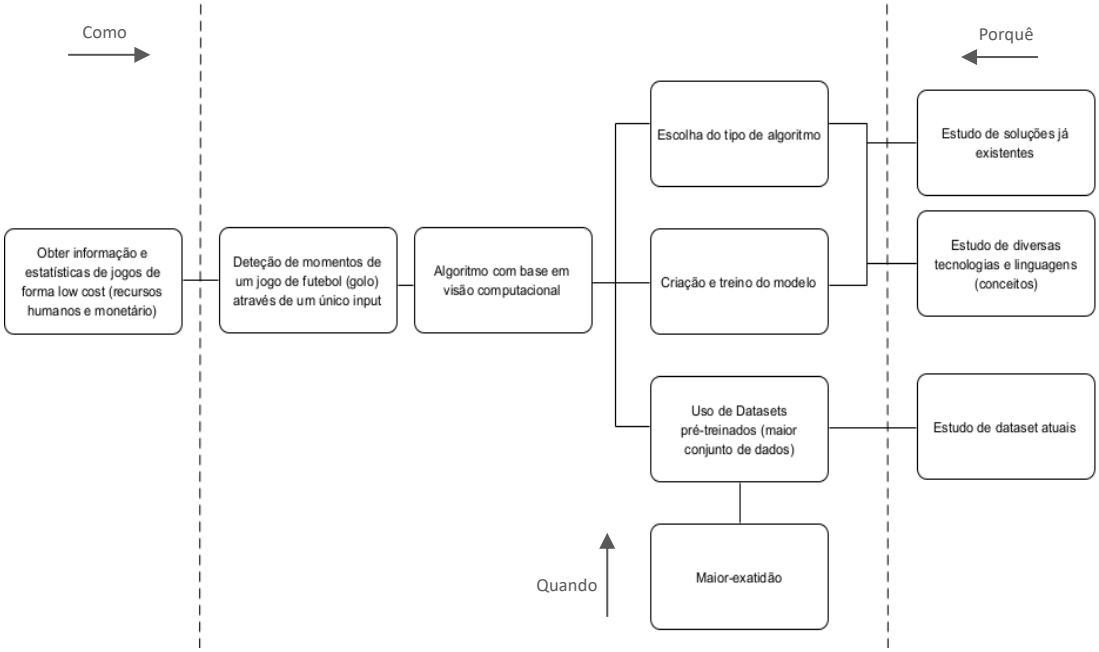


Figura 2 – Diagrama *FAST*

Para o caso deste problema, temos que o requisito principal é a obtenção de informação de jogos de futebol, momentos de golo, a partir de um processo com baixo número de recursos (monetários e humanos), como expressado no diagrama da Figura 2.

Desta forma existe a necessidade de um algoritmo capaz de detetar automaticamente os mesmos a partir de entradas de vídeos simples. Para tal, é necessário escolher o melhor tipo de algoritmo para este caso, construir o modelo e treinar o mesmo e, de preferência, utilizar um *dataset* já existente para fornecer maior exatidão.

Para esta implementação e tomada de decisões, por exemplo qual o melhor algoritmo para este tipo de problema, é necessária a realização de um estudo aprofundado de diversos temas, tecnologias e até mesmo soluções ou propostas dentro do mesmo domínio de problema.

1.6 Estrutura

No primeiro capítulo deste documento é feita uma introdução ao problema, aos objetivos e contributos da possível solução, de forma a contextualizar o leitor do problema que será abordado ao longo do documento. Neste é ainda idealizada a abordagem a seguir a partir da utilização da técnica *FAST* e descrita a estrutura do documento.

No capítulo seguinte, estado da arte, encontra-se o estudo feito sobre o domínio do problema, os conhecimentos necessários para possivelmente solucionar o mesmo e outras soluções já existentes.

No terceiro capítulo é discutido o conceito de golo, as alternativas para a arquitetura do modelo e o fluxo para a deteção de golos em vídeos de jogos de futebol. Por fim, é ainda utilizado o método *AHP* para selecionar qual a framework a utilizar durante o desenvolvimento.

No capítulo de implementação é demonstrado o processo de desenvolvimento das arquiteturas e fluxos descritos no capítulo de Análise e *Design* e todas as tarefas necessárias para alcançar esses. Como por exemplo, a fase de experimentação, as instalações e configurações necessárias e a escolho do dataset e processamento do mesmo.

No capítulo seguinte, Avaliação, é apresentado o estudo das métricas e técnicas de avaliação, analisados os resultados obtidos e a devida avaliação da solução.

Por fim, são apresentadas quais as limitações que ocorreram durante este projeto, possíveis melhorias e trabalho futuro, se os objetivos foram compridos e uma apreciação final do projeto como um todo.

2 Estado da Arte

Neste capítulo é feito um estudo dos diversos conceitos e tecnologias relevantes para o entendimento do problema e desenho da solução.

Este estudo alberga a identificação e aprofundamento dos conceitos mais relevantes, a análise de diferentes abordagens, estudo e comparação das tecnologias existentes. Para a análise das abordagens não só são identificadas as vantagens e desvantagens destas, como é feito um estudo das abordagens propostas em trabalhos relacionados.

2.1 Visão computacional

A visão é um dos sentidos mais importantes para os seres humanos, sendo que nos permite observar, detetar e identificar o que nos rodeia. Este dá-nos a capacidade de identificar as características (forma, cor, dimensão) de diversos objetos e indivíduos, e até mesmo conseguir identificar emoção através de fotografias de alguém (Szeliski, 2021). Estas capacidades são depois utilizadas por nós no nosso quotidiano, nos nossos trabalhos e até mesmo por outros seres vivos para garantir a sua sobrevivência.

A visão computacional pode ser considerada uma replicação artificial dos processos de visão biológicos. Esta realiza estes processos de forma automática, através de algoritmos de aprendizagem automática, e encontra-se em constante crescimento (Szeliski, 2021).

Esta tem várias aplicações e pode ser usada da forma que mais ajude o utilizador, tal como nos seguintes setores (Ajgaonkar, 2021) (Szeliski, 2021):

- Energia - monitorização dos equipamentos de forma a garantir a eficácia destes e a segurança. Como por exemplo a monitorização de equipamentos e materiais nas centrais nucleares;
- Fabrico – uso de modelos *3D*, regularização de máquinas, inspeção das mesmas e avaliação de produtos, o que melhora a eficácia da fábrica e ajuda também na segurança. Tudo isto

através de deteção de anomalias, como a deteção de um defeito no produto ou de falhas de segurança (ex. falta de uso de capacete);

- Transporte – através da implementação de condução automática, a partir da deteção de linhas, sinais, semáforos e até mesmo objetos inesperados, o que garante a segurança. Esta aplicação é dos maiores exemplos do uso de visão computacional no nosso quotidiano e espera-se que se veja intensificar cada vez mais;
- Retalho/Vendas – monitorização dos locais de venda, gestão do inventário e, atualmente, para lojas totalmente autónomas;
- Medicina/saúde – em análise de imagens médicas (ex. *MRI*);
- Segurança – como já mencionado nos outros setores, a visão computacional garante segurança ao realizar monitorização de trabalhadores e equipamentos, mas esta também pode ser utilizada em biometria e para vigilância (monitorizar armazéns ou tráfego);
- Indústria cinematográfica – como o uso modelos *3D* e na captura do movimento, por exemplo para criação de animações.

2.1.1 Deteção de objetos

A deteção de objetos carateriza-se pelo reconhecimento de um ou múltiplos objetos de interesse, de uma determinada categoria, presentes num conjunto de imagens, como representado na Figura 3, ou sequências de imagens (*frames* de vídeos) (Alake, 2020).

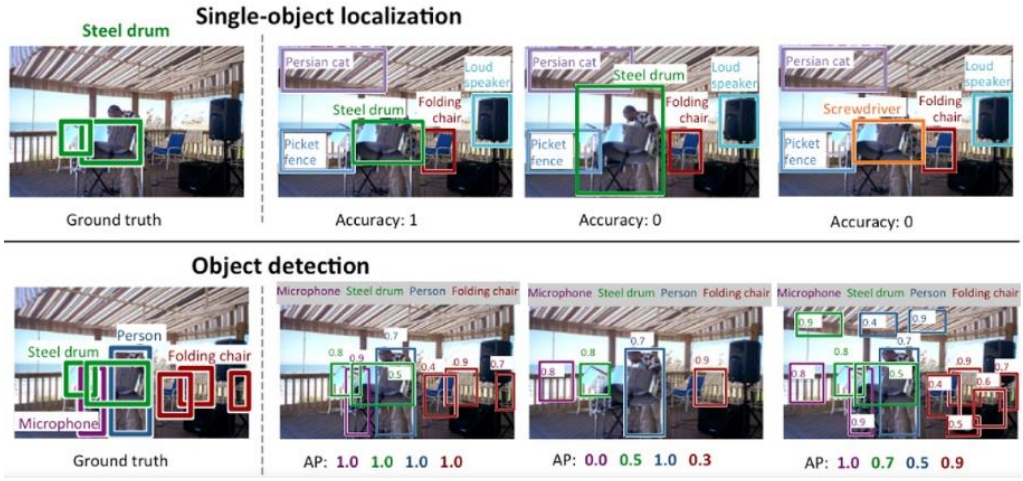


Figura 3 – Deteção de objetos (Russakovsky et al., 2015)

A implementação da deteção de objetos pode ser abordada de duas formas diferentes. Numa das abordagens é necessário a criação e treino de algoritmos sem qualquer base, o que implica maior conhecimento e maior tempo de treino e da própria implementação. A outra abordagem permite diminuir o tempo de treino e aumentar a exatidão a partir do uso da técnica de aprendizagem por transferência (Alake, 2020), explicado na secção de Redes neuronais deste documento.

2.1.2 Deteção de eventos

Enquanto na deteção de objetos são procurados os objetos para o qual a máquina foi treinada, na deteção de eventos é feito o reconhecimento de clipes que contêm eventos predefinidos (Feng et al., 2020), tal como o nome indica.

Para este projeto é importante considerar uma sequência de eventos dentro de um campo num jogo de futebol. Estes eventos podem variar desde faltas, aos cartões dados devido a estas, ao pontapé de meio-campo e aos momentos de golo, como demonstrado na Figura 4.

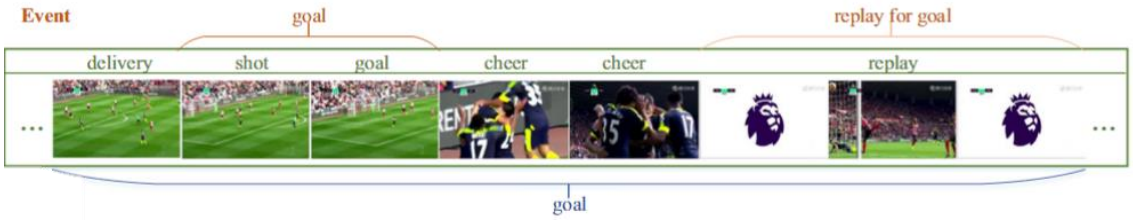


Figura 4 – Evento de golo de um jogo de futebol (Feng et al., 2020)

2.2 Aprendizagem automática

Como demonstra o esquema seguinte, Figura 5, a aprendizagem automática, conhecida como *machine learning (ML)*, é um subconjunto de inteligência artificial. IA representa todas as técnicas que permitem as máquinas "imitarem" a inteligência humana, sendo uma dessas técnicas retratada neste capítulo (Microsoft, 2022a).

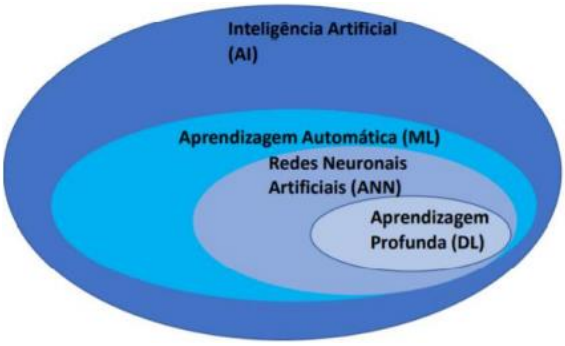


Figura 5 – Esquema dos conceitos de inteligência artificial (Ramos, 2021b)

Por sua vez, a aprendizagem profunda usa outras técnicas que permitem as máquinas beneficiarem da experiência para melhorar as suas tarefas (Microsoft, 2022a), ou seja, converterem experiência em conhecimento (Shalev-Shwartz & Ben-David, 2014).

Esta técnica de inteligência artificial é utilizada quando existe uma necessidade de adaptabilidade e o problema é complexo ao nível humano. Neste tipo de processo de aprendizagem é necessário introduzir dados de treinos e realizar o treino do modelo (Microsoft,

2022a). Quanto maior for a quantidade de dados de treino, maior será a experiência e melhor será o resultado (Shalev-Shwartz & Ben-David, 2014).

A aprendizagem, de forma geral, pode ser realizada de diversas maneiras e categorizada em diferentes tipos. A aprendizagem profunda não é exceção e também esta pode ser subdividida em tipos através de diversos parâmetros. Um destes baseia-se na natureza da interação do aluno com o ambiente e no tipo de supervisão realizado sobre esta (Oladipupo, 2010) (Shalev-Shwartz & Ben-David, 2014) (Saravanan & Sujatha, 2019).

2.2.1 Aprendizagem não supervisionada

Como o nome indica, numa aprendizagem não supervisionada não existe supervisão no processo de aprendizagem. Neste tipo de processo não é dada informação extra ou rótulos indicativos e os dados de treino são semelhantes aos dados usados para teste (Delua, 2021).

Por causa da natureza desta aprendizagem, os algoritmos são maioritariamente utilizados para problemas de associação e agrupamento, onde o conjunto de dados é subdividido em conjuntos com características semelhantes (Shalev-Shwartz & Ben-David, 2014) (Brownlee, 2016). Este tipo de agrupamento pode depois ser utilizado para deteção de anomalias, sendo a anomalia identificada por se encontrar fora de um padrão estabelecido por características semelhantes (Delua, 2021).

2.2.2 Aprendizagem supervisionada

Ao contrário da aprendizagem não supervisionada, este tipo de aprendizagem fornece, juntamente com os dados de treino, informação extra sobre como catalogar os mesmos. O objetivo desta aprendizagem é treinar o algoritmo a reconhecer a ligação entre o rótulos e os dados fornecidos, de forma a depois este conseguir catalogar dados de testes, sem essa informação de ajuda associada (Shalev-Shwartz & Ben-David, 2014).

Este tipo de aprendizagem é normalmente utilizado para problemas de catalogações, sendo que estes podem ser divididos em duas categorias, problemas de regressão e problemas de classificação (Brownlee, 2016) (Delua, 2021). Os problemas de regressão são problemas que pretendem catalogar os dados de forma quantitativa, por características como o peso, o preço e a quantidade. Já os problemas de classificação, pretendem catalogar os dados de forma qualitativa, por categorias como a cor, ausência ou presença de uma determinada característica (Brownlee, 2016), e como no caso do problema retratado neste documento, o tipo de momento de jogo (ex. golo, fora de jogo).

Na aprendizagem supervisionada virada para a classificação podemos encontrar diferentes tipos de algoritmos, cada um com as suas forças, fraquezas e tipos de utilidade dispersos (Oladipupo, 2010) (Microsoft, 2021).

Os classificadores lineares classificam com base no valor de combinação linear de todas as caraterísticas. Apesar do algoritmo da árvore de decisão ser considerado um dos algoritmos mais rápidos de classificação, os classificadores lineares também são utilizados para resolver problemas onde é necessária rapidez na classificação, principalmente em casos em que as características são consideradas dispersas. Estes têm melhor desempenho quando o número de dimensões das características é grande, por exemplo para a classificação de documentos (número de palavras por documento). A técnica de regressão logística, algoritmos *perceptron* e máquina de vetores de suporte são exemplos de classificadores lineares (Oladipupo, 2010) (Saravanan & Sujatha, 2019).

K-Nearest Neighbors (*KNN*) é um algoritmo de agrupamento não paramétrico, onde a classificação é conseguida através da identificação dos vizinhos mais próximos, *k*, e o cálculo da sua distância. Algoritmos deste tipo são considerados intuitivos e são aplicáveis não só para casos de classificação, mas também de regressão. No entanto, estes têm um melhor desempenho para volumes de dados menores e dependem da normalização das características (Rashidi et al., 2019).

A árvore de decisão e, por sua vez, o algoritmo *random forest* também pertencem à aprendizagem supervisionada. Este género de algoritmo assemelha-se a uma árvore real, onde a raiz representa a entrada, os nós internos as questões/decisões a ser tomadas e as ramificações as regras para chegar a uma categoria/decisão final (folha) (Rashidi et al., 2019).

As redes neuronais são maioritariamente utilizadas para grande volumes de dados e principalmente para problemas como o deste documento, que requer a utilização de visão computacional e análise de imagem/vídeo (Rashidi et al., 2019). Estas são exploradas nos próximos capítulos, onde é aprofundada a sua utilidade, tipos, o seu funcionamento e tecnologias existentes.

Outros exemplos de algoritmos desta aprendizagem, mas não aplicáveis para o caso do problema deste documento, são os algoritmos *Boosting*, classificadores quadráticos, redes bayesianas e classificadores probabilísticos, como os classificadores *Naive Bayes* (Oladipupo, 2010) (Saravanan & Sujatha, 2019).

2.2.3 Aprendizagem semi-supervisionada

Tal como o nome indica a aprendizagem semi-supervisionada utilizada ambas as técnicas das aprendizagens anteriormente mencionadas (aprendizagem supervisionada e aprendizagem não supervisionada). Desta forma, esta aprendizagem é feita usando uma mistura de dados de treino com e sem rótulos. O *ratio* entre a quantidade de dados de treino catalogados e não catalogados pode ser adaptado a cada caso e para a obtenção de maior precisão (Oladipupo, 2010) (Saravanan & Sujatha, 2019).

2.2.4 Aprendizagem por reforço

Nesta aprendizagem os dados de treino têm informação extra, mas não têm toda a informação que o algoritmo precisa de prever (Shalev-Shwartz & Ben-David, 2014). Deste modo, o algoritmo aprende com o ambiente em que se encontra e com as ações-consequências, ou seja, obtém *feedback* das ações no ambiente e identifica o comportamento ideal a ter (Oladipupo, 2010) (Saravanan & Sujatha, 2019).

2.3 Redes neuronais

De acordo com *Kevin Gurvey*, autor do livro "*An Introduction to Neural Networks*", uma rede neuronal pode ser definida como um conjunto interconectado de vários elementos de processamento (unidades ou nós) que visam funcionar como neurónios de seres vivos. (Gurney, 2018). Este autor descreve as semelhanças entre a comunicação dos neurónios dos seres vivos com os neurónios artificiais.

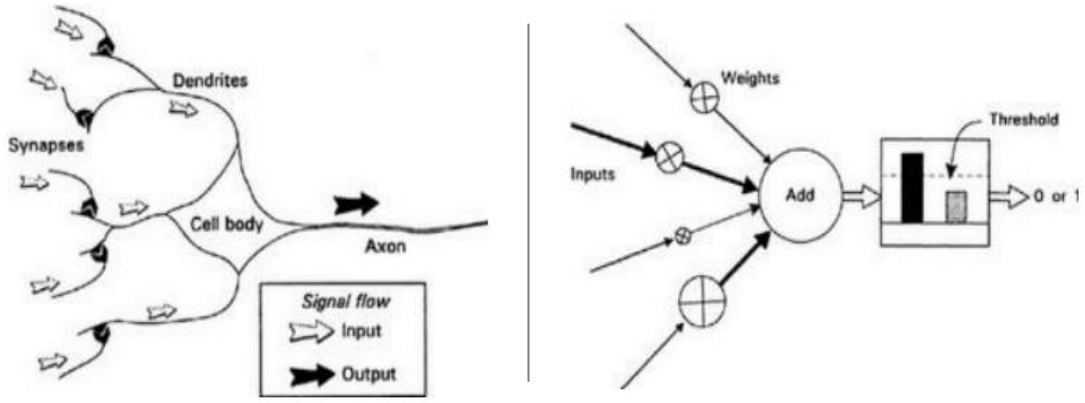


Figura 6 – Comparação entre componentes de um neurónio e componentes de um neurónio artificial (Gurney, 2018)

Os neurónios comunicam por impulsos (sinais elétricos) através de regiões responsáveis pela conexão e comunicação dos neurónios, sinapses. No caso de neurónios artificiais, como representado na figura de cima, são utilizados pesos para replicar as sinapses. Desta forma, cada valor de entrada funciona como a comunicação a passar e os pesos são as regiões de sinapse, o que significa que cada valor de entrada é multiplicado pelo respetivo peso. Posteriormente, os valores obtidos nesta fase são somados e fornecidos a um nó chamado nó de ativação. Todas estas fases podem ser representadas pela seguinte fórmula (Gurney, 2018):

$$\alpha = \sum_{i=1}^{n} w_i x_i \qquad (1)$$

Onde ∑ representa o somatório de 1 a n, com índice i, da multiplicação dos valores de *input* x com os respetivos pesos *w*. O resultado desta fórmula, *α*, é o valor fornecido ao nó de ativação que de seguida é utilizado na comparação com o *threshold*, também representado na Figura 6.

Nesta comparação, conhecida como *threshold logic unit* (*TLU*), se o valor de ativação for maior ou igual ao valor de *threshold* o resultado é o valor 1, mas se o valor de ativação for menor que o *threshold* o resultado é o valor 0. Esta comparação pode ser representada da seguinte forma (Gurney, 2018):

$$y = \begin{cases} 1, & \alpha > \theta \\ 0, & \alpha < \theta \end{cases} \quad (2)$$

Onde y exprime o resultado, α o valor de ativação previamente obtido e θ o valor de *threshold* definido.

A lógica aqui representada é depois usada em sistemas de neurónios que compõem as redes neuronais. Estas têm uma grande capacidade de processamento e, ainda segundo o autor *Kevin Gurney*, essa capacidade é armazenada nos pesos obtidos durante o processo de aprendizagem (Gurney, 2018).

2.4 Aprendizagem profunda

A aprendizagem profunda, também conhecida por *deep learning* (*DL*), como representada na Figura 5, pode ser considerada um tipo de arquitetura de redes neuronais e, por sua vez, é um subconjunto da aprendizagem automática.

Em comparação com os métodos tradicionais de aprendizagem automática, a aprendizagem profunda funciona como uma espécie de caixa preta e apresenta um melhor desempenho para grandes volumes de dados (Ramos, 2021b), como representado no esquema seguinte.

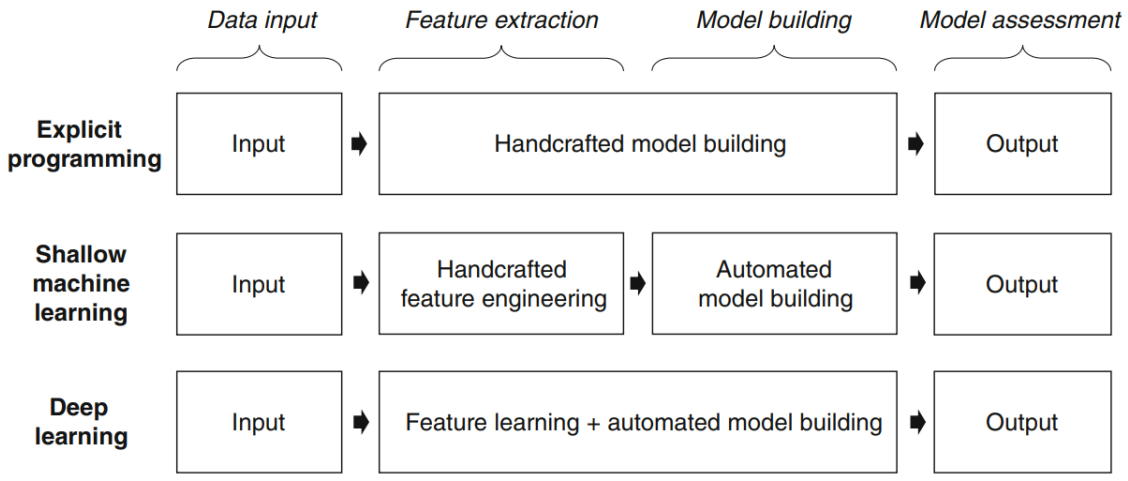


Figura 7 – Comparação entre aprendizagem automática e profunda (Janiesch et al., 2021a)

Como visto no esquema da Figura 7, os algoritmos de *ML* necessitam de um processo prévio de extração e processamento das caraterísticas. A qualidade deste processo de extração interfere com o desempenho do algoritmo de aprendizagem automática, o que se torna uma limitação para esses algoritmos (Janiesch et al., 2021a).

No entanto, no caso da arquitetura *DL,* o processo de extração e processamento é realizado de forma automática e sem grande esforço humano, o oposto de *ML*. Para conseguir realizar todo o processo de extração e classificação esta arquitetura é constituída por diversas camadas (Janiesch et al., 2021a).

Devido a estas características, este tipo de arquitetura de redes neuronais é ideal para problemas com dados de grande escala, possivelmente com pouca qualidade e não estruturados, como problemas complexos de visão computacional e análise de linguagem natural (Ramos, 2021b) (Microsoft, 2022a).

No entanto, o desempenho destas redes é dependente de hardware (uso de *GPUs* ou *FPGAs*), de um processo de treino mais longo e com grandes conjuntos de dados de treino (Ramos, 2021b) (Microsoft, 2022a). Devido ao maior tempo de treino que estas redes necessitam, estas estão mais sujeitas a *overfitting,* que torna o modelo inutilizável para dados fora do conjunto de treino. O *overfitting* acontece quando um modelo treina por demasiado tempo os mesmos dados de treino e/ou quando é demasiado complexo. Isto faz com que este comece a memorizar ruído presente nos dados de treino e se adeque demasiado a estes (Yamashita et al., 2018) (IBM, 2021b). Este problema pode ser reduzido pelo aumento do número dados de treino, aumento da diversidade dos mesmos, por exemplo através de transformações, como a rotação de imagens, e/ou através do uso de funções regularizadoras (Yamashita et al., 2018), mencionadas nos capítulos seguintes.

2.5 Redes neuronais profundas

Como já mencionado as redes neuronais profundas, *DNNs (Deep Neural Networks)*, têm não só múltiplas entradas e saídas, como também múltiplas camadas ocultas entre estas. Cada camada consegue transformar os dados em informação usável pela próxima camada, o que permite uma aprendizagem contínua, juntamente com o processamento de dados (Ramos, 2021b) (Microsoft, 2022a).

Existem vários tipos de redes neuronais profundas e cada uma tem os seus benefícios como descrito nos subcapítulos seguintes.

2.5.1 Rede neuronal convolucional

Uma rede neuronal convolucional (*CNN*) é composta por várias camadas que permitem a extração das caraterísticas e de seguida a classificação como representado na figura seguinte, Figura 8. Devido ao grande desempenho destas redes, estas são atualmente utilizadas não só

para processamento de imagens, como também para tratamento de vídeos e deteção de objetos (Ramos, 2021c).

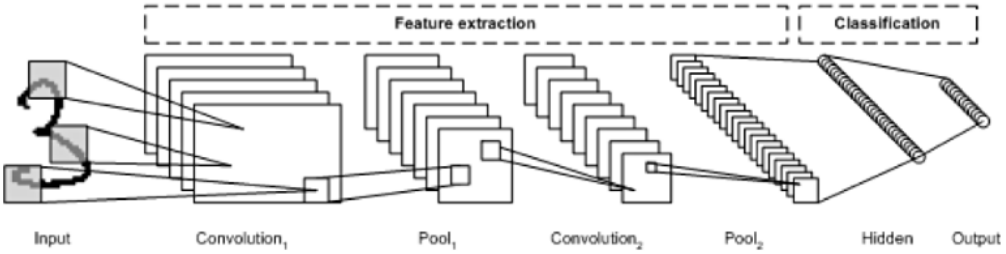


Figura 8 – Arquitetura rede neuronal convolucional (Jones, 2017)

Uma arquitetura usual de redes neuronais convolucionais é composta por camadas de convolução e agrupamento que permite extrair as características. Sendo que nestas camadas os neurónios ligam-se apenas a regiões dos neurónios de outra camada (Khan et al., 2020)(Microsoft, 2022a). Esta mistura de camadas é seguida por uma camada totalmente conectada. Para além de funções de mapeamento, existem ainda funções de regularização, como a normalização do lote e *dropout* que aumentam o desempenho da rede neuronal (Jones, 2017) (Khan et al., 2020).

Cada camada de uma *CNN* pode, como descrito na Figura 9, ser representada por um sólido com largura, altura e profundidade. Sendo que a largura e altura estão relacionadas com a dimensão (ex. da imagem) e a profundidade está associada à dimensão do elemento da matriz (ex. unidade do valor dos pixéis – código da cor). O volume inicial é maior que o volume final, devido à redução a um só vetor, como explicado mais à frente neste subcapítulo (CS231n, 2021).

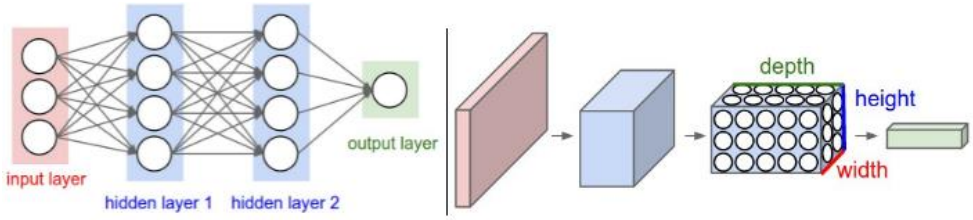


Figura 9 - Comparação entre rede neuronal e rede neuronal convolucional profunda (CS231n, 2021)

2.5.1.1 Camadas de Convolução

Ao realizar o processamento de uma imagem é necessário entender as suas características. Nós, seres humanos, conseguimos olhar para uma imagem e perceber o objeto na mesma, tal como a forma do objeto. As redes neuronais fazem esse processamento através dos pixéis da imagem.

Desta forma, uma imagem pode ser traduzida numa matriz de pixéis, como na Figura 10. Cada valor e a comparação com valores vizinhos permite à rede entender o que encontra nas imagens.

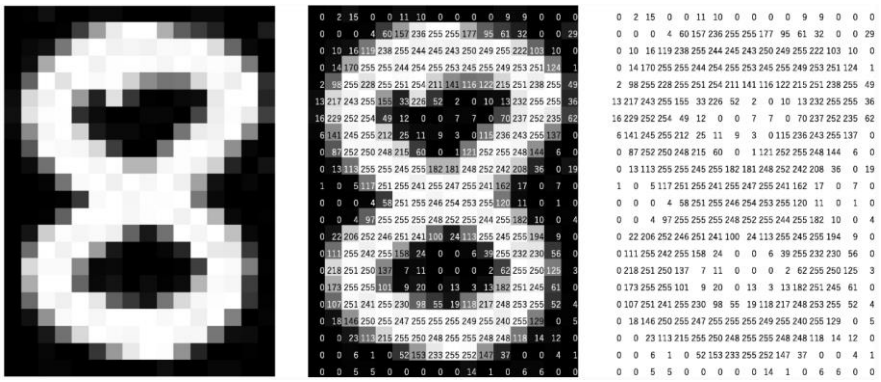


Figura 10 – Matriz de pixéis (Yamashita et al., 2018)

Uma imagem com melhor resolução, ou seja, com mais detalhes significa que tem um maior número de pixéis. Os pixéis podem variar entre 0 e 255 de forma a representar o nível de intensidade (Yamashita et al., 2018).

Para a leitura destes pixéis, comparação entre pixéis vizinhos, as camadas de convolução aplicam máscaras/filtros de convolução que dividem a imagem em pequenas partes, como na Figura 11. Estas pequenas regiões são chamadas de campos recetivos e são utilizadas por outras camadas (Singh, 2020) (Khan et al., 2020).

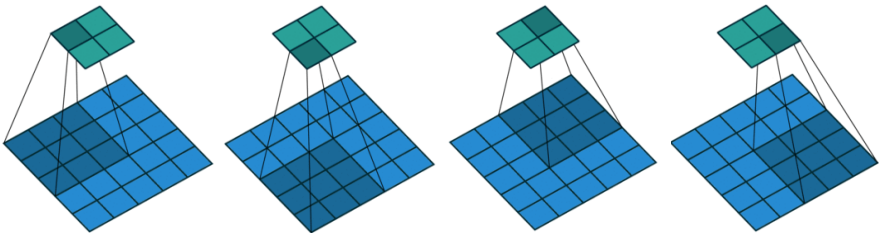


Figura 11 - Convolução da imagem com um filtro (Pai, 2020)

A convolução na sua forma mais básica, onde x representa os dados de entrada e f o filtro aplicado, pode ser representada através da seguinte expressão matemática (Singh, 2020):

$$z = x * \mathrm{f} \tag{3}$$

As camadas de convolução são consideradas operações locais e podem ser catalogadas pelo tipo de operação de convolução, tamanho dos filtros e outros aspetos (Khan et al., 2020).

2.5.1.2 Camadas de Agrupamento

As camadas de agrupamento ou camadas de redução de amostra (*down-sampling*) são operações locais que, como o nome indica, agrupam informação semelhante na vizinhança do campo recetivo (Khan et al., 2020). Este agrupamento de informação permite extrair mapas de caraterísticas que são considerados invariantes a deslocamentos translacionais e distorções. Deste modo, a quantidade de parâmetros é reduzida, tal como a complexidade e tamanho do

mapa, que por consequência reduz o *overfitting*, que faz aumentar o desempenho da rede (Yamashita et al., 2018).

O agrupamento pelo valor máximo, pela média, pela norma L2, por sobreposição e agrupamento de pirâmide espacial são exemplos de tipos de agrupamentos que podem ser executados nestas camadas. O agrupamento pelo valor máximo é o mais utilizado de todos e filtra as matrizes pelos valores máximos de cada quadrante, como descreve o esquema da Figura 12 (CS231n, 2021).

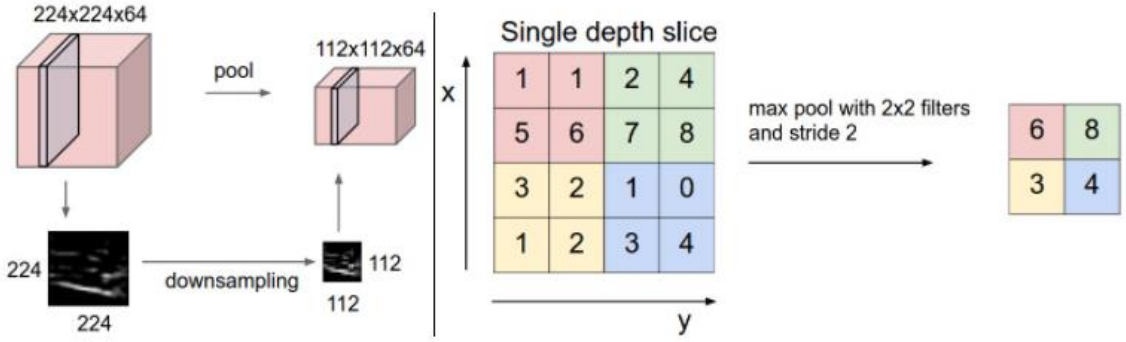


Figura 12 – Agrupamento de CNNs (CS231n, 2021)

O agrupamento, independentemente do tipo de operação, pode ser representado pela seguinte equação, onde Z_i^k representa o mapa de agrupamento da camada i do mapa de características k, F_i^k. O tipo de agrupamento é representado por $g_p(\_)$ (Khan et al., 2020):

$$Z_i^k = g_p(F_i^k) \tag{4}$$

2.5.1.3 Ativação

A função de ativação é uma função não linear que ajuda no processo de aprendizagem e na tomada de decisões, principalmente em casos de padrões complexos. Esta função pode ser representada pela seguinte fórmula matemática (Yamashita et al., 2018) (Khan et al., 2020):

$$T_i^k = g_a(F_i^k) \tag{5}$$

Onde F_i^k representa o mapa de características (campo recetivo), $g_a(\_)$ expressa a aplicação de uma função não linear – função de ativação e o T_i^k é o devido resultado transformado (Khan et al., 2020).

Existem diferentes operações de ativação, como a *ReLU* (*Rectified Linear Unit*), a *sigmoid* e a tangente hiperbólica (Khan et al., 2020), respetivamente representadas nos gráficos da Figura 13.

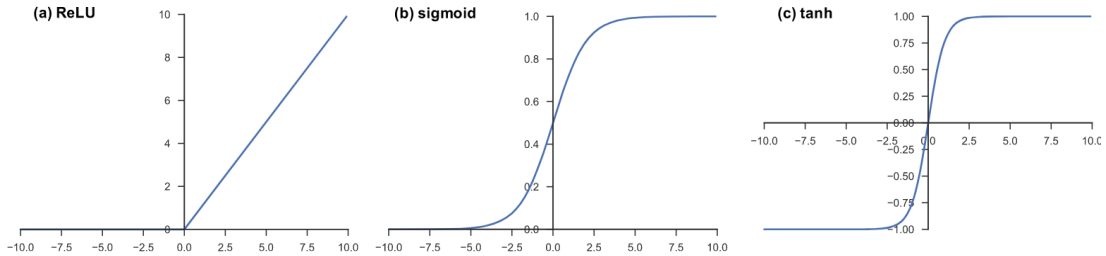


Figura 13 – Funções de ativação (Yamashita et al., 2018)

No entanto, a primeira é mais vantajosa devido ao seu cálculo mais simples e a não apresentar problemas de convergência ou variação abrupta da derivada como ocorre em outras (Khan et al., 2020) (Ramos, 2021c). A função *ReLU* coloca o limiar máximo como 0, significando que os valores negativos passam a 0, sem alterar o volume, usando a seguinte função (Yamashita et al., 2018):

$$f(x) = \max(0, x) \tag{6}$$

Este tipo de função é normalmente utilizado após uma camada de convolução (transformação linear), mas a sua aplicação é também importante na última camada. Para esta, outro tipo de função de ativação é considerado de acordo com a necessidade e objetivo do algoritmo. Na seguinte tabela, Tabela 1, são apresentados os tipos de funções mais aplicados na última camada de acordo com o objetivo (Yamashita et al., 2018).

Tabela 1 – Funções de ativação da última camada mais utilizadas de acordo com o objetivo final da rede neuronal

Objetivo	**Função de ativação**
Classificação binária	*Sigmoid*
Classificação multiclasse *OvR*	*SoftMax*
Classificação multiclasse *OvO*	*Sigmoid*
Regressão para valores contínuos	Identidade

No caso da classificação multiclasse existem dois tipos, o *OvR* (*One-vs-Rest*) que divide a classificação multiclasse em classificações binárias por classe e o *OvO* (*One-vs-One*) que realiza classificações binárias por cada par de classes (Brownlee, 2021).

2.5.1.4 Normalização de lote

A normalização do lote é uma função regularizadora utilizada para resolver problemas relacionados com possíveis alterações na distribuição de valores de unidades ocultas, que pode causar constrangimento na convergência. No caso de um mapa de caraterísticas transformado, F_i^k, a normalização do mesmo pode ser representada pela seguinte fórmula (Khan et al., 2020):

$$N_i^k = \frac{F_i^k - \mu_B}{\sqrt{+\varepsilon}} \tag{7}$$

Onde N_i^k representa o mapa normalizado, μ_B a média de um mapa de um lote, σ_B^2 a variância e ε é utilizado para evitar a divisão por 0 (Khan et al., 2020).

Apesar de algumas redes utilizarem funções de normalização, as camadas de normalização são pouco utilizadas atualmente já que a sua contribuição para a melhoria de desempenho da rede foi considerada mínima (CS231n, 2021).

2.5.1.5 Dropout

Dropout é uma técnica utilizada para a regularização da rede, onde são selecionadas e descartadas, de forma aleatória, unidades e conexões com determinadas probabilidades. O descarte das conexões é feito pela atribuição do valor 0 a determinadas ativações durante o processo de treino.

Este tipo de rede neuronal costuma ter múltiplas ligações, como se pode observar no primeiro esquema da Figura 14, o que aumenta a probabilidade de *overfitting*. Esta técnica é importante, porque permite reduzir o número de conexões e, por tal, melhorar o desempenho da rede e ajudar na redução de *overfitting* (Khan et al., 2020) (Szeliski, 2021).

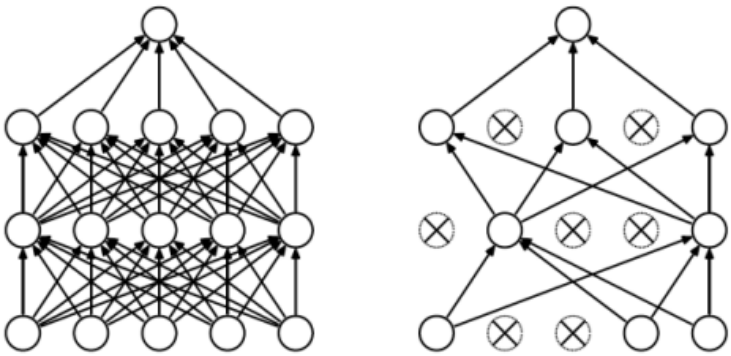


Figura 14 – *Dropout* (Szeliski, 2021)

2.5.1.6 Camada totalmente conectada

A camada totalmente conectada, normalmente a última camada da rede, é uma operação global ao contrário das camadas anteriores (camadas de convolução e agrupamento). Esta camada tem como função principal a classificação e, para tal, analisa de forma global os mapas de características extraídos pelas outras camadas (Khan et al., 2020).

2.5.2 Rede neuronal recorrente

Uma rede neuronal recorrente é uma rede artificial com a habilidade de lidar e reconhecer sequências. Devido a esta caraterística, este tipo de rede é normalmente utilizado para problemas de sequência de tempo, textuais e de áudio, tais como o reconhecimento de discurso, reconhecimento de caligrafia, análise sentimental e legenda de imagens (D. Gupta, 2017) (Jones, 2017) (Pai, 2020).

Enquanto as redes neuronais convolucionais são *feed-forward,* as redes neuronais recorrentes utilizam *feedback* nas camadas escondidas, também conhecido pela memória da rede (IBM Cloud Education, 2020) (Pai, 2020). Os esquemas seguintes representam a diferença entre redes recorrentes e redes *feed-forward*.

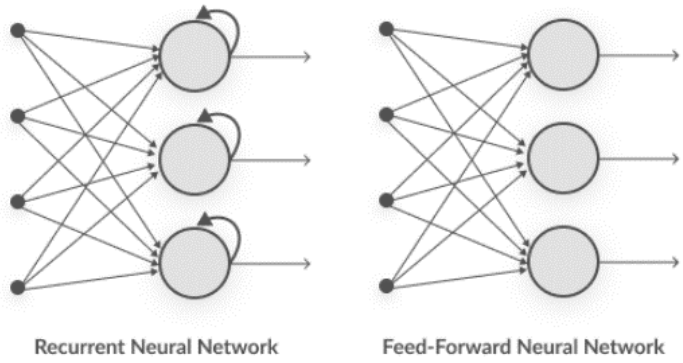


Figura 15 – Comparação da arquitetura de redes neuronais recorrentes com arquitetura *feed-forward* (Pai, 2020)

Uma *RNN* pode ser representada por duas vistas diferentes, *Rolled* com a vista de toda a rede e *Unrolled* com a representação dos vários momentos com o passar do tempo, tal como representado na Figura 16 (IBM Cloud Education, 2020). No caso do domínio do problema deste documento, uma vista *Rolled* representa o jogo por inteiro (sequência de eventos futebolísticos) e a vista *Unroled* da *RNN* seria cada momento/evento associado a um tempo.

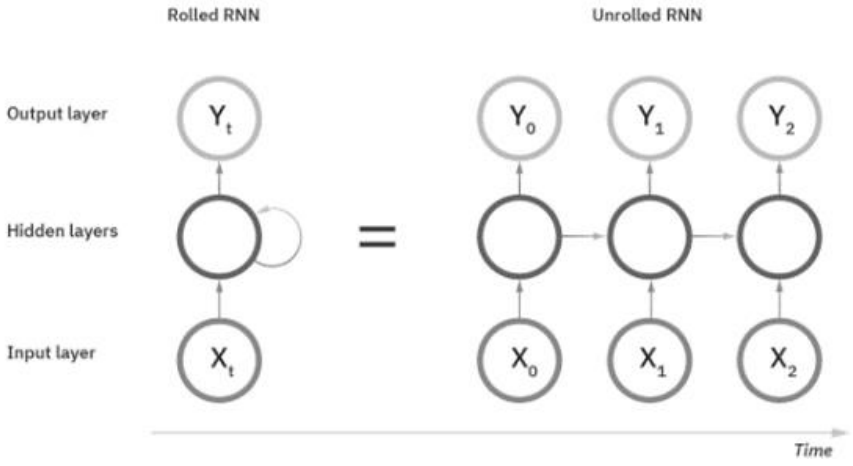


Figura 16 – Vistas de representações de *RNNs* (IBM Cloud Education, 2020)

Apesar de não ter as camadas de convolução das redes neuronais convolucionais, este tipo de rede neuronal utiliza funções de ativação tais como as funções mencionadas na secção de Ativação das *CNN*.

Um exemplo bastante popular deste tipo de redes são as redes *LSTM* (memória de curto e longo prazo), maioritariamente utilizadas para reconhecimento de linguagem gestual e análise de texto (Jones, 2017) (Ramos, 2021b).

2.5.3 Outras redes

Para além destes dois tipos de redes neuronais profundas existem muitas outras, que tal como as anteriores são especializadas para certas tarefas e têm um bom desempenho para grandes volumes.

Dentro das redes neuronais profundas supervisionadas existem ainda as redes recursivas que têm uma estrutura hierárquica. E em termos das redes neuronais profundas não supervisionadas, existem as redes pré-treinadas não supervisionadas (*UPN*) que funcionam como filtros de agrupamento e que incluem redes como as de crenças profundas (*DNB*), redes adversárias generativas (*GANs*) e autocodificadores (Jones, 2017) (Ramos, 2021b).

2.5.4 Aprendizagem por transferência

A aprendizagem por transferência, *TL*, é uma técnica muito popular com as redes neuronais profundas, pois o desempenho destas depende de grandes conjuntos de dados de treino e períodos de treino (Litjens et al., 2017). Esta técnica permite pré-treinar uma *DNN* com um *dataset* de maior dimensão, como por exemplo o *ImageNet* e depois adaptar esta para o objetivo pretendido com uso de um dataset mais pequeno e customizado ao problema (Yamashita et al., 2018) (Janiesch et al., 2021b).

Esta técnica de transferência de conhecimento pode ser aplicada de três formas diferentes, uso de uma rede pré-treinada, método de extração de recursos fixos e método de ajustes/retoques, respetivamente representados nos esquemas da figura seguinte para redes neuronais convolucionais.

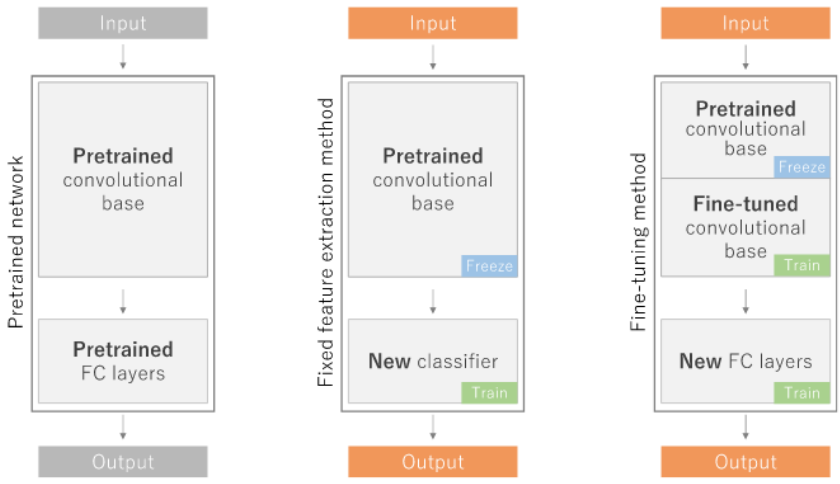


Figura 17 – Formas de aplicação da técnica de aprendizagem por transferência (Yamashita et al., 2018)

A primeira forma representada na Figura 17, uso de redes pré-treinadas, como o nome indica, refere-se à utilização de modelos pré-treinados por outros desenvolvedores (F. Li, 2020). Para

alcançar melhores resultados podem ainda ser utilizadas as outras formas mencionadas em seguida (Jones, 2019). Como exemplos destes modelos públicos temos a *AlexNet*, *ResNet*, *Inception, VGG*, *R-CNNs* e as próprias *frameworks* de implementação de *DL* tem os seus próprios modelos pré-treinados (Yamashita et al., 2018). O modelo de deteção de objetos e o modelo de *YOLO* da *TensorFlow* e o modelo de descoberta de sentimentos da *PyTorch* são exemplos dessas redes pré-treinadas (Jones, 2019).

O método de extração de recursos fixos refere-se à utilização de uma base pré-treinada, ou seja, a utilização de camadas de entrada e extração, com os respetivos pesos, usadas para um determinado problema e aplicação de uma camada de classificação para o domínio do problema a resolver. Este método tem um melhor desempenho quando o domínio do problema das camadas base e da última camada se assemelham (Jones, 2019).

Para casos em que os domínios dos problemas são menos semelhantes e necessitam de capturar diferentes características, surgiu uma alternativa com base no método anterior. Nesta alternativa também são utilizadas as camadas pré-treinadas e uma nova camada de classificação, no entanto são adicionadas novas camadas ajustadas especificamente para o domínio. Estes retoques são feitos de acordo com o erro de classificação, por retropropagação do erro (Jones, 2019) (F. Li, 2020).

2.6 Modelos de Visão Computacional

Como mencionado no capítulo de Aprendizagem por transferência, com a introdução de aprendizagem profunda e a utilização de métodos de aprendizagem por transferência surgiram alguns modelos públicos, principalmente para resolução de problemas de visão computacional.

Neste capítulo são mencionados alguns desses modelos que surgiram e que foram utilizados como base por outros.

2.6.1 R-CNN

As *R-CNNs,* redes neuronais convolucionais baseadas na região, surgiram em 2014 de modo a resolverem os problemas dos algoritmos de deteção de objetos devido à quantidade de regiões a analisar numa imagem. Estas redes aplicam métodos de pesquisa seletiva para obtenção de regiões candidatas e *CNN* para localizar e segmentar objetos (Girshick et al., 2014).

Como representa o esquema da Figura 18, este tipo de rede recebe uma imagem e numa primeira etapa extrai cerca de 2 mil regiões candidatas, que de seguida são utilizadas para gerar vetores de características através de uma rede convolucional na segunda etapa. Por último são utilizadas *SVMs* para classificar cada vetor e, por sua vez, cada região (Girshick et al., 2014).

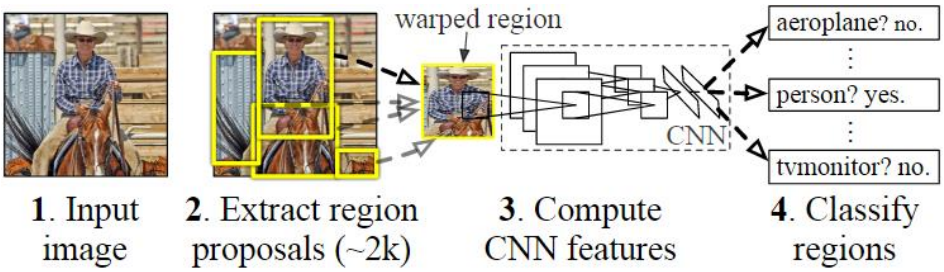


Figura 18 – Modelo *R-CNN* (Girshick et al., 2014)

2.6.2 Fast R-CNN

Em 2015 foi proposta a *Fast R-CNN,* uma versão melhorada da *R-CNN*, pelo autor *Ross Girshick*, também presente no artigo das *R-CNNs* originais. Este sugeriu um processo de deteção e classificação de duas etapas (Girshick, 2015), como representado no esquema da Figura 19.

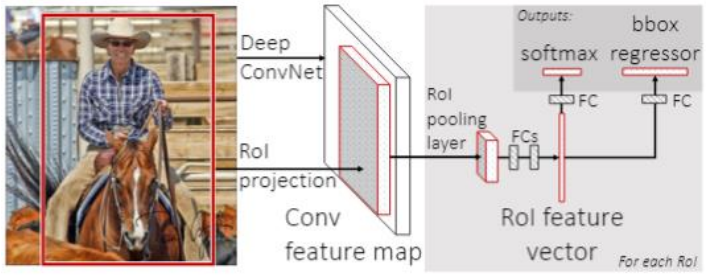


Figura 19 – Modelo *Fast R-CNN* (Girshick, 2015)

Este tipo de modelo recebe uma imagem e um conjunto de propostas de objetos de entrada. A imagem é processada como um todo por camadas de convolução e agrupamento, de modo a gerar os mapas de características e, por cada proposta de objeto fornecida, é utilizada uma camada de agrupamento de região de interesse (*RoI*) para extrair o vetor de características associado (Girshick, 2015). Os vetores são fornecidos a uma camada completamente conectada que se ramifica em duas camadas de saída (Girshick, 2015), visíveis no esquema da Figura 19:

- Camada de *softmax* – produção de estimativas de probabilidade *softmax* sobre a classe de geral (fundo) e *k* categorias de objetos, ou seja, produção de rótulos de classificação;

- Camada de regressão linear – produção de caixas delimitadoras (de objetos) dentro da imagem fornecida, através de um resultado conjunto de quatro valores reais (posição) para cada uma das *k* categorias.

2.6.3 Faster R-CNN

A *Faster R-CNN* foi proposta com o objetivo de melhorar as taxas de tempo referentes às *Fast R-CNNs*, principalmente as taxas de tempo da etapa de propostas de região. Sendo assim, foi

proposto um sistema dividido em dois módulos, onde é combinada uma rede de propostas de região (*RPN*) com um *Fast R-CNN* (Ren et al., 2015).

Como representado na Figura 20, a *RPN* é responsável por propor caixas delimitadoras de objetos propostos a partir da imagem de entrada. Esta atribui a cada caixa delimitadora uma probabilidade de exigência de objeto dentro da mesma, o que permite ao módulo seguinte saber as regiões relevantes (Ren et al., 2015).

O segundo módulo utiliza as zonas com maior probabilidade e é constituído pelo sistema de deteção *Fast R-CNN*, de modo a produzir as caixas delimitadoras finais e as respetivas classificações (Ren et al., 2015).

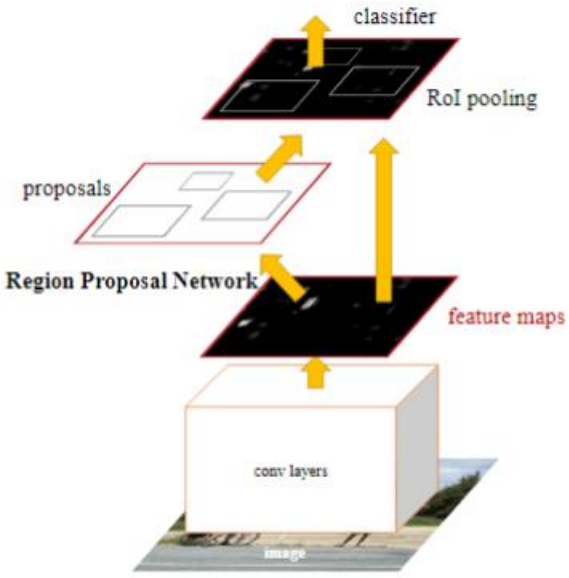


Figura 20 – Modelo *Faster R-CNN* (Ren et al., 2015)

2.6.4 Mask R-CNN

A *Mask R-CNN* foi proposta em 2017 com base na *Faster R-CNN*, o permitiu a esta alcançar melhores resultados em segmentação, deteção de objetos e pontos chaves e, por consequência, maior popularidade entre os desenvolvedores (He et al., 2017).

Este modelo utiliza como base a arquitetura *Faster R-CNN* e aplica uma ramificação adicional em cada *RoI,* para prever as máscaras de segmentação, em paralelo com a ramificação já existente para reconhecimento e produção das caixas delimitadoras dos mesmos, como representado na Figura 21 (He et al., 2017).

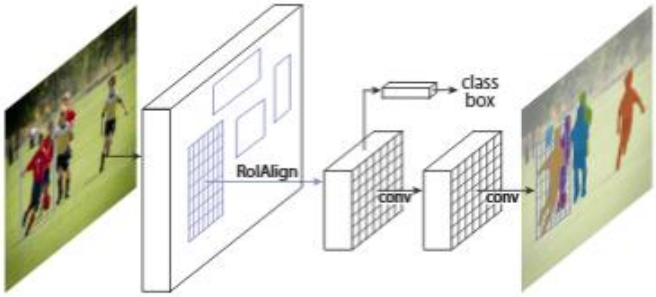


Figura 21 – Modelo *Mask R-CNN* (He et al., 2017)

Considera-se que as *Mask R-CNNs* combinam as vantagens dos métodos de agrupamento em pirâmide de características (*FPN*), redes totalmente convolucionais (*FCNs*), redes residuais e *Faster R-CNNs* (Brown, 2017).

2.6.5 YOLO

O modelo *YOLO*, "*You Only Look Once*", surgiu na mesma altura que a abordagem *Faster R-CNN,* mas com uma arquitetura mais simples com apenas uma fase, priorizando a velocidade/tempo de resposta à taxa de precisão (Redmon et al., 2015).

A arquitetura *YOLO* consiste na utilização de uma *CNN* para previsão simultânea das várias caixas delimitadoras possíveis e as respetivas categorias. Inicialmente a imagem é dividida por uma grelha *S* por *S,* como descrito no esquema da Figura 22, onde a responsabilidade de detetar cada objeto pertence às células da grelha com o centro do objeto. Essas células e as restantes são responsáveis pela produção de caixas delimitadoras (coordenadas do ponto central – *x* e *y*, altura – *h* e largura – *w*), o respetivo grau de confiança de presença de objeto dentro desta e a probabilidade do tipo de classe do objeto. Esta informação é depois utilizada para obter as previsões finais, ou seja, as caixas delimitadoras finais e as respetivas classes dos objetos (Redmon et al., 2015).

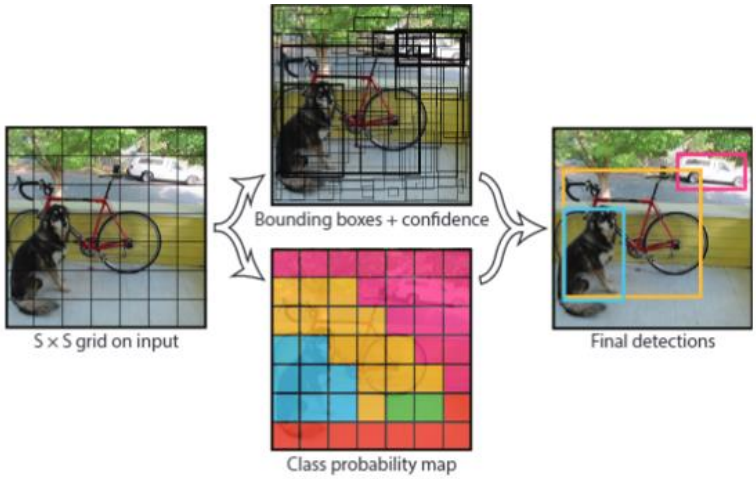


Figura 22 – Modelo *YOLO* (Redmon et al., 2015)

Este modelo permite processar cerca de 45 imagens por segundo (*FPS*) e conta com uma versão mais pequena do mesmo, *Fast YOLO*, que é capaz de processar 155 *FPS* e manter uma média de precisão considerada boa. No entanto, para problemas em que é necessário detetar objetos mais pequenos, principalmente em grupo, o *YOLO* fica abaixo da média (Redmon et al., 2015).

2.7 Datasets

Neste contexto, os *datasets* representam os conjuntos de dados a serem utilizados pelos algoritmos de aprendizagem automática para treino, validação e testes. Estes podem ser armazenados de diferentes formas, como por exemplo em base de dados, ficheiros e folhas de cálculo (Ramos, 2021a), e como mencionado no subcapítulo anterior, Aprendizagem por transferência, podem ser utilizados para pré-treinar as *DNNs*. Estes conjuntos de dados são de extrema importância principalmente para as redes neuronais profundas, pois estas dependem de grandes quantidades de dados de treino para o seu desempenho.

Os *datasets MNIST* (*Modified National Institute of Standards and Technology*) e *ImageNet* são dois dos maiores *datasets* de imagens existentes (Ramos, 2021a). O *MNIST* consiste em cerca de 60 mil imagens de treino e 10 mil imagens de teste de dígitos manuscritos (LeCun et al., 1998). O *ImageNet* consiste em vários subconjuntos com imagens de diversas categorias, sendo um dos subconjuntos mais utilizados o *dataset* de classificação e localização *ILSVRC* 2012-2017 (*ImageNet Large Scale Visual Recognition Challenge*) constituído por cerca de mil categorias de objetos, 1 milhão de imagens de treino, 50 mil imagens de validação e 100 mil imagens de teste. (Stanford Vision Lab et al., 2020).

Relativamente a vídeos e deteção de ações/eventos existem vários *datasets,* uns referentes a ações mais genéricas e outros específicos a um determinado tipo de ações. O *dataset THUMOS15* é genérico e composto por 200 vídeos de treino e 213 vídeos de testes com 20 categorias. Este foi seguido pelo *MultiThumos* com o mesmo conjunto de vídeo, mas com mais 45 categorias e um aumento do número máximo de ações distintas por vídeo de 3 para 25. Os datasets *ActivityNet* e *Hacs* são dos maiores *datasets* de vídeos existentes com quase 20 mil vídeos e 504 mil vídeos, respetivamente. Ambos são genéricos, mas com grande incidência em ações humanas, e dividem a informação em 200 categorias (Vahdani & Tian, 2021).

De acordo com o domínio do problema retratado neste documento foram explorados *datasets* relativos a eventos de futebol, especialmente *datasets* com segmentos de momentos de golos.

2.7.1 *Datasets* públicos em repositórios

Vários *datasets* podem ser encontrados por pesquisa online e leitura de artigos, no entanto existem repositórios abertos e motores de busca que facilitam essa procura.

O *Kaggle* permite desenvolvedores publicarem e disponibilizarem *datasets*, de modo a serem utilizados por outros desenvolvedores. Este repositório conta com mais de um milhão de

membros e várias competições de aprendizagem automática utilizam *datasets* presentes no mesmo (Ramos, 2021a).

Um motor de busca de *datasets* interessante pertence à *Google* (Google, 2022a), permitindo pesquisar diversos conjuntos de dados de vários sites/repositórios, incluindo do *Kaggle,* e autores com o uso de palavras-chave, como por exemplo “futebol” e “eventos”.

2.7.2 SoccerDB

O *dataset SoccerDB* surgiu em 2020 com cerca de 172 mil segmentos de vídeos de 346 jogos de futebol e com 11 categorias de eventos: plano de fundo, ferido, cartão vermelho/amarelos, passe, substituição, livre, canto, falta, golo, defesa e penalti. Este tem anotações não só referentes aos eventos mencionados, mas também para rastreamento de jogadores (Y. Jiang et al., 2020).

2.7.3 SoccerNet

SoccerNet foi idealizado em 2018 por *Giancola et al*. era composto por 500 jogos de futebol completos de ligas europeias ente 2014 e 2017, 764 horas de vídeos e 6 637 instâncias catalogadas em 3 eventos: golo, cartão amarelo/vermelho e substituição (Giancola et al., 2018).

Em 2021, foi proposta uma nova versão deste *dataset* com cerca de 300 mil anotações em 500 vídeos de jogos de futebol e mais 14 categorias de eventos futebolísticos para além dos 3 anteriormente captados. O *SoccerNet v2* propôs também à comunidade diversas competições, tal como as competições do *ImageNet* (Deliege et al., 2021). Toda a informação das competições pode ser encontrada em diversos artigos e no próprio *website* destes, sendo que existe uma competição em curso.

2.8 Trabalhos relacionados

Nesta secção é feita uma revisão de artigos, projetos em curso e soluções atualmente utilizadas relativas à deteção de eventos, principalmente de momentos de golo, em jogos de futebol.

O estudo dos artigos relacionados e projetos em curso permite perceber as abordagens até então propostas, os seus benefícios e desvantagens, as tecnologias existentes utilizadas por estes e obter dados de comparação para a avaliação da abordagem adotada neste documento.

Por outro lado, a revisão de soluções existentes sem grande contexto tecnológico-científico permite entender melhor a oportunidade e as utilidades da deteção de eventos em jogos de futebol.

2.8.1 Proposta de arquitetura de *Jiang et al. de 2017*

O artigo de *Jiang et al. de 2017* propõe uma solução baseada inteiramente em redes neuronais profundas para detetar eventos de futebol de forma automática. Os autores deste artigo, como descrito no esquema da Figura 23, propõem o uso de um modelo de **rede neuronal convolucional** para a extração de características dos eventos e uma **rede neuronal recorrente** para a relação de tempo das sequências de características, obtendo-se eventos (H. Jiang et al., 2017).

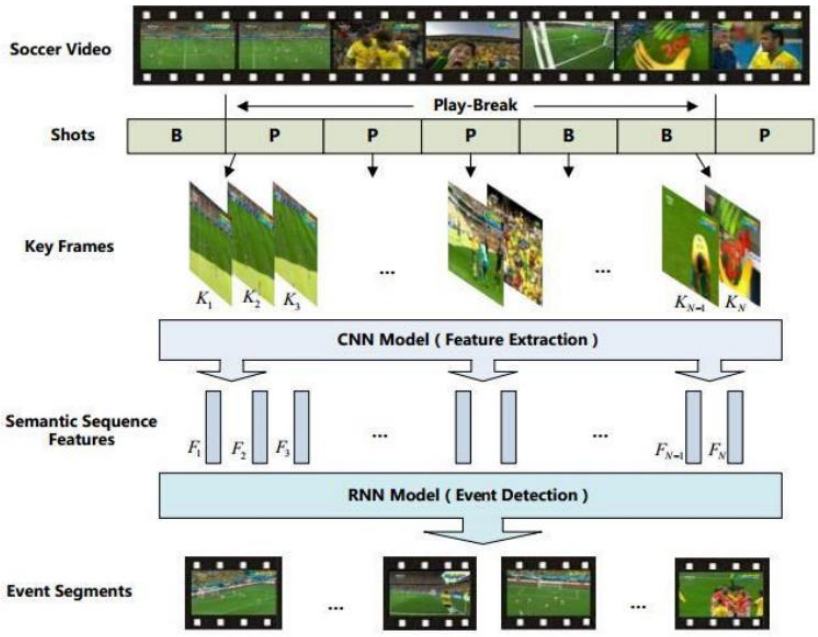


Figura 23 - Proposta de arquitetura por *Jiang et al.* (H. Jiang et al., 2017)

Como mencionado, nesta arquitetura a *CNN* foi utilizada para a obtenção de características distintas dos eventos presentes nas *frames*. Para o treino deste modelo foi utilizada a técnica de aprendizagem por transferência com o auxílio do *dataset ImageNet,* ou seja, a rede foi pré-treinada e depois adaptada e treinada por um *dataset* personalizado, para reconhecimento de características dos eventos definidos (canto, golo, oportunidade de golo e cartão). Para o reconhecimento destas caraterísticas, os autores dividiram as imagens em nove vistas. Estas vistas são classificadas pelo foco (longe, intermédio e perto), pela zona do campo e por quem se encontra na imagem (jogador, guarda-redes e árbitro). Para a implementação deste modelo os autores utilizaram *MatConvNet* (H. Jiang et al., 2017).

A rede neuronal recorrente, *RNN*, é considerada nesta proposta para resolver a relação temporal das sequências de visões e os eventos. Os autores usaram a *framework Keras* para a implementação desta rede neuronal (H. Jiang et al., 2017).

2.8.2 FootballAnalysis

O *FootballAnalysis* é um projeto *open-source* que tem como missão ajudar na automatização dos processos de análise de jogos de futebol e na extração de momentos de jogo e estatísticas (Akhaee et al., 2021a). Este projeto envolve diversos serviços, como representado na Figura 24, o desenvolvimento e *demos* para *PyTorch* destes podem ser encontrados no *GitHub* do projeto, com exceção da deteção de eventos baseada em vídeos (Akhaee et al., 2021b).

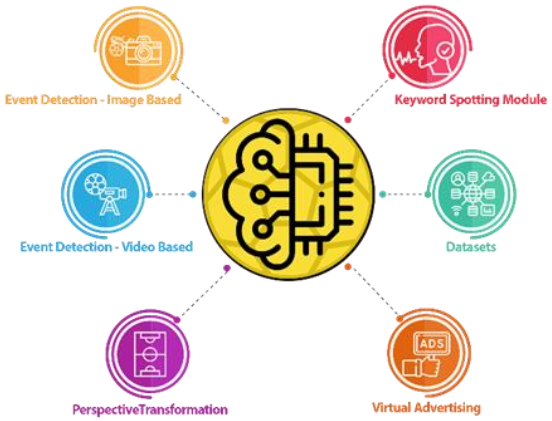


Figura 24 – Serviços *FootballAnalysis* (Akhaee et al., 2021b)

Este projeto tem como base a arquitetura de deteção de eventos no futebol proposta no artigo escrito por alguns dos desenvolvedores do projeto. Este artigo propõe uma arquitetura dividida em filtragem, classificação e classificação de granularidade fina (Karimi et al., 2021), como representado no esquema da Figura 25.

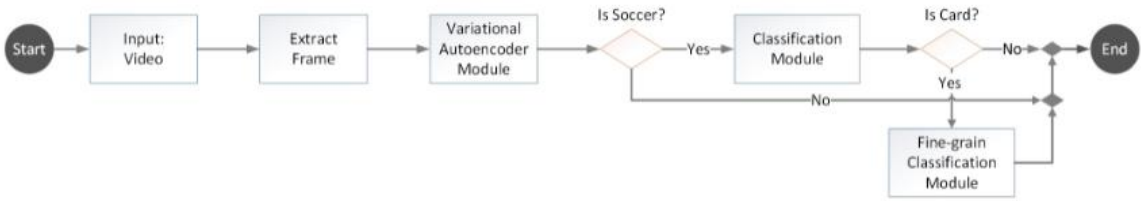


Figura 25 – Proposta de arquitetura por *Akhaee et al.* (Karimi et al., 2021)

No primeiro módulo, filtragem, o vídeo é dividido em *frames* e estas passam por um **autocodificador variacional (*VAE*)**, com utilização do *dataset SEV (*criado pelos autores*)*. Este valida se a imagem tem características semelhantes às imagens de eventos considerados (imagens do *SEV*) ou se pode ser descartada (Karimi et al., 2021).

O módulo de classificação recebe as *frames* relevantes e utiliza uma **rede neuronal convolucional** para classificar as mesmas em seis categorias de eventos (canto, livre, substituição, corte de bola, cartão e penálti). Se nenhum desses eventos forem cartões, a imagem é registada com esse evento. Se o evento encontrado for um cartão, a *frame* é processada por outro módulo (Karimi et al., 2021), como representado do esquema da Figura 25. Para este módulo foi utilizada a técnica de aprendizagem por transferência. Primeiramente

a *CNN* é treinada com auxílio de dados do *ImageNet* e, de seguida, é novamente treinada com o uso de dados do *dataset SEV* e ajustada (Karimi et al., 2021).

No último módulo apenas são analisadas imagens onde foi detetado um evento de cartão. Neste é feito o reconhecimento da cor do cartão (amarela ou vermelha) e depois fechado o fluxo com essa classificação. Este módulo utiliza conjuntos de dados do *dataset SEV* para o treino do modelo do mesmo (Karimi et al., 2021).

Por fim são analisadas as 14 *frames* vizinhas de uma *frame* de evento e, se mais de metade corresponderem à mesma categoria, estas e a própria são classificadas com esse mesmo evento. Um evento é considerado de *10* em *10* segundos e se acontecer o mesmo tipo de evento nesse espaço de tempo apenas será contado um (Karimi et al., 2021), o que levanta um problema nesta arquitetura.

2.8.3 Soluções atuais

De modo a melhorar a experiência do futebol, quer dentro do campo ou fora, a *FIFA* tem uma equipa dedicada à procura e experimentação de novas tecnologias e produtos. Esta equipa pretende trazer inovação juntamente com o futebol e possivelmente solucionar problemas atuais (FIFA, 2022a).

A partir desta equipa e do programa de qualidade da *FIFA*, já foram creditados alguns produtos relacionados com a deteção de eventos e objetos, para uso das equipas de futebol (FIFA, 2022a). Todos estes produtos e as empresas associadas encontram-se listados numa central de recursos da *FIFA,* conhecida como *Resource Hub* (FIFA, 2022b).

Dentro dos produtos relativos a reconhecimento e deteção de acontecimentos no campo, temos serviços como o *Coach160* do *Track160*, os sistemas de deteção de jogadores pertencentes à *Footovision* e à *Hawk-eye* e o *Tracab* da *ChyronHego* (FIFA, 2022b).

A *Hawk-eye* fornece serviços para diferentes setores e é reconhecida por ser pioneira em inovação na área de desporto, na sua transmissão e até mesmo no processo de arbitragem (*VAR*). Em termos de deteção em jogos de futebol, este é mais focado no rastreamento dos jogadores e bola com vista a análises e estatísticas. Este aposta bastante no uso de câmaras especializadas e na sua calibração, ou seja, em sistemas de rastreamento óticos (*OTS*) (Hawk-Eye, 2022).

Tracab tem três frentes, *Tracab Optical* que é um sistema de rastreamento ótico, baseado em rastreamento de câmara e técnicas de visão computacional, usado para a transmissão e análise de jogos em tempo real. *TRACAB GO*, um sistema de posicionamento global (*GPS*) que utiliza as redes de satélites, por exemplo para desportos que se realizam ao ar livre. E por último, *TRACAB RF*, um sistema de posicionamento local (*LPS*) com o uso de dispositivos pessoais (ChyronHego, 2021).

O *Track160* e *Footovision* focam-se em soluções dedicadas a futebol e ambas, para serviços de rastreamento, têm serviços de deteção automática e análise de eventos através de algoritmos de aprendizagem automática (Track160, 2021) (Footovision, 2021). O *Track160*, como representa a Figura 26, permite aos utilizadores verem diversos eventos, como golos e passes, e a informação de cada um desses eventos, por exemplo o jogador presente no mesmo. No entanto, este é dependente de vídeos de diversos ângulos, ou seja, uso de câmaras fixas 360 com operação remota ou diversas câmaras portáteis, disponibilizadas também pelo *Track160* (Track160, 2021).

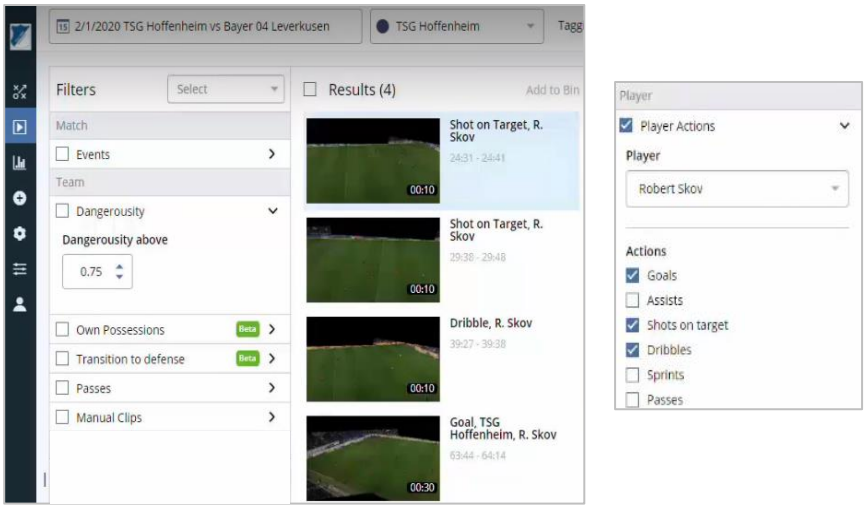


Figura 26 – Deteção e análise de eventos do *Track160* (Track160, 2021)

A *Footovision*, certificada no final do ano passado (FIFA, 2022b), ao contrário de todas as anteriores, afirma que não necessita de *hardware* dedicado e atua sobre qualquer tipo de entrada de vídeo. Estes utilizam visão computacional para a deteção de eventos e para obter informação sobre os jogadores, árbitros e posição da bola (Footovision, 2021).

2.9 Tecnologias

2.9.1 Python

Python é uma linguagem de programação de alto nível orientada a objetos com uma sintaxe simples e de fácil aprendizagem (Python Software Foundation, 2022). A simplicidade deste permite desenvolver programas complexos num curto período. Da mesma forma, a manutenção dos programas escritos nesta linguagem também é facilitada (Costa, 2020).

A simplicidade da sintaxe permite também uma melhor legibilidade do código, o que permite aos desenvolvedores seguir com mais facilidade exemplos e tutoriais disponibilizados e partilhar soluções e dúvidas dentro da comunidade (S. Gupta, 2021).

A simplificação é muito importante durante o desenvolvimento de algoritmos de aprendizagem profunda, mas *Python* ainda fornece outro benefício aos desenvolvedores. Esta linguagem tem a si associado um conjunto extenso de bibliotecas e *frameworks* relativos a aprendizagem automática, tais como (S. Gupta, 2021) (Costa, 2020):

- *Scikit-learn* – apropriado para o desenvolvimento de algoritmos básicos de aprendizagem automática;
- *Scikit-image* e *OpenCv* – utilizadas para tratamento e processamento de imagens, sendo que a *OpenCv* foi das mais utilizadas em aprendizagem automática antes da popularidade dos algoritmos e *frameworks* de aprendizagem profunda;
- *TensorFlow, Keras e PyTorch* – usadas para desenvolvimento de algoritmos de aprendizagem profunda. Por tal, estas vão ser mais aprofundadas e comparadas nos próximos subcapítulos.

Para além destas, podem ser encontradas outras bibliotecas que por sua vez são base para outras. Tal como *NumPy*, uma das bibliotecas mais populares de *Python*, que é utilizada como base da *framework TensorFlow* e *SciPy* ,usada em manipulação de imagem, e a base de *Scikit-learn* em conjunto com a *NumPy* (GeeksforGeeks, 2022).

De acordo com a estimativa de *SlashData*, em 2021 *Python* contava com cerca de 11.3 milhões de desenvolvedores (SlashData, 2021), como expressado no gráfico da Figura 27.

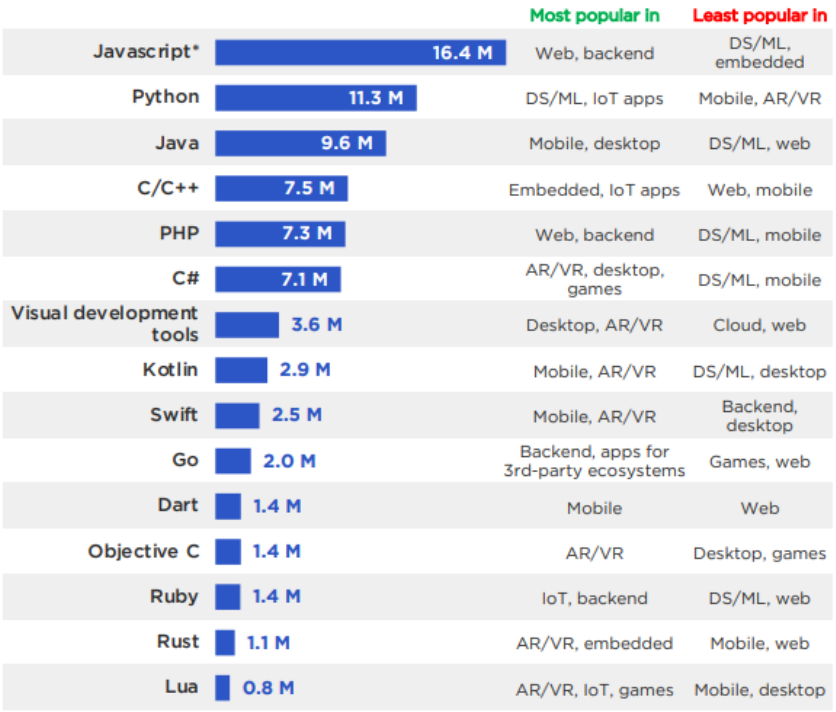


Figura 27 – Estimativa de *SlashData* de desenvolvedores mundiais por cada linguagem de programação (SlashData, 2021)

Neste gráfico observa-se ainda que *Python* é a linguagem mais utilizada no contexto de aprendizagem automática. E de acordo com o gráfico presente na Figura 28, dentro do conceito de IA e *ML* em comparação com outras ferramentas, como *Java*, *C++, R* e *Javascript, Python* é a linguagem mais pesquisada nos últimos cinco anos (Google, 2022b).

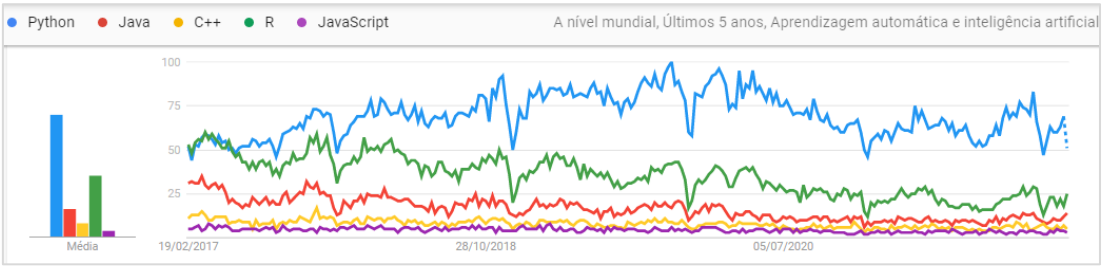


Figura 28 – Interesse a nível mundial das linguagens *Python*, *Java*, *C++, R* e *JavaScript* relativos a aprendizagem automática e inteligência artificial nos últimos cinco anos (Google, 2022b)

Devido aos benefícios mencionados e à sua comunidade extensa, esta linguagem é adotada para projetos de inteligência artificial por empresas com a *Apple, Microsoft, Meta e Google* (S. Gupta, 2021).

2.9.2 TensorFlow

TensorFlow é uma *framework* open-source para implementação de soluções de aprendizagem profunda lançada pela *Google AI* em 2015. Esta é conhecida pela sua vasta documentação e suporte, pela escalabilidade e com o novo lançamento, *TensorFlow* 2.0, pela sua usabilidade (TensorFlow, 2022j) (Waymo, 2022).

Esta *framework* é bastante flexível, com acesso a diversas bibliotecas, uso de modelos pré-treinados, suporte para várias plataformas e suporte de diferentes linguagens (*Python*, C*++, Java* e outras). *TensorFlow* é usado por organizações como a *Airbnb*, *Coca-Cola*, *Twitter* e pelo próprio desenvolvedor, *Google* (TensorFlow, 2022j).

2.9.2.1 Keras

Keras é uma *API open-source* de rede neuronal escrita em *Python*. Esta biblioteca foi desenhada para ser executada em *CNT*, *Theano* e *TensorFlow*. Sendo que o *TensorFlow* 2.0 já tem esta *API* integrada, *tf.keras* (TensorFlow, 2019). Esta é de fácil utilização e aprendizagem e permite aos utilizadores realizarem experiências com redes neuronais profundas (Atlas Systems, 2020) (KerasSIG, 2020).

Esta biblioteca permite escalabilidade e grande desempenho, sendo que atualmente é utilizada por empresas como a *NASA* e *Google*. No caso da *Google*, esta ferramenta é utilizada em projetos como o *Youtube* e em projetos ainda em desenvolvimento como o *Waymo*, relacionado com carros autónomos (KerasSIG, 2020) (Waymo, 2022).

2.9.3 PyTorch

PyTorch, tal como o *TensorFlow*, é uma *framework open-source* de aprendizagem profunda, no entanto, este foi desenvolvido pela *Meta AI* em 2016. Esta framework tem suporte em *Python* e é especialmente usada para problemas de processamento de linguagem natural (Atlas Systems, 2020).

Segundo o seu *website*, a *PyTorch* dá importância à flexibilidade, simplicidade e eficácia do uso da mesma. Por tal, esta tem uma sintaxe simples e apeladora aos utilizadores, permite o uso de modelos pré-treinados e de gráficos computacionais dinâmicos e acesso a múltiplas bibliotecas (PyTorch, 2022).

2.9.4 Comparação TensorFlow e PyTorch

As duas *frameworks* encontram-se a par uma da outra em termos de atualização da sua tecnologia e até mesmo nas suas funcionalidades, mas é possível fazer uma comparação das duas em aspetos como a sua facilidade de utilização, documentação e suporte da comunidade.

Em termos de facilidade de aprendizagem e usabilidade das ferramentas, a versão antiga do *Tensorflow* era considera a mais difícil de usar. No entanto, com a introdução da *API Keras* e o lançamento do *TensorFlow* 2.0, foram removidas várias redundâncias e incoerências e tornou a *framework* mais estável e mais prática para os utilizadores. *PyTorch* continua com a mesma abordagem, ou seja, semelhanças fortes à linguagem *Python* e um nível da *API* baixo, o que permite maior personalização por parte do utilizadores (Samaya et al., 2021).

PyTorch, ao contrário do *TensorFlow*, é mais dinâmico em termos de gráficos computacionais, permitindo a sua configuração, alteração e execução a qualquer momento (PyTorch, 2022) (TensorFlow, 2022j). Este permite também uma maior facilidade de realização de *debugging* para utilizadores familiarizados com *python*, pois utiliza os métodos de *debugging* da linguagem. A outra framework permite também um *debug* completo e claro, mas os utilizadores necessitam de primeiro aprender a utilizar o *debugger* deste (Samaya et al., 2021).

No entanto, em termos de implantação e suporte da comunidade, o *PyTorch* ainda fica um pouco aquém comparado com o *TensorFlow*. A framework da *Google* tem um módulo próprio, *TensorFlow Serve*, que permite a implantação dos modelos de forma prática (TensorFlow, 2022j). Já no caso da outra ferramenta apenas foi introduzido um modulo do género em 2020 (PyTorch, 2022).

Quanto à comunidade, apesar de ambos terem um vasto suporte, é possível encontrar mais tutoriais e outros recursos de *TensorFlow*, do que da framework lançada pelo *Meta*. O gráfico seguinte mostra uma porção dessas comunidades e recursos, retratando a percentagem de questões realizadas no *Stack Overflow* refentes às ferramentas.

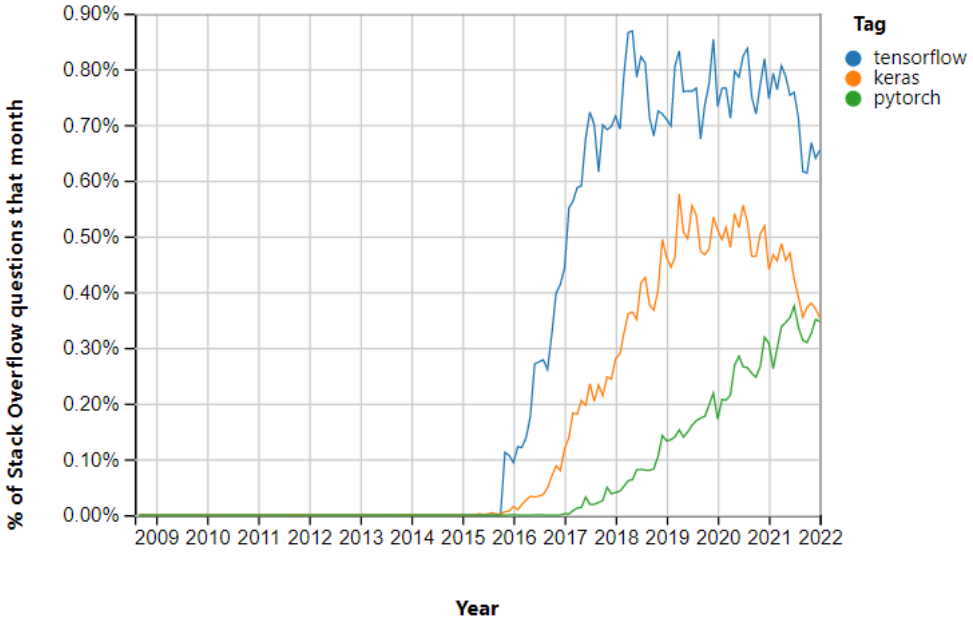


Figura 29 – Percentagem de questões realizadas no *Stack Overflow* referentes à *framework TensorFlow*, à *API Keras* e à *framework PyTorch* (Stack Overflow, 2022)

No gráfico da Figura 29 é ainda possível observar uma subida da percentagem de questões sobre *PyTorch,* demonstrando que esta se encontra em crescimento.

Atualmente, ambas as tecnologias são reconhecidas como boas *frameworks* de aprendizagem automática. Sendo que o *TensorFlow* se trata de uma *framework* mais robusta em termos de funcionalidades, bibliotecas e comunidade e o *PyTorch* uma ferramenta em desenvolvimento constante e mais simplista. Para facilitar a escolha entre as ferramentas foi utilizado o método de análise hierárquica (*AHP*), que se encontra no próximo capítulo.

3 Análise e *Design*

Neste capítulo é definido o evento de golo em jogos de futebol, assim como os eventos a captar para a sua classificação. De acordo com essa definição, é apresentado o desenho do modelo e o estudo das alternativas das arquiteturas para esse modelo.

De seguida é representado e discutido o fluxo da solução para deteção de golos em jogos e, por último, estudada a ferramenta a utilizar durante o desenvolvimento.

3.1 Golo

Na maioria dos casos, um golo pode ser explicado como a sequência dos seguintes eventos:

1. Oportunidade – remate na direção da baliza;
2. Momento de golo – entrada da bola na baliza;
3. Celebração – festejo da equipa que marcou e adeptos;
4. Pontapé de meio-campo – retorno a meio-campo por parte da equipa adversária para dar continuidade ao jogo.

No entanto, esta sequência pode não ser sempre verdadeira ou visível em vídeo. Por exemplo, o golo ou pontapé de meio-campo podem não ser captados pelas câmaras, ou pode não existir festejos por parte da equipa.

A existência em vídeo de cada um destes eventos por si só também não é suficiente para determinar a existência de golo. O início em meio-campo pode ocorrer em diversas circunstâncias, como início da primeira ou segunda parte. Inclusive, se um momento de golo for captado, este pode não ter sido classificado como golo, por exemplo, devido a fora de jogo.

Deste modo, foi definido que um golo, para ser classificado pela rede, deve conter o evento de momento de golo e o evento de pontapé de meio-campo. No caso de apenas ser detetado um dos eventos, este deve ser ignorado e descartado.

Considerou-se que em média os eventos de golo e pontapé de meio-campo no futebol duram aproximadamente 6 segundos. Este valor foi definido após visualização de alguns eventos e vídeos de futebol em geral.

3.2 Arquitetura do Modelo

De acordo com o estudo realizado na secção do Estado da Arte, os eventos necessários a classificar e o formato destes, foi desenhada a abordagem da Figura 30. Esta envolve a aplicação de *CNN e LSTM*, para garantir que as características extraídas em cada imagem são mantidas na sequência que ocorreram.

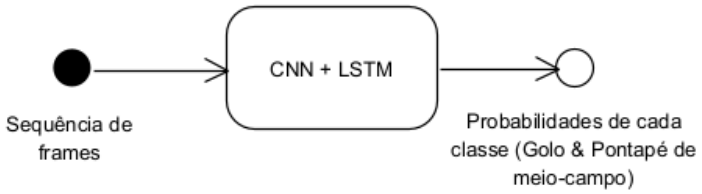


Figura 30 – Abordagem da arquitetura do modelo

Nesta, o *input* deve ser uma sequência de imagens, *frames* dos vídeos, de 3 dimensões (comprimento, altura e número de canais). O resultado deverá conter a probabilidade de cada classe/evento dessa sequência de *frames*.

Para a arquitetura deste tipo de modelo, com *CNN* e *LSTM*, foram exploradas três alternativas: *ConvLSTM, LRCN* ou a criação de um modelo para cada uma das redes e aplicação de modelos pré-treinados.

3.2.1 *ConvLSTM*

A rede *ConvLSTM* foi proposta em 2015 como solução para um problema de previsão de precipitação através da análise de sequências de imagens. Esta foi idealizada considerando as redes *FC-LSTM*. A arquitetura desta rede baseava-se na utilização de camadas recorrentes *ConvLSTM* em sequência (Shi et al., 2015).

Uma camada *Convolutional LSTM* (*ConvLSTM)* é uma variante da rede *LSTM*, onde algumas operações foram trocadas por outras convolucionais (Alexandre Xavier, 2019). Isto é, operações existentes em redes convolucionais foram integradas na rede *LSTM*.

Ao contrário da *LSTM*, que só aceita vetores unidimensionais como *input*, a *ConvLSTM* permite aceitar vetores tridimensionais (comprimento, altura e número de canais). Ou seja, esta é capaz de receber os dados em forma de imagem (Alexandre Xavier, 2019; Anwar et al., 2021). No caso de um modelo *ConvLSTM,* o input passa a ter 4 dimensões (número de imagens, comprimento, altura e número de canais), representativo de sequências de imagens (Anwar et al., 2021).

Os esquemas da Figura 31 representam a arquitetura de *LSTM* normal e a arquitetura de *ConvLSTM,* respetivamente.

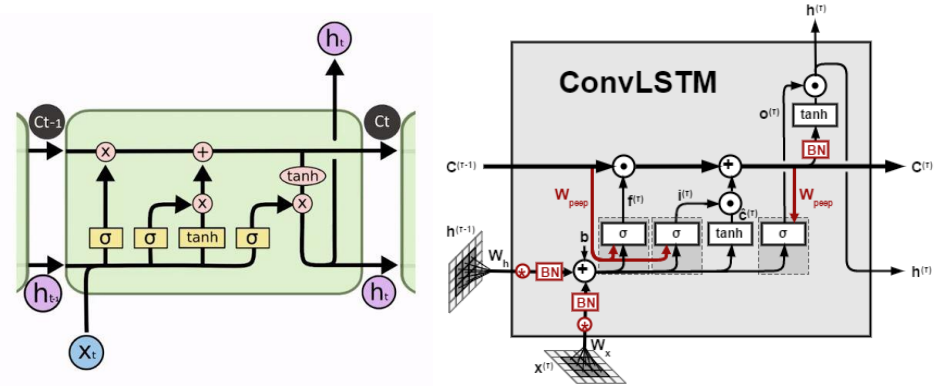


Figura 31 - Arquitetura LSTM (Olah, 2015) e arquitetura *ConvLSTM* (Alexandre Xavier, 2019)

Este tipo de arquitetura permite beneficiar do processamento de sequências temporais das redes recorrentes e das características de visão computacional das redes convolucionais. Isto é, ao aplicar este modelo a um vídeo, é possível extrair características de cada *frame* enquanto se mantém a sequência em que estas surgiram (Keras, 2022).

3.2.2 *LRCN*

A rede convolucional recorrente de longo prazo foi proposta em 2016 como uma combinação de *CNN* e *RNN* (Donahue et al., 2017). Na Figura 32, encontra-se representada a arquitetura deste tipo de rede.

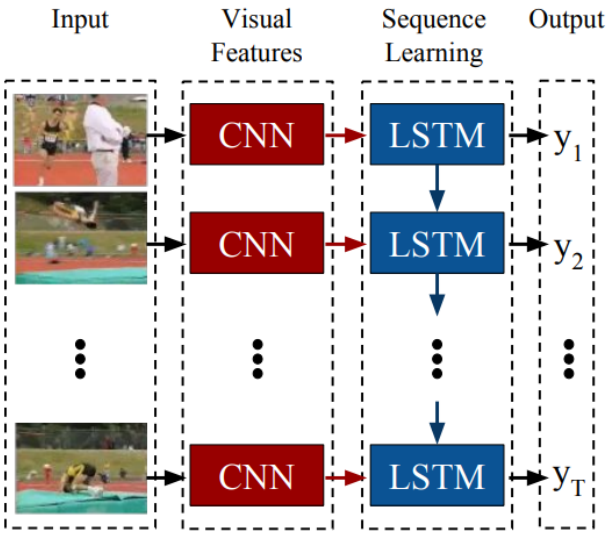


Figura 32 – Arquitetura *LRCN* (Donahue et al., 2017)

Nesta arquitetura, as camadas de convolução tratam de extrair as características e alimentar a(s) camada(s) *LSTM* para o processamento das mesmas em forma sequencial. Isto permite

guardar, analisar e classificar sequências pela ordem exata em que as características surgem durante esse excerto de vídeo (Anwar et al., 2021).

Tal como a alternativa anterior, esta rede beneficia da memória temporal das redes recorrentes e é ideal para reconhecimento de atividades/ações e descrição de vídeos (Donahue et al., 2017).

3.2.3 Uso de modelos pré-treinados

Outra alternativa seria a aplicação de modelos pré-treinados através de aprendizagem por transferência estudada na secção do Estado da Arte.

No entanto, não foram encontrados modelos públicos para redes *LRCN* ou *ConvLSTM*. Ou seja, a utilização de modelos pré-treinados implicava a necessidade de construção de uma rede do zero, por exemplo modelo *CNN* + modelo *LSTM*. Os benefícios das alternativas anteriores estudados eram perdidos, havia necessidade de mais recursos e os resultados não eram garantidos devido a problemas como *underfit* da aprendizagem por transferência. Deste modo, esta alternativa foi excluída e optou-se por testar e avaliar as outras duas alternativas (apenas um modelo).

3.3 Fluxo de deteção de Golos

Com os eventos e modelos estudados, foi desenhado o fluxo necessário para a deteção de golos em vídeos de jogos de futebol. Este fluxo encontra-se representado no diagrama de fluxo da Figura 33.

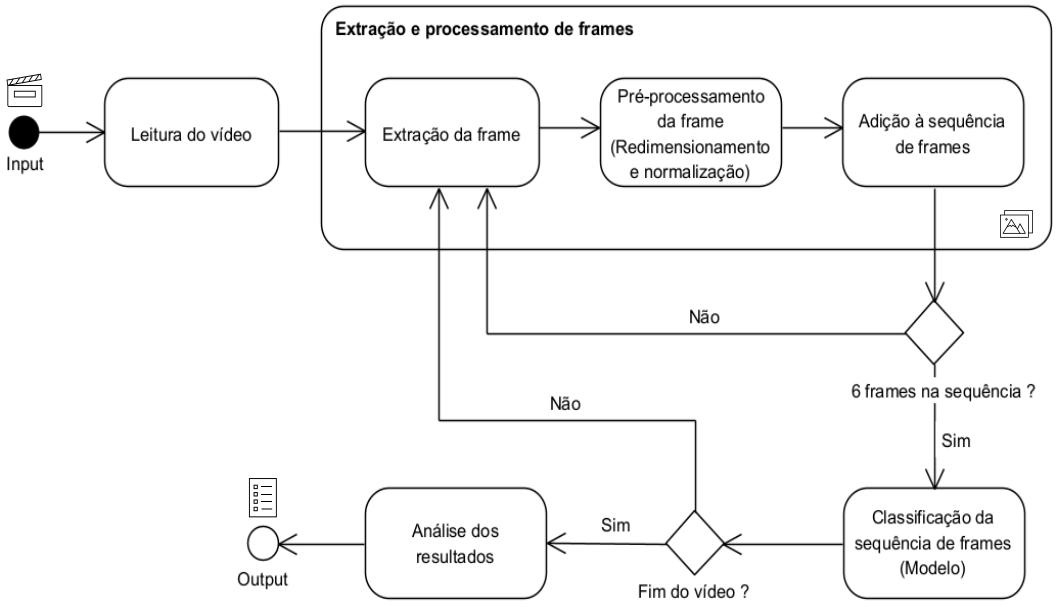


Figura 33 - Diagrama de fluxo da solução a desenvolver

Neste diagrama, foram definidas as fases de processamento, classificação e análise pelas quais o vídeo deve passar para devolver a informação sobre os golos.

O *input*, como representado figurativamente na Figura 33, deve ser o caminho para um vídeo de futebol. Este vídeo deve ser lido e devem ser extraídas *frames* a cada segundo em grupos de 6 (duração definida para o evento).

Cada uma das *frames* extraídas deve passar por um pré-processamento de uniformizar as mesmas (redimensionamento e normalização). Esta uniformização é importante, pois a qualidade dos vídeos utilizados para treino do modelo pode ser diferente do vídeo de *input* escolhido pelo utilizador. Também é importante definir o tamanho das imagens (comprimento e altura), para evitar incompatibilidades com o modelo.

Estas sequências de *frames* devem ser depois alimentadas ao modelo, de modo a este calcular as probabilidades dos dois tipos de eventos definidos (golo e pontapé de meio-campo). No caso de fim de vídeo, as probabilidades obtidas devem ser analisadas de modo a criar um ficheiro de texto com os minutos de jogo detetados como golo e a probabilidade associada.

3.4 Seleção da *framework* de desenvolvimento

A tecnologia a adotar durante o desenvolvimento foi uma das principais decisões de *design*. Para tal, foi adotado o método matemático de análise hierárquica (*AHP*) criado pelo professor *Thomas Saaty* em 1980. Este é utilizado para ajudar na tomada de decisão, através de critérios quantitativos e qualificativos (Saaty, 1990).

Numa primeira fase foi necessário definir a decisão a ser tomada (objetivo), os critérios a utilizar e as alternativas possíveis (Saaty, 1990). Para o caso do problema deste documento, e como previamente mencionado, o *AHP* é utilizado para a seleção de *framework* de trabalho mais adequada. Sendo as alternativas as últimas versões de duas *framework* competidoras, *TensorFlow* e *PyTorch,* como é possível observar na árvore hierárquica da Figura 34.

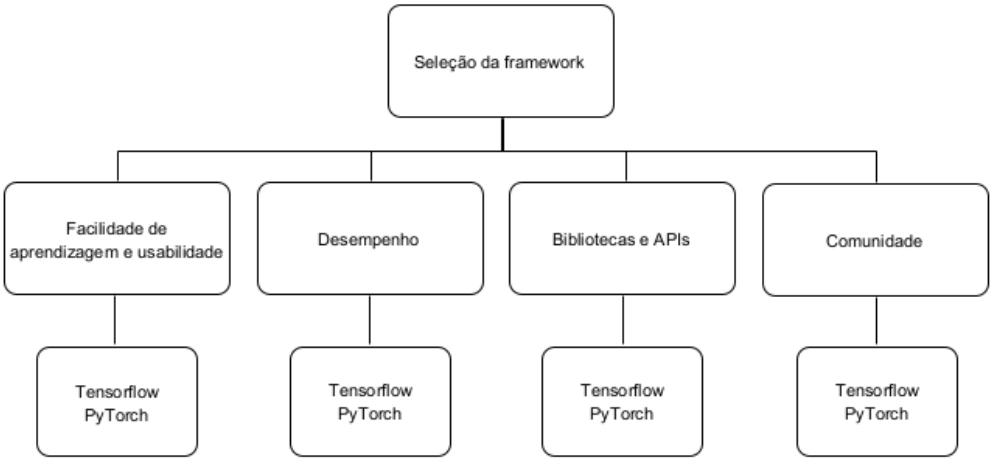


Figura 34 – Árvore hierárquica de decisão para seleção da *framework*

Assim, foram definidos e usados os quatro critérios seguintes:

- Facilidade de aprendizagem e usabilidade – usabilidade da ferramenta, tempo de aprendizagem, documentação e suporte para a aprendizagem;
- Desempenho – tempo de resposta e utilização de recursos;
- Bibliotecas e *APIs* - quantidade e qualidade das bibliotecas e *APIs* relativas a visão computacional (ex. *API* de deteção de objetos do *TensorFlow*);
- Comunidade – suporte da comunidade, pesquisas, artigos e questões feitas sobre a *framework* em questão.

Após estabelecer os critérios foi possível realizar a comparação destes, utilizando a escala fundamental de *Saaty* (Saaty, 1990). Na Tabela 2 é demonstrada essa mesma comparação e realizada a soma das mesmas.

Tabela 2 – Comparação de critérios

	Aprendizagem e usabilidade	Desempenho	Bibliotecas e *APIs*	Comunidade
Aprendizagem e usabilidade	1	2	1/3	2
Desempenho	1/2	1	1/2	3
Bibliotecas e *APIs*	3	2	1	6
Comunidade	1/2	1/3	1/6	1
Soma	**5**	**16/3**	**2**	**12**

A partir dos valores de comparação desta tabela, Tabela 2, foi possível construir a matriz de comparação, A:

$$A = \begin{bmatrix} 1 & 2 & 1/3 & 2 \\ 1/2 & 1 & 1/2 & 3 \\ 3 & 2 & 1 & 6 \\ 1/2 & 1/3 & 1/6 & 1 \end{bmatrix}$$

De seguida, foi realizada a normalização dos valores da matriz de comparação, A, e calculada a média aritmética de cada linha. Isto permitiu a obtenção da prioridade relativa de cada um dos critérios, ou seja, o peso de cada um destes (Saaty, 1990). A Tabela 3 serve como auxiliar para descrever os valores de comparação normalizados e o resultado da média aritmética de cada linha, prioridade relativa.

Tabela 3 – Normalização da matriz de comparação e prioridade relativa por critério

	Aprendizagem e usabilidade	Desempenho	Bibliotecas e *APIs*	Comunidade	**Prioridade relativa**
Aprendizagem e usabilidade	1/5	3/8	1/6	1/6	**0,2271**
Desempenho	1/10	3/16	1/4	1/4	**0,1969**
Bibliotecas e *APIs*	3/5	3/8	1/2	1/2	**0,4938**
Comunidade	1/10	1/16	1/12	1/12	**0,0823**

Com os valores obtidos na tabela anterior obteve-se o seguinte vetor de prioridades/próprio:

$$A = \begin{bmatrix} 1 & 2 & 1/3 & 2 \\ 1/2 & 1 & 1/2 & 3 \\ 3 & 2 & 1 & 6 \\ 1/2 & 1/3 & 1/6 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1/5 & 3/8 & 1/6 & 1/6 \\ 1/10 & 3/16 & 1/4 & 1/4 \\ 3/5 & 3/8 & 1/2 & 1/2 \\ 1/10 & 1/16 & 1/12 & 1/12 \end{bmatrix} \rightarrow X = \begin{bmatrix} 0{,}23 \\ 0{,}20 \\ 0{,}50 \\ 0{,}08 \end{bmatrix}$$

Onde A representa a matriz inicial de comparação, a segunda matriz representa a normalização de A e X o vetor de prioridades. De acordo com esse vetor o critério de facilidade de aprendizagem e usabilidade tem um peso de 0,23, o desempenho 0,20, o critério das bibliotecas e *APIs* um peso de 0,50 e a comunidade 0,08.

De forma a validar a consistência das prioridades relativas obtidas foi necessário calcular a Razão de Consistência (*RC*). Esta permite medir a consistência das avaliações em relação a grandes amostras de juízo aleatórias (Saaty, 1990).

Para realizar o cálculo de RC foi necessário começar por calcular o valor próprio, λ_{max}, através do vetor de prioridades, X, e da matriz de comparação previamente obtidos, considerando a seguinte fórmula (8) (Saaty, 1990):

$$A * X = \lambda_{max} * X \tag{8}$$

Ao substituir pelos valores previamente obtidos retira-se o seguinte:

$$\begin{bmatrix} 1 & 2 & 1/3 & 2 \\ 1/2 & 1 & 1/2 & 3 \\ 3 & 2 & 1 & 6 \\ 1/2 & 1/3 & 1/6 & 1 \end{bmatrix} * \begin{bmatrix} 0{,}23 \\ 0{,}20 \\ 0{,}50 \\ 0{,}08 \end{bmatrix} = \lambda_{max} \begin{bmatrix} 0{,}23 \\ 0{,}20 \\ 0{,}50 \\ 0{,}08 \end{bmatrix}$$

E após realizar a multiplicação da matriz de comparação, A, com o vetor de prioridade, X, foi obtido o seguinte resultado:

$$\begin{bmatrix} 0{,}95 \\ 0{,}81 \\ 2{,}07 \\ 0{,}35 \end{bmatrix} = \lambda_{max} \begin{bmatrix} 0{,}23 \\ 0{,}20 \\ 0{,}50 \\ 0{,}08 \end{bmatrix}$$

O valor próprio, λ_{max}, foi obtido a partir do cálculo da média da divisão de cada linha dos vetores resultantes, tal como de seguida representado (9):

$$\lambda_{max} = \frac{(0{,}95 \div 0{,}23) + (0{,}81 \div 0{,}20) + (2{,}07 \div 0{,}50) + (0{,}35 \div 0{,}08)}{4} \approx 4{,}17 \quad (9)$$

Uma vez calculado o valor próprio foi possível calcular o índice de consistência (IC) da seguinte forma (10) (Saaty, 1990):

$$IC = \frac{\lambda_{max} - n}{n - 1} = \frac{4.17 - 4}{4 - 1} \approx 0{,}06 \quad (10)$$

A razão de consistência, RC, é calculada pela divisão do índice de consistência, IC, já obtido, por um índice aleatório, IR, como mostra a seguinte fórmula (11) (Saaty, 1990):

$$RC = \frac{IC}{IR} \quad (11)$$

Para o índice aleatório foi utilizado um índice calculado para matrizes quadradas de ordem n, onde n é o número de critérios utilizados (Saaty, 1990). A Tabela 4 representa uma parte deste índice, sendo 4 o número de critérios utilizados neste estudo.

Tabela 4 – Índice aleatório calculado para matrizes quadradas de ordem n

1	2	3	**4**
0,00	0,00	0,58	**0,90**

Com o valor de IC calculado (0,06) e considerando o valor de índice aleatório como 0,90 foi possível calcular o valor da razão de consistência da seguinte forma (12):

$$RC = \frac{0{,}06}{0{,}90} \approx 0{,}07 < 0{,}1 \quad (12)$$

Como o valor de RC (0,07) é menor que 0,1 conclui-se que os pesos dados aos critérios são confiáveis e considerados consistentes.

Após obter e validar os valores das prioridades relativas avançou-se para a próxima fase, onde foram calculadas as matrizes de comparação, realizadas as suas normalizações e obtidos os vetores próprios de cada critério considerando as alternativas (Saaty, 1990). Os cálculos e matrizes para cada critério e alternativa estão representados em formato tabela, sendo estas seguidas pelo vetor resultante.

Facilidade de aprendizagem e usabilidade

Tabela 5 & Tabela 6 – Matriz de comparação, normalização e prioridade relativa do critério de facilidade de aprendizagem e usabilidade

Aprendizagem e usabilidade	TensorFlow	Pytorch
TensorFlow	1	1/2
Pytorch	2	1
Soma	**3**	**3/2**

Aprendizagem e usabilidade	TensorFlow	Pytorch	**Prioridade Relativa**
TensorFlow	1/3	1/3	0,3333
Pytorch	1/2	2/3	0,5833

$$X_{ferramenta} = \begin{bmatrix} 0{,}33 \\ 0{,}58 \end{bmatrix}$$

Bibliotecas e API

Tabela 7 & Tabela 8- Matriz de comparação, normalização e prioridade relativa do critério de bibliotecas e *APIs*

Bibliotecas e *APIs*	TensorFlow	Pytorch
TensorFlow	1	3
Pytorch	1/3	1
Soma	**4/3**	**5**

Bibliotecas e *APIs*	TensorFlow	Pytorch	**Prioridade Relativa**
TensorFlow	3/4	3/5	0,675
Pytorch	1/4	1/5	0,225

$$X_{ferramenta} = \begin{bmatrix} 0{,}68 \\ 0{,}23 \end{bmatrix}$$

Desempenho

Tabela 9 & Tabela 10 - Matriz de comparação, normalização e prioridade relativa do critério de desempenho

Desempenho	TensorFlow	Pytorch
TensorFlow	1	1
Pytorch	1	1
Soma	**2**	**2**

Desempenho	TensorFlow	Pytorch	**Prioridade Relativa**
TensorFlow	1/2	1/2	0,5
Pytorch	1/2	1/2	0,5

$$X_{ferramenta} = \begin{bmatrix} 0{,}5 \\ 0{,}5 \end{bmatrix}$$

Comunidade

Tabela 11 & Tabela 12- Matriz de comparação, normalização e prioridade relativa do critério de comunidade

Comunidade	TensorFlow	Pytorch
TensorFlow	1	5
Pytorch	1/5	1
Soma	6/5	6

Comunidade	TensorFlow	Pytorch	**Prioridade Relativa**
TensorFlow	5/6	5/6	0,8333
Pytorch	1/6	1/6	0,1667

$$X_{ferramenta} = \begin{bmatrix} 0{,}83 \\ 0{,}17 \end{bmatrix}$$

Terminando este processo, pode-se voltar a apresentar a árvore hierárquica de decisão, mas com os respetivos pesos, Figura 35:

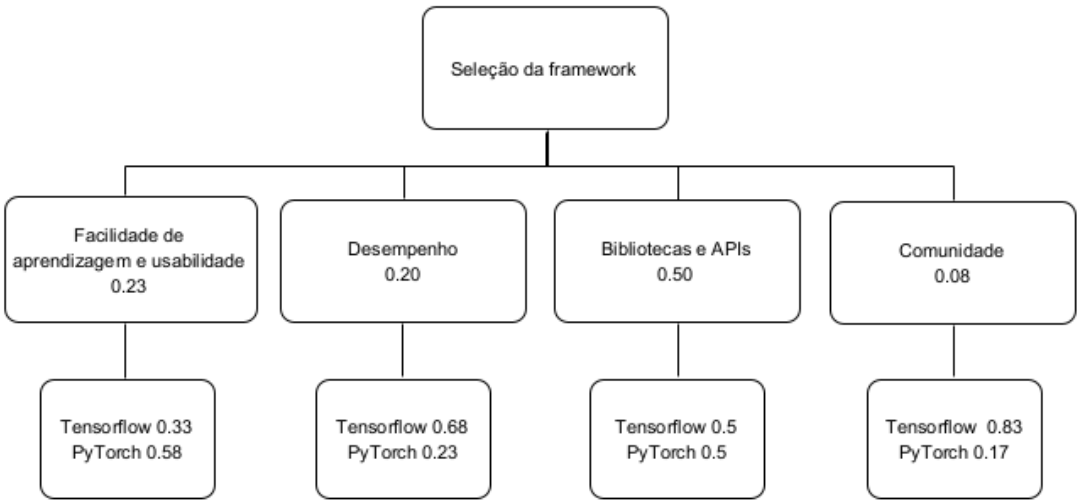


Figura 35 – Árvore hierárquica de decisão para seleção da framework

De modo a obter as prioridades compostas para as alternativas, foi formada uma matriz com os valores obtidos por cada critério-alternativa e multiplicado pelo vetor de prioridade relativa dos critérios, como representado:

$$\begin{bmatrix} 0{,}33 & 0{,}68 & 0{,}5 & 0{,}83 \\ 0{,}58 & 0{,}23 & 0{,}5 & 0{,}17 \end{bmatrix} * \begin{bmatrix} 0{,}23 \\ 0{,}20 \\ 0{,}50 \\ 0{,}08 \end{bmatrix} = \begin{bmatrix} \mathbf{0,53} \\ 0{,}43 \end{bmatrix}$$

Após este último cálculo conclui-se que ambas as tecnologias comprovaram ser competitivas em todos os aspetos, sendo que a *framework* ***TensorFlow*** (0,53) é considerada a melhor alternativa para este projeto relativo à *framework PyTorch* (0,43).

4 Implementação

Nesta secção é descrito o processo de implementação deste projeto, encontrando-se dividida em subcapítulos representativos do desenvolvimento: (i) instalação e configurações das ferramentas necessárias, (ii) experimentação e aprendizagem dessas ferramentas e modelos necessários, (iii) escolha e *download* do *dataset*, (iv) processamento dos dados do *dataset*, (v) criação e treino dos modelos, (vi) utilização dos mesmos para detetar golos e (vii) análise dos resultados. A implementação completa pode ser encontrada nos anexos (A, B, C, D, E e F).

4.1 Instalação

Neste subcapítulo são explicados os vários passos a serem seguidos de modo a conseguir experimentar e implementar modelos de *machine learning*, tais como a escolha da máquina, ferramentas e *frameworks* a instalar e configurações necessárias.

4.1.1 Máquina

Para a implementação deste projeto foi estudada qual a máquina a ser utilizada, devido à necessidade de capacidade de armazenamento (disco) – vídeos de treino e teste, processamento (*CPU* e memória *RAM*) – leitura e processamento de vídeos e processamento gráfico (*GPU*) – criação e treino do modelo.

Primeiramente foram exploradas máquinas virtuais na *Microsoft Azure* para este tipo de tarefas.

Tamanho da VM ↑↓	Tipo ↑↓	RAM (GiB) ↑↓	Custo/mês ↑↓
NC4as_T4_v3	GPU	28	518,34 €
NV4as_v4 ⓘ	GPU	14	292,41 €
NV8as_v4 ⓘ	GPU	28	584,83 €

Figura 36 – Custo das máquinas virtuais na *Microsoft Azure*

Contudo, como é possível observar na Figura 36, os preços das mesmas eram bastantes elevados mesmo para 14GB de *RAM*. Devido ao custo alto destas máquinas virtuais, não foi possível a sua utilização.

De seguida, foram exploradas e comparadas as máquinas físicas de modo a escolher qual a utilizar, como representado na Tabela 13.

Tabela 13 – Comparação de máquina físicas

	Fixo pessoal	**Portátil pessoal**	**Portátil de trabalho***
Processador	Intel(R) Core (TM) i5-10600 CPU @ 3.30GHz 3.31 GHz	Intel(R) Core (TM) i5-7300HQ CPU @ 2.50GHz 2.50 GHz	11th Gen Intel(R) Core (TM) i7-1195G7 CPU @ 2.90GHz 1.80 GHz
RAM	16.0 GB	8.00 GB	16.0 GB
Memória disco	SSD 465 GB	SSD 118 GB HDD 916 GB	SSD 476 GB
Placa Gráfica	NVIDIA GeForce GTX 1650 (4.0 GB dedicada – 8.0 GB partilhada)	NVIDIA GeForce GTX 1050 (2.0 GB dedicada – 3.9 GB partilhada)	Intel(R) Iris(R) Xe Graphics

*Recebido durante o processo de implementação.

Na tabela anterior, Tabela 13, observa-se que o terceiro portátil não tem uma placa gráfica dedicada e por isso foi logo excluído, tal como o computador que substituiu (antigo portátil utilizado – não representado na tabela). Quanto aos computadores pessoais, apesar do portátil ter mais memória de disco e, por isso, mais capacidade de armazenamento, o fixo tem um melhor processador, mais memória *RAM* e uma placa gráfica ligeiramente melhor (maior quantidade de *VRAM*). Desta forma, a primeira máquina, com maior capacidade de processamento gráfico e em geral, foi escolhida para ser utilizada durante a implementação.

Como suporte e aumento da capacidade de armazenamento, foi utilizado um disco externo com a capacidade máxima de 1TB. Durante a implementação, este disco serviu para armazenar os vários vídeos, de modo a libertar um pouco a memória da máquina.

4.1.2 *Python*

Para utilizar a linguagem *Python* e os comandos associados foi feita a transferência da última versão disponibilizada no site da mesma (Python, 2022). Para este projeto foi utilizada a versão 3.10.5.

4.1.3 *TensorFlow*

A instalação do *Tensorflow* requer o pacote *Microsoft Visual C++ Redistributable for Visual Studio 2015, 2017 and 2019* instalado e ter o *pip* atualizado (TensorFlow, 2022b). Desse modo, foi descarregado o pacote do site da *Microsoft* (Microsoft, 2022b), feita a instalação do mesmo e atualizado o *pip* através do seguinte comando na *powershell*: "pip install --upgrade pip".

Após terminar a atualização, foi possível instalar o *TensorFlow* através do comando "pip install tensorflow", como recomendado no site do mesmo (TensorFlow, 2022b).

4.1.4 *Notebooks*

O desenvolvimento poderia ser feito diretamente em scripts de *Python*, no entanto não seria tão intuitivo e percetível, pois seria necessário estar sempre a correr os mesmos por linha de comandos. Como tal, foram exploradas outras duas opções.

Os *notebooks* do *Google Colab* ou "Colaboratory", que permitem desenvolver e experimentar algoritmos de IA e outros métodos em *Python* a partir do navegador e, como tal, sem necessidade de configurações na máquina local. Esta opção permite ainda ter acesso a *GPU* remotos sem qualquer custo, guardar de forma simples e organizada todo o desenvolvimento e partilhar o mesmo de forma igualmente fácil (Google Colab, 2022). No entanto, foram detetados alguns problemas com os mesmos após algumas utilizações. Estes ficavam em "pausa" após algum tempo sem atividade, o que dificultava tarefas necessárias durante a implementação, como o treino de modelos, e tinham problemas com ligação a dispositivos externos, como por exemplo *webcams*.

Os *jupyter notebooks*, que apesar de não fornecerem *GPU* remotos e requererem instalações e configurações na máquina local, mantêm-se ativos e funcionais durante longos períodos sem atividade, tendo sido por isso a opção escolhida. A instalação dos mesmos foi feita via *powershell* com o comando "pip install notebook". O comando "jupyter notebook" é utilizado de modo a abrir uma página como a representada na Figura 37, que permite a utilização dos *notebooks* para desenvolvimento (Jupyter, 2022).

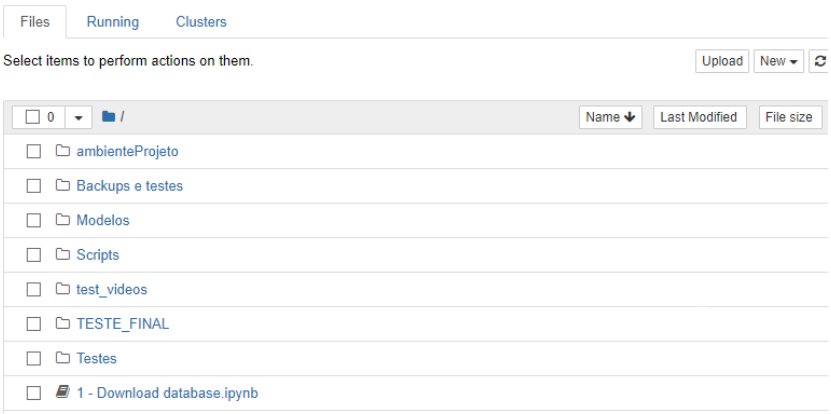


Figura 37 – Página inicial *Jupyter Notebooks*

A partir da página apresentada na Figura 37 é possível abrir o *notebook* em que se pretende trabalhar, criar *notebooks* e parar outros que estejam em execução e/ou a consumir recursos.

Como mencionado anteriormente, foi ainda necessário configurar e instalar outros recursos para a utilização e desempenho destes *notebooks*. Para garantir que os mesmos continuavam ativos, mesmo sem atividade por determinados períodos, o computador foi configurado para não suspender ou bloquear automaticamente. Já para a utilização de *GPU* em funções de IA, foram criados e configurados ambientes virtuais *conda* e instalado *tensorflow-gpu*, como explicado no próximo subcapítulo.

4.1.5 Ambiente virtual

Ao utilizar *jupyter notebook* é possível configurar o *kernel* a utilizar, de modo a escolher *CPU* ou *GPU*. No caso do uso de *CPU*, apenas é necessário criar um ambiente virtual na pasta do projeto e adicionar como *kernel* dos *notebooks* com os seguintes comandos (Renotte, 2021a):

```
python -m venv ambienteVirtual
pip install ipykernel
python -m ipykernel install --user --name=ambienteVirtual
.\tfod\Scripts\activate
jupyter notebook
```

Código 1 – Criação de ambiente virtual *CPU*

Para o uso de *GPU* são necessárias instalações extra para a criação do ambiente virtual, como por exemplo a instalação de *cuDNN* (*NVIDIA CUDA Deep Neural Network*). Esta é uma biblioteca responsável pela aceleração do *GPU* para o treino, compilação e avaliação de redes neuronais profundas. Para A instalação, foi seguido o guia de instalação de *cuDNN* da *NVIDIA*, sendo necessáriA a pré-instalação do *CUDA* e de bibliotecas como a "Zlib" (NVIDIA, 2022).

A utilização destas bibliotecas com *TensorFlow* requer cuidado com as versões, sendo que múltiplas versões ou versões incompatíveis não permitem prosseguir com o desenvolvimento (TensorFlow, 2022a). Para tornar este processo mais simples, tal como para instalação do *CUDA* e criação de ambientes *GPU*, optou-se por instalar *Miniconda* - versão simplificada e gratuita do instalador de *conda* (Anaconda, 2017).

Após todas as instalações necessárias, foi possível criar o ambiente virtual, utilizando o excerto de código apresentado em baixo (Código 2).

```
conda create -n ambienteVirtual-gpu
conda activate ambienteVirtual-gpu
pip install tensorflow-gpu
pip install ipykernel
python -m ipykernel install --user --name=ambienteVirtual-gpu
jupyter notebook
```

Código 2 – Criação de ambiente virtual *GPU*

Como para o caso do uso de *CPU*, esta criação deve ser realizada na pasta do projeto via linha de comandos e deve ser adicionado o novo *kernel*. Sendo a maior diferença, a forma como o ambiente é criado (*conda*) e a instalação do *tensorflow-gpu*.

O *kernel* e a sua configuração podem ser validados pelo código representado na seguinte Figura 38, de modo a garantir que o *GPU* se encontra ativo.

```
import tensorflow as tf
from tensorflow.python.client import device_lib
```

```
tf.test.is_built_with_cuda()
```

```
True
```

```
tf.config.list_physical_devices('GPU')
```

```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

```
device_lib.list_local_devices()
```

```
[name: "/device:CPU:0"
 device_type: "CPU"
 memory_limit: 268435456
 locality {
 }
 incarnation: 16621926873023532563
 xla_global_id: -1,
 name: "/device:GPU:0"
 device_type: "GPU"
 memory_limit: 1907854540
 locality {
   bus_id: 1
   links {
   }
 }
 incarnation: 8415198127611378451
 physical_device_desc: "device: 0, name: NVIDIA GeForce GTX 1650, pci bus id: 0000:01:00.0, compute capability: 7.5"
 xla_global_id: 416903419]
```

Figura 38 - Validação *kernel GPU* dentro do *notebook*

O código seguinte (Figura 39) foi também adicionado de modo a garantir que apenas é atribuída a memória de *GPU* consoante o seu uso. Sendo que por defeito o *TensorFlow* atribui grande parte de toda a memória para qualquer processo, para evitar fragmentação e aumentar a eficiência dos recursos (TensorFlow, 2022i). No entanto, para este projeto e recursos utilizados, a atribuição quase total da memória não era benéfica.

```
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    try:
        # Currently, memory growth needs to be the same across GPUs
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
            logical_gpus = tf.config.list_logical_devices('GPU')
            print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
    except RuntimeError as e:
        # Memory growth must be set before GPUs have been initialized
        print(e)
```

```
1 Physical GPUs, 1 Logical GPUs
```

Figura 39 – Ativação de crescimento de memória *GPU* (TensorFlow, 2022i)

4.2 Experimentação

Numa fase inicial do desenvolvimento foram realizados alguns tutoriais e minicursos, de modo a adquirir conhecimento e experiência com as ferramentas e linguagem escolhidas. Estes variaram ao longo do conhecimento obtido. Inicialmente foram seguidos guias do próprio *TensorFlow* para a utilização deste, seguido de tutoriais sobre conceitos base dos algoritmos de *ML*, experimentado outros algoritmos e, por fim, foram realizados tutoriais diretamente relacionados com o projeto (tema de futebol e categorização de ações/eventos).

Esta fase foi bastante importante, já que permitiu obter as bases necessárias para o desenvolvimento deste projeto e ainda explorar e aprender mais sobre esta área. Nos subcapítulos seguintes são apresentados os tutoriais/experiências realizadas com mais aplicabilidade ao longo da implementação.

4.2.1 Tutorial base

Após os primeiros testes com a *framework*, foi realizado um tutorial pensado para iniciantes de IA. Este está disponível em formato vídeo, apresentado por *Tim Ruscica* e também em *notebooks* do *Google Colab* (Ruscica, 2020).

A primeira parte deste tutorial é maioritariamente teórica, onde foi possível rever alguns conceitos estudados na secção do Estado da Arte, mas aplicados a exemplos. Após dados os fundamentais teóricos, foi possível explorar a utilização dos *notebooks* do *Google Colab* com a *framework TensorFlow*.

Antes da experimentação de algoritmos, foram ainda estudados e explorados conceitos base como *tensor* (quais os tipos existentes, como obter a sua forma e *rank,* e a aplicabilidade destes) e conceitos de treino como: *epochs*, validação e treino, *steps* de treino e *loss.*

Este tutorial permitiu experimentar e desenvolver algoritmos, de modo a explorar conceitos relativos a cada tipo, tais como: (i) regressão linear, (ii) redes neuronais – funções de ativação, *cost*/*loss*, gradiente descendente, *optimizers* e hiperparâmetros, (iii) *CNN* – pixéis, canais de cores, camadas (convolucionais, totalmente conectadas e de agrupamento), mapa de características e aplicação de *padding* e (iv) *RNN* e *LSTM* – conceitos relacionados com linguagem natural.

4.2.2 Tutorial de deteção de objetos

Este tutorial, apesar de um pouco fora do tema, permitiu entender melhor os conceitos de ambiente virtual e *kernel* e a utilização dos *jupyter notebooks.* Sendo este apresentado também em formato vídeo pelo próprio autor, *Nicholas Renotte*, e no *GitHub* (Renotte, 2021b).

O objetivo deste curso era a construção de uma solução de reconhecimento e categorização de sinais realizados com as mãos, utilizando conceitos de deteção de objetos. Como observável na Figura 40, foi possível implementar a solução e experimentar a mesma.

Figura 40 - Experimentação com algoritmo de deteção de objetos

4.2.3 Tutorial de reconhecimento de ações

Neste tutorial, foi possível experimentar ambas as alternativas de modelos estudadas na secção Análise e *Design* (*ConvLSTM* e *LRCN*). Sendo que o objetivo deste era explorar várias abordagens para o reconhecimento de ações humanas no quotidiano (Anwar et al., 2021). Tal como representado no teste da solução *LRCN* da Figura 41.

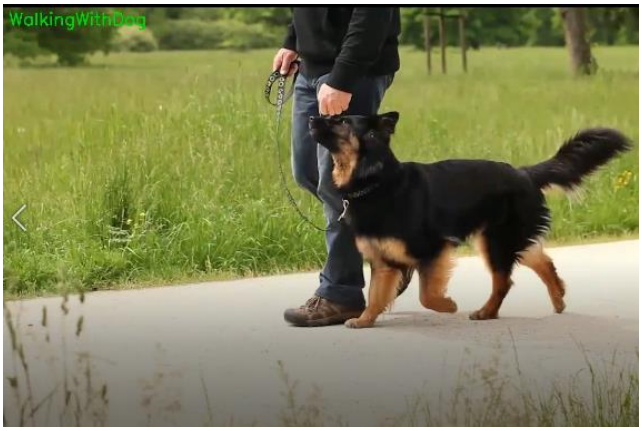


Figura 41 - Experimentação com algoritmo de reconhecimento de ações (*LRCN*)

4.3 *Dataset*

Após terminar a fase de experimentação com as ferramentas e algoritmos e antes de iniciar a implementação da solução, o estudo dos *datasets* feito no capítulo do Estado da Arte foi revisto, sendo estes explorados de uma forma mais profunda.

Aos dois principais *datasets* estudados anteriormente juntou-se um terceiro para esta fase - *Wyscout dataset,* presente no *Kaggle* previamente mencionado e com *website* próprio. Este *dataset* é mencionado noutros trabalhos relacionados e conta com eventos recolhidos em "sete competições (La Liga, Serie A, Bundesliga, Premier League, Ligue 1, FIFA World Cup 2018, UEFA Euro Cup 2016) ao longo de uma época" (Pappalardo & Massucco, 2019).

De modo a escolher o *dataset* a utilizar para o treino dos modelos e teste da solução, foi necessário comparar lado a lado os três *datasets* estudados – *SoccerNet* (versão 1 e 2), *SoccerDB* e o *Wyscout dataset.*

4.3.1 Comparação

Para a comparação dos *datasets* e de forma a escolher o que mais se aplicava à solução e mais benefícios traria a esta, foram considerados os seguintes critérios:

- Número de jogos
- Número de horas
- Número de anotações marcadas nos jogos
- Número de eventos
- Tipo de eventos anotados
- Qualidade dos vídeos

Na tabela seguinte, Tabela 14, pode-se observar a comparação dos quatro primeiros critérios mencionados na lista anterior através dos dados fornecidos pelos autores dos *datasets* (Deliege et al., 2021) (Y. Jiang et al., 2020) (Pappalardo et al., 2019).

Tabela 14 – Comparação *datasets*

	Vídeos/Jogos	**Horas**	**Ações/Eventos**	**Anotações**
***SoccerNet*-v1**	500	764	3	6 637
SoccerDB	346	668.6	11	37 715
SoccerNet-v2	500	764 (ações)	17	110 000 (ações; ao todo - 300 000)
Wyscout	1941	--	6 tipos + subtipos (golo é uma *tag* e não tipo/subtipo)	3 251 294

Numa primeira análise, o *dataset Wyscout* tem um maior número de jogos e anotações que os outros *datasets* estudados, mas este tem uma menor quantidade de tipos de eventos e encontra-se estruturado de forma diferente, pelo qual é necessário perceber o tipo de eventos de cada um.

O *SoccerNet-v1*, como previamente mencionado no Estado da Arte, apenas conta com três tipos de eventos – golo, cartão amarelo/vermelho e substituição, sendo que a versão 2 deste *dataset*

conta com dezassete tipos de eventos, representados na seguinte Figura 42, incluindo os tipos de eventos necessários para a implementação - golo e pontapé de meio-campo (Deliege et al., 2021).

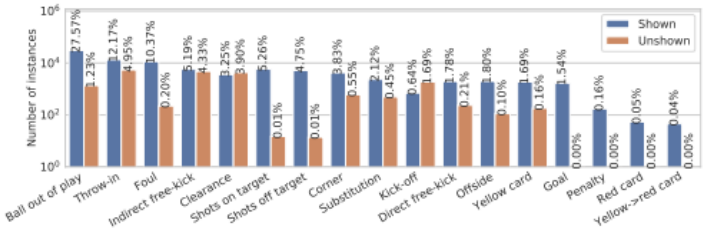


Figure 3. **SoccerNet-v2 actions.** Log-scale distribution of our **shown** and **unshown** actions among the 17 classes, and **proportion** that each class represents. The dataset is unbalanced, with some of the most important actions in the less abundant classes.

Figura 42 – Eventos do *dataset SoccerNet-v2* (Deliege et al., 2021)

O *SoccerDB* não só é o *dataset* com o menor número de jogos, representado na Tabela 14, como também não possui anotações de pontapé de meio-campo consideradas necessárias. Este tem em consideração onze tipos de eventos, como se pode observar na tabela da Figura 43.

Table 2: SoccerDB statistics. The dataset covers ten key events in soccer games. This table shows segment number, total time duration and playback segment number of each events. The unit of the duration is minute

Events	#Segments	Dur(min)	#Playback
Background(BK)	145473	25499.3	0
Injured(IJ)	1478	306.57	666
Red/Yellow Card(R/Y)	1160	244.08	219
Shot(SH)	14358	2125.35	8490
Substitution(SU)	867	298.92	14
Free Kick(FK)	3119	400.53	843
Corner(CO)	3275	424.08	668
Saves(SA)	5467	735.95	2517
Penalty Kick(PK)	156	28.25	130
Foul(FO)	5276	766.33	1015
Goal(GO)	2559	532.03	2559
Total	183188	31361.39	17121

Figura 43 – Eventos do *dataset SoccerDB* (Y. Jiang et al., 2020)

Ao contrário dos anteriores, o *Wyscout dataset* não considera só o tipo de evento como também subtipos e *tags*, sendo que golo é um subtipo de "free kick", como representado na Figura 44.

type	subtype	tags
pass	cross, simple pass	accurate, not accurate, key pass, opportunity, assist, goal
foul		no card, yellow, red, 2nd yellow
shot		accurate, not accurate, block, opportunity, assist, goal
duel	air duel, dribbles, tackles, ground loose ball	accurate, not accurate
free kick	corner, shot, goal kick, throw in, penalty, simple kick	accurate, not accurate, key pass, opportunity, assist, goal
offside touch	acceleration, clearance, simple touch	counter attack, dangerous ball lost, missed ball, interception, opportunity, assist, goal

Figura 44 – Tipos e subtipos de eventos do *Wyscout dataset* (Pappalardo et al., 2019)

Nesta tabela pode-se observar também a falta do tipo/subtipo *kick-off* necessário para a implementação, como descrito na secção de Análise e *Design*.

Em suma, apenas o *dataset SoccerNet-v2* tem ambas as anotações consideradas obrigatórias para a implementação (golo e pontapé de meio-campo), sendo o *dataset* escolhido.

4.3.2 *Download*

Para a utilização do *dataset* escolhido foi necessário transferir os vídeos e os ficheiros *json* associados a cada jogo pertencentes ao mesmo. Este *download* pode ser realizado por duas formas – utilização da *API* fornecida pelos autores do *dataset* ou *download* de ficheiros comprimidos fornecidos pelos autores (SoccerNet, 2022).

Para esta implementação, inicialmente foi realizado o *download* via *API*. No entanto, devido à falta de espaço e necessidade de escolha dos jogos e pastas de vídeos, foi também utilizado o *download* dos ficheiros.

A *API* fornecida facilita o *download* e a divisão entre treino, validação e teste dos vídeos e ficheiros de *labels.* Esta permite a escolha da qualidade dos vídeos (720p ou 224p), mas não permite escolher quais as ligas ou quantidade a transferir. Para usar esta *API* nos *notebooks* são necessários os seguintes *imports* (Código 3):

```
import SoccerNet
from SoccerNet.Downloader import SoccerNetDownloader
from SoccerNet.utils import getListGames
```

Código 3 – *Imports* para *download* do *dataset*

Para utilizar este método são também necessárias algumas configurações. De modo a obter os ficheiros *json* apenas é necessário configurar o local de armazenamento e executar a última linha do excerto de código representado no Código 4. Contudo, o *download* dos vídeos requer o preenchimento de um questionário com *NDA*, devido aos direitos dos vídeos dos jogos, e utilização de uma *password* enviada pelos autores do *dataset* (SoccerNet, 2022).

```
%store -r local_dataset_dir
mySoccerNetDownloader=SoccerNetDownloader(LocalDirectory =
local_dataset_dir)

%store -r soccernet_pw
mySoccerNetDownloader.password = soccernet_pw

mySoccerNetDownloader.downloadGames(files=["1_720p.mkv", "2_720p.mkv"])
mySoccerNetDownloader.downloadGames(files=["Labels-v2.json"])
```

Código 4 – *Download dataset*

O *download* dos vídeos é efetuado com a penúltima linha do Código 4 apresentado acima. Neste caso, apenas é realizado o carregamento de vídeos 720p e não é feita a divisão da informação.

Tal como mencionado, a divisão do *dataset* pode ser realizada diretamente durante o *download* do mesmo. Para tal, é necessário o parâmetro "split" na função *downloadGames()*, tanto para os vídeos como para os ficheiros *json* (Python Software Foundation, 2022). Esta divisão não foi utilizada para este projeto, já que não foi possível fazer *download* de todos os dados e alguns vídeos foram carregados mais tarde pelo outro método de transferência.

Para este projeto foram carregados 764GB de dados do *dataset Soccernet-v2*, contendo todos os jogos em 224p (cerca de 530 jogos) e uma parte dos jogos em 720p (cerca de 340 jogos).

4.3.3 Processamento dos dados

Como representado no diagrama da secção de Análise e *Design*, os vídeos irão passar por uma fase de processamento, de modo a obter sequências de *frames* pré-processadas e regularizadas. Esta fase deve ser aplicada aos dados de treino e validação, para garantir que são equivalentes. Este processamento é também necessário para obter a lista de *features* e as *labels* a estas associadas, que irão ser utilizadas durante o treino dos modelos, ou seja, a criação de um *dataset* processado e personalizado à necessidade dos mesmos.

De modo a facilitar a implementação, foram definidas algumas variáveis globais, para utilização ao longo dos vários *notebooks* de desenvolvimento, e outras apenas locais, referentes apenas ao *notebook* em execução. Entre essas variáveis constam as seguintes:

a) DATASET_DIR = "C:\…" - caminho do diretório que contém o *dataset*, utilizado para obter os vídeos dos vários jogos e os respetivos ficheiros de informação (*labels*);
b) FRAME_SECONDS_DIFF = [-2, -1, 0, 1, 2, 3] - lista com os tempos para extração das *frames*, tendo em conta o minuto do momento anotado no ficheiro de informação para extrair o momento em 6 *frames* no espaço de 6 segundos, desde 2 segundos antes do evento e até 3 segundos depois;
c) CLASSES_LIST = ["kick-off", "goal"] - lista de tipos de eventos a classificar;
d) IMAGE_HEIGHT, IMAGE_WIDTH = 72, 128 - altura e comprimento das *frames* em pixéis, para o redimensionamento das várias *frames*. Estes valores foram definidos tendo em consideração as proporções das *frames* dos vídeos de treino (720 x 1280 e 224 x 398), ou seja, 16 por 9;
e) VIDEO_QUALITY_FOLDERS = ["720p", "224p"] - lista de pastas de vídeos do *datatset*;
f) PARTS_1_FILE_NAME = "1", PARTS_1_FILE_NAME = "1", VIDEO_FILE_EXT = ".mkv", LABELS_FILE_NAME = "Labels-v2.json" – representando os nomes das partes dos jogos, a extensão dos vídeos e o nome dos ficheiros de informação, respetivamente, e utilizadas para validação das pastas dos jogos e leitura dos vários ficheiros
g) LABEL_ATTRIBUTE_NAME = "annotations" – nome da secção do ficheiro *json* que contém as anotações dos eventos.

Após a definição das variáveis anteriores, foram também criadas funções auxiliares para o cálculo dos minutos em segundos e para a extração de *frames*, representadas na Figura 45 e na Figura 46, respetivamente.

A primeira função tem como parâmetro uma *string* com o formato "MM:SS" (minutos : segundos) e retorna o mesmo em segundos. Esta é utilizada pela função principal de criação do *dataset* para calcular o segundo em que o momento ocorreu no vídeo, de acordo com os tempos anotados no ficheiro de informação.

```python
#Minutos(string) para segundos (número - int)
def get_seconds(time_in_minutes_str):
    m, s = time_in_minutes_str.split(':')
    return int(m) * 60 + int(s)
```

Figura 45 – Função auxiliar para converter minutos para segundos

A segunda função auxiliar é utilizada, como o nome indica, para extrair as *frames* representativas dos momentos. Esta também tem a responsabilidade de pré-processar cada *frame* através do redimensionamento e da normalização destas, como representado nesta Figura 46.

```python
def frames_extraction_per_label(video_path, time_in_seconds):
    # Lista de frames do vídeo
    frames_list = []

    # Leitura do vídeo - VideoCapture
    video_reader = cv2.VideoCapture(video_path)

    fps = video_reader.get(cv2.CAP_PROP_FPS)

    if fps <= 0:
        return frames_list

    frameRate  = fps / 100

    for frame_second_diff in FRAME_SECONDS_DIFF:

        if(time_in_seconds == 0):
            time = time_in_seconds + frame_second_diff + 2
        else:
            time = time_in_seconds + frame_second_diff

        time = time + frameRate
        time = round(time, 2)

        #Posição da frame atual
        #CAP_PROP_POS_FRAMES - 0-based index of the frame to be decoded/captured next
        #CAP_PROP_POS_MSEC - 0-based index of the frame to be decoded/captured next
        video_reader.set(cv2.CAP_PROP_POS_MSEC, time*1000)

        # Leitura da frame
        success, frame = video_reader.read()

        # Validação da leitura
        if not success:
            print(f'break')
            break

        # Resize da frame
        resized_frame = cv2.resize(frame, (IMAGE_WIDTH, IMAGE_HEIGHT))

        # Normalizar a frame (/255 -- cada pixel entre 0 e 1)
        normalized_frame = resized_frame / 255

        # Adicionar frame à lista
        frames_list.append(normalized_frame)

    # VideoCapture release
    video_reader.release()

    return frames_list
```

Figura 46 – Extração de *frames*

Esta função recebe como parâmetro o caminho do vídeo do qual vai retirar as *frames* e o tempo, em segundos, em que ocorreu o evento. Primeiramente é usado *OpenCV* para ler o vídeo e calcular o *frame rate* do mesmo a partir do atributo "CAP_PROP_FPS" (OpenCV, 2018). Com a lista das diferenças de tempo – variável b) mencionada anteriormente, o *frame rate* do vídeo e tempo exato do evento, são calculados os tempos, em milissegundos, em que serão retiradas as *frames* para formar a sequência representativa do momento de golo/pontapé de meio-campo. Para retirar cada *frame* é atribuído o tempo calculado ao atributo "CAP_PROP_POS_MSEC" (OpenCV, 2018) e cada uma destas é pré-processada.

A função principal, *createDataset(),* começa por procurar cada jogo a ser processado, tendo em consideração a estrutura de pastas em que estes estão armazenados:

- Qualidade dos vídeos (720p ou 224p)
    - Liga (por exemplo *france_ligue)*
        - Época (2014-1015, 2015-2016, 2026-2017)
            - Jogo
                - Primeira parte do jogo – *1_720p.mkv*
                - Segunda parte do jogo – *2_720p.mkv*
                - Ficheiro *json* com anotações – *Labels-v2.json*

Sendo que a quantidade de *kick-offs* e golos visíveis difere, foi adicionado um limite de 1136 para cada classe. Este valor foi escolhido por ser a quantidade máxima de *kick-offs* recolhidos. A utilização deste limite permite manter uma proporção 50/50 dos exemplos de cada classe no novo *dataset*.

Esta função percorre cada uma das pastas, valida se o caminho existe e se ainda não chegou ao limite definido para cada classe. Ao alcançar a pasta com os vídeos e ficheiro de *labels*, valida também se todas as partes existem, como representado no excerto de código da Figura 47. Este passo foi necessário, dado que nem todos os jogos possuem os três componentes presentes.

```
game_dir = os.path.join(season_dir, game)

print(f'Extracting data of folder - league - season - game: {video_quality_folder} - {league} - {season} - {game
files_list = os.listdir(game_dir)

# Validar se existe o ficheiro de labels
print(f'Extracting data of folder - league - season - game - labels: {video_quality_folder} - {league} - {season
labels_file_path = os.path.join(game_dir, LABELS_FILE_NAME)
labels_file_exists = os.path.exists(labels_file_path)

# Pastas com o ficheiro labels-v2.json, mas sem videos do jogo
## Validar se existe vídeo da primeira parte do video
fist_part_video_path = os.path.join(game_dir, PARTS_1_FILE_NAME + "_"  + video_quality_folder + VIDEO_FILE_EXT)
fist_part_video_exists = os.path.exists(fist_part_video_path)

## Validar se existe vídeo da segunda parte do video
second_part_video_path = os.path.join(game_dir, PARTS_2_FILE_NAME + "_" + video_quality_folder + VIDEO_FILE_EXT)
second_part_video_exists = os.path.exists(second_part_video_path)

if(labels_file_exists and fist_part_video_exists and second_part_video_exists):
```

Figura 47 – Validação dos componentes da pasta do jogo

Após esta validação, é feita a leitura do ficheiro *json* com as anotações. O *json* deste ficheiro segue a mesma estrutura para todos os jogos. Neste, as anotações são representadas numa lista com a informação do evento que aconteceu, quando e se é visível no vídeo. A Figura 48 apresenta um excerto desse *json* de um jogo.


```
{
    "UrlLocal": "italy_serie-a/2014-2015/2015-04-19 - 21-45 Inter 0 - 0 AC Milan/",
    "UrlYoutube": "",
    "annotations": [
        {
            "gameTime": "1 - 00:00",
            "label": "Kick-off",
            "position": "23",
            "team": "home",
            "visibility": "not shown"
        },
        {
            "gameTime": "1 - 01:15",
            "label": "Foul",
            "position": "75175",
            "team": "home",
            "visibility": "visible"
        },
        ...
    ],
    "gameAwayTeam": "AC Milan",
    "gameDate": "19/04/2015 - 21:45",
    "gameHomeTeam": "Inter",
    "gameScore": "0 - 0"
}
```


Figura 48 – Estrutura do *json* dos ficheiros de anotações dos jogos

Como visível na Figura 49, para esta leitura foram utilizadas as funções *apply*() e *glow*(). A leitura normal ou o *parse* para objeto não tratava corretamente a lista interna das anotações, não permitindo a leitura dos objetos dentro da mesma.

```
labels_file = pd.read_json(labels_file_path)

label_names = labels_file['annotations'].apply(lambda row: glom(row, 'label'))
game_times = labels_file['annotations'].apply(lambda row: glom(row, 'gameTime'))
visibility_info = labels_file['annotations'].apply(lambda row: glom(row, 'visibility'))

imp_annotations_indexs = [label_index for label_index, x in enumerate(label_names)
                          if x in CLASSES_LIST and
                          visibility_info[label_index] == "visible"]

imp_annotations_labels = [x for label_index, x in enumerate(label_names)
                          if x in CLASSES_LIST and
                          visibility_info[label_index] == "visible"]

imp_game_times = [game_time for game_time_index, game_time in enumerate(game_times)
                  if game_time_index in imp_annotations_indexs]
```

Figura 49 - Recolha das anotações de cada jogo

Com a aplicação destas funções foi possível obter três listas diferentes com o mesmo tamanho: (i) lista com os eventos que aconteceram em jogo, (ii) lista com os momentos em que cada evento ocorreu (parte e minuto do vídeo) e (iii) lista com visibilidade de cada um.

Para obter a listagem dos índices e *labels* dos eventos definidos, é realizada a filtragem da lista de eventos que ocorreram. Estas listas mantém a ordem cronológica pela qual os eventos

aconteceram no jogo. Com os índices obtidos, é filtrada a lista com a parte e momento em que os eventos ocorreram.

A partir da lista filtrada, é realizada a extração e armazenamento da sequência de *frames* de cada evento encontrado, como visível na Figura 50. A informação sobre a parte do vídeo é utilizada inicialmente para validar se o vídeo com essa parte existe. Se existir, a *string* com o minuto e segundo do momento do evento é transformada em número (segundos) pela função auxiliar previamente mencionada. Com o caminho do vídeo e momento do evento em segundos, é chamada a segunda função auxiliar para obter a sequência de *frames* desse evento.

```
for game_time_index, game_time in enumerate(imp_game_times): #game_times

    game_time_info = game_time.split(" - ")

    #part (1 or 2)
    video_name = game_time_info[0] + "_" + video_quality_folder + VIDEO_FILE_EXT
    print(f'Extracting data of folder - league - season - game - video: {video_quality

    video_path = os.path.join(game_dir, video_name)
    video_path_exists = os.path.exists(video_path)

    if video_path_exists:

        #time in seconds
        time_in_minutes_str = game_time_info[1]
        time_in_seconds = get_seconds(time_in_minutes_str)

        frames = frames_extraction_per_label(video_path, time_in_seconds)

        label_str = label_names[game_time_index]

        class_index = CLASSES_LIST.index(imp_annotations_labels[game_time_index])

        #check if frames found - nr of frames = tamanho da lista FRAME_SECONDS_DIFF
        if len(frames) == len(FRAME_SECONDS_DIFF):

            if class_index == 0 and nr_of_features_kick_off != KICK_OFF_MAX_QUANTITY:
                features.append(frames)
                labels.append(class_index)
                nr_of_features_kick_off = nr_of_features_kick_off + 1
                print(f'nr_of_features_kick_off: {nr_of_features_kick_off}')

            elif class_index == 1 and nr_of_features_goal != GOAL_MAX_QUANTITY:
                features.append(frames)
                labels.append(class_index)
                nr_of_features_goal = nr_of_features_goal + 1
                print(f'nr_of_features_goal: {nr_of_features_goal}')
```

Figura 50 - Extração das *frames* e armazenamento de cada *feature*

O número de *frames* da sequência é validado, para garantir que foi possível extrair 6 imagens sem anomalias. Caso o limite da classe do evento obtido não tenha sido alcançado, a sequência e *index* da classe são adicionadas a cada lista e é incrementado o contador de exemplo dessa classe.

Por fim, as listas com as sequências de *frame*s e com as classes de cada exemplo obtidas são transformadas em *arrays*, sendo estes retornados.

4.3.4 Divisão dos dados

Para garantir que a avaliação e validação dos modelos são imparciais e que a rede não está *overfitted* aos dados de treino, o *dataset* foi divido, através da técnica *HoldOut*, em três grupos – treino, teste e validação.

O *array,* com classes de cada sequência de *frames* recolhida, foi então convertido numa matriz binária com a função *to_categorical()* do *Keras* (TensorFlow, 2022h). Esta função utiliza o *array* e o número de classes como parâmetros, sendo esta conversão utilizada principalmente para problemas de *ML* com dados categóricos, como o caso, e é conhecida como *One-Hot Encoding* (Brownlee, 2017). A matriz resultante deste processo possui o número de colunas igual ao número de classes e o número de linhas corresponde ao tamanho do *array* de parâmetro. Ou seja, se o primeiro exemplo for de um golo, este estará representado no *array* como [1], mas na matriz estará [0,1].

A divisão dos dados foi feita através da função *train_test_split(x, y)* da biblioteca *Scikit-learn*, como representado nas Figura 51 e Figura 52. Nestas figuras pode-se ainda observar que foi utilizado o parâmetro *stratify.* Com a definição deste parâmetro para o conjunto y, são mantidas as proporções entre as classes em ambos os conjuntos. Para a sua utilização apenas é necessário garantir que o *shuffle* está definido como *True* (Scikit Learn, 2022) (Brownlee, 2020).

Primeiramente a função foi utilizada para obter os conjuntos de treino e teste a partir do *array* com os exemplos recolhidos (x) e da matriz binária das classes gerada previamente (y).

```
features_train, features_test, labels_train, labels_test = train_test_split(features,
                                                                            one_hot_encoded_labels,
                                                                            test_size = 0.2,
                                                                            shuffle = True,
                                                                            stratify=one_hot_encoded_labels)
```

Figura 51 – Divisão dos dados para teste

Devido ao tamanho do *dataset*, foi escolhido utilizar apenas 20% do mesmo para teste e o resto para treino. Os dados também são misturados antes da divisão, de modo a não seguirem a mesma ordem pela qual foram recolhidos.

A partir do conjunto de treino obtido, foi aplicada a mesma função de divisão para obter os subconjuntos de treino e validação. Na Figura 52, observa-se a aplicação dessa divisão e que o valor escolhido para validação foi 20% dos dados de treino.

```
features_train, features_val, labels_train, labels_val = train_test_split(features_train,
                                                                          labels_train,
                                                                          test_size=0.2,
                                                                          shuffle=True,
                                                                          stratify=labels_train)
```

Figura 52 - Divisão dos dados para treino e validação

Para a utilização destes subconjuntos noutros *notebooks*, estes tiveram de ser guardados fora de variáveis locais. Inicialmente, os mesmos estavam a ser guardados em variáveis globais,

como as constantes já mencionadas, mas os subconjuntos passaram o limite de 4GB. Devido às suas dimensões, optou-se por armazenar os mesmos em ficheiros na raiz do projeto, utilizando a função da Figura 53. Este método permitiu também a utilização destes subconjuntos sem ser necessário executar novamente o *notebook* dos mesmos.

Figura 53 – Armazenamento dos subconjuntos de treino, validação e teste em ficheiros

A função *save_info_to_file*() foi criada de modo a receber o nome do ficheiro e a informação a guardar, subconjuntos de treino e teste. Para a criação e escrita do ficheiro é utilizada a função *open()* e utilizado o modo "wb", ou seja, ficheiro em formato binário para escrita (Tutorials Teacher, 2022). Apesar desta função fazer *overwrite* se o ficheiro existir, optou-se por apagar o ficheiro caso um com o mesmo nome exista. Desta forma, é garantido que dados de execuções anteriores não afetam os resultados atuais.

Para a leitura destes ficheiros para utilização dos subconjuntos, foi utilizada novamente a função *open()*, mas com outro modo (*rb*). Este modo permite a leitura de ficheiros com formato binário (Tutorials Teacher, 2022). A Figura 54 demonstra a leitura dos vários ficheiros e o armazenamento do conteúdo destes em variáveis locais.

Figura 54 – Leitura dos ficheiros com os subconjuntos de treino, validação e teste

4.4 Modelo *ConvLSTM*

Para a construção do modelo *ConvLSTM* foram utilizadas camadas de *ConvLSTM2D* do *Keras* encadeadas. A arquitetura deste modelo foi construída tendo como base o modelo experimental construído no Tutorial de reconhecimento de ações. Com base nessa arquitetura,

foram alterados alguns parâmetros e até mesmo o número de camadas para condizer com o problema e a quantidade de recursos disponíveis.

Como representado na Figura 55, optou-se por ter apenas 3 camadas, em vez das 4 sugeridas no tutorial previamente mencionado. Esta alteração surgiu maioritariamente devido à complexidade da rede e capacidade da máquina. Após cada uma destas manteve-se a camada de agrupamento (*MaxPooling3D*) com os mesmos parâmetros e aplicou-se uma camada de *dropout*. Para esta o valor de *dropout* foi utilizado 0,2, sendo testado ainda o aumento para 0,5, que não mostrou grande diferença nos resultados. De modo a aplicar o *dropout* em todas as *frames* da sequência foi utilizado o *wrapper TimeDistributed ()* do *Keras* (TensorFlow, 2022e).

```
def create_convlstm_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura

    model.add(ConvLSTM2D(filters = 8, kernel_size = (3, 3), activation = 'tanh',data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True, input_shape = (NR_OF_FRAMES,
                                                                                      IMAGE_HEIGHT, IMAGE_WIDTH, 3)))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))


    model.add(ConvLSTM2D(filters = 16, kernel_size = (3, 3), activation = 'tanh', data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True))

    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))


    model.add(ConvLSTM2D(filters = 16, kernel_size = (3, 3), activation = 'tanh', data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))


    model.add(Flatten())

    model.add(Dense(len(CLASSES_LIST), activation = "sigmoid"))

    model.summary()

    return model
```

Figura 55 – Modelo *ConvLSTM* (Experiência B)

A função utilizada nas 3 primeiras camadas foi a tangente hiperbólica (*tanh*), sendo a função sugerida pelos autores da rede *ConvLSTM* (Shi et al., 2015). Um teste rápido através a troca desta função pela função de *ReLu* foi realizado e confirmado o benefício da função *tanh*.

Após estas camadas, a camada *Flatten* do *Keras*, tem a responsabilidade de transformar os *tensors* multidimensionais em unidimensionais para alimentar a última camada. Para esta última camada totalmente conectada (*Dense*), aplicou-se uma função de ativação *sigmoid*, tendo em conta o número de classes que necessita classificar (2).

De modo a tentar alcançar os melhores resultados, foram definas 12 experiências para testar diferentes configurações de arquitetura. Sendo o exemplo anterior, Figura 55, uma dessas experiências. Na Tabela 15 pode-se observar as diferentes configurações que foram definidas para cada experiência.

Tabela 15 – Experiências modelo *ConvLSTM*

Experiência	Número de filtros	*Kernel size*	Com *dropout*	*Batch size*
A	8 – 16 – 16	3 x 3	Não	8
B		3 x 3	Sim	
C		5 x 5	Não	
D		5 x 5	Sim	
E	4 – 8 – 8	3 x 3	Não	16
F		3 x 3	Sim	
G		5 x 5	Não	
H		5 x 5	Sim	
I	16 – 32 – 32	3 x 3	Não	4
J		3 x 3	Sim	
K		5 x 5	Não	
L		5 x 5	Sim	

Para o número de filtros foi escolhido um valor potência de base dois, entre 1 e a altura das imagens a atribuir para a segunda e terceira camada. Sendo que para a primeira camada foi utilizado metade desse valor.

O número máximo de filtros escolhido inicialmente foi 64, potência de base dois mais próximo da altura definida para as *frames*, mas a máquina não tinha capacidade para o treino de uma rede com essa dimensão. Desta forma, foram utilizados os valores 32, 16 e 8 como máximo.

No caso do *kernel size*, como representado na Tabela 15, foram utilizados os tamanhos genéricos: 3 por 3 e 5 por 5.

Ao contrário do próximo modelo apresentado, *LRCN*, o treino deste com 32 de *batch size* não foi possível (esgotamento de recursos). Devido à complexidade e peso desta rede, foi necessário alterar o tamanho dos lotes consoante o número de filtros utilizados. Ou seja, quantos mais filtros por camada, menor o *batch size* possível durante o treino.

A avaliação e análise destas experiências podem ser encontradas nos anexos (Anexos D) e, em forma de resumo, no subcapítulo de Avaliação dos modelos.

4.5 Modelo *LRCN*

Para a construção deste modelo, considerou-se como base a arquitetura apresentada na secção Análise e *Design* e nos tutoriais realizados. Tal como no modelo anterior, a arquitetura foi afinada de acordo com o problema (classificação de 2 classes) e recursos disponíveis.

Desta forma e como representado na Figura 56, este foi construído com 4 camadas convolucionais seguidas de 1 camada recorrente *LSTM*. A seguir a cada camada convolucional foram aplicadas camadas de agrupamento (*MaxPooling2D*) e *dropout de 0,2*. Sendo que para este modelo, foi necessário aplicar o *wrapper TimeDistributed()*, também utilizado no modelo

ConvLSTM, não só na camada de *dropout*, mas a todas estas camadas (convolucional, agrupamento e de *dropout*). Este *wrapper* tem como função permitir aplicar as camadas a cada *frame* pertencente à sequência. Ou seja, permitir que as camadas sejam capazes de receber vetores de 4 dimensões, em vez de apenas 3 e, deste modo, considerar o número de *frames* (TensorFlow, 2022e).

Para a camada totalmente conectada (última camada) foi aplicada a mesma camada utilizada no modelo anterior, dado que o número de classes é o mesmo.

Neste modelo, também foi aplicada uma camada *Flatten*, mas noutro local. Em vez de adicionada antes da última camada, necessitou de ser usada antes da camada recorrente já que esta apenas aceita vetores unidimensionais.

```
def create_LRCN_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura
    model.add(TimeDistributed(Conv2D(8, (3, 3), padding='same',activation = 'relu'),
                              input_shape = (NR_OF_FRAMES, IMAGE_HEIGHT, IMAGE_WIDTH, 3)))

    model.add(TimeDistributed(MaxPooling2D((4, 4))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Conv2D(16, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((4, 4))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Conv2D(32, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Conv2D(32, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Flatten()))

    model.add(LSTM(16))

    model.add(Dense(len(CLASSES_LIST), activation = 'sigmoid'))

    # Tabela
    model.summary()

    return model
```

Figura 56 – Modelo *LRCN* (Experiência B)

Para este também foram definidos 12 testes, representados na Tabela 16, no entanto foram utilizadas quantidades de filtros diferentes e o *batch size* para o treino manteve-se ao longo de todas as experiências (32).

Tabela 16 – Experiências modelo *LRCN*

Experiência	Número de filtros	*Kernel size*	Com *dropout*
A	8 – 16 – 32 – 32	3 x 3	Não
B		3 x 3	Sim
C		5 x 5	Não
D		5 x 5	Sim
E	4 – 8 – 16 – 16	3 x 3	Não
F		3 x 3	Sim
G		5 x 5	Não
H		5 x 5	Sim
I	16 – 32 – 64– 64	3 x 3	Não
J		3 x 3	Sim
K		5 x 5	Não
L		5 x 5	Sim

Para o número de filtros foram utilizados os valores: 64, 32 e 16. Sendo que para as camadas convolucionais optou-se pela seguinte sequência: metade da metade do valor – metade do valor – valor. Para a camada recorrente foi utilizado o mesmo valor que a camada anterior (camada convolucional).

Os testes e avaliação dos mesmos podem ser encontrados de forma completa na secção dos anexos (Anexos E). A avaliação destes pode também ser encontrada juntamente com a avaliação do modelo anterior no subcapítulo de Avaliação dos modelos.

4.6 Treino

Para o treino de ambos os modelos foi utilizado o mesmo excerto de código, apresentado na Figura 57. Para ambos foram utilizadas as mesmas funções de *callback*, hiperparâmetros e dados de validação, com a exceção do hiperparâmetro de tamanho do lote (*batch size*). Este último variou de acordo com o tamanho/complexidade da rede como explicado anteriormente.

```
# Callbacks
early_stopping = EarlyStopping(monitor="val_loss", patience=15, mode = 'min', restore_best_weights = True)
reduce_lr = ReduceLROnPlateau(monitor="val_loss", patience=5)

# Training hyperparameters
epochs = 100
batch_size = 4

convlstm_model.compile(loss = 'binary_crossentropy' , optimizer = 'Adam', metrics = ["accuracy"])

convlstm_model_training_history = convlstm_model.fit(
    FEATURES_TRAIN_STORED,
    LABELS_TRAIN_STORED,
    batch_size=batch_size,
    epochs=epochs,
    validation_data=(FEATURES_VAL_STORED, LABELS_VAL_STORED),
    callbacks=[early_stopping, reduce_lr],
    shuffle = True
)
```

```
Epoch 1/100
467/467 [==============================] - 50s 98ms/step - loss: 0.6946 - accuracy: 0.5043 - val_loss: 0.6928 - val_accuracy:
0.5000 - lr: 0.0010
Epoch 2/100
```

Figura 57 – Treino

4.6.1 Funções de regularização

No início do excerto do código da figura anterior, Figura 57 , pode-se observar as duas funções de *callback* para regularização definidas.

Early stopping é uma técnica de regularização, que monitoriza o *val_loss* (erro de validação) e para o treino assim que este erro tenha atingido um mínimo (Aurélien Géron, 2019).

Para este projeto, utilizou-se a função *EarlyStopping()* do *Keras* com 15 de *patience*, ou seja, o treino para se ao longo de 15 iterações (*epochs*) o erro de validação não baixar (*monitor="val_loss"* e *mode="min"*). O parâmetro "*restore_best_weights*" foi também configurado, de forma a restaurar os pesos da rede na iteração onde o *val_loss* foi mais baixo (TensorFlow, 2022c).

A segunda função configurada, *ReduceLROnPlateau*() do *Keras*, permite regularizar a taxa de aprendizagem. Esta monitoriza a métrica definida e, se a mesma não melhorar ao longo de um certo valor de iterações, reduz a taxa de aprendizagem (Chen, 2020).

Neste caso, optou-se por monitorizar o *val_loss*, tal como na função anterior, e usar 5 de *patience*. A taxa é reduzida em 10%, sendo utlizado o valor *default* da função (TensorFlow, 2022d). Com estes valores é possível decrementar o *learning rate* devagar e se a métrica não melhorar, após 3 descidas, no mínimo, parar o treino.

Esta técnica é importante, já que um *learning rate* demasiado alto pode levar a divergências no treino e se for demasiado baixo pode levar a exaustão dos recursos. A redução da taxa deve ser executada de forma lenta, já que oposto pode levar a ficar preso num mínimo local (Aurélien Géron, 2019).

4.6.2 Hiperparâmetros

Um hiperparâmetro é um parâmetro de *ML* que é configurado externamente ao modelo e permite regular a aprendizagem do modelo. O valor deste, ao contrário de outros parâmetros não pode ser obtido através dos dados nem treino (Brownlee, 2019).

Como representado na Figura 57, foram definidos dois hiperparâmetros: tamanho do lote e número de iterações. Sendo que o primeiro define o número de exemplos de treino trabalhados ao mesmo tempo. Isto é, no final de cada lote o modelo é melhorado através do cálculo do erro entre as previsões feitas e as *labels* e atualização dos parâmetros internos a partir do mesmo (Brownlee, 2022).

A literatura sugere que o número de lotes seja uma potência de base dois, tendo como máximo o tamanho do conjunto de treino. No caso de ser o tamanho total o algoritmo de aprendizagem é conhecido como "Batch Gradient Descent". Se for igual a 1, este é chamado de "Stochastic Gradient Descent" e se for um valor entre estes, é conhecido como "Mini-Batch Gradient Descent" (Brownlee, 2022) (Aurélien Géron, 2019).

Neste desenvolvimento, foram utilizados valores de tamanho de lote entre 4 e 32 devido à capacidade dos recursos. O número de iterações foi definido como 100, sendo que, devido à função de *early stopping*, este representa o número máximo de iterações de treino.

4.6.3 Compilação do modelo

Antes do treino do modelo, foi necessário configurar as métricas, o optimizador e função de perda a utilizar através da compilação do modelo (TensorFlow, 2022f).

Devido à utilizado da função de ativação *Sigmoid* na última camada de ambos os modelos, a função de perda configurada foi a *binary_crossentropy*. Para o optimizador foi utilizado o *Adam* e nas métricas a precisão, considerados *default* por vários tutoriais e pela própria ferramenta *TensorFlow* (TensorFlow, 2022g).

4.6.4 Treino do modelo

O treino do modelo foi realizado através da função *fit*() com a utilização dos hipérmetros e *callbacks* mencionados anteriormente. Nesta foi também definida a aleatoriedade dos dados a cada iteração, os dados a usar para validação (conjunto de exemplos e respetivas *labels* de validação) e os dados de treino (conjunto de treino com os exemplos e *labels* associadas).

4.7 Guardar Modelos

Após o treino e avaliação de cada experiência, os modelos treinados foram guardados em ficheiros para uso posterior. Para distinguir as várias experiências, o nome de cada modelo guardado seguiu o mesmo formato: nome da pasta do modelo / nome do modelo, experiência, data de treino e avaliação obtida (*loss* e *accuracy*), como visível no código da Figura 58.

Guardar modelo

```
model_evaluation_loss, model_evaluation_accuracy = model_evaluation_history

date_time_format = '%Y_%m_%d__%H_%M_%S'
current_date_time_dt = dt.datetime.now()
current_date_time_string = dt.datetime.strftime(current_date_time_dt, date_time_format)

model_file_name = f'{MODELS_DIR}/LRCN/LRCN_model_a___Date_Time_{current_date_time_string}___Loss_{model_evaluation_loss}___Accura
print(f'{model_file_name}')

LRCN_model.save(model_file_name)
```

```
C:/Users/catar/Documentos/Tese/Projeto/Modelos/LRCN/LRCN_model_a___Date_Time_2022_10_05__15_20_20___Loss_0.6053931713104248___A
ccuracy_0.6713147163391113.h5
```

Figura 58 – Guardar modelo após treino e avaliação

Para a utilização destes modelos guardados, como por exemplo para *predict,* a função *load_model()* do *keras* foi utilizada (TensorFlow, 2022g). O código na Figura 59 demonstra como obter estes modelos a partir dessa função, o caminho de cada tipo de modelo e a configuração do nome do modelo guardado a utilizar.

ConvLSTM

```
model = tf.keras.models.load_model(f'Modelos/ConvLSTM/{model_file_name}.h5')
```

LRCN

```
model = tf.keras.models.load_model(f'Modelos/LRCN/{model_file_name}.h5')
```

Figura 59 – Obter modelos previamente guardados

Guardar os modelos após treino ao longo das experiências durante o desenvolvimento facilitou a comparação e a experimentação dos mesmos para deteção de golos com vídeos fora do *dataset*.

4.8 Deteção de golos

Para testar a solução de deteção de golos foi utilizado um jogo fora do *dataset SoccerNet-v2* e que tivesse visíveis todos os seus golos e *kick-off*. Algo que nem todos os vídeos do *dataset* tinham, devido à sobreposição da repetição do golo. O jogo utilizado para teste da solução foi um jogo fora do *dataset* pertencente à 13ª jornada da Liga Nos 2016-2017 em 720p, com 3 golos em 1 hora e 43 minutos de vídeo.

A solução implementada segue o diagrama de fluxos apresentado na secção de Análise e *Design*. Como descrito nesse capítulo, o vídeo é processado e são extraídas *frames* em grupos de 6. Cada *frame* extraída é pré-processada, do mesmo modo que as *frames* de treino, e adicionada ao conjunto. Cada conjunto é depois passado pelo modelo escolhido e os resultados deste são processados (probabilidade de cada evento).

4.8.1 Criação do ficheiro de *output*

A função criada para deteção de golo, começa por criar o ficheiro de *output* com a identificação da data e hora em que este foi gerado. No excerto de código da Figura 60, pode-se observar essa criação através da função *open()* do *Python* com o modo "a". Este modo permite criar um ficheiro com o intuito de adicionar informação a este, pela mesma ordem em que é anexada (Tutorials Teacher, 2022).

```
#Data atual
date_time_format = '%Y_%m_%d__%H_%M_%S'
current_date_time_dt = dt.datetime.now()
current_date_time_string = dt.datetime.strftime(current_date_time_dt, date_time_format)

#Ficheiro
file_name = f'Predict_{current_date_time_string}.txt'
txt_file = open(file_name,"a")
```

Figura 60 – Inicio da escrita do ficheiro de *output*

4.8.2 Leitura e processamento do vídeo

De seguida, o vídeo no caminho passado é lido através da biblioteca *OpenCV* e são iniciadas as variáveis necessárias para o processamento das *frames* e classificação das sequências. Ainda nesta secção as *deques (Doubly Ended Queues)* de *frames* e *timestamps* associadas são configuradas para terem um máximo de 6 elementos. A adição de um novo elemento leva à perda de outro no início da fila, ou seja, a cada *frame* adicionada (mais 1 segundo) é mantido o valor de 6 *frames* por sequência (GeeksforGeeks, 2016). Por exemplo, inicialmente a fila tem *frames* do segundo 0 ao 6 e após processada avança para a sequência de *frames* do 1 ao 7.

As *frames*, como já mencionado, são recolhidas a cada segundo do vídeo e para tal é utilizada a variável "*count",* representativa do número de *frames* lidas do vídeo. Para este cálculo, o valor de *frames* por segundo previamente através da propriedade "CAP_PROP_FPS" do *OpenCV* (OpenCV, 2018). Sendo o contador incrementado com o valor de *FPS* recolhido do processamento e classificação de cada sequência.

A propriedade "*CAP_PROP_POS_FRAMES*" do *OpenCV* foi utilizada nesta função para definir qual a *frame* a extrair (OpenCV, 2018), tendo em consideração o valor do contador mencionado no paragrafo anterior.

Como representado na Figura 61 , cada *frame* é extraída e o seu *timestamp* é obtido através da utilização de outra propriedade do *OpenCV, "CAP*_PROP_POS_MSEC*"* (OpenCV, 2018).

```
while video_reader.isOpened():

    video_reader.set(cv2.CAP_PROP_POS_FRAMES, count)

    #Leitura da frame
    ok, frame = video_reader.read()

    #Se a frame não tiver sido lida corretamente é feito um break
    if not ok:
        break

    timestamp = video_reader.get(cv2.CAP_PROP_POS_MSEC);

    #Resize da frame
    resized_frame = cv2.resize(frame, (IMAGE_WIDTH, IMAGE_HEIGHT))

    #Normalizar a frame (/255 -- cada pixel entre 0 e 1)
    normalized_frame = resized_frame / 255

    #Adicionar a frame à queue
    frames_queue.append(normalized_frame)
    timestamps_queue.append(timestamp)

    #Validar a existência do nr de frames desejado (6)
    if len(frames_queue) == NR_OF_FRAMES:

        predict = model.predict(np.expand_dims(frames_queue, axis = 0), verbose=1)
        predicted_labels_probabilities = predict[0]
        predicted_label = np.argmax(predicted_labels_probabilities)
        prob = predicted_labels_probabilities[predicted_label]
```

Figura 61 – Extração e processamento de *frames*

As *frames* são depois redimensionadas e normalizadas com as mesmas configurações utilizadas no processamento aplicado durante a criação do *dataset* de treino e avaliação dos modelos, descrito no capítulo de *Datasets* da secção de Implementação.

Para a obtenção das probabilidades de cada evento das sequências, estas são alimentadas ao modelo pela função "predict" após serem expandidas para a forma e tipo de vetor necessário. O *index* da classe com a maior probabilidade, assim como o valor dessa probabilidade, é obtido através da aplicação da função *argmax*() ao *array* das probabilidades resultante do modelo, assim como a respetiva probabilidade. Este processo pode ser observado nas últimas linhas do código da Figura 61.

4.8.3 Processamento do resultado do modelo

Para excluir alguns falsos positivos, apenas são processadas as sequências com a probabilidade de uma das classes maior que um *threshold* predefinido de acordo com alguns testes iniciais (0,995). Foram considerados cinco possíveis cenários: início do evento/momento, continuação do momento, início de outro evento, fim do momento de jogo e nenhum evento encontrado.

Para o primeiro cenário foi desenvolvido o código apresentado na Figura 62. Neste é guardada a lista de *timestamps* associadas às *frames* da sequência a ser processada, qual o evento encontrado, o *timestamp* inicial e a probabilidade do evento.

```
last_timestamps = list(timestamps_queue)
last_predicted_class_index = predicted_label
moment_first_timestamp = timestamps_queue[0]
event_probs.append(prob)
```

Figura 62- Início de um momento de jogo (golo ou *kick-off*)

Se a classe prevista for igual à classe prevista para a sequência anterior e se ainda estiver na mesma altura do jogo (*timestamp)* pertencente à lista guardada anteriormente, ou seja, continuação do mesmo momento de jogo é aplicado o código da Figura 63. Neste, a lista de *timestamps* desta nova sequência é guardada, para prepósitos de validação de altura do jogo, e adicionada a probabilidade obtida à lista de probabilidades desse evento.

```
last_timestamps = list(timestamps_queue)
event_probs.append(prob)
```

Figura 63 – Continuação do momento captado

No caso da probabilidade ser maior que o *threshold*, mas não ser a primeira instância e tiver sido prevista outra classe ou ser outra altura do jogo, foi construído o código da Figura 64.

```
#record event
mean_prob = np.mean(event_probs)

if last_predicted_class_index == 1 :
    last_goal_moment_start_timestamp = moment_first_timestamp
    last_goal_moment_end_timestamp = last_timestamps[-1]
    last_goal_moment_prob = mean_prob

if last_predicted_class_index == 0 and last_goal_moment_prob != None and
    moment_first_timestamp >= last_goal_moment_end_timestamp + MIN_TIME and
    moment_first_timestamp <= last_goal_moment_end_timestamp + MAX_TIME :

    line = f'GOAL - time: {get_time_str(last_goal_moment_start_timestamp)} - {get_time_str(last_goal_moment_end_timestamp)} = {last_goal_moment_prob}'
    txt_file.write('\n')
    txt_file.write(line)
    txt_file.write('\n')
    last_goal_moment_start_timestamp = None
    last_goal_moment_end_timestamp = None
    last_goal_moment_prob = None

event_probs = []

#start new event
last_predicted_class_index = predicted_label
moment_first_timestamp = timestamps_queue[0]
last_timestamps = list(timestamps_queue)
event_probs.append(prob)
```

Figura 64 – Alteração de momento de jogo

Se o último evento classificado tiver sido golo (1) as variáveis responsáveis por armazenar este são preenchidas (*timestamp* inicial e final do evento de golo recolhido e média das probabilidades do evento nas sequências recolhidas). De seguida, o início do novo evento é registado.

Por outro lado, se o último evento classificado tiver sido *kick-off e* antes desse tiver existido um evento de golo com um intervalo predefinido (pelo menos 1 minuto depois - *MIN_TIME* e no máximo 2 minutos depois - *MAX_TIME,*) entre estes, o golo é registado no ficheiro de output. Sendo anotado o momento inicial e final da previsão e a certeza da previsão. As variáveis com a informação sobre o evento de golo são limpas e dá-se início a um novo evento.

De modo à informação sobre a previsão do momento de golo ser mais percetível, foi desenvolvida e aplicada a função auxiliar da Figura 65. Esta tem o objetivo de converter o *timestamp* obtido em milissegundos numa uma *string* com o formato “HH:MM:SS.sss” (horas : minutos : segundos . milissegundos).

```
#milisegundos(número) para time (string)
def get_time_str(milliseconds):

    seconds, milliseconds = divmod(milliseconds, 1000)
    minutes, seconds = divmod(seconds, 60)
    hours, minutes = divmod(minutes, 60)

    return f'{int(hours):02d}:{int(minutes):02d}:{int(seconds):02d}.{int(milliseconds):03d}'
```

Figura 65 – Função auxiliar para converter milissegundos em uma *string* com o formato “HH:MM:SS.sss”

Para o fim do momento de jogo, sem início de um novo foram aplicadas as mesmas regras que a situação anterior, mas sem o registo de um novo evento. Neste caso, as variáveis são limpas como representado na Figura 66. Tal como no caso de não ter previsto nenhum evento (probabilidade menor que o threshold) e não existir nenhum evento a ser processado.

```
#no event
last_timestamps = []
last_predicted_class_index = None
moment_first_timestamp = None
event_probs = []
```

Figura 66 – Nenhum momento encontrado

4.8.4 Resultados

Vários testes foram feitos com os modelos construídos e o vídeo escolhido, Figura 67, um exemplo dos resultados obtidos utilizando a rede *LRCN*, especificamente a experiência H desse modelo representada na Tabela 16.

Nesta figura, Figura 67, pode-se observar que foram detetados 22 eventos de golo ao longo de todo o vídeo. As linhas representadas a verde representam momentos de golo previstos de forma correta (golos reais). Todos os outros (19) são falsos positivos que a rede atribuiu grande probabilidade na mesma. Este fenómeno é facilmente justificado, já que os modelos foram treinados para conhecer golos e *kick-off*, mas nada mais. Em jogos não só existem outros tipos de eventos (classes desconhecidas à rede), como momentos semelhantes a todas as classes.

1. GOAL - time: 00:13:30.000 - 00:13:35.000 = 0.9958712458610535
2. GOAL - time: 00:17:58.000 - 00:18:06.000 = 0.9955296516418457
3. GOAL - time: 00:20:37.000 - 00:20:46.000 = 0.995695948600769
4. GOAL - time: 00:22:40.000 - 00:22:45.000 = 0.9954503178596497
5. GOAL - time: 00:25:34.000 - 00:25:39.000 = 0.9957428574562073
6. GOAL - time: 00:29:04.000 - 00:29:11.000 = 0.9957109093666077
7. GOAL - time: 00:35:17.000 - 00:35:28.000 = 0.9958730340003967
8. GOAL - time: 00:38:08.000 - 00:38:16.000 = 0.9955413937568665
9. GOAL - time: 00:45:42.000 - 00:45:50.000 = 0.9955376386642456
10. GOAL - time: 00:47:37.000 - 00:47:50.000 = 0.9960616230964661
11. GOAL - time: 00:50:14.000 - 00:50:22.000 = 0.9957449436187744
12. GOAL - time: 00:54:50.000 - 00:55:03.000 = 0.9958983659744263
13. GOAL - time: 00:57:13.000 - 00:57:18.000 = 0.995009183883667
14. GOAL - time: 01:02:31.000 - 01:02:38.000 = 0.9955580234527588
15. GOAL - time: 01:07:38.000 - 01:07:45.000 = 0.9953688979148865
16. GOAL - time: 01:14:22.000 - 01:14:27.000 = 0.9951130747795105
17. GOAL - time: 01:17:07.000 - 01:17:13.000 = 0.995363175868988
18. GOAL - time: 01:19:31.000 - 01:19:41.000 = 0.995989978313446
19. GOAL - time: 01:23:21.000 - 01:23:27.000 = 0.9956538081169128
20. GOAL - time: 01:30:54.000 - 01:31:05.000 = 0.9960201978683472
21. GOAL - time: 01:35:40.000 - 01:35:51.000 = 0.9957922697067261
22. GOAL - time: 01:40:43.000 - 01:40:49.000 = 0.9956063032150269

Figura 67 – Exemplo do resultado deteção de golos com o Modelo *LRCN* (Experiência H)

Um exemplo de um caso semelhante e que pode (e induziu) em erro a rede é o ataque por parte da equipa A perto da baliza da equipa B, seguido do contra-ataque por parte da equipa B (meio-campo). A rede não tendo conhecimento destes momentos semelhantes (parecido, mas nem golo nem *kick-off*) tenta atribuir uma das classes e entra qual a que tem mais semelhança. Após visualização do vídeo nos momentos dos falsos positivos, notou-se que a maioria se tratava do exemplo anterior.

Para a alternativa *ConvLSTM* notou-se o mesmo fenómeno, mas ainda mais agravado, tendo como resultado a quantidade maior de falsos positivos. E após visualização dos instantes classificados como golo não foi possível observar momentos semelhantes nesses que levassem a erro do algoritmo. Após aumento do threshold para 0,999, a quantidade de falsos positivos diminui, mas não encontrou o primeiro golo real, como representado na Figura 68,

1. GOAL - time: 00:12:45.000 - 00:12:51.000 = 0.9999908804893494
2. GOAL - time: 00:22:39.000 - 00:22:45.000 = 0.9999741315841675
3. GOAL - time: 00:26:09.000 - 00:26:15.000 = 0.9999359846115112
4. GOAL - time: 00:30:47.000 - 00:30:53.000 = 0.9999955296516418
5. GOAL - time: 00:35:53.000 - 00:36:01.000 = 0.9999969005584717
6. GOAL - time: 00:50:59.000 - 00:51:04.000 = 0.9999830722808838
7. GOAL - time: 00:54:55.000 - 00:55:03.000 = 0.9999593496322632
8. GOAL - time: 00:56:50.000 - 00:56:56.000 = 0.9999485015869141
9. GOAL - time: 01:00:26.000 - 01:00:32.000 = 0.9999819993972778
10. GOAL - time: 01:02:25.000 - 01:02:32.000 = 0.9999507069587708
11. GOAL - time: 01:09:11.000 - 01:09:17.000 = 0.9999697208404541
12. GOAL - time: 01:11:36.000 - 01:11:42.000 = 0.9999629259109497
13. GOAL - time: 01:13:52.000 - 01:13:57.000 = 0.9999289512634277
14. GOAL - time: 01:17:08.000 - 01:17:14.000 = 0.9999566674232483
15. GOAL - time: 01:27:47.000 - 01:27:52.000 = 0.9999034404754639
16. GOAL - time: 01:31:00.000 - 01:31:05.000 = 0.9999406337738037
17. GOAL - time: 01:33:39.000 - 01:33:46.000 = 0.9999787211418152
18. GOAL - time: 01:35:40.000 - 01:35:46.000 = 0.9999485611915588
19. GOAL - time: 01:40:44.000 - 01:40:49.000 = 0.9999897480010986

Figura 68 – Exemplo do resultado deteção de golos com o Modelo *ConvLSTM* (Experiência G)

5 Avaliação

Nesta secção é descrito o processo de avaliação do sistema, identificando a hipótese a ser testada, as métricas de avaliação a serem utlizadas e a metodologia de avaliação a seguir.

Por fim, são analisados os resultados obtidos e avaliada a solução final de deteção de golos, refletindo na hipótese proposta.

5.1 Hipótese

Uma hipótese de investigação deve ter em conta os objetivos previamente definidos para o projeto e o que se espera alcançar.

O objetivo principal deste projeto, como já referido, é a construção de uma solução que automaticamente consiga analisar um vídeo de um jogo de futebol e catalogue os momentos de golo. A solução tem em mente a sua utilização por parte de equipas de futebol, principalmente sem grandes recursos humanos e monetários, para análise e formação de estatísticas do jogo. Como tal, a solução deve ter em conta esses recursos e deve permitir a análise sem grandes necessidades óticas (câmaras).

Com base no objetivo principal temos a questão de investigação "Como automatizar o processo de detetar e catalogar os momentos de golo de um jogo de futebol?", que juntamente com as necessidades do cliente permite formular a hipótese deste documento: "É possível automatizar o processo de deteção e catalogação dos momentos de golo de jogo sem grandes equipamentos óticos".

5.2 Indicadores e fontes de informação

Para validar a hipótese formulada é necessário perceber o desempenho do algoritmo de classificação. Uma forma de o fazer seria pela avaliação da sua exatidão a partir de diversos tipos de vídeos de jogos de futebol. No entanto, a exatidão por si só pode não ser a abordagem

mais correta, principalmente para problemas mais complexos e por isso o uso de uma matriz de confusão que envolve diversas métricas de avaliação é o mais aconselhado (Jason Brownlee, 2016). Os valores usados pela matriz de confusão e, consequentemente, pelas métricas, devem ser obtidos a partir do algoritmo e um conjunto de dados de teste diferente do conjunto de treino. Os dados devem também variar entre si, por exemplo na qualidade, ângulos e vista do jogo de futebol.

5.2.1 Matriz de confusão

A matriz de confusão, como mencionado, é uma matriz n por n, onde n representa o número de classes utilizadas na avaliação do desempenho de algoritmos de classificação. Esta compara a classificação obtida com a suposta e permite ter uma vista geral sobre o desempenho, e que tipo de erros podem estar a acontecer (Kozhaya, 2017).

A título de exemplo vai ser utilizada uma matriz de confusão 2 por 2, ou seja, classificação binária com dois valores possíveis (positivo e negativo). Na matriz representada na Figura 69, observa-se que as colunas representam os valores reais e as linhas os valores previstos/obtidos pelo algoritmo (Aniruddha Bhandari, 2020).

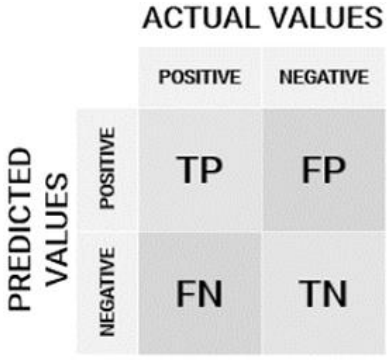


Figura 69 – Matriz de confusão para um classificador binário (Aniruddha Bhandari, 2020)

Neste exemplo da Figura 69, encontram-se também representados os quatro conceitos em que esta se baseia (Aniruddha Bhandari, 2020):

- Verdadeiro positivo (*TP*) – valor real igual ao suposto com ambos os valores positivos, ou seja, uma classificação positiva correta;
- Falso positivo (*FP*) - valor real diferente ao suposto com o valor real negativo e o previsto positivo, classificação incorreta de um valor negativo como positivo;
- Verdadeiro negativo (*TN*) - valor real igual ao suposto, mas o valor real é negativo e o previsto é negativo, ou seja, classificação negativa correta;
- Falso negativo (*FN*) - valor real diferente do suposto, com o valor real positivo e o previsto negativo, classificação incorreta de um valor positivo como negativo.

5.2.2 Métricas de avaliação

A exatidão, a precisão e o *recall* são as métricas de avaliação mais utilizadas pelos artigos estudados com o mesmo domínio de problema que o deste documento (Wickramaratna et al., 2005) (Ma et al., 2020) (Hong et al., 2018) (Huang et al., 2006) (Karimi et al., 2021) (H. Jiang et al., 2017). Outra métrica interessante, mas pouco mencionada, é a *F1-Measure* que relaciona a *precisão* e o *recall* (IBM, 2021a).

5.2.2.1 Exatidão

A exatidão de um algoritmo de classificação pode ser representada pela seguinte expressão (Jason Brownlee, 2016) (Aniruddha Bhandari, 2020):

$$\text{Exatidão} = \frac{\text{Nr de previsões corretas}}{\text{Nr de previsões}} = \frac{TP + TN}{TP + FP + TN + FN} \tag{13}$$

5.2.2.2 Precisão

A precisão representa quantas das precisões corretas foram de positivos, relevantes para o domínio do problema, o que determina se o modelo de classificação é de confiança. Esta métrica é calculada de acordo com a seguinte expressão (Aniruddha Bhandari, 2020):

$$\text{Precisão} = \frac{TP}{TP + FP} \tag{14}$$

5.2.2.3 *Recall*

Por outro lado, o *recall* permite medir quantos casos positivos foram previstos corretamente e é calculado segundo a seguinte expressão (Aniruddha Bhandari, 2020):

$$Recall = \frac{TP}{TP + FN} \tag{15}$$

5.2.2.4 *F1-Measure*

A *F1-Measure* representa a média harmónica entre a precisão e o *recall*, como descreve a expressão do seu cálculo (Aniruddha Bhandari, 2020) (IBM, 2021a):

$$F1 = \frac{2}{\frac{1}{\text{Precisão}} + \frac{1}{Recall}} = 2 * \frac{\text{Precisão} * Recall}{\text{Precisão} * Recall} \tag{16}$$

5.3 Metodologia de avaliação

A validação da robustez e exatidão do algoritmo para diferentes tipos de dados é importante para descartar *overfiting* do mesmo aos dados de treino e garantir o seu desempenho.

Desta forma, surgiram algumas técnicas para avaliação como o método simples de *HoldOut*, onde o conjunto de dados é divido aleatoriamente por dois subconjuntos, de treino e de teste.

No entanto, por ser feita uma divisão aleatória pode existir uma grande variância entre os subconjuntos, ou seja, a exatidão é dependente da divisão feita (Lakshana, 2021).

Outra técnica utilizada é *K-Fold Cross-Validation*, onde o conjunto de dados é dividido em *k*-conjuntos, onde *k*-1 conjuntos são utilizados para treino e o conjunto restante para teste, esta divisão é feita pelo método anterior, mas especificando o número de conjuntos (Lakshana, 2021). Este processo é realizado por *k* iterações e o conjunto de teste varia ao longo destas, do primeiro até *k*-ésimo conjunto, como representado no esquema seguinte (Figura 70).

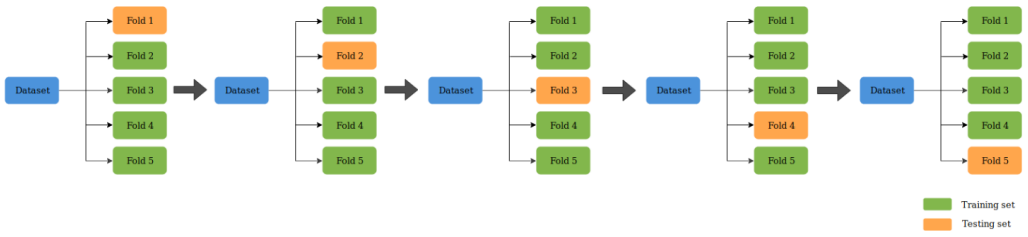


Figura 70 – Técnica *K-Fold Cross-Validation* (Rastogi, 2021)

O artigo de *Wickramaratna et al.* com o mesmo domínio de problema que o endereçado neste documento, utilizou para o estudo experimental do algoritmo proposto esta mesma técnica (Wickramaratna et al., 2005). Desta forma, seria interessante utilizar esta para a avaliação neste contexto, juntamente com as métricas de avaliação mencionadas.

5.4 Avaliação dos modelos

Apesar dos benefícios mencionados no uso da técnica *K-Fold Cross-Validation* para avaliação dos modelos, esta foi descartada durante o desenvolvimento. Isto deveu-se maioritariamente ao consumo excessivo de recursos na utilização desta técnica e, após mais algum estudo, foi desaconselhada por alguns autores no caso de redes neuronais profundas. Deste modo, a técnica utilizada para divisão de dados e, por consequência, avaliação destes foi a *HoldOut.*

Na Tabela 17 são apresentadas, resumidamente, as avaliações feitas a cada experiência de ambos os modelos implementados (*ConvLSTM* e *LSTM*). Os valores apresentados nesta tabela têm como unidade percentagem, sendo que os valores das duas primeiras colunas de avaliação foram arredondados a duas casas decimais e os restantes à unidade. A informação detalhada desta avaliação, como gráficos das curvas da exatidão e *loss* durante o treino e as matrizes de confusão de cada umas destas experiências podem ser encontradas na secção de Anexos (Anexos D e E).

Para esta avaliação foi utilizado o mesmo conjunto de testes, definido anteriormente no subcapítulo de Divisão dos dados, para todas as experiências. Sendo este composto por 228 exemplos de *kick-off* e 227 exemplos de golo, com um total de 455 exemplos.

A avaliação em si foi também realizada com o mesmo método para todas as experiências de modo a ser possível comparar as mesmas pelas suas configurações. Para obtenção dos valores de exatidão e *loss* da rede foi utilizada a função *evaluate*(), como representado na Figura 71.

```
model_evaluation_history = LRCN_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED)
print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

Figura 71 – Avaliação do modelo

Para o desenho dos gráficos com informação sobre a evolução da exatidão/exatidão de validação e *loss*/*loss* de validação, ao longo do processo de treino de cada modelo, foi utilizado o código representado na Figura 72. Estes gráficos podem ser encontrados na secção dos anexos (Anexos D e E)., tal como previamente mencionado.

```
def plot_metric(model_training_history, metric_name_1, metric_name_2, plot_name):

    metric_value_1 = model_training_history.history[metric_name_1]
    metric_value_2 = model_training_history.history[metric_name_2]

    epochs = range(len(metric_value_1))

    plt.plot(epochs, metric_value_1, 'blue', label = metric_name_1)
    plt.plot(epochs, metric_value_2, 'red', label = metric_name_2)

    plt.title(str(plot_name))

    plt.legend()
```

```
plot_metric(LRCN_model_training_history, 'loss', 'val_loss', 'Total Loss vs Total Validation Loss')
```

```
plot_metric(LRCN_model_training_history, 'accuracy', 'val_accuracy', 'Total Accuracy vs Total Validation Accuracy')
```

Figura 72 – Método de desenho das curvas de *accuracy* - *val_accuracy* e *loss* - *val_loss*

Como demonstra a Figura 73, para o desenho das matrizes de confusão e a obtenção das métricas: precisão, *recall* e *F1-measure*, associada a estas foram utilizadas as funções *confusion_matrix()* e *classification_report(),* respetivamente. Para a apresentação das matrizes de confusão nos *notebooks* foi ainda utilizado o método *ConfusionMatrixDisplay*() da biblioteca *Scikit-learn,* como as funções anteriores. Estas matrizes podem ser encontradas nos anexos (Anexos D e E).

```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import numpy as np
```

```
ypred = LRCN_model.predict(FEATURES_TEST_STORED, verbose=1)
#print(ypred[0])

ypred = np.argmax(ypred, axis=1)
#print(ypred)

test_labels = np.argmax(LABELS_TEST_STORED, axis=1)
print(classification_report(test_labels, ypred, target_names=CLASSES_LIST))

cm = confusion_matrix(test_labels, ypred)

## binary 0-1
#TN (True negative) -- FP (false positive)
#FN (false negative) -- TP (True positive)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=CLASSES_LIST)

disp.plot(cmap=plt.cm.Blues)
plt.show()

del ypred
```

Figura 73 – Método de desenho da matriz de confusão

Da informação da Tabela 17, pode-se concluir que ambos os modelos obtiveram bons resultados na categorização de golo ou *kick-off* ao ser apresentado um exemplo de um destes. Desta é ainda possível refletir sobre as diferentes configurações testadas (quantidade de filtros, *kernel size* e uso de *dropout*) e comparar os melhores resultados das melhores experiências.

Tabela 17 – Avaliação das experiências dos modelos *ConvLSTM* e *LRCN*

Modelo	Experiência	Exatidão	*Loss*	Precisão	*Recall*	F1-Measure
ConvLSTM	A	97,58	8,16	98 - 97	97 - 98	98
	B	98,02	7,77	98 - 98	98 - 98	98
	C	97,14	9,22	97 - 97	97 - 97	97
	D	98,02	8,42	97 - 99	99 - 97	98
	E	97,58	7,96	98 - 97	97 - 98	98
	F	98,66	8,15	99 - 99	99 - 99	99
	G	98,24	7,68	98 - 99	99 - 98	98
	H	96,70	11,63	95 - 98	98 - 95	97
	I	97,14	15,19	97 - 98	98 - 96	97
	J	98,02	13,15	97 - 99	99 - 98	98
	K	97,80	11,88	96 - 100	100 - 96	98
	L	97,14	14,38	95 - 100	100 - 95	97
LRCN	A	98,68	4,32	99 - 98	98 - 99	99
	B	98,90	4,89	99 - 99	99 - 99	99
	C	99,12	3,75	99 - 100	100 - 99	99
	D	99,56	2,95	100 - 100	100 - 100	100
	E	98,68	5,83	99 - 99	99 - 99	99
	F	98,68	4,62	98 - 100	100 - 98	99
	G	99,12	3,53	99 - 99	99 - 99	99
	H	99,56	2,86	100 - 100	100 - 100	100
	I	98.90	4,56	99 - 99	99 - 99	99
	J	99,12	4,35	99 - 99	99 - 99	99
	K	99,12	4,06	99 - 100	100 - 99	99
	L	99,56	3,26	100 - 100	100 - 100	100

*Nas colunas de precisão e *recall*, o primeiro valor é referente ao evento de pontapé de meio-campo (*kick-off)* e o segundo ao evento de golo.

Para a abordagem *LRCN*, a utilização de uma camada mostrou ser benéfica em geral (diminuição de *loss* de teste) e as experiências com o *kernel size* configurado para 5x5 obteve melhores resultados que os testes equivalentes configurados com 3x3. Em termos de número de filtros, não foi possível retirar nenhuma conclusão, por exemplo os valores das experiências D, H e L (equivalentes, sendo a única diferença o número de filtros por camada) são quase iguais. A partir desta avaliação pode-se concluir que a experiência com os melhores resultados nesta abordagem era a H.

No caso do modelo *ConvLSTM,* pode-se concluir que o *dropout* teve o efeito contrário (valores de *loss* mais elevados para experiências em que foram aplicadas camadas de *dropout*). Em termos da configuração *kernel size*, os valores obtidos nas experiências com 3x3 e 5x5 foram semelhantes, não indicando o benefício de um sobre o outro. No entanto, ao contrário dos

testes da rede *LRCN*, pode-se observar que a *loss* de teste aumentou ao longo do aumento do número de filtros por camada. Isto é, as experiências E, F, G e H, configuradas com menos filtros por cada camada (8-8-4), apresentaram melhores resultados que as experiências com 16-16-8 filtros, que por sua vez, obtiveram melhores resultados que os modelos com 32-32-16 filtros. Para este modelo pode-se concluir que a experiência que atingiu melhores resultados foi a G.

Ao comparar as duas abordagens implementadas através das suas melhores experiências (H do *LRCN* e G do *ConvLSTM*), é possível observar que o modelo *LRCN* obteve melhores resultados. Sendo bastante visível a diferença entre os valores de *loss* de teste obtidos na comparação entre as duas melhores experiências (4,82 de diferença).

5.5 Avaliação solução final

Como visto no capítulo de Deteção de golos, apesar da avaliação de ambos os modelos ser positiva, os resultados obtidos durante a solução final não são os melhores. Esta discrepância deve-se maioritariamente a sequências que são desconhecidas à rede. Ou seja, a rede sabe distinguir, com bons resultados, um golo de um *kick-off* e durante o treino aprendeu as características que determinam cada evento. No entanto, ao lhe ser apresentado uma sequência que o modelo não reconhece, este tenta classificar como um dos eventos definidos, principalmente em situações semelhantes a estes.

Numa tentativa de melhorar a quantidade de falsos positivos e obter avaliações dos modelos mais fidedignas, foi configurada uma terceira classe "Outro". Nesta foram adicionados exemplos variados de outros eventos que acontecem em jogo, utilizando o mesmo dataset. Contudo os resultados obtidos em teste foram piores, dado que essa classe tinha características muito genéricas, o que levava a rede a classificar tudo como "outro".

Outra tentativa foi a introdução de mais classes (3/4), como por exemplo penálti, mas não fez grande diferença já que apenas esta tinha semelhanças com uma das classes. Isto é, a rede continuava a classificar de forma errada sequências semelhantes a golo ou *kick-off*, por exemplo passe lento na zona do meio-campo.

Outra possível solução idealizada, mas não explorada, foi a utilização de um modelo para a classificação de cada evento, com as seguintes classes: "sim", "não, mas semelhante" e "não". Desta forma, seria possível mitigar os falsos positivos e ainda aumentar a aprendizagem de características específicas a cada evento. Contudo, não foi encontrado um *dataset* com as anotações necessárias, nem foi possível recolher estes manualmente, principalmente devido à quantidade necessária para obter bons resultados e questões de tempo. Para além disso, seria o dobro do treino necessário e iria ser necessário ainda mais recursos, sendo eles escassos. Para esta abordagem seria também necessário alterar a solução de deteção de golos, para passar cada sequência por ambos os modelos e analisar os dados por sobreposição.

Em suma, pode-se concluir que a hipótese proposta no início desta secção é possível, mas que não foi totalmente alcançada neste projeto.

6 Conclusão

Esta última secção é usada para refletir sobre as limitações sentidas ao longo do projeto e as possíveis melhorias e trabalho futuro.

Por fim, é feita uma apreciação geral do projeto discutindo os objetivos alcançados.

6.1 Limitações

A maior limitação sentida ao longo deste projeto foi a capacidade da máquina utilizada. Apesar de ter sido apresentada como a melhor opção a utilizar, o processamento e extração de *frames* de vídeo exige uma grande quantidade de *RAM* e o treino requer bastante memória da placa gráfica (*VRAM*). Sendo assim, certas experiências realizadas tiveram de ser modificadas devido à exaustão dos recursos, como por exemplo a alteração do *batch size* consoante a quantidade de filtros nas experiências *ConvLSTM.*

Outra limitação relativa à capacidade dos recursos foi o armazenamento de jogos de futebol, devido ao tamanho de cada ficheiro. Para ultrapassar parte desta limitação, foi utilizado um disco externo para garantir mais capacidade de armazenamento (1TB). Contudo, mesmo com a utilização desse disco não foi possível usufruir de todos os jogos presentes no *dataset*, mostrando a necessidade de ainda maior capacidade de armazenamento.

Por último, apesar da quantidade de vídeos, nem sempre estes tinham os eventos necessários visíveis. Esta limitação fez-se notar principalmente na recolha de exemplos de *kick-off,* já que os pontapés de meio-campo iniciais eram visíveis, mas o mesmo não acontecia para despois de um golo. Grande parte deste tipo de eventos está classificado como "não visível" devido à existência da repetição do golo e outros ângulos da camara para, por exemplo, mostrar a celebração dos adeptos.

6.2 Melhorias

Consoante mencionado nas limitações, a capacidade da máquina afeta o desempenho e, até mesmo a forma como os modelos e solução é construída e treinada. Desta forma, uma das melhorias seria a obtenção e utilização de uma máquina virtual ou física com maior capacidade.

Relativamente aos modelos, uma melhoria já mencionada anteriormente, seria o dar a conhecer aos modelos parte do "desconhecido", de modo a diminuir os falsos positivos obtidos. Uma possível solução seria o treino e classificação de exemplos de momentos de jogo semelhantes às classes originais, tal como descrito na secção de Avaliação.

6.3 Trabalho futuro

Após as melhorias, um possível primeiro passo seria a criação de uma *dataset* próprio com apenas vídeos de jogos de equipas amadoras/juniores. Deste modo, era possível treinar e avaliar a solução para jogos gravados com menos recursos e ainda, obter mais exemplos de *kick-off*, já que normalmente os vídeos deste tipo de jogos não têm edições.

Também seria interessante, retirar e catalogar nesse *dataset* momentos de celebração (jogadores e adeptos), sendo que este faz parte da definição de golo feita na secção de Análise e *Design*.

O uso de áudio para o reconhecimento dos momentos de golo também seria um desafio interessante. Sendo que este permitiria adicionar mais uma camada de garantia aos momentos previstos e, ajudar nos casos onde a sequência de golo mínima (golo + *kick-off*) não é visível.

Outro trabalho futuro não tão importante, já que o objetivo é a utilização de vídeos sem grandes edições, seria o reconhecimento e extração das repetições para também diminuir falsas classificações. Este processo poderia ser feito, por exemplo pelo reconhecimento das *frames* de transição que normalmente aparecem entre as repetições.

6.4 Apreciação final

Em termos dos objetivos propostos na secção de Introdução, pode-se concluir que foram maioritariamente alcançados, mas com uma margem de erro. A solução desenvolvida permite interpretar vídeos de jogos de futebol e catalogar os momentos de golo, registando o instante em que o mesmo aconteceu. No entanto, cataloga também vários falsos positivos, principalmente resultantes de situações semelhantes, como por exemplo, oportunidade de golo seguida de um contra-ataque (meio-campo).

O algoritmo de classificação de golos foi treinado utilizando vídeos de diferentes qualidades (224p e 720p), tal como os objetivos requeriam. Contudo, foram utilizados vídeos de

transmissão televisiva de jogos de equipas profissionais, ou seja, com vários ângulos de diversas câmaras, não sendo possível realizar a deteção de golos em jogos de equipas amadores gravados por dispositivos não profissionais.

Em suma, o projeto foi bastante desafiante devido à falta de experiência com algoritmos de visão computacional e IA no geral. Contudo este permitiu estudar e explorar conceitos de IA, fora e dentro do tema, permitindo obter competências numa área cada vez mais abrangente e em constante expansão.

Referências

Ajgaonkar, A. (2021). The Value of Computer Vision: More Than Meets the Eye. *Tech Journal*. https://www.insight.com/en_US/content-and-resources/tech-journal/spring-2021/the-value-of-computer-vision--more-than-meets-the-eye.html

Akhaee, A., Toosi, R., Yaghoobpour, A., Masroori, I., Aghadavood, A., Kazerooni, A., Karimi, A., Alavi, M., & Rostami, A. (2021a). *Football Analysis*. https://footballanalysis.github.io/#featured-services

Akhaee, A., Toosi, R., Yaghoobpour, A., Masroori, I., Aghadavood, A., Kazerooni, A., Karimi, A., Alavi, M., & Rostami, A. (2021b). *FootballAnalysis · GitHub*. https://github.com/FootballAnalysis

Alake, R. (2020, January 2). *Understanding Motion Analysis in Machine Learning*. https://towardsdatascience.com/understanding-motion-analysis-in-machine-learning-f504e9987413

Alexandre Xavier. (2019). *An introduction to ConvLSTM. Nowadays it is quite common to find… | by Alexandre Xavier | Neuronio | Medium*. https://medium.com/neuronio/an-introduction-to-convlstm-55c9025563a7

Anaconda. (2017). *Miniconda*. https://docs.conda.io/en/latest/miniconda.html

Aniruddha Bhandari. (2020). Confusion Matrix for Machine Learning. In *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2020/04/confusion-matrix-machine-learning/

Anwar, T., Naeem, R., & Anjum, M. (2021). *Human Activity Recognition using TensorFlow (CNN + LSTM) | 2 Methods - YouTube*. https://www.youtube.com/watch?v=QmtSkq3DYko

Atlas Systems. (2020). *Tensorflow vs Keras vs Pytorch: Which Framework is the Best?* Medium. https://medium.com/@AtlasSystems/tensorflow-vs-keras-vs-pytorch-which-framework-is-the-best-f92f95e11502

Aurélien Géron. (2019). Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems. In *O'Reilly Media* (2nd ed.). O'Reilly. https://learning.oreilly.com/library/view/hands-on-machine-learning/9781492032632/ch04.html#idm45022191905928

Brown, A. (2017). *Introduction to Object Detection & Image Segmentation*. nvidia. https://on-demand.gputechconf.com/gtcdc/2017/presentation/dc7217-abel-brown-deep-learning-object-detection-and-segmentation.pdf

Brownlee, J. (2016). *Supervised and Unsupervised Machine Learning Algorithms*. Machine Learning Mastery. https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

Brownlee, J. (2017). *Why One-Hot Encode Data in Machine Learning?* Machine Learning Mastery. https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-

learning/

Brownlee, J. (2019, June 17). *What is the Difference Between a Parameter and a Hyperparameter?* Machine Learning Mastery. https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/

Brownlee, J. (2020). Train-Test Split for Evaluating Machine Learning Algorithms. In *Machine Learning Mastery* (pp. 4–5). https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/

Brownlee, J. (2021, April 27). *One-vs-Rest and One-vs-One for Multi-Class Classification*. Machine Learning Mastery. https://machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/

Brownlee, J. (2022, August 15). *Difference Between a Batch and an Epoch in a Neural Network*. Machine Learning Mastery. https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

Chen, B. (2020). A Practical Introduction to Keras Callbacks in TensorFlow 2. *Towards Data Science*, 1. https://towardsdatascience.com/a-practical-introduction-to-keras-callbacks-in-tensorflow-2-705d0c584966

ChyronHego. (2021). *Tracab*. https://tracab.com/products/tracab-technologies/

Costa, C. D. (2020). *Python Libraries for Machine Learning and Deep Learning | by Claire D. Costa | Towards Data Science*. Towardsdatascience. https://towardsdatascience.com/best-python-libraries-for-machine-learning-and-deep-learning-b0bd40c7e8c

CS231n. (2021). CS231n Convolutional Neural Networks for Visual Recognition. *Stanford University*, 1–18. https://cs231n.github.io/convolutional-networks/#conv

Deliege, A., Cioppa, A., Giancola, S., Seikavandi, M. J., Dueholm, J. V, Nasrollahi, K., Ghanem, B., Moeslund, T. B., & Van Droogenbroeck, M. (2021). SoccerNet-v2: A dataset and benchmarks for holistic understanding of broadcast soccer videos. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 4503–4514. https://doi.org/10.1109/CVPRW53098.2021.00508

Delua, J. (2021). Supervised vs. Unsupervised Learning: What's the Difference? In *IBM*. https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning

Donahue, J., Hendricks, L. A., Rohrbach, M., Venugopalan, S., Guadarrama, S., Saenko, K., & Darrell, T. (2017). Long-Term Recurrent Convolutional Networks for Visual Recognition and Description. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(4), 677–691. https://doi.org/10.1109/TPAMI.2016.2599174

F. Li, J. J. and Y. S. (2020). CS231n: Convolutional Neural Networks for Visual Recognition. *Stanford University*, 1–9. https://cs231n.github.io/transfer-learning/#tf

Feng, N., Song, Z., Yu, J., Chen, Y. P. P., Zhao, Y., He, Y., & Guan, T. (2020). SSET: a dataset for

shot segmentation, event detection, player tracking in soccer videos. *Multimedia Tools and Applications*, *79*(39–40), 28971–28992. https://doi.org/10.1007/s11042-020-09414-3

FIFA. (2022a). *Football Technology & Innovation*. https://www.fifa.com/technical/football-technology

FIFA. (2022b). *Resource Hub*. https://www.fifa.com/technical/football-technology/resource-hub?QualityProgram=6Sshn3qiYsRBq6muymEEtY&Category=21vlZTNlv31aveduLGFmDi

Footovision. (2021). *Footovision*. https://www.footovision.com/

GeeksforGeeks. (2016). *Deque in Python - GeeksforGeeks*. https://www.geeksforgeeks.org/deque-in-python/

GeeksforGeeks. (2022). *Best Python libraries for Machine Learning*. GeeksforGeeks. https://www.geeksforgeeks.org/best-python-libraries-for-machine-learning/

Giancola, S., Amine, M., Dghaily, T., & Ghanem, B. (2018). SoccerNet: A scalable dataset for action spotting in soccer videos. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, *2018-June*, 1792–1802. https://doi.org/10.1109/CVPRW.2018.00223

Girshick, R. (2015). Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, *2015 Inter*, 1440–1448. https://doi.org/10.1109/ICCV.2015.169

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 580–587. https://doi.org/10.1109/CVPR.2014.81

Google. (2022a). *Dataset Search*. https://datasetsearch.research.google.com/

Google. (2022b). *Python, Java, C++, R, JavaScript*. Google Trends. https://trends.google.pt/trends/explore?cat=1299&date=today 5-y&q=%2Fm%2F05z1_,%2Fm%2F07sbkfb,%2Fm%2F0jgqg,R,%2Fm%2F02p97

Google Colab. (2022). *Damos-lhe as boas-vindas ao Colaboratory - Colaboratory*. https://colab.research.google.com/

Gupta, D. (2017). *Recurrent Neural Network | Fundamentals Of Deep Learning*. Analytics Vidya. https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/?utm_source=blog&utm_medium=cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning

Gupta, S. (2021). *What Is the Best Language for Machine Learning?* Springboard. https://www.springboard.com/blog/ai-machine-learning/best-language-for-machine-learning/

Gurney, K. (2018). *An Introduction to Neural Networks* (1st Editio). CRC Press. https://doi.org/10.1201/9781315273570

Hawk-Eye. (2022). *Hawk-Eye*. https://www.hawkeyeinnovations.com/about

He, K., Gkioxari, G., Dollar, P., & Girshick, R. (2017). Mask R-CNN. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *2018-Jan*(2), 386–397. https://doi.org/10.1109/TPAMI.2018.2844175

Hong, Y., Ling, C., & Ye, Z. (2018). End-to-end soccer video scene and event classification with deep transfer learning. *2018 International Conference on Intelligent Systems and Computer Vision, ISCV 2018*, *2018-May*, 1–4. https://doi.org/10.1109/ISACV.2018.8369043

Huang, C. L., Shih, H. C., & Chao, C. Y. (2006). Semantic analysis of soccer video using dynamic bayesian network. *IEEE Transactions on Multimedia*, *8*(4), 749–759. https://doi.org/10.1109/TMM.2006.876289

IBM. (2021a). *Quality metrics overview*. https://www.ibm.com/docs/en/cloud-paks/cp-data/3.0.1?topic=openscale-quality-metrics-overview

IBM. (2021b). *What is Overfitting? | IBM*. IBM. https://www.ibm.com/cloud/learn/overfitting

IBM Cloud Education. (2020). *Recurrent Neural Networks*. IBM. https://www.ibm.com/cloud/learn/recurrent-neural-networks

Janiesch, C., Zschech, P., & Heinrich, K. (2021a). Machine learning and deep learning. *Electronic Markets*, *31*(3), 685–695. https://doi.org/10.1007/s12525-021-00475-2

Janiesch, C., Zschech, P., & Heinrich, K. (2021b). Machine learning and deep learning. *Electronic Markets*, *31*(3), 685–695. https://doi.org/10.1007/s12525-021-00475-2

Jason Brownlee. (2016). What is a Confusion Matrix in Machine Learning. *Machinelearningmastery.Com*, 1–39. https://machinelearningmastery.com/confusion-matrix-machine-learning/

Jiang, H., Lu, Y., & Xue, J. (2017). *Automatic Soccer Video Event Detection Based on a Deep Neural Network Combined CNN and RNN*. 490–494. https://doi.org/10.1109/ictai.2016.0081

Jiang, Y., Cui, K., Chen, L., Wang, C., & Xu, C. (2020). SoccerDB: A Large-Scale Database for Comprehensive Video Understanding. *MMSports 2020 - Proceedings of the 3rd International Workshop on Multimedia Content Analysis in Sports*, *8*, 1–8. https://doi.org/10.1145/3422844.3423051

Jones, M. T. (2017). Arquiteturas de aprendizado profundo. In *IBM*. https://developer.ibm.com/br/articles/cc-machine-learning-deep-learning-architectures/

Jones, M. T. (2019). *Transfer learning for deep learning*. Https://Developer.Ibm.Com/. https://developer.ibm.com/articles/transfer-learning-for-deep-learning/

Jupyter. (2022). *Project Jupyter | Installing Jupyter*. https://jupyter.org/install

Karimi, A., Toosi, R., & Akhaee, M. A. (2021). *Soccer Event Detection Using Deep Learning*. https://arxiv.org/abs/2102.04331v1

Keras. (2022). *Next-Frame Video Prediction with Convolutional LSTMs*. https://keras.io/examples/vision/conv_lstm/

KerasSIG. (2020). *About Keras*. Keras Special Interest Group. https://keras.io/about/

Khan, A., Sohail, A., Zahoora, U., & Qureshi, A. S. (2020). A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, *53*(8), 5455–5516. https://doi.org/10.1007/s10462-020-09825-6

Kozhaya, J. (2017). *Train and evaluate custom machine learning models*. IBM Developer Blog. https://developer.ibm.com/blogs/train-and-evaluate-custom-machine-learning-models/

Lakshana. (2021). *4 Ways to Evaluate your Machine Learning Model: Cross-Validation Techniques*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/05/4-ways-to-evaluate-your-machine-learning-model-cross-validation-techniques-with-python-code/

LeCun, Y., Cortes, C., & Burges, C. J. C. (1998). *MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges*. http://yann.lecun.com/exdb/mnist/

Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., van der Laak, J. A. W. M., van Ginneken, B., & Sánchez, C. I. (2017). A survey on deep learning in medical image analysis. In *Medical Image Analysis* (Vol. 42, pp. 60–88). Elsevier. https://doi.org/10.1016/j.media.2017.07.005

Ma, S., Shao, E., Xie, X., & Liu, W. (2020). Event Detection in Soccer Video Based on Self-Attention. *2020 IEEE 6th International Conference on Computer and Communications, ICCC 2020*, 1852–1856. https://doi.org/10.1109/ICCC51575.2020.9344896

Microsoft. (2021). *Referência de componente de algoritmo & para designer de Azure Machine Learning*. https://docs.microsoft.com/pt-pt/azure/machine-learning/component-reference/component-reference

Microsoft. (2022a). *Aprendizagem profunda vs. machine Learning no Azure Machine Learning*. https://docs.microsoft.com/pt-pt/azure/machine-learning/concept-deep-learning-vs-machine-learning#deep-learning-machine-learning-and-ai

Microsoft. (2022b). *Downloads do Pacote Redistribuível do Visual C++ com suporte mais recentes | Microsoft Learn*. https://learn.microsoft.com/pt-PT/cpp/windows/latest-supported-vc-redist?view=msvc-170

NVIDIA. (2022, October 3). *Installation Guide :: NVIDIA Deep Learning cuDNN Documentation*. https://docs.nvidia.com/deeplearning/cudnn/install-guide/index.html#install-windows https://developer.nvidia.com/rdp/cudnn-download

Oladipupo, T. (2010). Types of Machine Learning Algorithms. In *New Advances in Machine Learning*. https://doi.org/10.5772/9385

Olah, C. (2015). Understanding LSTM Networks [Blog]. *Web Page*, 1–13. https://colah.github.io/posts/2015-08-Understanding-LSTMs/

OpenCV. (2018). *OpenCV: Flags for video I/O*. https://docs.opencv.org/3.4/d4/d15/group__videoio__flags__base.html

Pai, A. (2020). *Types of Neural Networks*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/

Pappalardo, L., Cintia, P., Rossi, A., Massucco, E., Ferragina, P., Pedreschi, D., & Giannotti, F. (2019). A public data set of spatio-temporal match events in soccer competitions. *Scientific Data*, *6*(1). https://doi.org/10.1038/s41597-019-0247-7

Pappalardo, L., & Massucco, E. (2019). *Soccer match event dataset. figshare.* https://doi.org/10.6084/M9.FIGSHARE.C.4415000

Python. (2022). *Download Python | Python.org*. Python. https://www.python.org/downloads/

Python Software Foundation. (2022). What is Python? Executive Summary. *Python Software Foundation*. https://www.python.org/doc/essays/blurb/

PyTorch. (2022). *PyTorch*. https://pytorch.org/features/

Ramos, C. (2021a). *Conjuntos de Dados , Treino, Validação e Teste de Redes Neuronais*. Apontamentos da Unidade Curricular de Aprendizagem Automática 2 do Mestrado em Engenharia de Inteligência Artificial do ISEP.

Ramos, C. (2021b). *Introdução à Aprendizagem Profunda (Deep Learning)*. Apontamentos da Unidade Curricular de Aprendizagem Automática 2 do Mestrado em Engenharia de Inteligência Artificial do ISEP.

Ramos, C. (2021c). *Redes Neuronais Convolucionais*. Apontamentos da Unidade Curricular de Aprendizagem Automática 2 do Mestrado em Engenharia de Inteligência Artificial do ISEP.

Rashidi, H. H., Tran, N. K., Betts, E. V., Howell, L. P., & Green, R. (2019). Artificial Intelligence and Machine Learning in Pathology: The Present Landscape of Supervised Methods. In *Academic Pathology* (Vol. 6, p. 2374289519873088). SAGE PublicationsSage CA: Los Angeles, CA. https://doi.org/10.1177/2374289519873088

Rastogi, S. (2021). *How to Apply K-Fold Averaging on Deep Learning Classifier*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/09/how-to-apply-k-fold-averaging-on-deep-learning-classifier/

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2015). You only look once: Unified, real-time object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-Dec*, 779–788. https://doi.org/10.1109/CVPR.2016.91

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object

Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *2016-Jan*, 1137–1149. https://doi.org/10.1109/TPAMI.2016.2577031

Renotte, N. (2021a). *GitHub - nicknochnack/TFODCourse*. https://github.com/nicknochnack/TFODCourse

Renotte, N. (2021b). Tensorflow Object Detection in 5 Hours with Python | Full Course with 3 Projects. In *https://www.youtube.com/watch?v=yqkISICHH-U&t=4387s&ab_channel=NicholasRenotte*. https://www.youtube.com/watch?v=yqkISICHH-U

Ruscica, T. (2020, March 3). *TensorFlow 2.0 Complete Course - Python Neural Networks for Beginners Tutorial - YouTube*. https://www.youtube.com/watch?v=tPYj3fFJGjk&list=LL&index=17

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., & Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, *115*(3), 211–252. https://doi.org/10.1007/s11263-015-0816-y

Saaty, T. L. (1990). How to make a decision: The analytic hierarchy process. *European Journal of Operational Research*, *48*(1), 9–26. https://doi.org/10.1016/0377-2217(90)90057-I

Samaya, M., Sidra, A., Vinay, R., & Anto, J. (2021, March 12). *Compare deep learning frameworks – IBM Developer*. IBM. https://developer.ibm.com/articles/compare-deep-learning-frameworks/#

Saravanan, R., & Sujatha, P. (2019). A State of Art Techniques on Machine Learning Algorithms: A Perspective of Supervised Learning Approaches in Data Classification. *Proceedings of the 2nd International Conference on Intelligent Computing and Control Systems, ICICCS 2018*, 945–949. https://doi.org/10.1109/ICCONS.2018.8663155

Scikit Learn. (2022). *sklearn.model_selection.train_test_split — scikit-learn 1.1.2 documentation*. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. In *Understanding Machine Learning: From Theory to Algorithms* (Vol. 9781107057). https://doi.org/10.1017/CBO9781107298019

Shi, X., Chen, Z., Wang, H., Yeung, D. Y., Wong, W. K., & Woo, W. C. (2015). Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *Advances in Neural Information Processing Systems*, *2015-Janua*, 802–810. https://doi.org/10.48550/arxiv.1506.04214

Singh, A. (2020). Demystifying the Mathematics Behind Convolutional Neural Networks (CNNs). *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2020/02/mathematics-behind-convolutional-neural-network/?utm_source=blog&utm_medium=cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning

SlashData. (2021). *State of the Developer Nation 21st Edition - Q3 2021*. https://www.developernation.net/resources/reports/

SoccerNet. (2022). *SoccerNet - Data*. https://www.soccer-net.org/data

Stack Overflow. (2022). *Stack Overflow Trends*. Stack Overflow Insights. https://insights.stackoverflow.com/trends?tags=tensorflow%2Ckeras%2Cpytorch

Stanford Vision Lab, Stanford University, & Princeton University. (2020). *ImageNet*. https://image-net.org/download.php

Steiner, S., & Allan, V. (2021). *How digital innovation evolved the fan experience*. SIRCuit. https://sirc.ca/blog/digital-innovation-and-the-fan-experience/

Szeliski, R. (2021). Computer Vision : Algorithms and Applications 2nd Edition. In *Springer*. https://szeliski.org/Book,

TensorFlow. (2019). *Announcing TensorFlow 2.0 (Coding TensorFlow)*. https://www.youtube.com/watch?v=EqWsPO8DVXk

TensorFlow. (2022a). *Build from source*. https://www.tensorflow.org/install/source_windows#gpu

TensorFlow. (2022b). *Install TensorFlow*. Https://Www.Tensorflow.Org/Install/Pip. https://www.tensorflow.org/install/pip#windows-native_1

TensorFlow. (2022c). *tf.keras.callbacks.EarlyStopping | TensorFlow v2.10.0*. https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping

TensorFlow. (2022d). *tf.keras.callbacks.ReduceLROnPlateau | TensorFlow v2.10.0*. https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ReduceLROnPlateau

TensorFlow. (2022e). *tf.keras.layers.TimeDistributed | TensorFlow v2.10.0*. https://www.tensorflow.org/api_docs/python/tf/keras/layers/TimeDistributed

TensorFlow. (2022f). *tf.keras.Model | TensorFlow v2.10.0*. https://www.tensorflow.org/api_docs/python/tf/keras/Model

TensorFlow. (2022g). *tf.keras.models.load_model | TensorFlow v2.10.0*. https://www.tensorflow.org/api_docs/python/tf/keras/models/load_model

TensorFlow. (2022h). *tf.keras.utils.to_categorical | TensorFlow v2.10.0*. https://www.tensorflow.org/api_docs/python/tf/keras/utils/to_categorical

TensorFlow. (2022i, January 19). *Use a GPU | TensorFlow Core*. https://www.tensorflow.org/guide/gpu

TensorFlow. (2022j). *TensorFlow*. https://doi.org/10.5281/ZENODO.5949169

Track160. (2021). *Track160*. https://www.track160.com/football-analytics-technology

Tutorials Teacher. (2022). *Python - Read and Write Files*. https://www.tutorialsteacher.com/python/python-read-write-file

Vahdani, E., & Tian, Y. (2021). *Deep Learning-based Action Detection in Untrimmed Videos: A Survey*. https://arxiv.org/abs/2110.00111v1

Value Analysis Canada. (2017). *Function Analysis system Technique (FAST) - Canadian Society of Value Analysis*. https://www.valueanalysis.ca/fast.php

Waymo. (2022). *Waymo*. https://waymo.com

Wickramaratna, K., Chen, M., Shu-Ching Chen, & Mei-Ling Shyu. (2005). Neural network based framework for goal event detection in soccer videos. *Seventh IEEE International Symposium on Multimedia (ISM'05), 2005*, 8 pp. https://doi.org/10.1109/ISM.2005.83

Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. In *Insights into Imaging* (Vol. 9, Issue 4, pp. 611–629). SpringerOpen. https://doi.org/10.1007/s13244-018-0639-9

Anexo A

Notebook 1 – *Download dataset*

```
import SoccerNet
from SoccerNet.Downloader import SoccerNetDownloader
from SoccerNet.utils import getListGames
```

```
%store -r local_dataset_dir

#disco externo
local_dataset_dir = "D:/cat/labels"

mySoccerNetDownloader=SoccerNetDownloader(LocalDirectory = local_dataset_dir)
```

```
%store -r soccernet_pw
mySoccerNetDownloader.password = soccernet_pw
```

```
mySoccerNetDownloader.downloadGames(files=["1_720p.mkv", "2_720p.mkv"], split=["train","valid","test","challenge"])
```

```
mySoccerNetDownloader.downloadGames(files=["1_720p.mkv", "2_720p.mkv"])
```

```
mySoccerNetDownloader.downloadGames(files=["Labels-v2.json"], split=["train","valid","test"])
```

```
mySoccerNetDownloader.downloadGames(files=["Labels-v2.json"])
```

```
print(getListGames(split="test")) # return list of games recommended for testing
```

```
print(getListGames(split="train")) # return list of games recommended for training
```

```
print(getListGames(split="valid")) # return list of games recommended for validation
```

```
print(getListGames(split=["train", "valid", "test", "challenge"])) # return list of games for training, validation and testing
```

Anexo B

Notebook 2 – Configurações

Diretórios

```
local_dir = "C:/Users/catar/Documentos/Tese/Projeto"
%store local_dir
del local_dir
```

```
Stored 'local_dir' (str)
```

```
local_models_dir = "C:/Users/catar/Documentos/Tese/Projeto/Modelos"
%store local_models_dir
del local_models_dir
```

```
Stored 'local_models_dir' (str)
```

```
test_videos_dir = "C:/Users/catar/Documentos/Tese/Projeto/test_videos"
%store test_videos_dir
del test_videos_dir
```

```
Stored 'test_videos_dir' (str)
```

```
dataset_dir = "D:/cat"
%store dataset_dir
del dataset_dir
```

```
Stored 'dataset_dir' (str)
```

Frames

```
# 2 before, 1 on the same minute, 3 after
frame_seconds_diff = [-2, -1, 0, 1, 2, 3]
%store frame_seconds_diff
del frame_seconds_diff
```

```
Stored 'frame_seconds_diff' (list)
```

Lista das classes

```
classes_list = ["Kick-off", "Goal"]
%store classes_list
del classes_list
```

```
Stored 'classes_list' (list)
```

Altura e largura para cada frame

```
image_width = 128
%store image_width
del image_width
```

```
Stored 'image_width' (int)
```

```
image_height= 72
%store image_height
del image_height
```

```
Stored 'image_height' (int)
```

Check gpu ¶

```
import tensorflow as tf
from tensorflow.python.client import device_lib
```

```
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    try:
        # Currently, memory growth needs to be the same across GPUs
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
            logical_gpus = tf.config.list_logical_devices('GPU')
            print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
    except RuntimeError as e:
        # Memory growth must be set before GPUs have been initialized
        print(e)
```

```
1 Physical GPUs, 1 Logical GPUs
```

```
tf.test.is_built_with_cuda()
```

```
True
```

```
tf.config.list_physical_devices('GPU')
```

```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

```
device_lib.list_local_devices()
```

```
[name: "/device:CPU:0"
 device_type: "CPU"
 memory_limit: 268435456
 locality {
 }
 incarnation: 11995790138886803972
 xla_global_id: -1,
 name: "/device:GPU:0"
 device_type: "GPU"
 memory_limit: 2254123828
 locality {
   bus_id: 1
   links {
   }
 }
 incarnation: 5121855470925404363
 physical_device_desc: "device: 0, name: NVIDIA GeForce GTX 1650, pci bus id: 0000:01:00.0, compute capability: 7.5"
 xla_global_id: 416903419]
```

Anexo C

Notebook 3 – Processamento do *dataset*

Constantes

In [1]:
```
# Altura e largura para cada frame
%store -r local_dir
LOCAL_DIR = local_dir

# Diretório do dataset
%store -r dataset_dir
DATASET_DIR = dataset_dir

# Classes para treino
%store -r classes_list
CLASSES_LIST = classes_list

# Altura de cada frame
%store -r image_height
IMAGE_HEIGHT = image_height

# Largura de cada frame
%store -r image_width
IMAGE_WIDTH = image_width
```

```
#disco externo
VIDEO_QUALITY_FOLDERS = ["720p", "224P"]

#Number of examples per classe
KICK_OFF_MAX_QUANTITY = 1136
GOAL_MAX_QUANTITY = 1136

# Partes dos vídeos do jogo
PARTS_1_FILE_NAME = "1"
PARTS_2_FILE_NAME = "2"

# Extensão dos vídeos
VIDEO_FILE_EXT = ".mkv"

# Nome do ficheiro das labels
LABELS_FILE_NAME = "Labels-v2.json"

#Nome do atribute das anotações no ficheiro das labels
LABELS_ATTRIBUE_NAME = "annotations"

#Frames - 2 before, 1 on the same minute, 3 after
%store -r frame_seconds_diff
FRAME_SECONDS_DIFF = frame_seconds_diff
```

Imports

```
import cv2
import os
import pandas as pd
import numpy as np
import random
import tensorflow as tf

from glom import glom

from tensorflow.keras.utils import to_categorical

from sklearn.model_selection import train_test_split
```

Auxiliares

```
#Minutos(string) para segundos (número - int)
def get_seconds(time_in_minutes_str):
    m, s = time_in_minutes_str.split(':')
    return int(m) * 60 + int(s)
```

Pre-processamentos (frames)

```
def frames_extraction_per_label(video_path, time_in_seconds):
    # Lista de frames do vídeo
    frames_list = []

    # Leitura do vídeo - VideoCapture
    video_reader = cv2.VideoCapture(video_path)

    fps = video_reader.get(cv2.CAP_PROP_FPS)

    if fps <= 0:
        return frames_list

    frameRate  = fps / 100

    for frame_second_diff in FRAME_SECONDS_DIFF:

        if(time_in_seconds == 0):
            time = time_in_seconds + frame_second_diff + 2
        else:
            time = time_in_seconds + frame_second_diff

        time = time + frameRate
        time = round(time, 2)
```

```
        video_reader.set(cv2.CAP_PROP_POS_MSEC, time*1000)

        # Leitura da frame
        success, frame = video_reader.read()

        # Validação da leitura
        if not success:
            print(f'break')
            break

        # Resize da frame
        resized_frame = cv2.resize(frame, (IMAGE_WIDTH, IMAGE_HEIGHT))

        # Normalizar a frame (/255 -- cada pixel entre 0 e 1)
        normalized_frame = resized_frame / 255

        # Adicionar frame à lista
        frames_list.append(normalized_frame)

    # VideoCapture release
    video_reader.release()

    return frames_list
```

Criação do dataset processado ¶

```
def create_dataset():
    features = []
    labels = []
    video_files_paths = []

    nr_of_features_kick_off = 0
    nr_of_features_goal = 0

    for video_quality_folder in VIDEO_QUALITY_FOLDERS:

        print(f'Extracting data of folder: {video_quality_folder}')
        folder_dir = os.path.join(DATASET_DIR, video_quality_folder)
        path_exists = os.path.exists(folder_dir)
        print(f'folder_dir: {video_quality_folder}')

        if (nr_of_features_kick_off != KICK_OFF_MAX_QUANTITY or nr_of_features_goal != GOAL_MAX_QUANTITY) and path_exists:

            leagues = os.listdir(folder_dir)

            for league in leagues:

                print(f'Extracting data of folder - league: {video_quality_folder} - {league}')
                league_dir = os.path.join(folder_dir, league)

                if (nr_of_features_kick_off != KICK_OFF_MAX_QUANTITY or nr_of_features_goal != GOAL_MAX_QUANTITY):

                    seasons_in_league = os.listdir(league_dir)

                    for season in seasons_in_league:

                        print(f'Extracting data of folder - league - season: {video_quality_folder} - {league} - {season}')
                        season_dir = os.path.join(league_dir, season)

                        if (nr_of_features_kick_off != KICK_OFF_MAX_QUANTITY or nr_of_features_goal != GOAL_MAX_QUANTITY):

                            games = os.listdir(season_dir)

                            for game in games:

                                if (nr_of_features_kick_off != KICK_OFF_MAX_QUANTITY or nr_of_features_goal != GOAL_MAX_QUANTITY):

                                    game_dir = os.path.join(season_dir, game)

                                    print(f'Extracting data of folder - league - season - game: {video_quality_folder} - {league} - {season} - {game}')
                                    files_list = os.listdir(game_dir)

                                    # Validar se existe o ficheiro de labels
                                    print(f'Extracting data of folder - league - season - game - labels: {video_quality_folder} - {league} - {season} - {game} - {LABELS_FILE_NAME}')
                                    labels_file_path = os.path.join(game_dir, LABELS_FILE_NAME)
                                    labels_file_exists = os.path.exists(labels_file_path)

                                    # Pastas com o ficheiro Labels-v2.json, mas sem videos do jogo
                                    ## Validar se existe vídeo da primeira parte do video
                                    fist_part_video_path = os.path.join(game_dir, PARTS_1_FILE_NAME + "_"  + video_quality_folder + VIDEO_FILE_EXT)
                                    fist_part_video_exists = os.path.exists(fist_part_video_path)
```

```
                                    ## Validar se existe vídeo da segunda parte do video
                                    second_part_video_path = os.path.join(game_dir, PARTS_2_FILE_NAME + "_" + video_quality_folder + VIDEO_FILE_EXT)
                                    second_part_video_exists = os.path.exists(second_part_video_path)

                                    if(labels_file_exists and fist_part_video_exists and second_part_video_exists):

                                        labels_file = pd.read_json(labels_file_path)

                                        label_names = labels_file['annotations'].apply(lambda row: glom(row, 'label'))
                                        game_times = labels_file['annotations'].apply(lambda row: glom(row, 'gameTime'))
                                        visibility_info = labels_file['annotations'].apply(lambda row: glom(row, 'visibility'))

                                        imp_annotations_indexs = [label_index for label_index, x in enumerate(label_names)
                                                                  if x in CLASSES_LIST and
                                                                  visibility_info[label_index] == "visible"]

                                        imp_annotations_labels = [x for label_index, x in enumerate(label_names)
                                                                  if x in CLASSES_LIST and
                                                                  visibility_info[label_index] == "visible"]

                                        imp_game_times = [game_time for game_time_index, game_time in enumerate(game_times)
                                                          if game_time_index in imp_annotations_indexs]

                                        for game_time_index, game_time in enumerate(imp_game_times): #game_times

                                            game_time_info = game_time.split(" - ")

                                            #part (1 or 2)
                                            video_name = game_time_info[0] + "_" + video_quality_folder + VIDEO_FILE_EXT
                                            print(f'Extracting data of folder - league - season - game - video: {video_quality_folder} - {league} - {season} - {game} - {video_name}')

                                            video_path = os.path.join(game_dir, video_name)
                                            video_path_exists = os.path.exists(video_path)

                                            if video_path_exists:

                                                #time in seconds
                                                time_in_minutes_str = game_time_info[1]
                                                time_in_seconds = get_seconds(time_in_minutes_str)

                                                frames = frames_extraction_per_label(video_path, time_in_seconds)

                                                label_str = label_names[game_time_index]

                                                class_index = CLASSES_LIST.index(imp_annotations_labels[game_time_index])

                                                #check if frames found - nr of frames = tamanho da lista FRAME_SECONDS_DIFF
                                                if len(frames) == len(FRAME_SECONDS_DIFF):

                                                    if class_index == 0 and nr_of_features_kick_off != KICK_OFF_MAX_QUANTITY:
                                                        features.append(frames)
                                                        labels.append(class_index)
                                                        nr_of_features_kick_off = nr_of_features_kick_off + 1
                                                        print(f'nr_of_features_kick_off: {nr_of_features_kick_off}')

                                                    elif class_index == 1 and nr_of_features_goal != GOAL_MAX_QUANTITY:
                                                        features.append(frames)
                                                        labels.append(class_index)
                                                        nr_of_features_goal = nr_of_features_goal + 1
                                                        print(f'nr_of_features_goal: {nr_of_features_goal}')
```

```
                                                    del frames
                                                    del class_index

    # List to numpy arrays
    features = np.asarray(features)
    labels = np.array(labels)

    return features, labels
```

```
features, labels = create_dataset()
```

```
Extracting data of folder - league - season - game - video: 720p - england_epl - 2014-2015
break
Extracting data of folder - league - season - game: 720p - england_epl - 2014-2015 - 2015-
Extracting data of folder - league - season - game - labels: 720p - england_epl - 2014-201
Extracting data of folder - league - season - game - video: 720p - england_epl - 2014-2015
nr_of_features_kick_off: 3
Extracting data of folder - league - season - game - video: 720p - england_epl - 2014-2015
nr_of_features_goal: 2
Extracting data of folder - league - season - game - video: 720p - england_epl - 2014-2015
nr_of_features_kick_off: 4
Extracting data of folder - league - season - game - video: 720p - england_epl - 2014-2015
break
Extracting data of folder - league - season - game - video: 720p - england_epl - 2014-2015
nr_of_features_kick_off: 5
Extracting data of folder - league - season - game - video: 720p - england_epl - 2014-2015
nr_of_features_goal: 3
Extracting data of folder - league - season - game: 720p - england_epl - 2014-2015 - 2015-
Extracting data of folder - league - season - game - labels: 720p - england_epl - 2014-201
Extracting data of folder - league - season - game - video: 720p - england_epl - 2014-2015
nr_of_features_kick_off: 6
```

```
features.shape
```

```
(2272, 6, 72, 128, 3)
```

```
len(features)
```

```
2272
```

```
len(labels)
```

```
2272
```

```
# Using Keras's to_categorical method to convert labels into one-hot-encoded vectors
one_hot_encoded_labels = to_categorical(labels, len(CLASSES_LIST))
```

```
labels.shape
```

```
(2272,)
```

```
one_hot_encoded_labels.shape
```

```
(2272, 2)
```

```
del labels
```

Split data (Treino e teste)

```
features_train, features_test, labels_train, labels_test = train_test_split(features,
                                                                            one_hot_encoded_labels,
                                                                            test_size = 0.20,
                                                                            shuffle = True,
                                                                            stratify=one_hot_encoded_labels)
```

```
features_train, features_val, labels_train, labels_val = train_test_split(features_train,
                                                                          labels_train,
                                                                          test_size=0.20,
                                                                          shuffle=True,
                                                                          stratify=labels_train)
```

```
len(labels_test)
```

```
455
```

```
len(labels_val)
```

```
364
```

```
len(labels_train)
```

```
1453
```

```
del one_hot_encoded_labels
del features
```

Guardar arrays para uso no treino, validação e teste

```
def save_info_to_file(file_name, info_to_save):
    file_path = os.path.join(LOCAL_DIR, file_name)
    file_exists = os.path.exists(file_path)

    if file_exists:
        os.remove(file_path)

    file = open(file_name, "wb")
    np.save(file, info_to_save)
    file.close
```

```
save_info_to_file("features_train_file", features_train)
del features_train
```

```
save_info_to_file("features_test_file", features_test)
del features_test
```

```
save_info_to_file("features_val_file", features_val)
del features_val
```

```
save_info_to_file("labels_train_file", labels_train)
del labels_train
```

```
save_info_to_file("labels_test_file", labels_test)
del labels_test
```

```
save_info_to_file("labels_val_file", labels_val)
del labels_val
```

Anexo D

***Notebooks* 4 (a,b,c,d,e,f,g,h,i) - ConvLSTM**

Constantes

```
# Diretório do dataset
%store -r local_dataset_dir
DATASET_DIR = local_dataset_dir

# Diretório modelos
%store -r local_models_dir
MODELS_DIR = local_models_dir

# Frames
%store -r frame_seconds_diff
FRAME_SECONDS_DIFF = frame_seconds_diff
NR_OF_FRAMES = len(FRAME_SECONDS_DIFF)

# Altura de cada frame
%store -r image_height
IMAGE_HEIGHT = image_height

# Largura de cada frame
%store -r image_width
IMAGE_WIDTH = image_width

# Classes para treino
%store -r classes_list
CLASSES_LIST = classes_list
```

Imports

```
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import *

from tensorflow.keras.utils import plot_model

from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ReduceLROnPlateau

import datetime as dt
import matplotlib.pyplot as plt

import numpy as np
```

Features e labels

```
file = open("features_train_file", "rb")
FEATURES_TRAIN_STORED = np.load(file)
len(FEATURES_TRAIN_STORED)
```

```
1453
```

```
file = open("features_test_file", "rb")
FEATURES_TEST_STORED = np.load(file)
len(FEATURES_TEST_STORED)
```

```
455
```

```
file = open("features_val_file", "rb")
FEATURES_VAL_STORED = np.load(file)
len(FEATURES_VAL_STORED)
```

```
364
```

```
file = open("labels_train_file", "rb")
LABELS_TRAIN_STORED = np.load(file)
len(LABELS_TRAIN_STORED)
```

```
1453
```

```
file = open("labels_test_file", "rb")
LABELS_TEST_STORED = np.load(file)
len(LABELS_TEST_STORED)
```

```
455
```

```
file = open("labels_val_file", "rb")
LABELS_VAL_STORED = np.load(file)
len(LABELS_VAL_STORED)
```

```
364
```

```
FEATURES_TRAIN_STORED.shape
```

```
(1453, 6, 72, 128, 3)
```

```
FEATURES_TEST_STORED.shape
```

```
(455, 6, 72, 128, 3)
```

```
FEATURES_VAL_STORED.shape
```

```
(364, 6, 72, 128, 3)
```

Treino

```
# Callbacks
early_stopping = EarlyStopping(monitor="val_loss", patience=15, mode = 'min', restore_best_weights = True)
reduce_lr = ReduceLROnPlateau(monitor="val_loss", patience=5)

# Training hyperparameters
epochs = 100
batch_size = 8

convlstm_model.compile(loss = 'binary_crossentropy' , optimizer = 'Adam', metrics = ["accuracy"])

convlstm_model_training_history = convlstm_model.fit(
    FEATURES_TRAIN_STORED,
    LABELS_TRAIN_STORED,
    batch_size=batch_size,
    epochs=epochs,
    validation_data=(FEATURES_VAL_STORED, LABELS_VAL_STORED),
    callbacks=[early_stopping, reduce_lr],
    shuffle = True
)
```

Plot Model's Loss & Accuracy Curves

```
def plot_metric(model_training_history, metric_name_1, metric_name_2, plot_name):

    metric_value_1 = model_training_history.history[metric_name_1]
    metric_value_2 = model_training_history.history[metric_name_2]

    epochs = range(len(metric_value_1))

    plt.plot(epochs, metric_value_1, 'blue', label = metric_name_1)
    plt.plot(epochs, metric_value_2, 'red', label = metric_name_2)

    plt.title(str(plot_name))

    plt.legend()
```

```
plot_metric(convlstm_model_training_history, 'loss', 'val_loss', 'Total Loss vs Total Validation Loss')
```

```
plot_metric(convlstm_model_training_history, 'accuracy', 'val_accuracy', 'Total Accuracy vs Total Validation Accuracy')
```

Matriz de confusão

```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import numpy as np
```

```
ypred = convlstm_model.predict(FEATURES_TEST_STORED, verbose=1)
print(ypred[0])

ypred = np.argmax(ypred, axis=1)
print(ypred)

test_labels = np.argmax(LABELS_TEST_STORED, axis=1)
print(classification_report(test_labels, ypred, target_names=CLASSES_LIST))

cm = confusion_matrix(test_labels, ypred)

## binary 01 (SIM_nao)
#TN (True negative) -- FP (false positive)
#FN (false negative) -- TP (True positive)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=CLASSES_LIST)

disp.plot(cmap=plt.cm.Blues)
plt.show()

del ypred
```

Guardar modelo

```
model_evaluation_loss, model_evaluation_accuracy = model_evaluation_history

date_time_format = '%Y_%m_%d__%H_%M_%S'
current_date_time_dt = dt.datetime.now()
current_date_time_string = dt.datetime.strftime(current_date_time_dt, date_time_format)

model_file_name = f'{MODELS_DIR}/ConvLSTM/convlstm_model_a___Date_Time_{current_date_time_string}___Loss_{model_evaluation_loss}_
print(f'{model_file_name}')

convlstm_model.save(model_file_name)
```

```
C:/Users/catar/Documentos/Tese/Projeto/Modelos/ConvLSTM/convlstm_model_a___Date_Time_2022_10_08__21_10_36___Loss_0.081628121435
64224___Accuracy_0.9758241772651672.h5
```

***Notebook* 4a**

Contrução do modelo

```
def create_convlstm_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura
    model.add(ConvLSTM2D(filters = 8, kernel_size = (3, 3), activation = 'tanh', data_format = "channels_last",
                         return_sequences=True, input_shape = (NR_OF_FRAMES,IMAGE_HEIGHT, IMAGE_WIDTH, 3)))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))


    model.add(ConvLSTM2D(filters = 16, kernel_size = (3, 3), activation = 'tanh', data_format = "channels_last",
                         return_sequences=True))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))


    model.add(ConvLSTM2D(filters = 16, kernel_size = (3, 3), activation = 'tanh', data_format = "channels_last",
                         return_sequences=True))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))


    #model.add(ConvLSTM2D(filters = 32 , kernel_size = (3, 3), activation = 'tanh', data_format = "channels_last",
                         #return_sequences=True))
    #model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))


    model.add(Flatten())


    model.add(Dense(len(CLASSES_LIST), activation = "sigmoid"))

    # Tabela
    model.summary()

    return model
```

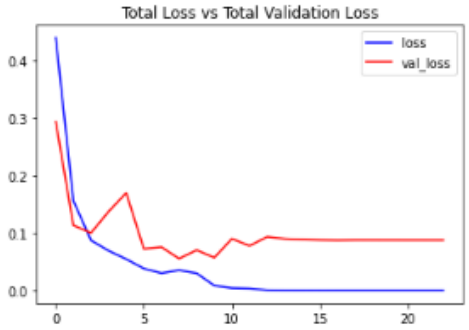
```
convlstm_model = create_convlstm_model()
```

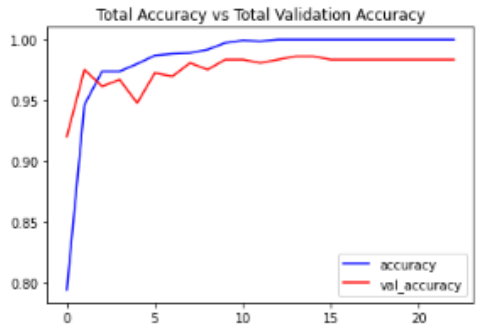
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_lstm2d (ConvLSTM2D)    (None, 6, 70, 126, 8)     3200

 max_pooling3d (MaxPooling3D  (None, 6, 35, 63, 8)     0
 )

 conv_lstm2d_1 (ConvLSTM2D)  (None, 6, 33, 61, 16)     13888

 max_pooling3d_1 (MaxPooling  (None, 6, 17, 31, 16)    0
 3D)

 conv_lstm2d_2 (ConvLSTM2D)  (None, 6, 15, 29, 16)     18496

 max_pooling3d_2 (MaxPooling  (None, 6, 8, 15, 16)     0
 3D)

 flatten (Flatten)           (None, 11520)             0

 dense (Dense)               (None, 2)                 23042

=================================================================
Total params: 58,626
Trainable params: 58,626
Non-trainable params: 0
_________________________________________________________________
```

Avaliação do modelo

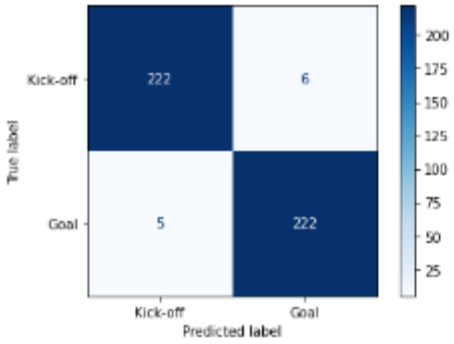
```
model_evaluation_history = convlstm_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED)
print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

```
15/15 [==============================] - 2s 97ms/step - loss: 0.0816 - accuracy: 0.9758
Accuracy: 97.58%
```





	precision	recall	f1-score	support
Kick-off	0.98	0.97	0.98	228
Goal	0.97	0.98	0.98	227
accuracy			0.98	455
macro avg	0.98	0.98	0.98	455
weighted avg	0.98	0.98	0.98	455



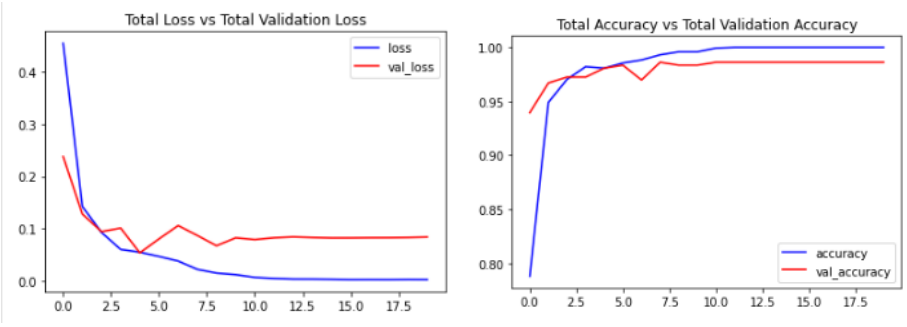
***Notebook* 4b**

```
def create_convlstm_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura

    model.add(ConvLSTM2D(filters = 8, kernel_size = (3, 3), activation = 'tanh',data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True, input_shape = (NR_OF_FRAMES,
                                                                                      IMAGE_HEIGHT, IMAGE_WIDTH, 3)))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))


    model.add(ConvLSTM2D(filters = 16, kernel_size = (3, 3), activation = 'tanh', data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True))

    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))


    model.add(ConvLSTM2D(filters = 16, kernel_size = (3, 3), activation = 'tanh', data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))


    model.add(Flatten())

    model.add(Dense(len(CLASSES_LIST), activation = "sigmoid"))

    model.summary()

    return model
```

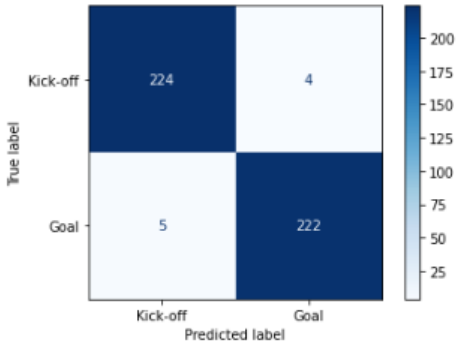
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_lstm2d (ConvLSTM2D)    (None, 6, 70, 126, 8)     3200

 max_pooling3d (MaxPooling3D  (None, 6, 35, 63, 8)     0
 )

 time_distributed (TimeDistr  (None, 6, 35, 63, 8)     0
 ibuted)

 conv_lstm2d_1 (ConvLSTM2D)  (None, 6, 33, 61, 16)     13888

 max_pooling3d_1 (MaxPooling  (None, 6, 17, 31, 16)    0
 3D)

 time_distributed_1 (TimeDis  (None, 6, 17, 31, 16)    0
 tributed)

 conv_lstm2d_2 (ConvLSTM2D)  (None, 6, 15, 29, 16)     18496

 max_pooling3d_2 (MaxPooling  (None, 6, 8, 15, 16)     0
 3D)

 time_distributed_2 (TimeDis  (None, 6, 8, 15, 16)     0
 tributed)

 flatten (Flatten)           (None, 11520)             0

 dense (Dense)               (None, 2)                 23042

=================================================================
Total params: 58,626
Trainable params: 58,626
Non-trainable params: 0
_________________________________________________________________
```

```
model_evaluation_history = convlstm_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED)
print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

```
15/15 [==============================] - 2s 97ms/step - loss: 0.0777 - accuracy: 0.9802
Accuracy: 98.02%
```



	precision	recall	f1-score	support
Kick-off	0.98	0.98	0.98	228
Goal	0.98	0.98	0.98	227
accuracy			0.98	455
macro avg	0.98	0.98	0.98	455
weighted avg	0.98	0.98	0.98	455



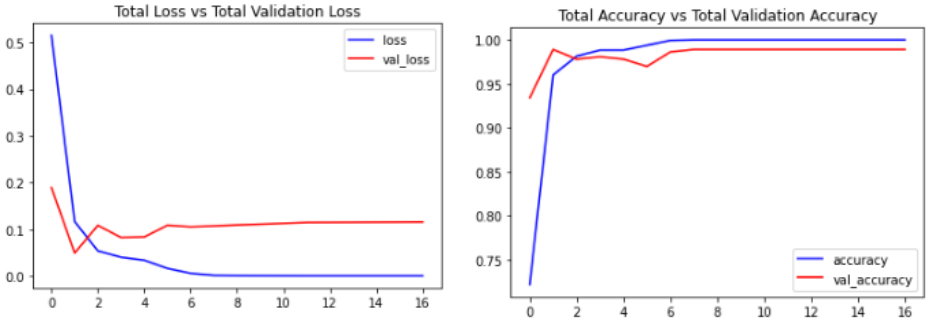
Notebook 4c

```
def create_convlstm_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura
    model.add(ConvLSTM2D(filters = 8, kernel_size = (5, 5), activation = 'tanh',data_format = "channels_last",
                         return_sequences=True, input_shape = (NR_OF_FRAMES,IMAGE_HEIGHT, IMAGE_WIDTH, 3)))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))


    model.add(ConvLSTM2D(filters = 16, kernel_size = (5, 5), activation = 'tanh', data_format = "channels_last",
                         return_sequences=True))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))


    model.add(ConvLSTM2D(filters = 16, kernel_size = (5, 5), activation = 'tanh', data_format = "channels_last",
                         return_sequences=True))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))

    model.add(Flatten())

    model.add(Dense(len(CLASSES_LIST), activation = "sigmoid"))

    model.summary()
    return model
```

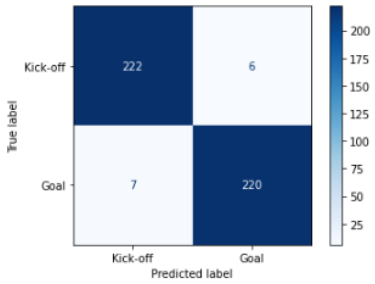
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_lstm2d (ConvLSTM2D)    (None, 6, 68, 124, 8)     8832

 max_pooling3d (MaxPooling3D  (None, 6, 34, 62, 8)     0
 )

 conv_lstm2d_1 (ConvLSTM2D)  (None, 6, 30, 58, 16)     38464

 max_pooling3d_1 (MaxPooling  (None, 6, 15, 29, 16)    0
 3D)

 conv_lstm2d_2 (ConvLSTM2D)  (None, 6, 11, 25, 16)     51264

 max_pooling3d_2 (MaxPooling  (None, 6, 6, 13, 16)     0
 3D)

 flatten (Flatten)           (None, 7488)              0

 dense (Dense)               (None, 2)                 14978

=================================================================
Total params: 113,538
Trainable params: 113,538
Non-trainable params: 0
_________________________________________________________________
```

```
model_evaluation_history = convlstm_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED)
print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

```
15/15 [==============================] - 2s 123ms/step - loss: 0.0922 - accuracy: 0.9714
Accuracy: 97.14%
```



	precision	recall	f1-score	support
Kick-off	0.97	0.97	0.97	228
Goal	0.97	0.97	0.97	227
accuracy			0.97	455
macro avg	0.97	0.97	0.97	455
weighted avg	0.97	0.97	0.97	455



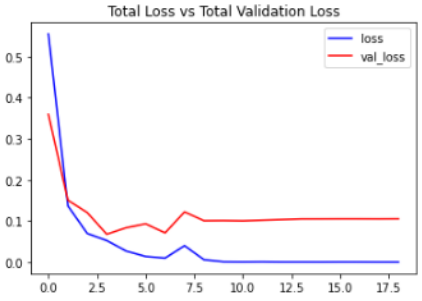
Notebook 4d

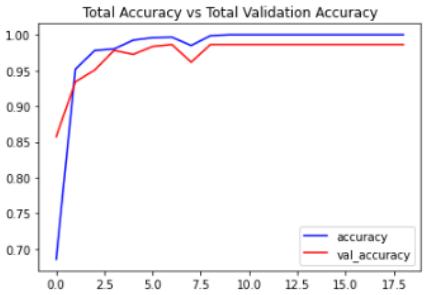
```
def create_convlstm_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura
    model.add(ConvLSTM2D(filters = 8, kernel_size = (5, 5), activation = 'tanh',data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True, input_shape = (NR_OF_FRAMES,
                                                                                      IMAGE_HEIGHT, IMAGE_WIDTH, 3)))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))


    model.add(ConvLSTM2D(filters = 16, kernel_size = (5, 5), activation = 'tanh', data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))


    model.add(ConvLSTM2D(filters = 16, kernel_size = (5, 5), activation = 'tanh', data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(Flatten())

    model.add(Dense(len(CLASSES_LIST), activation = "sigmoid"))

    # Summary.
    model.summary()

    return model
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_lstm2d (ConvLSTM2D)    (None, 6, 68, 124, 8)     8832

 max_pooling3d (MaxPooling3D  (None, 6, 34, 62, 8)     0
 )

 time_distributed (TimeDistr  (None, 6, 34, 62, 8)     0
 ibuted)

 conv_lstm2d_1 (ConvLSTM2D)  (None, 6, 30, 58, 16)     38464

 max_pooling3d_1 (MaxPooling  (None, 6, 15, 29, 16)    0
 3D)

 time_distributed_1 (TimeDis  (None, 6, 15, 29, 16)    0
 tributed)

 conv_lstm2d_2 (ConvLSTM2D)  (None, 6, 11, 25, 16)     51264

 max_pooling3d_2 (MaxPooling  (None, 6, 6, 13, 16)     0
 3D)

 time_distributed_2 (TimeDis  (None, 6, 6, 13, 16)     0
 tributed)

 flatten (Flatten)           (None, 7488)              0

 dense (Dense)               (None, 2)                 14978

=================================================================
Total params: 113,538
Trainable params: 113,538
Non-trainable params: 0
_________________________________________________________________
```

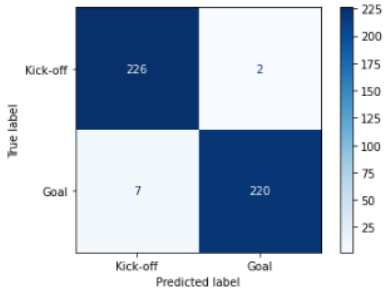
```
model_evaluation_history = convlstm_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED)
print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

```
15/15 [==============================] - 2s 124ms/step - loss: 0.0842 - accuracy: 0.9802
Accuracy: 98.02%
```





	precision	recall	f1-score	support
Kick-off	0.97	0.99	0.98	228
Goal	0.99	0.97	0.98	227
accuracy			0.98	455
macro avg	0.98	0.98	0.98	455
weighted avg	0.98	0.98	0.98	455



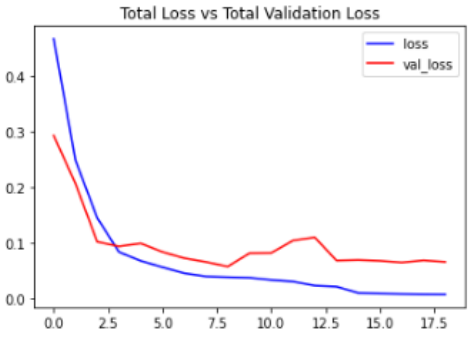
Notebook 4e

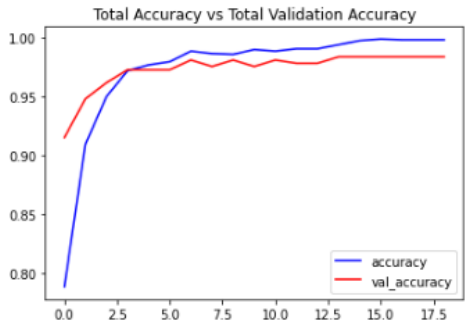
```
def create_convlstm_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura
    model.add(ConvLSTM2D(filters = 4, kernel_size = (3, 3), activation = 'tanh',
                         data_format = "channels_last", return_sequences=True,
                         input_shape = (NR_OF_FRAMES,IMAGE_HEIGHT, IMAGE_WIDTH, 3)))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))


    model.add(ConvLSTM2D(filters = 8, kernel_size = (3, 3), activation = 'tanh',
                         data_format = "channels_last",return_sequences=True))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))


    model.add(ConvLSTM2D(filters = 8, kernel_size = (3, 3), activation = 'tanh',
                         data_format = "channels_last", return_sequences=True))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))

    model.add(Flatten())

    model.add(Dense(len(CLASSES_LIST), activation = "sigmoid"))

    # Tabela
    model.summary()

    return model
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_lstm2d (ConvLSTM2D)    (None, 6, 70, 126, 4)     1024

 max_pooling3d (MaxPooling3D  (None, 6, 35, 63, 4)     0
 )

 conv_lstm2d_1 (ConvLSTM2D)  (None, 6, 33, 61, 8)      3488

 max_pooling3d_1 (MaxPooling  (None, 6, 17, 31, 8)     0
 3D)

 conv_lstm2d_2 (ConvLSTM2D)  (None, 6, 15, 29, 8)      4640

 max_pooling3d_2 (MaxPooling  (None, 6, 8, 15, 8)      0
 3D)

 flatten (Flatten)           (None, 5760)              0

 dense (Dense)               (None, 2)                 11522

=================================================================
Total params: 20,674
Trainable params: 20,674
Non-trainable params: 0
_________________________________________________________________
```

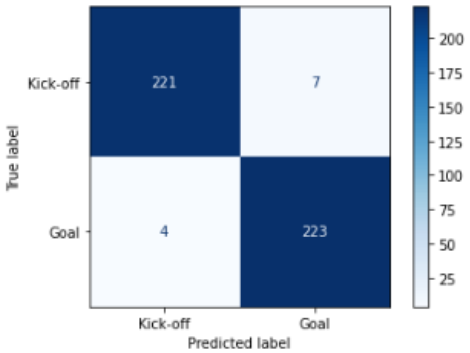
```
: model_evaluation_history = convlstm_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED)
  print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

```
15/15 [==============================] - 1s 68ms/step - loss: 0.0796 - accuracy: 0.9758
Accuracy: 97.58%
```





	precision	recall	f1-score	support
Kick-off	0.98	0.97	0.98	228
Goal	0.97	0.98	0.98	227
accuracy			0.98	455
macro avg	0.98	0.98	0.98	455
weighted avg	0.98	0.98	0.98	455



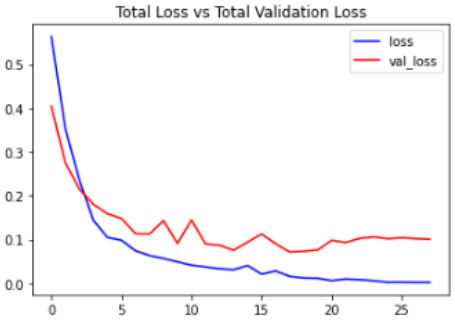
Notebook 4f

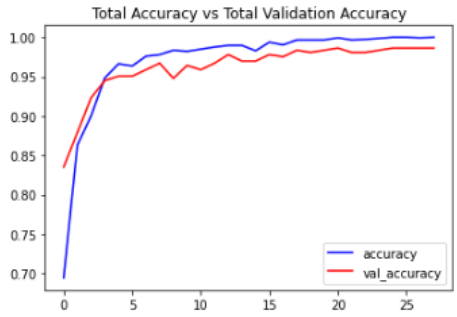
```
def create_convlstm_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura
    model.add(ConvLSTM2D(filters = 4, kernel_size = (3, 3), activation = 'tanh',data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True,
                         input_shape = (NR_OF_FRAMES, IMAGE_HEIGHT, IMAGE_WIDTH, 3)))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))


    model.add(ConvLSTM2D(filters = 8, kernel_size = (3, 3), activation = 'tanh', data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))


    model.add(ConvLSTM2D(filters = 8, kernel_size = (3, 3), activation = 'tanh', data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))


    model.add(Flatten())

    model.add(Dense(len(CLASSES_LIST), activation = "sigmoid"))

    model.summary()

    return model
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_lstm2d (ConvLSTM2D)    (None, 6, 70, 126, 4)     1024

 max_pooling3d (MaxPooling3D  (None, 6, 35, 63, 4)     0
 )

 time_distributed (TimeDistr  (None, 6, 35, 63, 4)     0
 ibuted)

 conv_lstm2d_1 (ConvLSTM2D)  (None, 6, 33, 61, 8)      3488

 max_pooling3d_1 (MaxPooling  (None, 6, 17, 31, 8)     0
 3D)

 time_distributed_1 (TimeDis  (None, 6, 17, 31, 8)     0
 tributed)

 conv_lstm2d_2 (ConvLSTM2D)  (None, 6, 15, 29, 8)      4640

 max_pooling3d_2 (MaxPooling  (None, 6, 8, 15, 8)      0
 3D)

 time_distributed_2 (TimeDis  (None, 6, 8, 15, 8)      0
 tributed)

 flatten (Flatten)           (None, 5760)              0

 dense (Dense)               (None, 2)                 11522

=================================================================
Total params: 20,674
Trainable params: 20,674
Non-trainable params: 0
```

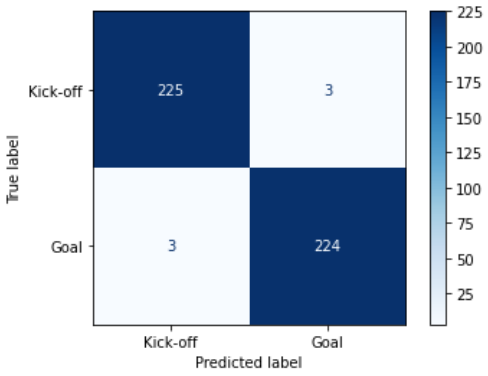
```
model_evaluation_history = convlstm_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED
print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

```
15/15 [==============================] - 1s 68ms/step - loss: 0.0815 - accuracy: 0.9868
Accuracy: 98.68%
```





	precision	recall	f1-score	support
Kick-off	0.99	0.99	0.99	228
Goal	0.99	0.99	0.99	227
accuracy			0.99	455
macro avg	0.99	0.99	0.99	455
weighted avg	0.99	0.99	0.99	455



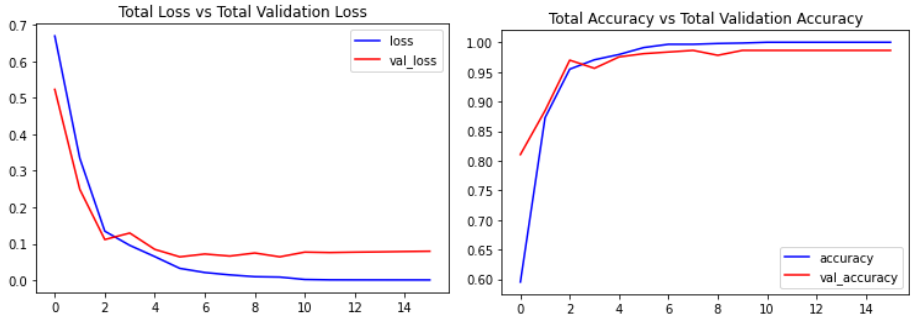
Notebook 4g

```
def create_convlstm_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura
    model.add(ConvLSTM2D(filters = 4, kernel_size = (5, 5), activation = 'tanh',data_format = "channels_last",
                         return_sequences=True, input_shape = (NR_OF_FRAMES,IMAGE_HEIGHT, IMAGE_WIDTH, 3)))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))


    model.add(ConvLSTM2D(filters = 8, kernel_size = (5, 5), activation = 'tanh', data_format = "channels_last",
                         return_sequences=True))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))


    model.add(ConvLSTM2D(filters = 8, kernel_size = (5, 5), activation = 'tanh', data_format = "channels_last",
                         return_sequences=True))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))

    model.add(Flatten())

    model.add(Dense(len(CLASSES_LIST), activation = "sigmoid"))

    model.summary()
    return model
```

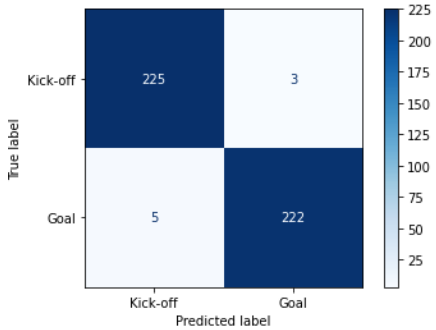
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_lstm2d (ConvLSTM2D)    (None, 6, 68, 124, 4)     2816

 max_pooling3d (MaxPooling3D  (None, 6, 34, 62, 4)     0
 )

 conv_lstm2d_1 (ConvLSTM2D)  (None, 6, 30, 58, 8)      9632

 max_pooling3d_1 (MaxPooling  (None, 6, 15, 29, 8)     0
 3D)

 conv_lstm2d_2 (ConvLSTM2D)  (None, 6, 11, 25, 8)      12832

 max_pooling3d_2 (MaxPooling  (None, 6, 6, 13, 8)      0
 3D)

 flatten (Flatten)           (None, 3744)              0

 dense (Dense)               (None, 2)                 7490

=================================================================
Total params: 32,770
Trainable params: 32,770
Non-trainable params: 0
_________________________________________________________________
```

```
model_evaluation_history = convlstm_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED)
print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

```
15/15 [==============================] - 1s 84ms/step - loss: 0.0768 - accuracy: 0.9824
Accuracy: 98.24%
```



	precision	recall	f1-score	support
Kick-off	0.98	0.99	0.98	228
Goal	0.99	0.98	0.98	227
accuracy			0.98	455
macro avg	0.98	0.98	0.98	455
weighted avg	0.98	0.98	0.98	455



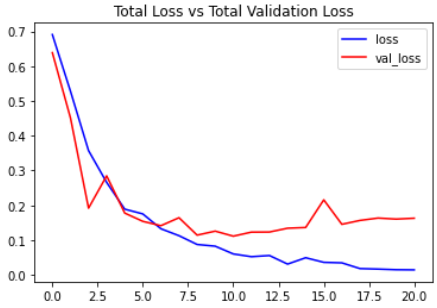
Notebook 4h

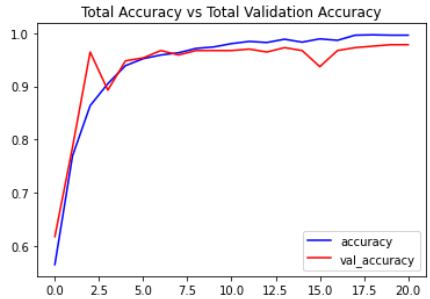
```
def create_convlstm_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura
    model.add(ConvLSTM2D(filters = 4, kernel_size = (5, 5), activation = 'relu',data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True, input_shape = (NR_OF_FRAMES,
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))


    model.add(ConvLSTM2D(filters = 8, kernel_size = (5, 5), activation = 'relu', data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))


    model.add(ConvLSTM2D(filters = 8, kernel_size = (5, 5), activation = 'relu', data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(Flatten())

    model.add(Dense(len(CLASSES_LIST), activation = "sigmoid"))

    # Summary.
    model.summary()

    return model
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_lstm2d (ConvLSTM2D)    (None, 6, 68, 124, 4)     2816

 max_pooling3d (MaxPooling3D  (None, 6, 34, 62, 4)     0
 )

 time_distributed (TimeDistr  (None, 6, 34, 62, 4)     0
 ibuted)

 conv_lstm2d_1 (ConvLSTM2D)  (None, 6, 30, 58, 8)      9632

 max_pooling3d_1 (MaxPooling  (None, 6, 15, 29, 8)     0
 3D)

 time_distributed_1 (TimeDis  (None, 6, 15, 29, 8)     0
 tributed)

 conv_lstm2d_2 (ConvLSTM2D)  (None, 6, 11, 25, 8)      12832

 max_pooling3d_2 (MaxPooling  (None, 6, 6, 13, 8)      0
 3D)

 time_distributed_2 (TimeDis  (None, 6, 6, 13, 8)      0
 tributed)

 flatten (Flatten)           (None, 3744)              0

 dense (Dense)               (None, 2)                 7490

=================================================================
Total params: 32,770
Trainable params: 32,770
Non-trainable params: 0
_________________________________________________________________
```

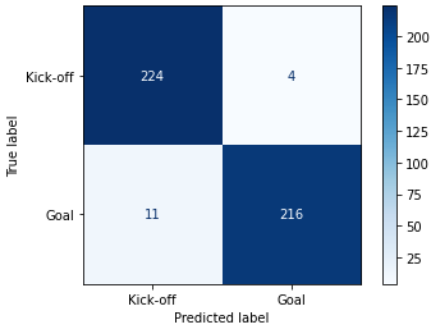
```
model_evaluation_history = convlstm_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED)
print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

```
15/15 [==============================] - 1s 84ms/step - loss: 0.1163 - accuracy: 0.9670
Accuracy: 96.70%
```





	precision	recall	f1-score	support
Kick-off	0.95	0.98	0.97	228
Goal	0.98	0.95	0.97	227
accuracy			0.97	455
macro avg	0.97	0.97	0.97	455
weighted avg	0.97	0.97	0.97	455



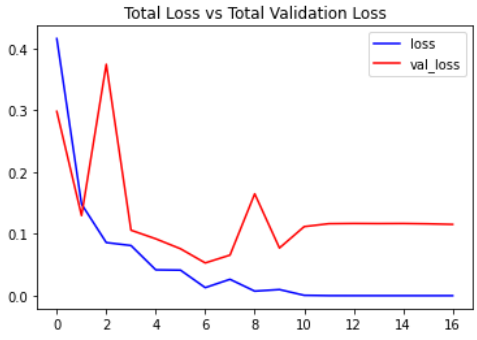
Notebook 4i

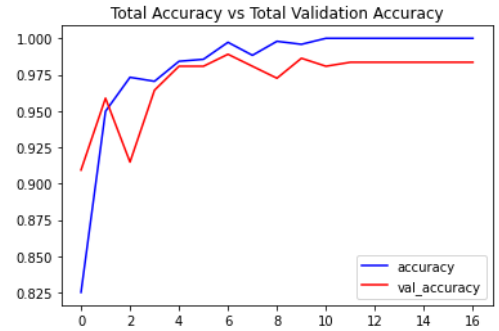
```
def create_convlstm_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura
    model.add(ConvLSTM2D(filters = 16, kernel_size = (3, 3), activation = 'tanh',data_format = "channels_last",
                         return_sequences=True, input_shape = (NR_OF_FRAMES,IMAGE_HEIGHT, IMAGE_WIDTH, 3)))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))


    model.add(ConvLSTM2D(filters = 32, kernel_size = (3, 3), activation = 'tanh', data_format = "channels_last",
                         return_sequences=True))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))


    model.add(ConvLSTM2D(filters = 32, kernel_size = (3, 3), activation = 'tanh', data_format = "channels_last",
                         return_sequences=True))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))


    model.add(Flatten())


    model.add(Dense(len(CLASSES_LIST), activation = "sigmoid"))

    # Tabela
    model.summary()

    return model
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_lstm2d (ConvLSTM2D)    (None, 6, 70, 126, 16)    11008

 max_pooling3d (MaxPooling3D  (None, 6, 35, 63, 16)    0
 )

 conv_lstm2d_1 (ConvLSTM2D)  (None, 6, 33, 61, 32)     55424

 max_pooling3d_1 (MaxPooling  (None, 6, 17, 31, 32)    0
 3D)

 conv_lstm2d_2 (ConvLSTM2D)  (None, 6, 15, 29, 32)     73856

 max_pooling3d_2 (MaxPooling  (None, 6, 8, 15, 32)     0
 3D)

 time_distributed (TimeDistr  (None, 6, 8, 15, 32)     0
 ibuted)

 flatten (Flatten)           (None, 23040)             0

 dense (Dense)               (None, 2)                 46082

=================================================================
Total params: 186,370
Trainable params: 186,370
Non-trainable params: 0
_________________________________________________________________
```

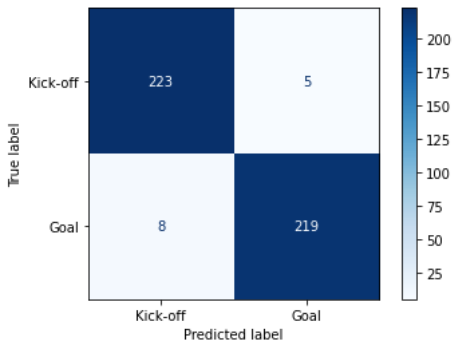
```
model_evaluation_history = convlstm_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED)
print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

```
15/15 [==============================] - 3s 156ms/step - loss: 0.1519 - accuracy: 0.9714
Accuracy: 97.14%
```





	precision	recall	f1-score	support
Kick-off	0.97	0.98	0.97	228
Goal	0.98	0.96	0.97	227
accuracy			0.97	455
macro avg	0.97	0.97	0.97	455
weighted avg	0.97	0.97	0.97	455



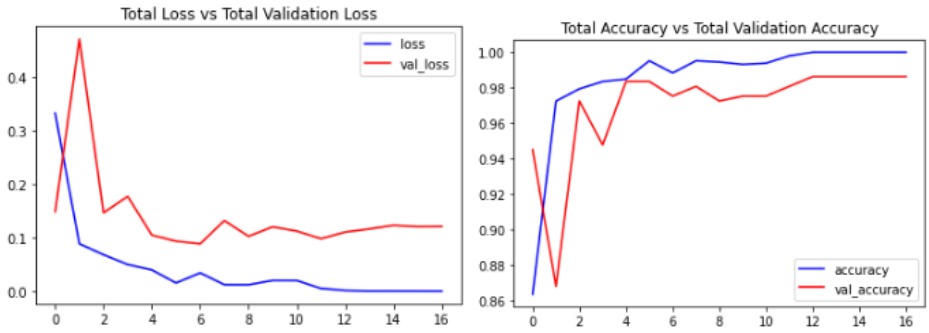
Notebook 4j

```
def create_convlstm_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura
    model.add(ConvLSTM2D(filters = 16, kernel_size = (3, 3), activation = 'tanh',data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True, input_shape = (NR_OF_FRAMES,
                                                                                      IMAGE_HEIGHT, IMAGE_WIDTH, 3))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))


    model.add(ConvLSTM2D(filters = 32, kernel_size = (3, 3), activation = 'tanh', data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True))

    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))


    model.add(ConvLSTM2D(filters = 32, kernel_size = (3, 3), activation = 'tanh', data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True))

    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(Flatten())

    model.add(Dense(len(CLASSES_LIST), activation = "sigmoid"))

    model.summary()

    return model
```

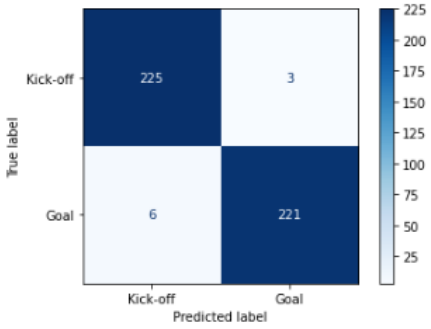
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_lstm2d (ConvLSTM2D)    (None, 6, 70, 126, 16)    11008

 max_pooling3d (MaxPooling3D  (None, 6, 35, 63, 16)    0
 )

 time_distributed (TimeDistr  (None, 6, 35, 63, 16)    0
 ibuted)

 conv_lstm2d_1 (ConvLSTM2D)  (None, 6, 33, 61, 32)     55424

 max_pooling3d_1 (MaxPooling  (None, 6, 17, 31, 32)    0
 3D)

 time_distributed_1 (TimeDis  (None, 6, 17, 31, 32)    0
 tributed)

 conv_lstm2d_2 (ConvLSTM2D)  (None, 6, 15, 29, 32)     73856

 max_pooling3d_2 (MaxPooling  (None, 6, 8, 15, 32)     0
 3D)

 time_distributed_2 (TimeDis  (None, 6, 8, 15, 32)     0
 tributed)

 flatten (Flatten)           (None, 23040)             0

 dense (Dense)               (None, 2)                 46082

=================================================================
Total params: 186,370
Trainable params: 186,370
Non-trainable params: 0
_________________________________________________________________
```

```
model_evaluation_history = convlstm_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED)
print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

```
15/15 [==============================] - 3s 153ms/step - loss: 0.1315 - accuracy: 0.9802
Accuracy: 98.02%
```



	precision	recall	f1-score	support
Kick-off	0.97	0.99	0.98	228
Goal	0.99	0.97	0.98	227
accuracy			0.98	455
macro avg	0.98	0.98	0.98	455
weighted avg	0.98	0.98	0.98	455



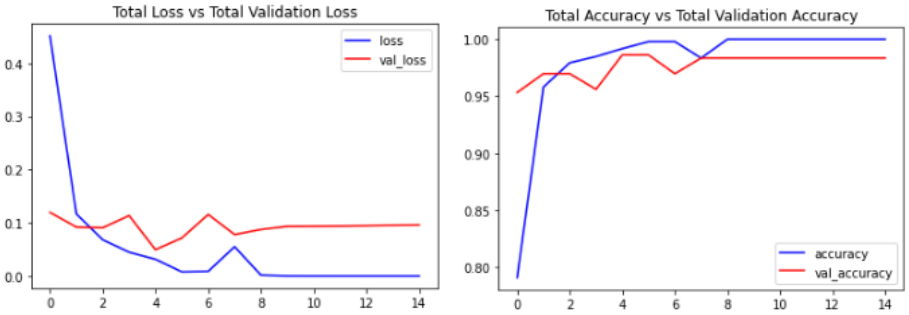
***Notebook* 4k**

```
def create_convlstm_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura
    model.add(ConvLSTM2D(filters = 16, kernel_size = (5, 5), activation = 'tanh',data_format = "channels_last",
                         return_sequences=True, input_shape = (NR_OF_FRAMES,IMAGE_HEIGHT, IMAGE_WIDTH, 3)))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))

    model.add(ConvLSTM2D(filters = 32, kernel_size = (5, 5), activation = 'tanh', data_format = "channels_last",
                         return_sequences=True))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))

    model.add(ConvLSTM2D(filters = 32, kernel_size = (5, 5), activation = 'tanh', data_format = "channels_last",
                         return_sequences=True))
    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))

    model.add(Flatten())

    model.add(Dense(len(CLASSES_LIST), activation = "sigmoid"))

    model.summary()
    return model
```

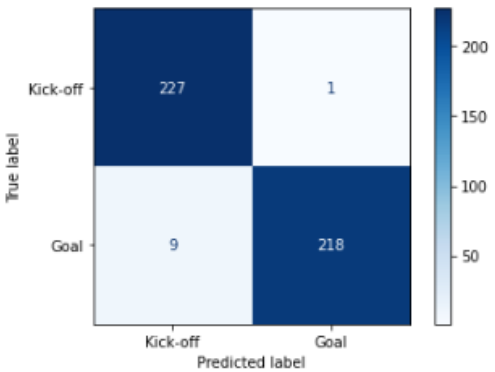
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_lstm2d (ConvLSTM2D)    (None, 6, 68, 124, 16)    30464

 max_pooling3d (MaxPooling3D  (None, 6, 34, 62, 16)    0
 )

 conv_lstm2d_1 (ConvLSTM2D)  (None, 6, 30, 58, 32)     153728

 max_pooling3d_1 (MaxPooling  (None, 6, 15, 29, 32)    0
 3D)

 conv_lstm2d_2 (ConvLSTM2D)  (None, 6, 11, 25, 32)     204928

 max_pooling3d_2 (MaxPooling  (None, 6, 6, 13, 32)     0
 3D)

 flatten (Flatten)           (None, 14976)             0

 dense (Dense)               (None, 2)                 29954

=================================================================
Total params: 419,074
Trainable params: 419,074
Non-trainable params: 0
_________________________________________________________________
```

```
model_evaluation_history = convlstm_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED)
print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

```
15/15 [==============================] - 4s 198ms/step - loss: 0.1188 - accuracy: 0.9780
Accuracy: 97.80%
```



	precision	recall	f1-score	support
Kick-off	0.96	1.00	0.98	228
Goal	1.00	0.96	0.98	227
accuracy			0.98	455
macro avg	0.98	0.98	0.98	455
weighted avg	0.98	0.98	0.98	455



***Notebook* 4l**

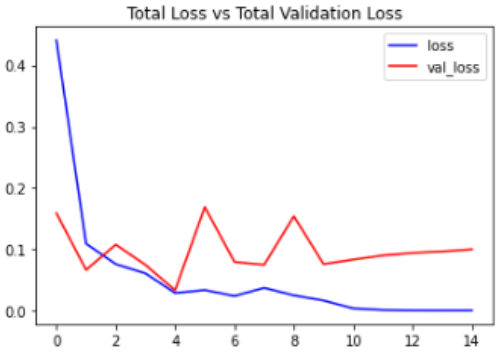
```
def create_convlstm_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura
    model.add(ConvLSTM2D(filters = 16, kernel_size = (5, 5), activation = 'tanh',data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True, input_shape = (NR_OF_FRAMES,
                                                                                      IMAGE_HEIGHT, IMAGE_WIDTH, 3)))

    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))


    model.add(ConvLSTM2D(filters = 32, kernel_size = (5, 5), activation = 'tanh', data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True))

    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))


    model.add(ConvLSTM2D(filters = 32, kernel_size = (5, 5), activation = 'tanh', data_format = "channels_last",
                         recurrent_dropout=0.2, return_sequences=True))

    model.add(MaxPooling3D(pool_size=(1, 2, 2), padding='same', data_format='channels_last'))
    model.add(TimeDistributed(Dropout(0.2)))


    model.add(Flatten())

    model.add(Dense(len(CLASSES_LIST), activation = "sigmoid"))

    # Summary.
    model.summary()

    return model
```

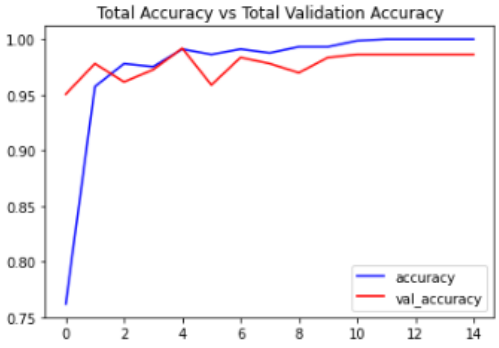
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_lstm2d (ConvLSTM2D)    (None, 6, 68, 124, 16)    30464

 max_pooling3d (MaxPooling3D  (None, 6, 34, 62, 16)    0
 )

 time_distributed (TimeDistr  (None, 6, 34, 62, 16)    0
 ibuted)

 conv_lstm2d_1 (ConvLSTM2D)  (None, 6, 30, 58, 32)     153728

 max_pooling3d_1 (MaxPooling  (None, 6, 15, 29, 32)    0
 3D)

 time_distributed_1 (TimeDis  (None, 6, 15, 29, 32)    0
 tributed)

 conv_lstm2d_2 (ConvLSTM2D)  (None, 6, 11, 25, 32)     204928

 max_pooling3d_2 (MaxPooling  (None, 6, 6, 13, 32)     0
 3D)

 time_distributed_2 (TimeDis  (None, 6, 6, 13, 32)     0
 tributed)

 flatten (Flatten)           (None, 14976)             0

 dense (Dense)               (None, 2)                 29954

=================================================================
Total params: 419,074
Trainable params: 419,074
Non-trainable params: 0
```

Avaliação do modelo

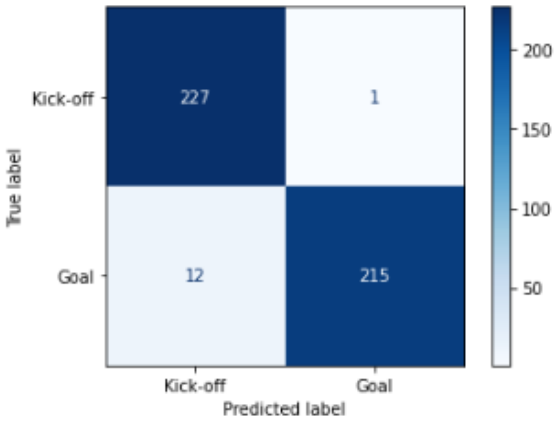
```
model_evaluation_history = convlstm_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED)
print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

```
15/15 [==============================] - 4s 199ms/step - loss: 0.1438 - accuracy: 0.9714
Accuracy: 97.14%
```





	precision	recall	f1-score	support
Kick-off	0.95	1.00	0.97	228
Goal	1.00	0.95	0.97	227
accuracy			0.97	455
macro avg	0.97	0.97	0.97	455
weighted avg	0.97	0.97	0.97	455



Anexo E

Notebooks* 5 (a,b,c,d,e,f,g,h,i) - *LRCN

Constantes

```
# Diretório do dataset
%store -r local_dataset_dir
DATASET_DIR = local_dataset_dir

# Diretório modelos
%store -r local_models_dir
MODELS_DIR = local_models_dir

# Frames
%store -r frame_seconds_diff
FRAME_SECONDS_DIFF = frame_seconds_diff
NR_OF_FRAMES = len(FRAME_SECONDS_DIFF)

# Altura de cada frame
%store -r image_height
IMAGE_HEIGHT = image_height

# Largura de cada frame
%store -r image_width
IMAGE_WIDTH = image_width

# Classes para treino
%store -r classes_list
CLASSES_LIST = classes_list
```

Imports

```
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import *

from tensorflow.keras.utils import plot_model

from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ReduceLROnPlateau

import datetime as dt
import matplotlib.pyplot as plt

import numpy as np
```

```
file = open("features_train_file", "rb")
FEATURES_TRAIN_STORED = np.load(file)
len(FEATURES_TRAIN_STORED)
```

```
1453
```

```
file = open("features_train_file", "rb")
FEATURES_TRAIN_STORED = np.load(file)
len(FEATURES_TRAIN_STORED)
```

```
1453
```

```
file = open("features_test_file", "rb")
FEATURES_TEST_STORED = np.load(file)
len(FEATURES_TEST_STORED)
```

```
455
```

```
file = open("features_test_file", "rb")
FEATURES_TEST_STORED = np.load(file)
len(FEATURES_TEST_STORED)
```

```
455
```

```
file = open("features_val_file", "rb")
FEATURES_VAL_STORED = np.load(file)
len(FEATURES_VAL_STORED)
```

```
364
```

```
file = open("features_val_file", "rb")
FEATURES_VAL_STORED = np.load(file)
len(FEATURES_VAL_STORED)
```

```
364
```

```
file = open("labels_train_file", "rb")
LABELS_TRAIN_STORED = np.load(file)
len(LABELS_TRAIN_STORED)
```

```
1453
```

```
file = open("labels_train_file", "rb")
LABELS_TRAIN_STORED = np.load(file)
len(LABELS_TRAIN_STORED)
```

```
1453
```

Treino

```
# Callback
early_stopping = EarlyStopping(monitor="val_loss", patience=10, mode = 'min', restore_best_weights = True)
reduce_lr = ReduceLROnPlateau(monitor="val_loss", patience=5)

# Training hyperparameters.
epochs = 100
batch_size = 32

LRCN_model.compile(loss = 'binary_crossentropy' , optimizer = 'Adam', metrics = ["accuracy"])

LRCN_model_training_history = LRCN_model.fit(
    x = FEATURES_TRAIN_STORED,
    y = LABELS_TRAIN_STORED,
    batch_size=batch_size,
    epochs=epochs,
    validation_data=(FEATURES_VAL_STORED, LABELS_VAL_STORED),
    callbacks=[early_stopping, reduce_lr],
    shuffle =  True # False
)
```

***Notebook* 5a**

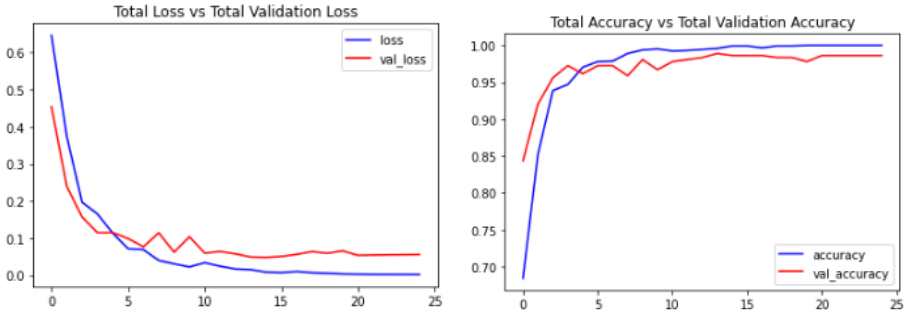
```
def create_LRCN_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura
    model.add(TimeDistributed(Conv2D(8, (3, 3), padding='same',activation = 'relu'),
                              input_shape = (NR_OF_FRAMES, IMAGE_HEIGHT, IMAGE_WIDTH, 3)))

    model.add(TimeDistributed(MaxPooling2D((4, 4))))

    model.add(TimeDistributed(Conv2D(16, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((4, 4))))

    model.add(TimeDistributed(Conv2D(32, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))

    model.add(TimeDistributed(Conv2D(32, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))

    model.add(TimeDistributed(Flatten()))

    model.add(LSTM(16))

    model.add(Dense(len(CLASSES_LIST), activation = 'sigmoid'))

    # Tabela
    model.summary()

    return model
```

```
LRCN_model = create_LRCN_model()
```

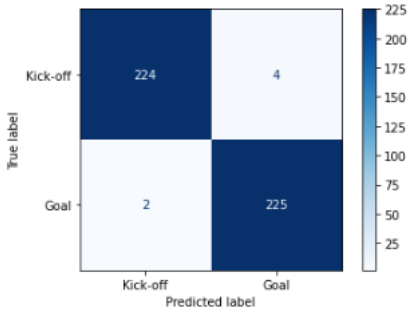
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 time_distributed (TimeDistr  (None, 6, 72, 128, 8)    224
 ibuted)

 time_distributed_1 (TimeDis  (None, 6, 18, 32, 8)     0
 tributed)

 time_distributed_2 (TimeDis  (None, 6, 18, 32, 16)    1168
 tributed)

 time_distributed_3 (TimeDis  (None, 6, 4, 8, 16)      0
 tributed)

 time_distributed_4 (TimeDis  (None, 6, 4, 8, 32)      4640
 tributed)

 time_distributed_5 (TimeDis  (None, 6, 2, 4, 32)      0
 tributed)

 time_distributed_6 (TimeDis  (None, 6, 2, 4, 32)      9248
 tributed)

 time_distributed_7 (TimeDis  (None, 6, 1, 2, 32)      0
 tributed)

 time_distributed_8 (TimeDis  (None, 6, 64)            0
 tributed)

 lstm (LSTM)                 (None, 16)                5184

 dense (Dense)               (None, 2)                 34

=================================================================
Total params: 20,498
Trainable params: 20,498
Non-trainable params: 0
_________________________________________________________________
```

```
model_evaluation_history = LRCN_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED)
print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

```
15/15 [==============================] - 0s 22ms/step - loss: 0.0432 - accuracy: 0.9868
Accuracy: 98.68%
```



	precision	recall	f1-score	support
Kick-off	0.99	0.98	0.99	228
Goal	0.98	0.99	0.99	227
accuracy			0.99	455
macro avg	0.99	0.99	0.99	455
weighted avg	0.99	0.99	0.99	455



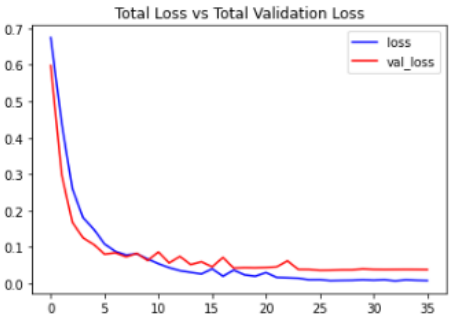
***Notebook* 5b**

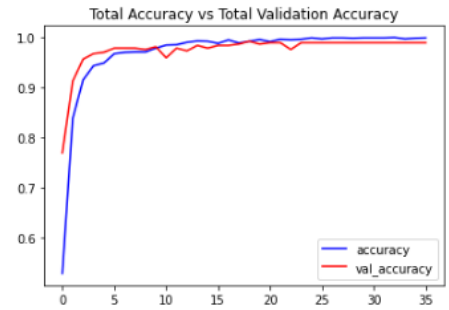
```
def create_LRCN_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura
    model.add(TimeDistributed(Conv2D(8, (3, 3), padding='same',activation = 'relu'),
                              input_shape = (NR_OF_FRAMES, IMAGE_HEIGHT, IMAGE_WIDTH, 3)))

    model.add(TimeDistributed(MaxPooling2D((4, 4))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Conv2D(16, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((4, 4))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Conv2D(32, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Conv2D(32, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Flatten()))

    model.add(LSTM(16))

    model.add(Dense(len(CLASSES_LIST), activation = 'sigmoid'))

    # Tabela
    model.summary()

    return model
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 time_distributed (TimeDistr  (None, 6, 72, 128, 8)    224
 ibuted)

 time_distributed_1 (TimeDis  (None, 6, 18, 32, 8)     0
 tributed)

 time_distributed_2 (TimeDis  (None, 6, 18, 32, 8)     0
 tributed)

 time_distributed_3 (TimeDis  (None, 6, 18, 32, 16)    1168
 tributed)

 time_distributed_4 (TimeDis  (None, 6, 4, 8, 16)      0
 tributed)

 time_distributed_5 (TimeDis  (None, 6, 4, 8, 16)      0
 tributed)

 time_distributed_6 (TimeDis  (None, 6, 4, 8, 32)      4640
 tributed)

 time_distributed_7 (TimeDis  (None, 6, 2, 4, 32)      0
 tributed)

 time_distributed_8 (TimeDis  (None, 6, 2, 4, 32)      0
 tributed)

 time_distributed_9 (TimeDis  (None, 6, 2, 4, 32)      9248
 tributed)

 time_distributed_10 (TimeDi  (None, 6, 1, 2, 32)      0
 stributed)

 time_distributed_11 (TimeDi  (None, 6, 1, 2, 32)      0
 stributed)

 time_distributed_12 (TimeDi  (None, 6, 64)            0
 stributed)

 lstm (LSTM)                 (None, 16)                5184

 dense (Dense)               (None, 2)                 34

=================================================================
Total params: 20,498
Trainable params: 20,498
Non-trainable params: 0
_________________________________________________________________
```

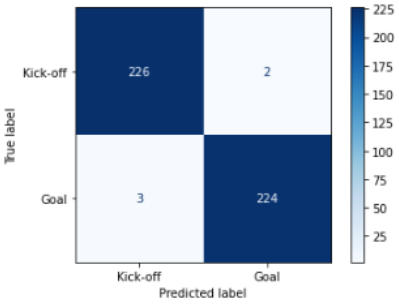
```
model_evaluation_history = LRCN_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED)
print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

```
15/15 [==============================] - 0s 19ms/step - loss: 0.0489 - accuracy: 0.9890
Accuracy: 98.90%
```





	precision	recall	f1-score	support
Kick-off	0.99	0.99	0.99	228
Goal	0.99	0.99	0.99	227
accuracy			0.99	455
macro avg	0.99	0.99	0.99	455
weighted avg	0.99	0.99	0.99	455



***Notebook* 5c**

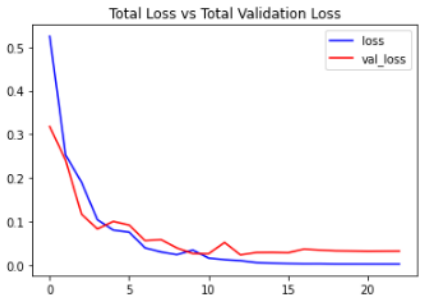
```
def create_LRCN_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura
    model.add(TimeDistributed(Conv2D(8, (5,5), padding='same',activation = 'relu'),
                              input_shape = (NR_OF_FRAMES, IMAGE_HEIGHT, IMAGE_WIDTH, 3)))

    model.add(TimeDistributed(MaxPooling2D((4, 4))))

    model.add(TimeDistributed(Conv2D(16, (5,5), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((4, 4))))

    model.add(TimeDistributed(Conv2D(32, (5,5), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))

    model.add(TimeDistributed(Conv2D(32, (5,5), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))

    model.add(TimeDistributed(Flatten()))

    model.add(LSTM(16))

    model.add(Dense(len(CLASSES_LIST), activation = 'sigmoid'))

    # Tabela
    model.summary()

    return model
```

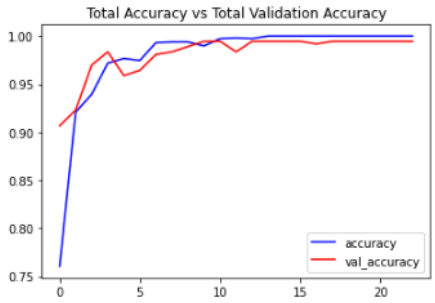
```
LRCN_model = create_LRCN_model()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 time_distributed (TimeDistr  (None, 6, 72, 128, 8)    608
 ibuted)

 time_distributed_1 (TimeDis  (None, 6, 18, 32, 8)     0
 tributed)

 time_distributed_2 (TimeDis  (None, 6, 18, 32, 16)    3216
 tributed)

 time_distributed_3 (TimeDis  (None, 6, 4, 8, 16)      0
 tributed)

 time_distributed_4 (TimeDis  (None, 6, 4, 8, 32)      12832
 tributed)

 time_distributed_5 (TimeDis  (None, 6, 2, 4, 32)      0
 tributed)

 time_distributed_6 (TimeDis  (None, 6, 2, 4, 32)      25632
 tributed)

 time_distributed_7 (TimeDis  (None, 6, 1, 2, 32)      0
 tributed)

 time_distributed_8 (TimeDis  (None, 6, 64)            0
 tributed)

 lstm (LSTM)                 (None, 16)                5184

 dense (Dense)               (None, 2)                 34

=================================================================
Total params: 47,506
Trainable params: 47,506
Non-trainable params: 0
_________________________________________________________________
```

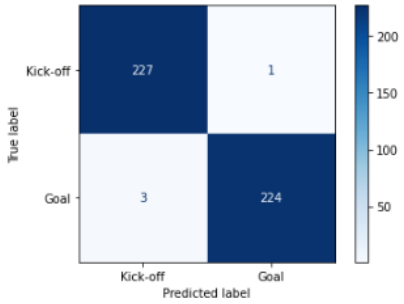
```
model_evaluation_history = LRCN_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED)
print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

```
15/15 [==============================] - 0s 21ms/step - loss: 0.0375 - accuracy: 0.9912
Accuracy: 99.12%
```





	precision	recall	f1-score	support
Kick-off	0.99	1.00	0.99	228
Goal	1.00	0.99	0.99	227
accuracy			0.99	455
macro avg	0.99	0.99	0.99	455
weighted avg	0.99	0.99	0.99	455



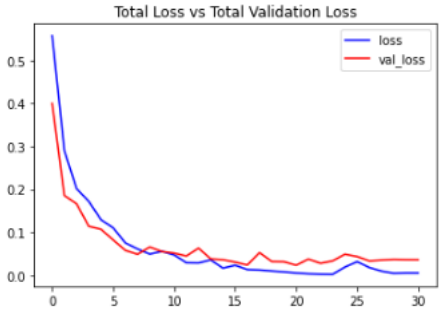
Notebook 5d

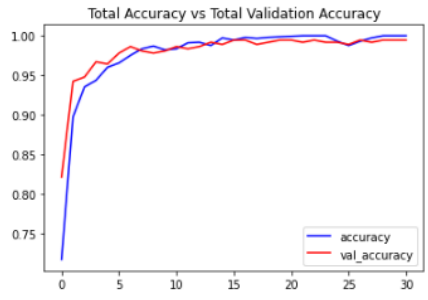
```
def create_LRCN_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura
    model.add(TimeDistributed(Conv2D(8, (5,5), padding='same',activation = 'relu'),
                              input_shape = (NR_OF_FRAMES, IMAGE_HEIGHT, IMAGE_WIDTH, 3)))

    model.add(TimeDistributed(MaxPooling2D((4, 4))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Conv2D(16, (5,5), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((4, 4))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Conv2D(32, (5,5), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Conv2D(32, (5,5), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Flatten()))

    model.add(LSTM(16))

    model.add(Dense(len(CLASSES_LIST), activation = 'sigmoid'))

    # Tabela
    model.summary()

    return model
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 time_distributed (TimeDistr  (None, 6, 72, 128, 8)    608
 ibuted)

 time_distributed_1 (TimeDis  (None, 6, 18, 32, 8)     0
 tributed)

 time_distributed_2 (TimeDis  (None, 6, 18, 32, 8)     0
 tributed)

 time_distributed_3 (TimeDis  (None, 6, 18, 32, 16)    3216
 tributed)

 time_distributed_4 (TimeDis  (None, 6, 4, 8, 16)      0
 tributed)

 time_distributed_5 (TimeDis  (None, 6, 4, 8, 16)      0
 tributed)

 time_distributed_6 (TimeDis  (None, 6, 4, 8, 32)      12832
 tributed)

 time_distributed_7 (TimeDis  (None, 6, 2, 4, 32)      0
 tributed)

 time_distributed_8 (TimeDis  (None, 6, 2, 4, 32)      0
 tributed)

 time_distributed_9 (TimeDis  (None, 6, 2, 4, 32)      25632
 tributed)

 time_distributed_10 (TimeDi  (None, 6, 1, 2, 32)      0
 stributed)

 time_distributed_11 (TimeDi  (None, 6, 1, 2, 32)      0
 stributed)

 time_distributed_12 (TimeDi  (None, 6, 64)            0
 stributed)

 lstm (LSTM)                 (None, 16)                5184

 dense (Dense)               (None, 2)                 34

=================================================================
Total params: 47,506
Trainable params: 47,506
Non-trainable params: 0
_________________________________________________________________
```

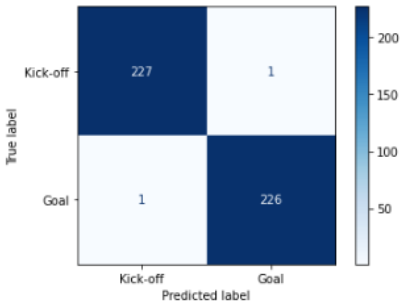
```
model_evaluation_history = LRCN_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED)
print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

```
15/15 [==============================] - 0s 21ms/step - loss: 0.0295 - accuracy: 0.9956
Accuracy: 99.56%
```





	precision	recall	f1-score	support
Kick-off	1.00	1.00	1.00	228
Goal	1.00	1.00	1.00	227
accuracy			1.00	455
macro avg	1.00	1.00	1.00	455
weighted avg	1.00	1.00	1.00	455



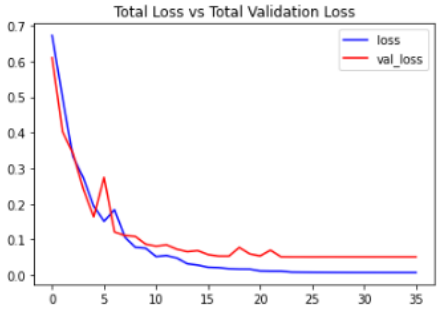
***Notebook* 5e**

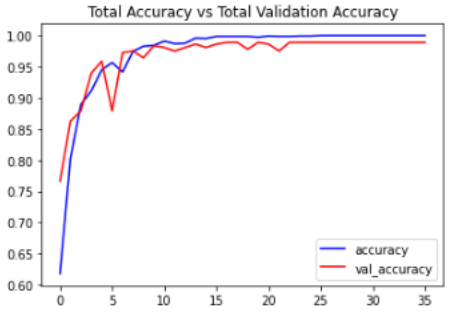
```
def create_LRCN_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura
    model.add(TimeDistributed(Conv2D(4, (3, 3), padding='same',activation = 'relu'),
                              input_shape = (NR_OF_FRAMES, IMAGE_HEIGHT, IMAGE_WIDTH, 3)))

    model.add(TimeDistributed(MaxPooling2D((4, 4))))

    model.add(TimeDistributed(Conv2D(8, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((4, 4))))

    model.add(TimeDistributed(Conv2D(16, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))

    model.add(TimeDistributed(Conv2D(16, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))

    model.add(TimeDistributed(Flatten()))

    model.add(LSTM(8))

    model.add(Dense(len(CLASSES_LIST), activation = 'sigmoid'))

    # Tabela
    model.summary()

    return model
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 time_distributed (TimeDistr  (None, 6, 72, 128, 4)    112
 ibuted)

 time_distributed_1 (TimeDis  (None, 6, 18, 32, 4)     0
 tributed)

 time_distributed_2 (TimeDis  (None, 6, 18, 32, 8)     296
 tributed)

 time_distributed_3 (TimeDis  (None, 6, 4, 8, 8)       0
 tributed)

 time_distributed_4 (TimeDis  (None, 6, 4, 8, 16)      1168
 tributed)

 time_distributed_5 (TimeDis  (None, 6, 2, 4, 16)      0
 tributed)

 time_distributed_6 (TimeDis  (None, 6, 2, 4, 16)      2320
 tributed)

 time_distributed_7 (TimeDis  (None, 6, 1, 2, 16)      0
 tributed)

 time_distributed_8 (TimeDis  (None, 6, 32)            0
 tributed)

 lstm (LSTM)                 (None, 8)                 1312

 dense (Dense)               (None, 2)                 18

=================================================================
Total params: 5,226
Trainable params: 5,226
Non-trainable params: 0
_________________________________________________________________
```

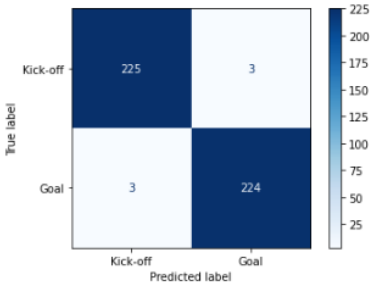
```
model_evaluation_history = LRCN_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED)
print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

```
15/15 [==============================] - 0s 17ms/step - loss: 0.0583 - accuracy: 0.9868
Accuracy: 98.68%
```





	precision	recall	f1-score	support
Kick-off	0.99	0.99	0.99	228
Goal	0.99	0.99	0.99	227
accuracy			0.99	455
macro avg	0.99	0.99	0.99	455
weighted avg	0.99	0.99	0.99	455



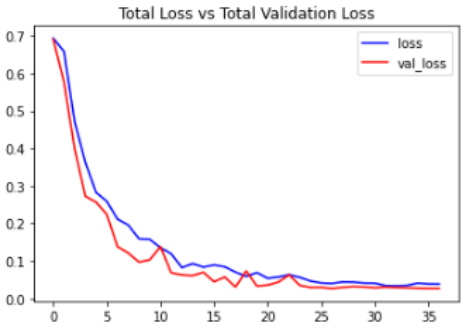
Notebook 5f

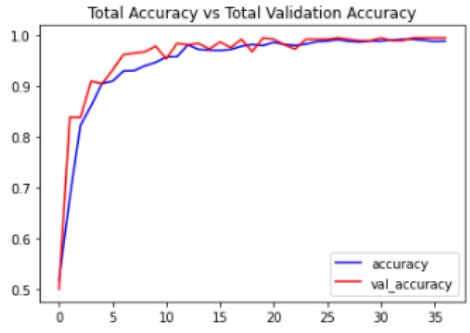
```
def create_LRCN_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura
    model.add(TimeDistributed(Conv2D(4, (3, 3), padding='same',activation = 'relu'),
                              input_shape = (NR_OF_FRAMES, IMAGE_HEIGHT, IMAGE_WIDTH, 3)))

    model.add(TimeDistributed(MaxPooling2D((4, 4))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Conv2D(8, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((4, 4))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Conv2D(16, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Conv2D(16, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Flatten()))

    model.add(LSTM(8))

    model.add(Dense(len(CLASSES_LIST), activation = 'sigmoid'))

    # Tabela
    model.summary()

    return model
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 time_distributed (TimeDistr  (None, 6, 72, 128, 4)    112
 ibuted)

 time_distributed_1 (TimeDis  (None, 6, 18, 32, 4)     0
 tributed)

 time_distributed_2 (TimeDis  (None, 6, 18, 32, 4)     0
 tributed)

 time_distributed_3 (TimeDis  (None, 6, 18, 32, 8)     296
 tributed)

 time_distributed_4 (TimeDis  (None, 6, 4, 8, 8)       0
 tributed)

 time_distributed_5 (TimeDis  (None, 6, 4, 8, 8)       0
 tributed)

 time_distributed_6 (TimeDis  (None, 6, 4, 8, 16)      1168
 tributed)

 time_distributed_7 (TimeDis  (None, 6, 2, 4, 16)      0
 tributed)

 time_distributed_8 (TimeDis  (None, 6, 2, 4, 16)      0
 tributed)

 time_distributed_9 (TimeDis  (None, 6, 2, 4, 16)      2320
 tributed)

 time_distributed_10 (TimeDi  (None, 6, 1, 2, 16)      0
 stributed)

 time_distributed_11 (TimeDi  (None, 6, 1, 2, 16)      0
 stributed)

 time_distributed_12 (TimeDi  (None, 6, 32)            0
 stributed)

 lstm (LSTM)                 (None, 8)                 1312

 dense (Dense)               (None, 2)                 18

=================================================================
Total params: 5,226
Trainable params: 5,226
Non-trainable params: 0
_________________________________________________________________
```

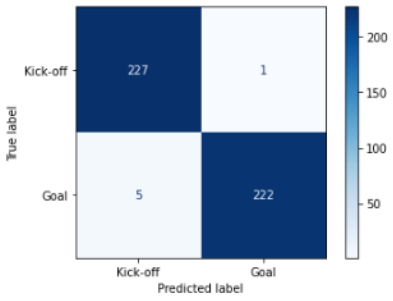
```
model_evaluation_history = LRCN_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED)
print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

```
15/15 [==============================] - 0s 17ms/step - loss: 0.0462 - accuracy: 0.9868
Accuracy: 98.68%
```





	precision	recall	f1-score	support
Kick-off	0.98	1.00	0.99	228
Goal	1.00	0.98	0.99	227
accuracy			0.99	455
macro avg	0.99	0.99	0.99	455
weighted avg	0.99	0.99	0.99	455



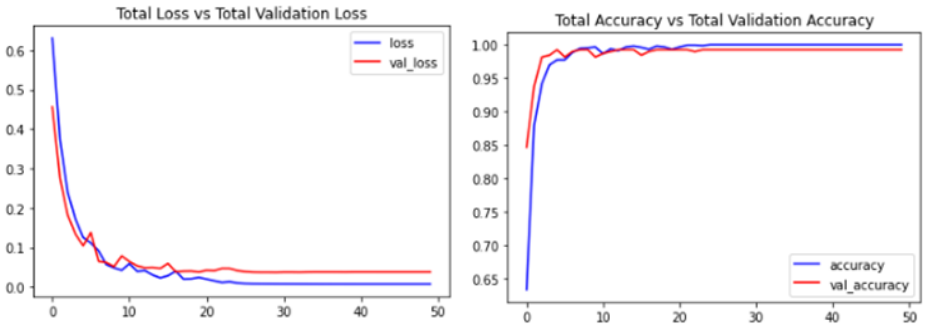
***Notebook* 5g**

```
def create_LRCN_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura
    model.add(TimeDistributed(Conv2D(4, (5,5), padding='same',activation = 'relu'),
                              input_shape = (NR_OF_FRAMES, IMAGE_HEIGHT, IMAGE_WIDTH, 3)))

    model.add(TimeDistributed(MaxPooling2D((4, 4))))

    model.add(TimeDistributed(Conv2D(8, (5,5), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((4, 4))))

    model.add(TimeDistributed(Conv2D(16, (5,5), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))

    model.add(TimeDistributed(Conv2D(16, (5,5), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))

    model.add(TimeDistributed(Flatten()))

    model.add(LSTM(8))

    model.add(Dense(len(CLASSES_LIST), activation = 'sigmoid'))

    # Tabela
    model.summary()

    return model
```

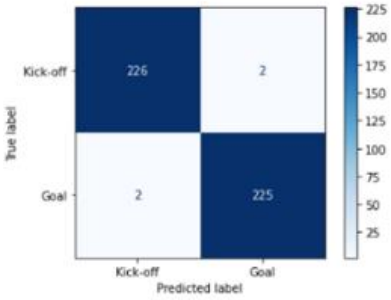
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 time_distributed (TimeDistr  (None, 6, 72, 128, 4)    304
 ibuted)

 time_distributed_1 (TimeDis  (None, 6, 18, 32, 4)     0
 tributed)

 time_distributed_2 (TimeDis  (None, 6, 18, 32, 8)     808
 tributed)

 time_distributed_3 (TimeDis  (None, 6, 4, 8, 8)       0
 tributed)

 time_distributed_4 (TimeDis  (None, 6, 4, 8, 16)      3216
 tributed)

 time_distributed_5 (TimeDis  (None, 6, 2, 4, 16)      0
 tributed)

 time_distributed_6 (TimeDis  (None, 6, 2, 4, 16)      6416
 tributed)

 time_distributed_7 (TimeDis  (None, 6, 1, 2, 16)      0
 tributed)

 time_distributed_8 (TimeDis  (None, 6, 32)            0
 tributed)

 lstm (LSTM)                 (None, 8)                 1312

 dense (Dense)               (None, 2)                 18

=================================================================
Total params: 12,074
Trainable params: 12,074
Non-trainable params: 0
_________________________________________________________________
```

```
model_evaluation_history = LRCN_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED)
print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

```
15/15 [==============================] - 0s 19ms/step - loss: 0.0353 - accuracy: 0.9912
Accuracy: 99.12%
```



	precision	recall	f1-score	support
Kick-off	0.99	0.99	0.99	228
Goal	0.99	0.99	0.99	227
accuracy			0.99	455
macro avg	0.99	0.99	0.99	455
weighted avg	0.99	0.99	0.99	455



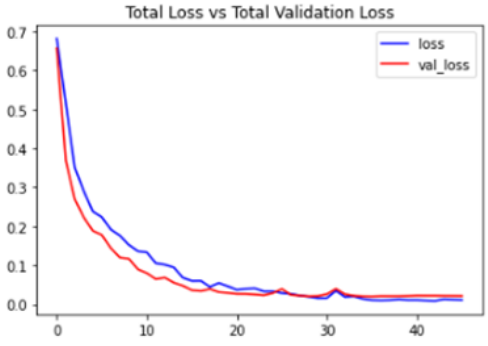
***Notebook* 5h**

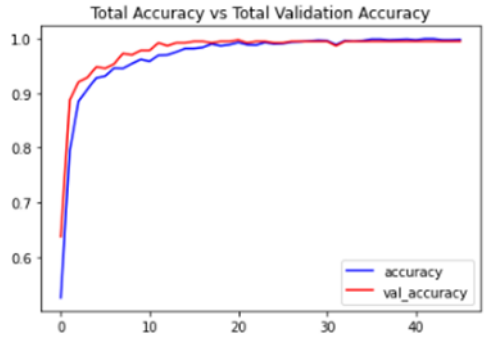
```
def create_LRCN_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura
    model.add(TimeDistributed(Conv2D(4, (5,5), padding='same',activation = 'relu'),
                              input_shape = (NR_OF_FRAMES, IMAGE_HEIGHT, IMAGE_WIDTH, 3)))

    model.add(TimeDistributed(MaxPooling2D((4, 4))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Conv2D(8, (5,5), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((4, 4))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Conv2D(16, (5,5), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Conv2D(16, (5,5), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Flatten()))

    model.add(LSTM(8))

    model.add(Dense(len(CLASSES_LIST), activation = 'sigmoid'))

    # Tabela
    model.summary()

    return model
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 time_distributed (TimeDistr  (None, 6, 72, 128, 4)    304
 ibuted)

 time_distributed_1 (TimeDis  (None, 6, 18, 32, 4)     0
 tributed)

 time_distributed_2 (TimeDis  (None, 6, 18, 32, 4)     0
 tributed)

 time_distributed_3 (TimeDis  (None, 6, 18, 32, 8)     808
 tributed)

 time_distributed_4 (TimeDis  (None, 6, 4, 8, 8)       0
 tributed)

 time_distributed_5 (TimeDis  (None, 6, 4, 8, 8)       0
 tributed)

 time_distributed_6 (TimeDis  (None, 6, 4, 8, 16)      3216
 tributed)

 time_distributed_7 (TimeDis  (None, 6, 2, 4, 16)      0
 tributed)

 time_distributed_8 (TimeDis  (None, 6, 2, 4, 16)      0
 tributed)

 time_distributed_9 (TimeDis  (None, 6, 2, 4, 16)      6416
 tributed)

 time_distributed_10 (TimeDi  (None, 6, 1, 2, 16)      0
 stributed)

 time_distributed_11 (TimeDi  (None, 6, 1, 2, 16)      0
 stributed)

 time_distributed_12 (TimeDi  (None, 6, 32)            0
 stributed)

 lstm (LSTM)                 (None, 8)                 1312

 dense (Dense)               (None, 2)                 18

=================================================================
Total params: 12,074
Trainable params: 12,074
Non-trainable params: 0
_________________________________________________________________
```

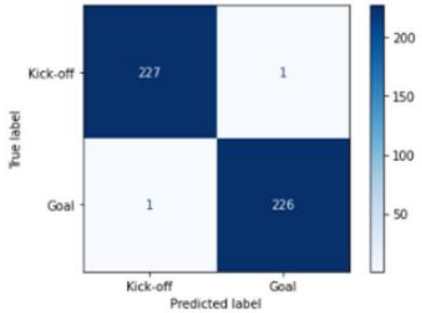
```
model_evaluation_history = LRCN_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED)
print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

```
15/15 [==============================] - 0s 20ms/step - loss: 0.0286 - accuracy: 0.9956
Accuracy: 99.56%
```





	precision	recall	f1-score	support
Kick-off	1.00	1.00	1.00	228
Goal	1.00	1.00	1.00	227
accuracy			1.00	455
macro avg	1.00	1.00	1.00	455
weighted avg	1.00	1.00	1.00	455



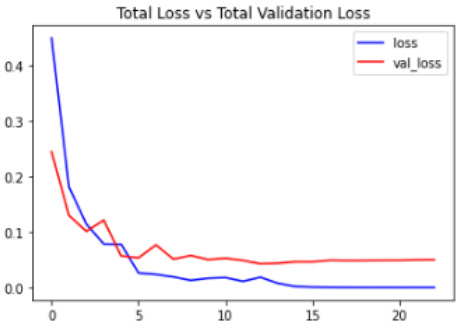
***Notebook* 5i**

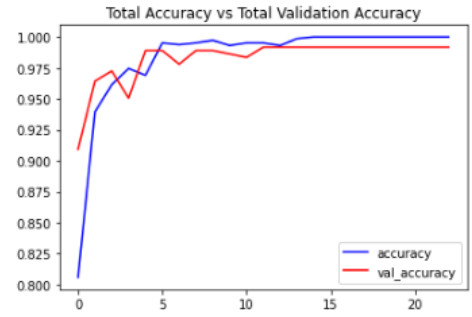
```
def create_LRCN_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura
    model.add(TimeDistributed(Conv2D(16, (3, 3), padding='same',activation = 'relu'),
                              input_shape = (NR_OF_FRAMES, IMAGE_HEIGHT, IMAGE_WIDTH, 3)))

    model.add(TimeDistributed(MaxPooling2D((4, 4))))

    model.add(TimeDistributed(Conv2D(32, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((4, 4))))

    model.add(TimeDistributed(Conv2D(64, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))

    model.add(TimeDistributed(Conv2D(64, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))

    model.add(TimeDistributed(Flatten()))

    model.add(LSTM(32))

    model.add(Dense(len(CLASSES_LIST), activation = 'sigmoid'))

    # Tabela
    model.summary()

    return model
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 time_distributed (TimeDistr  (None, 6, 72, 128, 16)   448
 ibuted)

 time_distributed_1 (TimeDis  (None, 6, 18, 32, 16)    0
 tributed)

 time_distributed_2 (TimeDis  (None, 6, 18, 32, 32)    4640
 tributed)

 time_distributed_3 (TimeDis  (None, 6, 4, 8, 32)      0
 tributed)

 time_distributed_4 (TimeDis  (None, 6, 4, 8, 64)      18496
 tributed)

 time_distributed_5 (TimeDis  (None, 6, 2, 4, 64)      0
 tributed)

 time_distributed_6 (TimeDis  (None, 6, 2, 4, 64)      36928
 tributed)

 time_distributed_7 (TimeDis  (None, 6, 1, 2, 64)      0
 tributed)

 time_distributed_8 (TimeDis  (None, 6, 128)           0
 tributed)

 lstm (LSTM)                 (None, 32)                20608

 dense (Dense)               (None, 2)                 66

=================================================================
Total params: 81,186
Trainable params: 81,186
Non-trainable params: 0
_________________________________________________________________
```

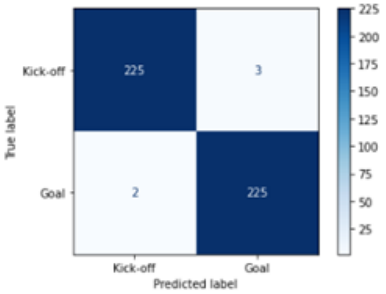
```
model_evaluation_history = LRCN_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED)
print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

```
15/15 [==============================] - 0s 24ms/step - loss: 0.0456 - accuracy: 0.9890
Accuracy: 98.90%
```





	precision	recall	f1-score	support
Kick-off	0.99	0.99	0.99	228
Goal	0.99	0.99	0.99	227
accuracy			0.99	455
macro avg	0.99	0.99	0.99	455
weighted avg	0.99	0.99	0.99	455



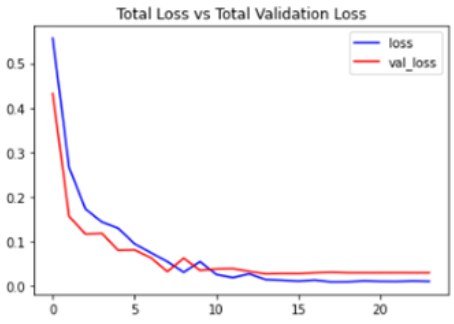
***Notebook* 5j**

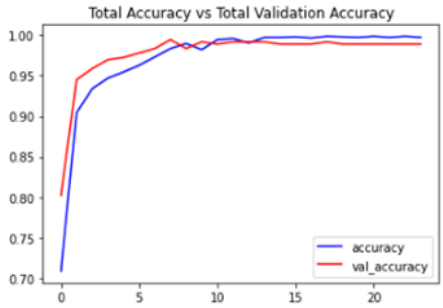
```
def create_LRCN_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura
    model.add(TimeDistributed(Conv2D(16, (3, 3), padding='same',activation = 'relu'),
                              input_shape = (NR_OF_FRAMES, IMAGE_HEIGHT, IMAGE_WIDTH, 3)))

    model.add(TimeDistributed(MaxPooling2D((4, 4))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Conv2D(32, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((4, 4))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Conv2D(64, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Conv2D(64, (3, 3), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Flatten()))

    model.add(LSTM(32))

    model.add(Dense(len(CLASSES_LIST), activation = 'sigmoid'))

    # Tabela
    model.summary()

    return model
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 time_distributed (TimeDistr  (None, 6, 72, 128, 16)   448
 ibuted)

 time_distributed_1 (TimeDis  (None, 6, 18, 32, 16)    0
 tributed)

 time_distributed_2 (TimeDis  (None, 6, 18, 32, 16)    0
 tributed)

 time_distributed_3 (TimeDis  (None, 6, 18, 32, 32)    4640
 tributed)

 time_distributed_4 (TimeDis  (None, 6, 4, 8, 32)      0
 tributed)

 time_distributed_5 (TimeDis  (None, 6, 4, 8, 32)      0
 tributed)

 time_distributed_6 (TimeDis  (None, 6, 4, 8, 64)      18496
 tributed)

 time_distributed_7 (TimeDis  (None, 6, 2, 4, 64)      0
 tributed)

 time_distributed_8 (TimeDis  (None, 6, 2, 4, 64)      0
 tributed)

 time_distributed_9 (TimeDis  (None, 6, 2, 4, 64)      36928
 tributed)

 time_distributed_10 (TimeDi  (None, 6, 1, 2, 64)      0
 stributed)

 time_distributed_11 (TimeDi  (None, 6, 1, 2, 64)      0
 stributed)

 time_distributed_12 (TimeDi  (None, 6, 128)           0
 stributed)

 lstm (LSTM)                 (None, 32)                20608

 dense (Dense)               (None, 2)                 66

=================================================================
Total params: 81,186
Trainable params: 81,186
Non-trainable params: 0
_________________________________________________________________
```

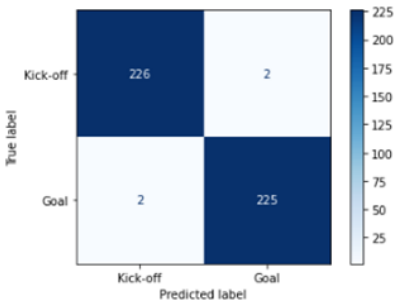
```
model_evaluation_history = LRCN_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED)
print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

```
15/15 [==============================] - 0s 23ms/step - loss: 0.0435 - accuracy: 0.9912
Accuracy: 99.12%
```





	precision	recall	f1-score	support
Kick-off	0.99	0.99	0.99	228
Goal	0.99	0.99	0.99	227
accuracy			0.99	455
macro avg	0.99	0.99	0.99	455
weighted avg	0.99	0.99	0.99	455



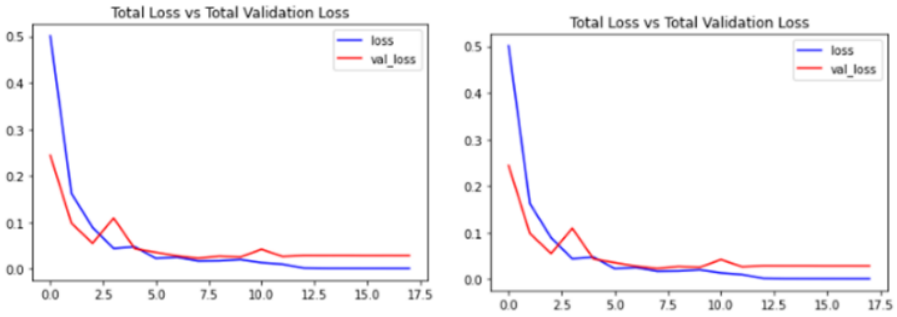
***Notebook* 5k**

```
def create_LRCN_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura
    model.add(TimeDistributed(Conv2D(16, (5,5), padding='same',activation = 'relu'),
                              input_shape = (NR_OF_FRAMES, IMAGE_HEIGHT, IMAGE_WIDTH, 3)))

    model.add(TimeDistributed(MaxPooling2D((4, 4))))

    model.add(TimeDistributed(Conv2D(32, (5,5), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((4, 4))))

    model.add(TimeDistributed(Conv2D(64, (5,5), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))

    model.add(TimeDistributed(Conv2D(64, (5,5), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))

    model.add(TimeDistributed(Flatten()))

    model.add(LSTM(32))

    model.add(Dense(len(CLASSES_LIST), activation = 'sigmoid'))

    # Tabela
    model.summary()

    return model
```

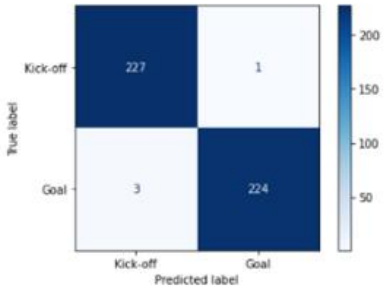
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 time_distributed (TimeDistr  (None, 6, 72, 128, 16)   1216
 ibuted)

 time_distributed_1 (TimeDis  (None, 6, 18, 32, 16)    0
 tributed)

 time_distributed_2 (TimeDis  (None, 6, 18, 32, 32)    12832
 tributed)

 time_distributed_3 (TimeDis  (None, 6, 4, 8, 32)      0
 tributed)

 time_distributed_4 (TimeDis  (None, 6, 4, 8, 64)      51264
 tributed)

 time_distributed_5 (TimeDis  (None, 6, 2, 4, 64)      0
 tributed)

 time_distributed_6 (TimeDis  (None, 6, 2, 4, 64)      102464
 tributed)

 time_distributed_7 (TimeDis  (None, 6, 1, 2, 64)      0
 tributed)

 time_distributed_8 (TimeDis  (None, 6, 128)           0
 tributed)

 lstm (LSTM)                 (None, 32)                20608

 dense (Dense)               (None, 2)                 66

=================================================================
Total params: 188,450
Trainable params: 188,450
Non-trainable params: 0
_________________________________________________________________
```

```
model_evaluation_history = LRCN_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED)
print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

```
15/15 [==============================] - 0s 26ms/step - loss: 0.0406 - accuracy: 0.9912
Accuracy: 99.12%
```



	precision	recall	f1-score	support
Kick-off	0.99	1.00	0.99	228
Goal	1.00	0.99	0.99	227
accuracy			0.99	455
macro avg	0.99	0.99	0.99	455
weighted avg	0.99	0.99	0.99	455



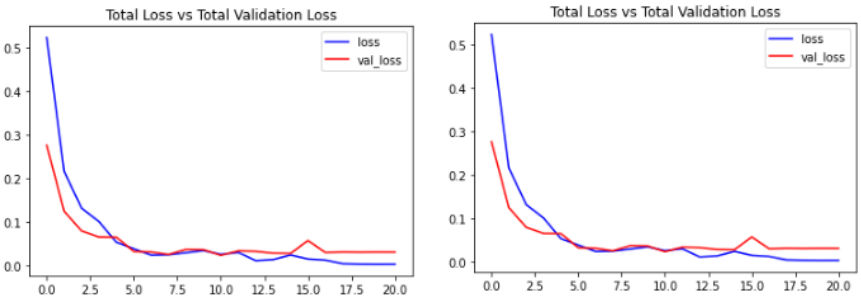
***Notebook* 5l**

```
def create_LRCN_model():

    # Modelo sequencial
    model = Sequential()

    # Arquitetura
    model.add(TimeDistributed(Conv2D(16, (5,5), padding='same',activation = 'relu'),
                              input_shape = (NR_OF_FRAMES, IMAGE_HEIGHT, IMAGE_WIDTH, 3)))

    model.add(TimeDistributed(MaxPooling2D((4, 4))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Conv2D(32, (5,5), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((4, 4))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Conv2D(64, (5,5), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Conv2D(64, (5,5), padding='same',activation = 'relu')))
    model.add(TimeDistributed(MaxPooling2D((2, 2))))
    model.add(TimeDistributed(Dropout(0.2)))

    model.add(TimeDistributed(Flatten()))

    model.add(LSTM(32))

    model.add(Dense(len(CLASSES_LIST), activation = 'sigmoid'))

    # Tabela
    model.summary()

    return model
```

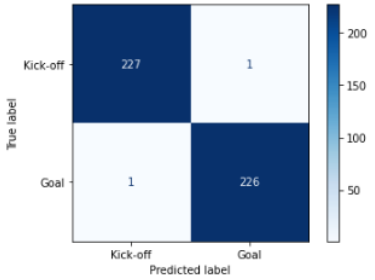
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 time_distributed (TimeDistr  (None, 6, 72, 128, 16)   1216
 ibuted)

 time_distributed_1 (TimeDis  (None, 6, 18, 32, 16)    0
 tributed)

 time_distributed_2 (TimeDis  (None, 6, 18, 32, 16)    0
 tributed)

 time_distributed_3 (TimeDis  (None, 6, 18, 32, 32)    12832
 tributed)

 time_distributed_4 (TimeDis  (None, 6, 4, 8, 32)      0
 tributed)

 time_distributed_5 (TimeDis  (None, 6, 4, 8, 32)      0
 tributed)

 time_distributed_6 (TimeDis  (None, 6, 4, 8, 64)      51264
 tributed)

 time_distributed_7 (TimeDis  (None, 6, 2, 4, 64)      0
 tributed)

 time_distributed_8 (TimeDis  (None, 6, 2, 4, 64)      0
 tributed)

 time_distributed_9 (TimeDis  (None, 6, 2, 4, 64)      102464
 tributed)

 time_distributed_10 (TimeDi  (None, 6, 1, 2, 64)      0
 stributed)

 time_distributed_11 (TimeDi  (None, 6, 1, 2, 64)      0
 stributed)

 time_distributed_12 (TimeDi  (None, 6, 128)           0
 stributed)

 lstm (LSTM)                 (None, 32)                20608

 dense (Dense)               (None, 2)                 66

=================================================================
Total params: 188,450
Trainable params: 188,450
Non-trainable params: 0
_________________________________________________________________
```

```
model_evaluation_history = LRCN_model.evaluate(FEATURES_TEST_STORED, LABELS_TEST_STORED)
print("Accuracy: %.2f%%" % (model_evaluation_history[1]*100))
```

```
15/15 [==============================] - 0s 26ms/step - loss: 0.0326 - accuracy: 0.9956
Accuracy: 99.56%
```



	precision	recall	f1-score	support
Kick-off	1.00	1.00	1.00	228
Goal	1.00	1.00	1.00	227
accuracy			1.00	455
macro avg	1.00	1.00	1.00	455
weighted avg	1.00	1.00	1.00	455



Anexo F

Notebooks* 6 - *Predict

Constantes

```
# Frames
%store -r frame_seconds_diff
FRAME_SECONDS_DIFF = frame_seconds_diff
NR_OF_FRAMES = len(FRAME_SECONDS_DIFF)

# Altura de cada frame
%store -r image_height
IMAGE_HEIGHT = image_height

# Largura de cada frame
%store -r image_width
IMAGE_WIDTH = image_width

# Classes para treino
%store -r classes_list
CLASSES_LIST = classes_list

# Classes para treino
%store -r test_videos_dir
TEST_VIDEOS_DIR = test_videos_dir
```

Imports

```
import tensorflow as tf

import cv2
from collections import deque
import numpy as np
import time
import datetime as dt
import json

import os
```

Obter modelo guardado

```
#model_file_name= "LRCN_model_h___Date_Time_2022_10_09__11_05_22___Loss_0.02855760045349598___Accuracy_0.995604395866394"
#is_convLSTM = False

#model_file_name="convlstm_model_g__Date_Time_2022_10_08__23_51_05___Loss_0.07677458226680756___Accuracy_0.9824175834655762"
model_file_name="convlstm_model_f___Date_Time_2022_10_08__13_37_08___Loss_0.6920326352119446___Accuracy_0.578671208381653"
is_convLSTM = True
```

```
if is_convLSTM:
    model = tf.keras.models.load_model(f'Modelos/ConvLSTM/{model_file_name}.h5')
    # THRESHOLD
    THRESHOLD = 0.995
    MIN_TIME = 60000 #10000
    MAX_TIME = 120000 #60000

else:
    model = tf.keras.models.load_model(f'Modelos/LRCN/{model_file_name}.h5')
    # THRESHOLD
    THRESHOLD = 0.995
    MIN_TIME = 60000
    MAX_TIME = 120000
```

Auxiliares

```
#milisegundos(número) para time (string)
def get_time_str(milliseconds):

    seconds, milliseconds = divmod(milliseconds, 1000)
    minutes, seconds = divmod(seconds, 60)
    hours, minutes = divmod(minutes, 60)

    return f'{int(hours):02d}:{int(minutes):02d}:{int(seconds):02d}.{int(milliseconds):03d}'
```

```python
def predict_on_video(video_file_path, SEQUENCE_LENGTH):

    #Data atual
    date_time_format = '%Y_%m_%d__%H_%M_%S'
    current_date_time_dt = dt.datetime.now()
    current_date_time_string = dt.datetime.strftime(current_date_time_dt, date_time_format)

    #Ficheiro
    file_name = f'Predict_{current_date_time_string}.txt'
    txt_file = open(file_name,"a")

    txt_file.write('\n')
    txt_file.write('início do vídeo')
    txt_file.write('\n')

    #Contador das frames
    count = 0

    #Inicializar objeto VideoCapture para ler do path do vídeo
    video_reader = cv2.VideoCapture(video_file_path)

    #FPS
    fps = video_reader.get(cv2.CAP_PROP_FPS)

    #Queue para guardar as frames (Adiciona 1 - tira 1 do início)
    frames_queue = deque(maxlen = NR_OF_FRAMES)

    #Queue para guardar os timestamps das frames
    timestamps_queue = deque(maxlen = NR_OF_FRAMES)

    #Primeiro timestamp de um momento
    moment_first_timestamp = None

    #Última index de class registada
    last_predicted_class_index = None

    event_probs = []

    last_goal_moment_start_timestamp = None
    last_goal_moment_end_timestamp = None
    last_goal_moment_prob = None

    last_timestamps = []
```

```python
    #Enquanto o video for acessivel
    while video_reader.isOpened():

        video_reader.set(cv2.CAP_PROP_POS_FRAMES, count)

        #Leitura da frame
        ok, frame = video_reader.read()

        #Se a frame não tiver sido lida corretamente é feito um break
        if not ok:
            break

        timestamp = video_reader.get(cv2.CAP_PROP_POS_MSEC);

        #Resize da frame
        resized_frame = cv2.resize(frame, (IMAGE_WIDTH, IMAGE_HEIGHT))

        #Normalizar a frame (/255 -- cada pixel entre 0 e 1)
        normalized_frame = resized_frame / 255

        #Adicionar a frame à queue
        frames_queue.append(normalized_frame)
        timestamps_queue.append(timestamp)

        #Validar a existência do nr de frames desejado (6)
        if len(frames_queue) == NR_OF_FRAMES:

            predict = model.predict(np.expand_dims(frames_queue, axis = 0), verbose=1)
            predicted_labels_probabilities = predict[0]
            predicted_label = np.argmax(predicted_labels_probabilities)
            prob = predicted_labels_probabilities[predicted_label]

            #Se a prob for maior que threshold
            if prob > THRESHOLD :

                #first game moment
                if last_predicted_class_index == None:

                    last_timestamps = list(timestamps_queue)
                    last_predicted_class_index = predicted_label
                    moment_first_timestamp = timestamps_queue[0]
                    event_probs.append(prob)

                #same game moment
                elif last_predicted_class_index == predicted_label and timestamps_queue[0] in last_timestamps:

                    last_timestamps = list(timestamps_queue)
                    event_probs.append(prob)
```

```
            #game moment changed or different game time (can be the same class, but is a different game moment)
            else:

                #record event
                mean_prob = np.mean(event_probs)

                if last_predicted_class_index == 1 :
                    last_goal_moment_start_timestamp = moment_first_timestamp
                    last_goal_moment_end_timestamp = last_timestamps[-1]
                    last_goal_moment_prob = mean_prob

                if last_predicted_class_index == 0 and last_goal_moment_prob != None and
                    moment_first_timestamp >= last_goal_moment_end_timestamp + MIN_TIME and
                    moment_first_timestamp <= last_goal_moment_end_timestamp + MAX_TIME :

                    line = f'GOAL - time: {get_time_str(last_goal_moment_start_timestamp)} - {get_time_str(last_goal_moment_end_timestamp)} = {last_goal_moment_prob}'
                    txt_file.write('\n')
                    txt_file.write(line)
                    txt_file.write('\n')
                    last_goal_moment_start_timestamp = None
                    last_goal_moment_end_timestamp = None
                    last_goal_moment_prob = None

                event_probs = []

                #start new event
                last_predicted_class_index = predicted_label
                moment_first_timestamp = timestamps_queue[0]
                last_timestamps = list(timestamps_queue)
                event_probs.append(prob)

        #end of moment but no new event
        elif prob < THRESHOLD and last_predicted_class_index != None:

            #record event
            mean_prob = np.mean(event_probs)

            if last_predicted_class_index == 1 :
                last_goal_moment_start_timestamp = moment_first_timestamp
                last_goal_moment_end_timestamp = last_timestamps[-1]
                last_goal_moment_prob = mean_prob

            if last_predicted_class_index == 0 and last_goal_moment_prob != None and moment_first_timestamp >= last_goal_moment_end_timestamp + MIN_TIME and moment_first_timestamp <= last_goal_moment_end_timestamp + MAX_TIME :

                line = f'GOAL - time: {get_time_str(last_goal_moment_start_timestamp)} - {get_time_str(last_goal_moment_end_timestamp)} = {last_goal_moment_prob}'
                txt_file.write('\n')
                txt_file.write(line)
                txt_file.write('\n')

                last_goal_moment_start_timestamp = None
                last_goal_moment_end_timestamp = None
                last_goal_moment_prob = None
```

```
                #no event
                last_timestamps = []
                last_predicted_class_index = None
                moment_first_timestamp = None
                event_probs = []

            #no begin or end of moment
            else:

                #no event
                last_timestamps = []
                last_predicted_class_index = None
                moment_first_timestamp = None
                event_probs = []


        count += fps

    txt_file.write('\n')
    txt_file.write('fim do vídeo')
    txt_file.write('\n')

    # Release video_reader and txt_file
    video_reader.release()
    txt_file.close()
```

```
input_video_file_path = os.path.join(TEST_VIDEOS_DIR, "8", "Benfica 2 x 1 Sporting JOGO COMPLETO 13ª Jornada Liga NOS 2016 17 HD.mp4")
input_video_file_path_exists = os.path.exists(input_video_file_path)

if input_video_file_path_exists:
    predict_on_video(input_video_file_path, NR_OF_FRAMES)
    print("done")
else:
    print("file not found")
```