INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Gerador de código HTML a partir de maquetes

PAULO FRANCISCO FERREIRA PEREIRA
Outubro de 2018

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Gerador de código HTML a partir de maquetes

Paulo Francisco Ferreira Pereira

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

Orientador: Doutora Maria de Fátima Coutinho Rodrigues

Coorientador: Doutor Carlos Manuel Abreu Gomes Ferreira

Supervisor: Engenheiro Francisco Correira

Porto, 14 de Outubro 2018

Dedicatória

"Ao meu avô que sempre me ensinou a importância

do esforço e da aprendizagem"

Resumo

A automação das tarefas é cada vez mais uma prática atual no ambiente organizacional, sendo que esta prática permite diminuir a necessidade de mão de obra e muitas vezes diminuir os erros associados ao fator humano.

O processo atual de desenvolvimento de software na Glintt – HealthCare Solutions tem diversas fases, sendo uma delas a elaboração de protótipos visuais por parte da equipa de design. Estes protótipos visuais são posteriormente transformados em código fonte HTML/CSS/Javascript pela equipa de desenvolvimento. Este processo pode demorar entre dois dias a oito dias dependendo da complexidade do ecrã.

No presente documento será apresentada uma solução para gerar o código fonte a partir de uma maquete de forma automática tendo como *input* uma imagem correspondente ao protótipo. De forma a conceber este trabalho serão usadas técnicas de *Deep Learning*, em especial *Convolutional Neural Networks* para a deteção e classificação de objetos em imagens.

Palavras-chave: Automação, Deep Learning, Convolutional Neural Networks

Abstract

The automation of tasks is increasingly a current practice in the organizational environment, and this practice reduces the need for manpower and often reduces the errors associated with the human factor.

The current software development process at Glintt - HealthCare Solutions has several phases, one of which is the designing of visual prototypes by the design team. These visual prototypes are later transformed into HTML/CSS/Javascript source code by the development team. This process can take from two days to eight days depending on the complexity of the screen.

In the present document a solution will be presented to generate the source code of a mockup automatically, having as input an image corresponding to the prototype. In the development of this project techniques of Deep Learning will be used, especially Convolutional Neural Networks for the detection and classification of objects in images.

Keywords: Automation, Deep Learning, Convolutional Neural Networks

Agradecimentos

Gostaria de começar por agradecer à minha orientadora, Professora Doutora Fátima Rodrigues, que foi incansável na dedicação ao sucesso deste projeto, despendendo muito tempo em reuniões de orientação, fazendo correções ao meu trabalho e incutindo a exploração de novas abordagens ao problema. Fico muito agradecido também pela disponibilidade do Professor Doutor Carlos Ferreira em ser coorientador do projeto, também ele ajudou, com a sua experiência na área, como abordar o problema e que tecnologias deveriam ser usadas para a execução da solução.

Gostaria de agradecer à Instituição Superior de Engenharia do Porto (ISEP), por fornecer um ensino de excelência e me tornar mais capaz tanto a nível profissional como académico.

Aproveito também para agradecer à minha entidade empregadora, a Glintt – Healthcare Solutions S.A. por facilitar a utilização dos dados necessário para a realização deste projeto e ao Engenheiro Francisco Correia pelo tempo alocado ao acompanhamento do projeto e pelas opiniões sugeridas.

Também gostaria de salientar o papel dos meus colegas de mestrado, em especial, ao Engenheiro Carlos Silva, Engenheiro Francisco Ramalho e ao Engenheiro José Cabeda, pela partilha de conhecimento e pela companhia durante estes dois anos.

Um agradecimento especial também para o meu irmão, Engenheiro Tiago Pereira, que foi o grande responsável pelo meu interesse na área de Engenharia Informática e pela tecnologia em geral.

Gostaria de agradecer à Marta Moreira, por estar sempre ao meu lado em todas as ocasiões e por me motivar a alcançar os meus objetivos e também por ajudar durante o processo de desenvolvimento do projeto.

Por último, mas não menos importante, gostaria de agradecer à minha família, especialmente aos meus pais e avô, que sempre me motivaram e relembraram a importância do estudo, pelo apoio incondicional durante todos estes anos de estudo e por todos os sacrifícios, sem eles não teria sido possível.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

CNN	Convolutional Neural Networks
R-CNN	Region-based Convulotional Neural Networks
YOLO	You Only Look Once
RNN	Recurrent Neural Network
LSTM	Long-Short Term Memory
DSL	Domain Specific Language
Glintt-HS	Glintt HealthCare Solutions
GPU	Graphic Processing Unit
CPU	Central Processing Unit
CRISP-DM	Cross Industry Standard Process for Data Mining
NCD	New Concept Development Model
AHP	Analytic Hierarchy Process
SAVE	Society of American Value Engineers
IVM	Institute of Value Management
RPN	Regional Proposal Network
FEI	Front End of Inovation
GUI	Graphics User Interface
HTML	HyperText Markup Language
CSS	Cascading Style Sheets

1. Introdução

Neste capítulo serão apresentadas as razões que levaram à escrita deste documento, uma breve descrição do modelo de processo atual na Glintt – HealthCare Solutions e o problema que existe atualmente no processo de desenvolvimento. Depois será apresentado de forma breve a proposta de solução ao problema e os objetivos que pretendem ser atingidos, vem como os resultados esperados com a implementação desta solução.

Por fim, será apresentado o potencial valor que o projeto pode trazer para a Glintt-HS, apontado as razões pelas quais este projeto beneficia a organização. Na última subsecção será também apresentada a estrutura do presente documento de modo a facilitar a navegação no documento por parte do leitor.

1.1 Contexto

Atualmente, na Glintt - HealthCare Solutions, a idealização de uma nova funcionalidade ou de várias novas funcionalidades, tem um custo demasiado alto associado. Este processo consiste em levantar as necessidades do cliente ou potenciais funcionalidades que podem interessar ao mercado, e elaborar um documento de especificação. Após a especificação os consultores reúnem-se com a equipa de *design* para idealizarem um protótipo da futura aplicação que irá ser desenvolvida para o cliente. Finalmente depois da equipa de *design* e a equipa de produto estarem de acordo com a aparência e comportamento que os ecrãs da aplicação deverão ter, é transmitido à equipa de desenvolvimento o documento de especificação acompanhado com o desenho dos ecrãs para procederem à implementação.

1.2 Problema

Nos ecrãs definidos para as novas aplicações são muitas vezes usados componentes que já foram implementados diversas vezes, o que leva a que exista um trabalho repetitivo, ou reutilização de código que pode conduzir à propagação de erros por diversas aplicações.

Também é necessário que exista um conjunto de pessoas especializadas na tradução de ecrãs para código, e que conheçam muito bem a *framework* interna de controlos da Glintt-HS, o "Glinttology". Existe também um custo muito grande associado à formação de um novo colaborador para aprender a trabalhar com esta *framework*.

Apesar do aspeto estético da aplicação ser importante para o cliente, acaba por ser uma tarefa acessória, uma vez que a estética por si só não conduz à resolução das funcionalidades

pretendidas pelo cliente. Por isso é importante automatizar ou reduzir o tempo despendido nas tarefas acessórias e focar em resolver realmente o problema dos clientes, dando mais importância à implementação das regras de negócio.

1.3 Objetivo

Este projeto tem como objetivo reduzir os custos associados à transformação de maquetes em código. Para resolver este problema serão usadas técnicas de *Deep Learning*, como por exemplo deteção de objetos em imagens.

A implementação de novas funcionalidades deverá ser mais rápida e com menos erros do que acontece com o atual processo de desenvolvimento. Atualmente implementar um novo ecrã simples com apenas um desenvolvedor, demora aproximadamente dois dias, sendo expectável que o novo sistema não demore mais do que um minuto a gerar o código correspondente, no entanto este código gerado pode eventualmente necessitar de algum desenvolvimento. No caso de um ecrã mais complexo o tempo de implementação pode chegar a ser de oito dias, no entanto, com o novo sistema, o tempo de realização será o mesmo independentemente da complexidade da maquete.

1.4 Resultados Esperados

Com a implementação desta solução é expectável que se automatize o processo de desenvolvimento de *software* na Glintt-HS, e também que esta solução cumpra os objetivos de realizar esta automação com a taxa de erros abaixo dos 20% e com o tempo de implementação abaixo de um minuto.

Através desta implementação também é pretendido que a Glintt-HS reconheça o potencial da área de *Machine Learning* e que comece a apostar mais nesta área, que aliás tem muita aplicabilidade na área da saúde onde tem uma forte presença no mercado.

Também é esperado que este projeto demonstre à comunidade de *Machine Learning* que a área de automação de geração de código é uma área com grande potencial e com grande aplicabilidade no mundo empresarial e incentivar ao desenvolvimento de novas soluções que estimulem o crescimento desta área.

1.5 Análise de valor

A implementação desta solução tem um valor considerável se for considerado a eficiência no processo de desenvolvimento que este projeto trará à Glintt-HS. Este projeto permite a diminuição da mão de obra necessária para implementar uma solução de *software*, diminuindo assim os custos com colaboradores.

Para além de diminuir os gastos com colaboradores, também irá diminuir os custos associados à formação, uma vez que deixa de ser necessário instruir uma quantidade muito grande de colaboradores a aprender os processos de implementação do código associado à maquete.

Por último a introdução desta solução no processo atual da Glintt-HS irá fazer com que as entregas de *software* sejam feitas de forma mais rápida uma vez que o código relativo às maquetes será gerado num período mais curto comparado com o que ocorria no processo de desenvolvimento anterior.

1.6 Abordagem Preconizada

Para construir a solução que irá resolver o problema e atingir os objetivos apresentados anteriormente, será seguido um processo desenvolvimento próprio da área de *Machine Learning.*

Num processo de desenvolvimento de *software*, como por exemplo no SCRUM, existe um conjunto de requisitos a resolver durante um certo período de tempo. Durante este tempo a equipa comunica entre si e resolve as tarefas planeadas, testa devidamente o seu código e é entregue ao cliente o trabalho relativo a esse período de tempo. Este tipo de projeto é iterativo e incremental e acaba quando não existem mais casos de uso no *backlog*.

No entanto na área de *Machine Learning* o processo é diferente. A metodologia CRISP-DM (Cross Industry Standard Process for Data Mining) (Wirth & Hipp, 2000) de suporte a projetos de Data Mining, descreve um modelo de referência que define as fases a seguir, as tarefas a executar e os resultados esperados pela realização de cada uma das tarefas. Esta metodologia encontra-se como o standard mais utilizado, devido à flexibilidade da sua implementação em qualquer domínio e à compatibilidade com qualquer ferramenta de Data Mining. Esta metodologia envolve os seguintes passos:

1. **Recolha de dados:** durante esta fase será necessário recolher dados, como por exemplo, recolher o código de ecrãs já implementados e a maquete correspondente a esse mesmo ecrã. Esta fase é muito importante, pois a quantidade e qualidade dos dados recolhidos influenciam diretamente o desempenho do modelo.
2. **Pré-processamento:** na fase de pré-processamento, é preciso modificar algumas características nos dados, como por exemplo, no caso das imagens pode-se ter que redimensionar as imagens para terem todas o mesmo número de pixéis.
3. **Treino:** durante a fase de treino será escolhido um conjunto de algoritmos que seja adequado ao tipo de dados. Para imagens, normalmente, escolhe-se entre vários tipos de *Convolutional Neural Networks* existentes atualmente. Depois serão separados os dados em duas partes, a parte treino e a parte de testes. Para a parte de treino serão usados 70% dos dados e para a parte de testes usar-se-à 20% dos dados recolhidos para avaliar o modelo treinado.

4. **Avaliação do modelo gerado:** após a fase de treino será avaliado o desempenho do modelo gerado, segundo a precisão que o modelo obteve ao processar a parte destinada como teste. No caso do modelo não ter um desempenho satisfatório pode-se escolher outros algoritmos, ou modificar o algoritmo usado, aplicando novas camadas ou modificando os parâmetros e voltamos a treinar e avaliar, até que o modelo gerado tenha um nível de desempenho satisfatório.
5. **Produção:** após ter-se encontrado o modelo que tenha uma boa precisão, será colocado em produção para que possa começar a fazer a previsões em contexto real.

1.7 Organização do Documento

O primeiro capítulo do presente documento apresenta uma breve introdução e interpretação do problema, apresentando em que contexto o problema surge. No mesmo capítulo também são apresentados os objetivos que a solução pretende alcançar, os resultados esperados e por último será apresentada de forma sucinta a abordagem preconizada.

No segundo capítulo é apresentada a análise de valor, onde são expostas as metodologias de análise de valor da solução, como por exemplo, o modelo de Canvas, modelo NCD e o modelo de AHP. À medida que estas metodologias vão sendo apresentadas também são aplicadas ao problema em questão para dar a conhecer o valor da solução a ser desenvolvida.

No estado da arte, são apresentadas as tecnologias inerentes ao problema em questão, começando pela apresentação do tema de *Deep Learning* seguido de alguns algoritmos de deteção de objetos em imagens e de processamento de linguagem. Ainda neste capítulo são apresentadas as tecnologias existentes atualmente que poderão ser usadas no desenvolvimento da solução.

No capítulo cinco é apresentado sem grande detalhe o design para a solução, começando por descrever a possível arquitetura para a solução e as suas alternativas.

No capítulo seis são confrontadas as tecnologias descritas no capítulo três de maneira a esclarecer as vantagens e as desvantagens de cada uma e posteriormente a escolha da *stack* tecnológica para o desenvolvimento.

O sétimo e último capítulo apresenta as conclusões relativamente ao trabalho desenvolvido.

2. Análise de Valor

O conceito de análise de valor tem várias definições, sendo que a Society of American Value Engineers (SAVE) descreve valor como uma abordagem sistemática e estruturada para melhorar projetos, produtos e processo. A análise de valor ajuda a alcançar um equilíbrio entre função, desempenho, qualidade, segurança e custo (SAVE, 2016).

A SAVE vai ainda mais longe na sua definição e chega mesmo a definir uma fórmula de calculo de valor:

$$Valor = \frac{Função}{Custo},$$

sendo que a função são as funcionalidades do produto e custo, os recursos necessários para criar o produto.

A Institute of Value Management (IVM), define valor como a relação entre a satisfação das necessidades e expectativas e os recursos necessários para alcançá-los.

Através das definições de valor mencionadas anteriormente, pode-se então concordar que valor é traduzido no equilíbrio entre a satisfação do cliente e o preço. O conceito de valor vai de encontro ao princípio em que o consumidor procura sempre o melhor produto ao menor preço.

O cliente nem sempre sabe todos os benefícios ou funcionalidades que um produto ou serviço pode trazer apesar de saber exatamente o preço, por isso é normal ter uma perceção de valor que pode não corresponder à realidade do produto.

Pode-se então afirmar que a perceção de valor de um produto é o valor que um produto ou serviços têm antes de o cliente usufruir de todas as funcionalidades e de conhecer os benefícios que traz para si. É comum as organizações usarem marketing não só para divulgar os seus produtos e serviços, mas também para aumentar a perceção de valor relativo ao produto.

Enquadrando a perceção do valor para os clientes, os desenvolvedores podem por exemplo ter a perceção que este projeto irá gerar código cem por cento livre de erros e que não terão de alterar uma linha de código daquele que é gerado pela plataforma, mas após utilizar o serviço podem notar que aquilo que estavam à espera, não está de acordo com a realidade, uma vez que a plataforma pode ter erros de deteção.

O valor para o cliente é normalmente explicado como a diferença entre os benefícios e os custos associados a um produto ou serviços. Quando os benefícios são superiores aos custos, o valor para o cliente é positivo e traduz-se num bom investimento, por outro lado, se os custos forem superiores aos benefícios, o valor para o cliente é negativo e levar o cliente a

considerar que é um mau investimento. O valor para ao cliente pode ser demonstrado através do seguinte cálculo,

$$Valor\ para\ o\ cliente = Beneficio - Custo$$

Uma perspetiva longitudinal de valor, que pode ser observada na Figura 1, foi apresentada por Tony Woodall, que estabelece 4 etapas temporais, desde a pré-compra até o após uso do cliente.

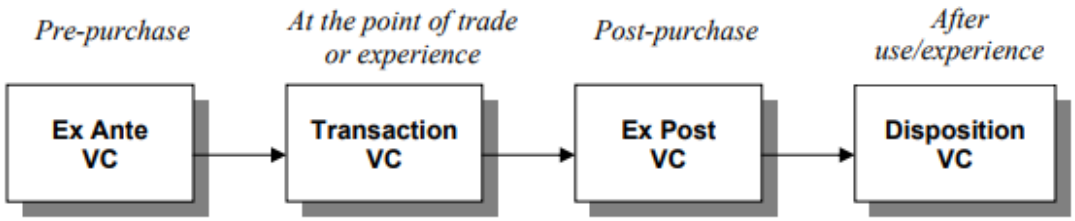


Figura 1 – Perspetiva longitudinal do valor para o cliente (Woodall, 2003).

Na fase de pré-compra é tentado perceber qual é a perceção que o público-alvo tem dos serviços e produtos que a empresa está a oferecer. Na fase de ponto de negócio é caracterizada pela sensação de valor para o cliente no ponto onde é efetuada a compra. Na fase de pós-compra é recebido os resultados das experiências dos clientes e finalmente, a fase pós-experiência que é a caracterizada pela reflexão sobre do ponto de venda.

Na Tabela 1 pode-se observar os benefícios e sacrifícios em cada fase da perspetiva longitudinal de valor de Woodal aplicada à Glintt-HS e ao projeto em questão.

Tabela 1 – Benefícios e sacrifícios em cada fase da perspetiva longitudinal de valor.

	Benefícios	**Sacrifícios**
Ex Ante VC	• Benefícios de logística • Benefícios funcionais	• Esforço • Custo de procura
Transaction VC	• Confiança	• Tempo
Ex Post VC	• Qualidade de performance	• Tempo • Esforço
Disposition VC	• Qualidade • Suporte do serviço	

2.1 Proposta de valor

A proposta de valor é a "(…) definição de como itens de valor, como produtos e serviços, bem como serviços complementares de valor agregado, são embalados e oferecidos para atender a necessidade do cliente (Osterwalder & Pigneur, 2003).

O desenvolvimento de técnicas automáticas de geração de código a partir de imagens são uma prática muito recente, sendo poucas as soluções existentes no mercado atualmente e a popularidade destes serviços ainda é muito baixa.

A implementação desta solução permite reduzir o número de colaboradores a fazer o trabalho repetitivo de transformar maquetes em código e alocar estes mesmos recursos no desenvolvimento das áreas *core* do negócio, ou então, usar este recursos para implementar boas práticas nos projetos em que estão inseridos para garantir a eficácia e eficiência da solução, como por exemplo, através do aumento de testes unitários e de integração ou de implementação e manutenção de um sistema integração contínua e entrega contínua.

2.2 New Concept Development Model – NCD Model

O New Concept Development Model (NCD) é um modelo constituído por três parte principais que podem ser observadas na Figura 2, sendo essas partes:

1. As cinco chaves que fazem parte do Front End of Inovation (FEI).
2. O *Engine* ou Motor, que influencia os cinco elementos do FEI e que é influenciada pela liderança e cultura da organização.
3. Os *Influencing factors* ou fatores de influência, são os fatores externos que influenciam a organização como por exemplo, fatores económicos, culturais ou legais.

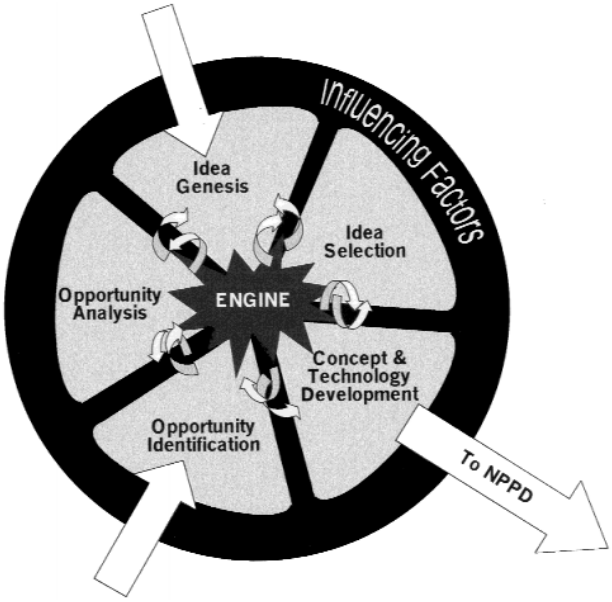


Figura 2 – Representação do New Concept Development Model (NCD) (Koen, 2001).

As cinco chaves do modelo de NCD descrevem os passos desde a identificação da oportunidade até à conceção e implementação da ideia.

2.2.1 Identificação da oportunidade

A identificação de oportunidades, é caracterizada pela análise de mercado e identificação de oportunidades que façam sentido para a estratégia e valores da organização. Existem diversas técnicas como o *brainstorming*, mapeamento mental e outras técnicas menos formais, como simples discussões em grupo que permitem a melhor identificação destas oportunidades.

A geração de código automático através de maquetes partiu da identificação da oportunidade de usar tecnologia recente, como o *Deep Learning* de modo a automatizar processos existentes na Glintt.

2.2.2 Análise de oportunidade

Após a identificação da oportunidade é necessário estudar e aprofundar o conhecimento na área identificada e perceber de que modo a oportunidade pode ser aproveitada e consequentemente se tem um valor de negócio que compense a empresa fazer o investimento.

De modo a saber se a utilização das tecnologias de Deep Learning para automatizar os processos na Glintt, representam uma real oportunidade para a empresa é necessário saber que tipo de recursos são necessários e o investimento necessário. Apesar da organização não possuir mão de obra qualificada na área, o investimento neste projeto é uma boa oportunidade uma vez que com a redução de tempo em formação e codificação, a longo prazo trará valor para a organização.

2.2.3 Ideia

Durante esta fase nasce a ideia concretamente, sendo normalmente discutida dentro da organização e consequentemente poderá mudar relativamente à ideia original, aperfeiçoando-se num processo contínuo de divulgação e *feedback.*

Nesta fase é espectável que o projeto de geração de código seja discutido entre os envolvidos no processo, como por exemplo a equipa de design e os *Front End Developers,* podendo desta forma levar a contributos que não tenham sido pensados inicialmente e pode trazer um valor acrescido para o projeto.

2.2.4 Seleção da ideia

A partir da ideia inicial nascem novas ideias, sendo necessário decidir em algum ponto qual ou quais as ideias que irão trazer maior valor para a organização.

Um sistema de geração automática de código a partir de maquetes pode ser implementado de várias formas e usando diversas tecnologias, no entanto, é necessário escolher entre as opções qual irá trazer mais benefícios e terá o menor custo.

2.2.5 Conceito e desenvolvimento tecnológico

O passo final do modelo NCD "(…) envolve o desenvolvimento de um caso de negócios baseado em estimativas de potencial de mercado, necessidades de clientes, requisitos de investimento, avaliações de concorrentes, desconhecidos de tecnologia e risco geral de projeto" (Koen, 2001).

Durante esta fase o projeto de geração de código automático a partir de maquetes já deve estar totalmente estruturado e alinhado com as partes interessadas e com o design definido e pronto a ser implementado, bem como a estimativa de custos.

2.3 Modelo Canvas

O modelo Canvas foi desenvolvido por Alexander Osterwalder em 2008, e desde então tem sido amplamente utilizado na definição de planos de negócio de forma rápida, e tem adaptado conforme as circunstâncias.

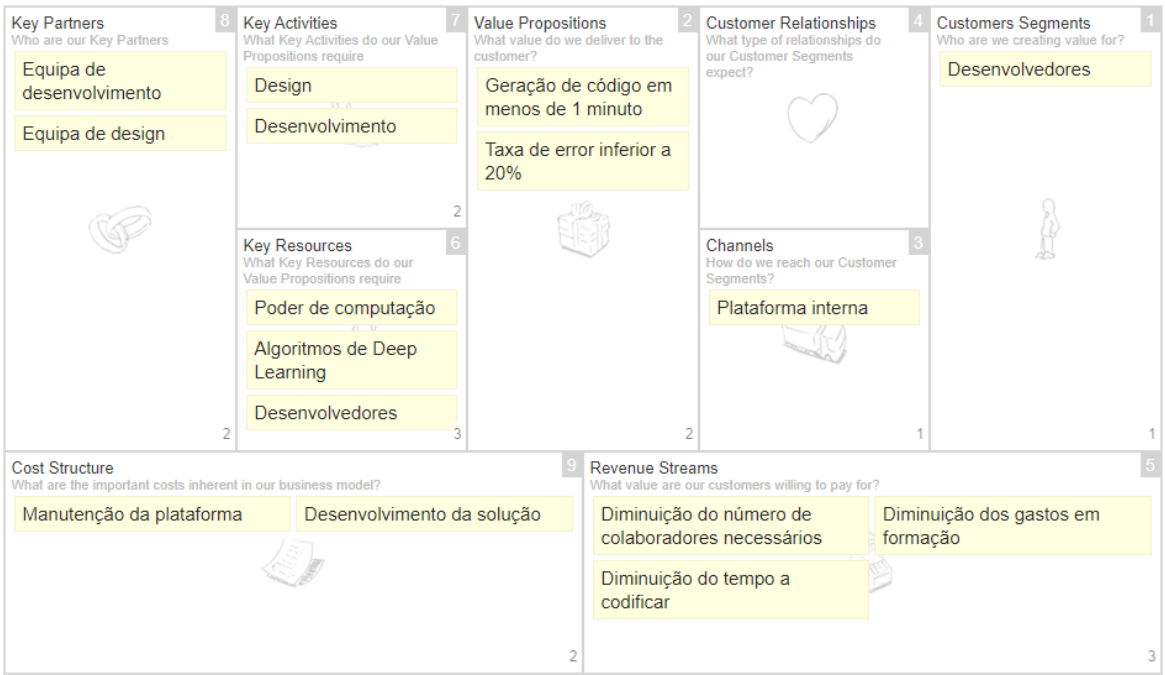


Figura 3 – Modelo de negócio de canvas

Como se pode observar na Figura 3, o modelo de canvas está dividido em nove secções, sendo elas os parceiros chave, atividades chave, recursos chave, propostas de valor, relação com os clientes, canais, segmentos de clientes, estrutura de custos e fontes de receita.

2.3.1 Parceiros chave

"Parceiros chave descrevem a rede de fornecedores e os parceiros que põem o modelo de negócios para funcionar" (Osterwalder & Pigneur, 2010).

Os parceiros chave são a equipa de desenvolvimento, porque é a equipa que irá utilizar a aplicação e que poderá dar feedback, e a equipa de design que irá ser responsável pelo desenvolvimento dos componentes que terão que ser identificados pela aplicação.

2.3.2 Atividades chave

"Atividades chave descreve as ações mais importantes que uma empresa deve realizar para fazer seu modelo de negócios funcionar" (Osterwalder & Pigneur, 2010).

A principal e única atividade deste projeto será gerar código a partir de maquetes desenvolvidas pela equipa de design.

2.3.3 Recursos chave

"Recursos principais descreve os recursos mais importantes exigidos para fazer um modelo de negócios funcionar" (Osterwalder & Pigneur, 2010).

Os recursos chave para que este modelo de negócio funcione são os desenvolvedores, pois são estes que fazem a manutenção do produto, os algoritmos de *Deep Learning* e o poder de computação que são os recursos que tornam o projeto realizável.

2.3.4 Propostas de valor

"Proposta de valor descreve o pacote de produtos e serviços que criam valor para um segmento de clientes específico" (Osterwalder & Pigneur, 2010).

Apesar de já existirem soluções semelhantes no mercado, este projeto pretende trazer uma taxa de erro inferior às demais soluções existentes, fixando a taxa de erros abaixo de 20%. Também é pretendido que o tempo de geração de código de uma maquete não seja superior a 1 minuto.

2.3.5 Relação com os clientes

"Relacionamento com os clientes descreve os tipos de relação que uma empresa estabelece com segmentos de clientes específicos" (Osterwalder & Pigneur, 2010).

Os clientes são os desenvolvedores da Glintt e não existe nenhum esforço para comunicar ou trazer mais clientes para a plataforma uma vez que é uma ferramenta interna.

2.3.6 Canais

"Os canais descrevem como uma empresa se comunica e alcança seus segmentos de clientes para entregar uma proposta de valor" (Osterwalder & Pigneur, 2010).

O principal canal de comunicação entre os clientes e o produto é a plataforma interna da Glintt, que será o local onde os desenvolvedores irão visitar usufruir das funcionalidades do produto.

2.3.7 Segmentos de clientes

"Segmentos de clientes define os diferentes grupos de pessoas ou organizações que uma empresa busca alcançar e servir" (Osterwalder & Pigneur, 2010).

Os clientes são os desenvolvedores que pretendem usar a plataforma para obter o código correspondente à maquete.

2.3.8 Estrutura de custos

"A estrutura de custo descreve todos os custos envolvidos na operação de um modelo de negócios" (Osterwalder & Pigneur, 2010).

Os principais custos associados ao projeto é o custo de desenvolvimento e o custo de manutenção.

2.3.9 Fontes de receita

"Fontes de receita representa o dinheiro que uma empresa gera a partir de cada segmento de clientes (os custos devem ser subtraídos da renda para gerar o lucro)" (Osterwalder & Pigneur, 2010).

Apesar da receita não se traduzir em aumento de lucro, pode ser traduzida na maior eficiência por parte da Glintt-HS na disponibilização das aplicações ao cliente, pois esta solução irá contribuir para a diminuição do tempo de desenvolvimento e maior disponibilidade dos

recursos existentes na Glintt-HS, uma vez que esta solução permite diminuir o tempo de implementação dos ecrãs e também a diminuição dos custos em formação.

2.4 Analytic Hierarchy Process

O método Analytic Hierarchy Process (AHP), é um método desenvolvido na década de 1970 por Thomas Saaty. Este método permite ajudar a tomar uma decisão fundamentada recorrendo a calculo matemáticos.

Para tomar uma decisão é necessário conhecer o problema, a necessidade e o propósito da decisão, o critério de decisão, as partes interessadas e as ações alternativas a tomar. Após identificar todos estes fatores pode-se então determinar a melhor alternativa (Saaty, 2008).

De modo a implementar este processo é necessário seguir 5 passos:

1. Definir o problema e os critérios.
2. Classificar as preferências entre os critérios.
3. Classificar as preferências entre subcritérios.
4. Construir um conjunto de matrizes segundo as preferências atribuídas anteriormente aos critérios e subcritérios.
5. Calcular os valores de prioridades para cada alternativa.

No contexto deste projeto, poderá ser aplicado por exemplo a escolha do algoritmo de deteção e classificação de objetos numa imagem. Começando então por definir a decisão a ser tomada que é como foi dito anteriormente "Qual o melhor algoritmo de deteção e classificação de objetos numa imagem?". Os critérios para classificar seriam por exemplo a média de tempo de deteção de objetos e a taxa de erros na classificação, como se pode verificar na Tabela 2. Depois de classificar segundo estes critérios as alternativas, como representado na Tabela 3 e na Tabela 4, são construídas as matrizes e obtidos os valores de prioridades.

Para realizar os cálculos já existem diversas ferramentas online que permitem obter as prioridades. A ferramenta utilizada para efetuar os cálculos neste caso foi a calculadora do site Business Performance Management Singapore que pode ser acedida através do link https://bpmsg.com/academic/ahp_calc.php.

Pode-se então agora classificar em termos de taxa de erros de classificação e média de tempo de deteção de objetos.

	Taxa de erros na classificação	Média de tempo de deteção de objetos
Taxa de erros na classificação	1	1/2
Média de tempo de deteção de objetos	2	1

Tabela 2 - Matriz emparelhada entre taxa de erros de classificação e a média de tempo de deteção de objetos numa imagem.

Inserindo os valores na calculadora obtemos as prioridades de 0.333 para a taxa de erros na classificação e 0.666 para a média de deteção de objetos. Com esta informação pode-se afirmar que o critério de taxa de erros na classificação é preferível à média de tempo de deteção de objetos.

As alternativas de algoritmos a usar seriam por exemplo CNN, Faster R-CNN e YOLO. Estes algoritmos serão abordados com mais detalhe na seção 3.5.1, 3.5.2 e 3.5.3 respetivamente.

	CNN	Faster R-CNN	YOLO
CNN	1	4	7
Faster R-CNN	1/4	1	4
YOLO	1/7	1/4	1

Tabela 3 - Matriz relação de taxa de erros de classificação entre as alternativas existentes.

Inserindo os valores na calculadora, pode-se verificar que a prioridade atribuída ao algoritmo CNN é de 0.696, ao Faster R-CNN 0.229 e ao YOLO 0.075 no que toca à taxa de erros de classificação.

	CNN	Faster R-CNN	YOLO
CNN	1	1/3	1/9
Faster R-CNN	3	1	1/5
YOLO	9	5	1

Tabela 4 - Matriz relação da média de tempo de deteção de objetos numa imagem.

Inserido os valores na calculadora, verificamos que o YOLO se destaca no critério de média de tempo de deteção de objetos numa imagem com a prioridade de 0.751, o segundo com melhor média é o Faster R-CNN com 0.178 e por último o CNN com 0.071.

	Taxa de erros de classificação	Média de tempo	Prioridade
CNN	0.2318	0.0473	0.2791
Faster R-CNN	0.0763	0.1186	0.1949
YOLO	0.0249	0.5001	0.525
Peso	0.333	0.666	1

Tabela 5 - Tabela de tomada de decisão

Na Tabela 5, pode-se verificar as prioridades para as alternativas existentes que respondem à questão “Qual o melhor algoritmo de deteção e classificação de imagem?”. Segundo a tabela o melhor algoritmo, de acordo com os critérios definidos como preferenciais, é o YOLO.

A avaliação relativa à taxa de erros e à média de tempo de deteção de objetos foram usados dados recolhidos de várias fontes na internet sendo que não é garantido que estes algoritmos irão ter o mesmo desempenho no neste caso específico, uma vez que o desempenho destes algoritmos é altamente dependente do volume e características dos dados.

3. Estado da Arte

As técnicas de transformação de imagens ou *mockups* em código-fonte são uma área de pesquisa muito recente, sendo que os primeiros resultados práticos foram demonstrados durante o ano de 2017. Tony Beltramelli, o fundador da empresa UIzard especializada em geração de código através de técnicas de *Machine Learning*, escreveu num dos seus artigos sobre estas técnicas: "O nosso trabalho é, no melhor de nosso conhecimento, o primeiro trabalho a tentar abordar o problema da geração de código de interface do utilizador a partir de entradas visuais (...)" (Beltramelli, 2017).

3.1 Deep Learning

Deep Learning está a despertar cada vez mais o interesse da comunidade científica, e também da indústria, sendo que atualmente já é usado numa grande variedade de produtos no mercado, como por exemplo, na tecnologia de condução autónoma dos carros e também na deteção de problemas de saúde, como cancro, ou Parkinson.

Apesar de já existirem investigações nesta área desde os anos 60, só atualmente despertou a atenção da comunidade científica e empresarial, devido ao aumento do poder de computação, uma vez que este tipo de tecnologia requer muito poder de computação e ao aumento de dados existentes.

Outra das razões pela qual este tipo de tecnologia está a ser usado em detrimento de outros algoritmos, é o facto de esta tecnologia ter um nível de performance elevado mesmo com um número elevado de dados, enquanto que outros algoritmos mais antigos começam a estagnar o seu nível de performance com o aumento do volume de dados, como é possível verificar na Figura 4.

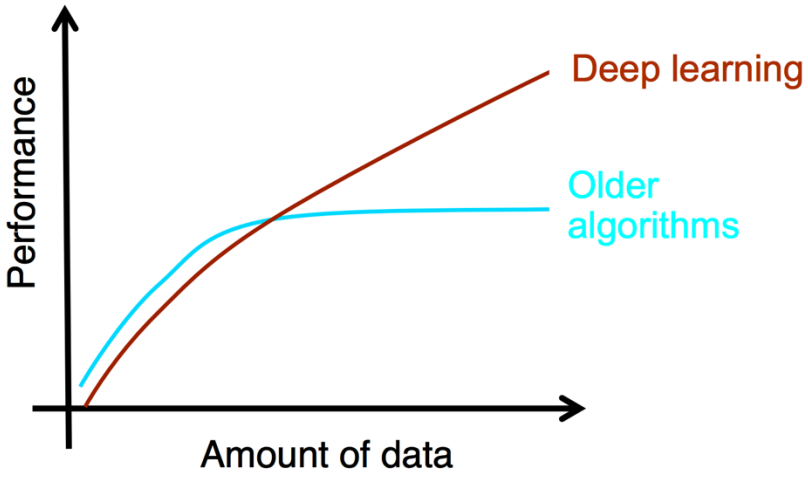


Figura 4 - Comparação da performance entre *Deep Learning* e outros algoritmos mais antigos. (Ng, 2017)

"*Deep Learning* permite modelos computacionais que são compostos de múltiplas camadas de processamento para aprender representações de dados com múltiplos níveis de abstração" (Yann LeCun et al., 2015).

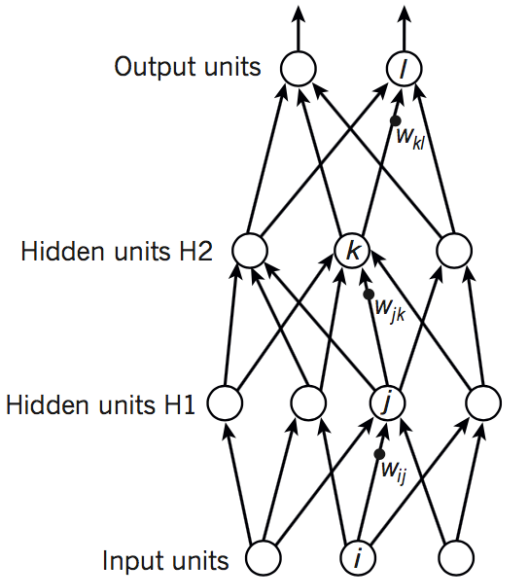


Figura 5 - Arquitetura de uma rede neuronal artificial. (Yann LeCun et al., 2015)

A Figura 5, representa a arquitetura de uma rede neural. Este sistema foi baseado no comportamento do cérebro humano, sendo que os círculos representados na figura representam os neurónios na rede, as setas são ligações entre os neurónios, uma analogia às sinapses do cérebro humano e cada conjunto de neurónios horizontal é apelidado de camada. As camadas entre a camada de entrada e a camada de saída são chamadas de camadas escondidas.

Cada neurónio representa um estado, e as ligações representam um conjunto de pesos que são ajustados durante o processo de treino, este comportamento é o que faz a rede realmente aprender a classificar corretamente os dados de input. O processo de ajustamento é feito através de um algoritmo chamado *backpropagation.* Este processo é descrito como um procedimento que repetidamente ajusta os pesos das conexões na rede de maneira a minimizar o valor da diferença entre o output atual e o output desejado (E.Rumelhart et al, 1986).

O processo de treino de uma rede neuronal pode ser feito de três maneiras, supervisionado, não supervisionado e aprendizagem reforçada. No processo de treino supervisionado é fornecido à rede um conjunto de treino acompanhado da correta previsão, que é suposto a rede aprender através da adaptação dos pesos nas ligações entre os neurónios. No processo não supervisionado é a rede que agrupa os dados em categorias segundo o calculo das distâncias do conjunto de características dos dados fornecidos. Também é possível ajustar os pesos da rede dinamicamente através de introdução no sistema de uma recompensa que permite saber que aquela previsão está correta e deve atualizar os valores, este processo é chamado aprendizagem reforçada.

3.1.1 Convolutional Neural Network – CNN

Convolutional Neural Networks é um tipo de rede de *Deep Learning* usado especialmente na área de *Computer Vision* e é inspirado no córtex visual dos animais.

Uma imagem é representada pelo conjunto de valores RGB em cada pixel, assim sendo uma imagem de 22 pixéis de altura e 22 pixéis de largura irá ter 22x22x3 valores. Durante o processo de convolução é aplicado um filtro à matriz representativa da imagem que irá procurar por um certo padrão e originar uma nova matriz de valores, de dimensão menor que a matriz original, a cada passo é criado um nível de abstração da imagem, até um ponto em que resta apenas um valor.

Após o processo de convolução é também aplicado o processo de *pooling* que é semelhante ao processo de convolução, no entanto em vez de procurar por um padrão, é procurado o maior valor através de uma mascara na matriz.

Os processos descritos anteriormente podem ser observados na Figura 6.

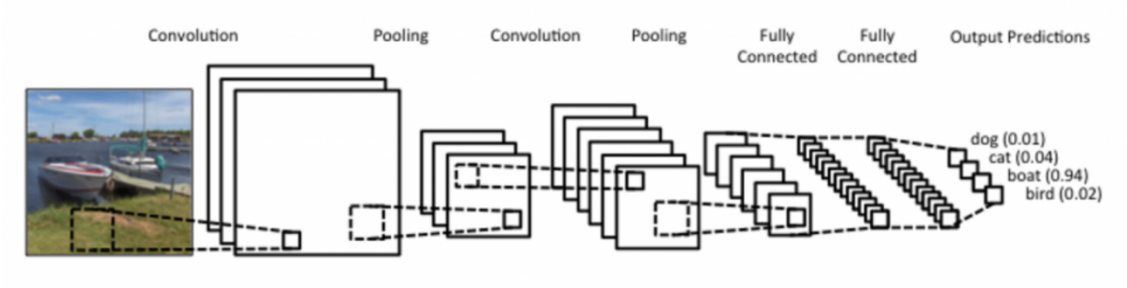


Figura 6 - Arquitetura de uma Convolutional Neural Network (Britz, 2015)

Em 2012, a rede AlexNet (Krizhevsky et al., 2012) foi a grande surpresa no concurso de avaliação de performance de algoritmos de reconhecimento de imagens, ao vencer todos os outros algoritmos por uma larga maioria e reduzindo o erro do ano anterior em mais de 10%. Desde então este tipo de rede tem sido o centro das atenções no reconhecimento de imagem, tendo surgido outras redes baseadas nesta inicial e com o objetivo de diminuir o tempo de classificação. Alguns exemplos das novas redes criadas são: Regional CNN (Girshick et al., 2013), Fast R-CNN (Girshick, 2015), Faster R-CNN (Ren et al, 2015) e Mask R-CNN (He et al., 2017).

3.1.2 Faster Region-based Convolutional Neural Network – Faster RCNN

Faster Region-based Convolutional Neural Network, é um tipo de CNN aperfeiçoada de modo a conseguir fazer a sua previsão num espaço de tempo mais reduzido.

A Regional Proposal Network (RPN) distingue-se das outras redes CNN uma vez que compartilha recursos convolucionais da imagem com a rede de deteção, permitindo propostas de região quase sem custos. Um RPN é uma rede totalmente convolutiva que, simultaneamente, prevê limites de objeto e pontuação de objeto em cada posição (Ren et al., 2015).

Durante o processo de deteção de objetos na imagem, em vez de procurar exaustivamente em todos os pixéis por objetos, este algoritmo define primeiro as áreas de interesse na imagem e depois aplica o classificador a essas áreas. Este fluxo pode ser visualizado na Figura 7.

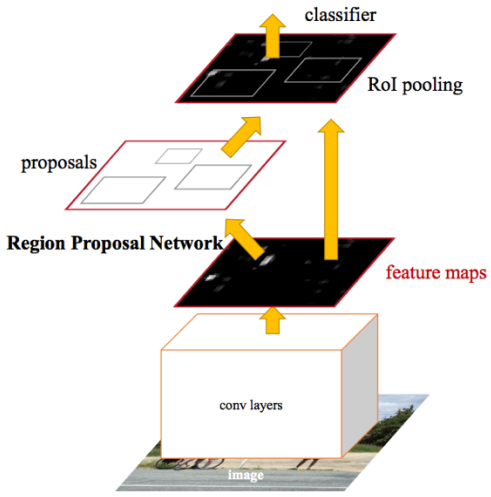


Figura 7 - Arquitetura de uma Faster R-CNN (Ren et al., 2016).

3.1.3 You Only Look Once – YOLO

You Only Look Once (Redmon et al., 2016) é uma rede neuronal focada em reduzir o espaço temporal, sendo capaz de detetar objetos em tempo real, como por exemplo identificar objetos durante a visualização de um vídeo.

Esta rede é composta apenas por uma rede CNN que simultaneamente prediz um conjunto áreas candidatas a um objeto e a probabilidade de uma certa classe para cada uma das áreas (Redmon et al., 2016). Apesar de identificar rapidamente objetos nas imagens, este algoritmo tem a sua eficácia reduzida quando se trata de objetos pequenos.

Durante o processo de deteção e classificação de objetos o algoritmo divide a imagem a ser classificada numa matriz SxS, como se pode verificar na Figura 8. No caso de o centro do objeto ficar numa das posições da matriz, essa mesma posição é responsável por detetar o objeto em questão. O mesmo é aplicado para cada posição da matriz que é responsável por detetar as áreas envolventes e a probabilidade de uma certa classe estar nessa área (Redmon et al., 2016).

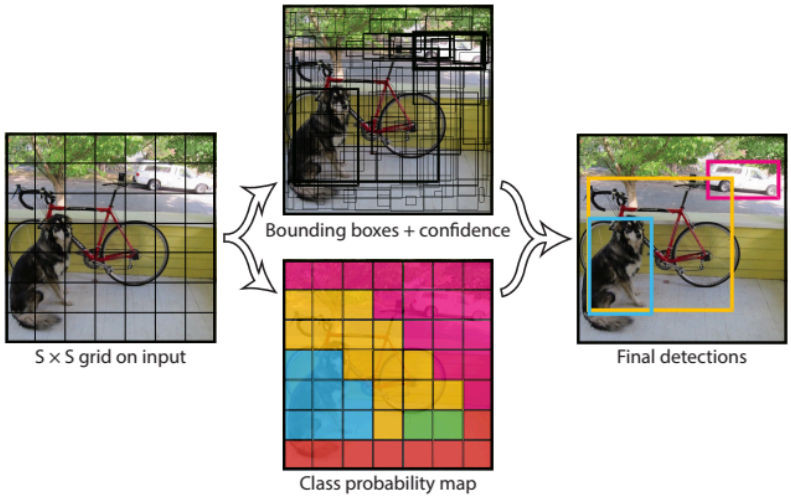


Figura 8 - Processo de deteção de objeto usando YOLO (Redmon et al., 2016).

Este algoritmo apesar de ser muito rápido na classificação dos objetos, não consegue identificar bem os detalhes das imagens, isto acontece porque durante a classificação as posições da matriz apenas procuram por uma classe, mas se nesse espaço da imagem existir mais do que um objeto, então apenas um deles será classificado.

3.1.4 Recurrent Neural Networks - RNN

Ao contrário das outras redes neuronais, as Recurrent Neural Networks, não se limitam a classificar ou prever os eventos atuais, estas redes têm um sistema de "memória" que permite relacionar os eventos do passado com os eventos atuais (Britz, 2015).

Este tipo de rede é muito usado no reconhecimento de discurso e reprodução automática de texto, porque permite estabelecer uma probabilidade de aparecer a próxima palavra baseada nas palavras anteriores.

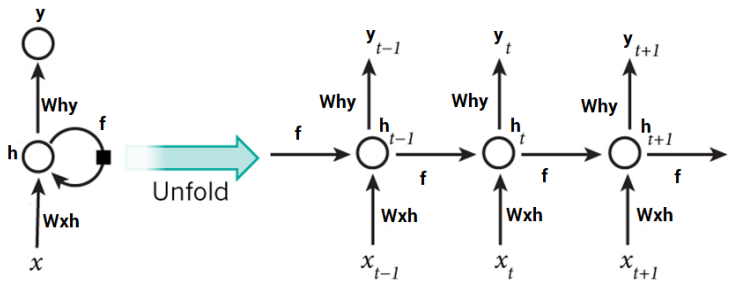


Figura 9 - Desdobramento de uma Recurrent Neural Network (Britz, 2015).

A Figura 9 mostra o desdobramento de uma Recurrent Neural Network, ou seja, mostra como é feita a apresentação daquilo que foi guardado na rede durante o processo de previsão. Por exemplo imagine-se que esta rede foi submetida a uma imagem, em que foi possível observar

vários objetos classificados, como por exemplo uma menina, uma bola. No desdobramento da rede uma possível descrição textual seria a frase uma "menina joga com a bola".

3.1.5 Long Short Term Memory - LSTM

Long Short Term Memory é um tipo de RNN desenvolvida pela necessidade das RNN guardarem informação com grande espaço temporal. Por exemplo para uma RNN adivinhar a palavra no final da seguinte frase "as nuvens estão no céu", será uma tarefa muito fácil, uma vez que o contexto necessário para predizer a palavra "céu" está muito próximo (Olah, 2015).

No entanto se o contexto estiver muito distante da palavra a ser predita, então as redes RNN normais não conseguem conectar a informação. Como por exemplo, "Eu sou português, falo muitas línguas entre elas ..., no entanto, a minha língua nativa é o português". Neste caso para uma RNN seria mais desafiante porque o contexto que permite dizer que a língua nativa é o português, está distante.

3.2 Abordagens existentes

Nesta secção serão apresentados alguns projetos que partilham alguns objetivos com o projeto a ser desenvolvido. A compreensão destes projetos é fundamental para perceber de que forma abordaram os problemas que tinham para resolver e de que forma se pode adaptar a resolução desses mesmos problemas ao contexto do projeto a ser desenvolvido.

3.2.1 pix2code

Este projeto foi realizado por Tony Beltramelli no ano de 2017, e foi pioneiro na área de automação da transformação de *mockups* em código HTML/CSS. O modelo treinado gerou código para várias plataformas como por exemplo, iOS, Android e Web com eficácia de 77%.

O modelo desenvolvido por Beltramelli, e publicado no artigo científico que descreve o seu projeto, é assente em dois principais componentes. O primeiro componente é uma Convolution Neural Network (CNN) responsável por mapear a imagem de input em elementos reconhecidos pelo modelo, o segundo componente é uma Recurrent Neural Network (RNN) cuja responsabilidade é associar uma descrição textual aos elementos identificados. A arquitetura deste projeto pode ser observada na Figura 10.

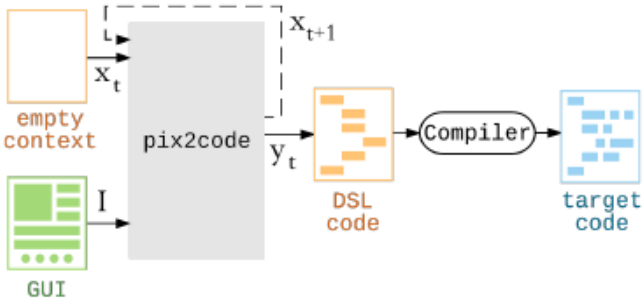


Figura 10 - pix2code e os seus artefactos de input e output.

3.2.2 sketch2code

No ano de 2017, também a empresa norte-americana Airbnb, decidiu modificar a maneira como o design é transformado em código fonte. No entanto não foi publicado nenhum artigo ou outro tipo de comunicação sobre a arquitetura e implementação desta tecnologia.

Esta aplicação é capaz de interpretar os componentes desenhados à mão (*sketch*), ou seja, sem um nível de detalhe muito elevado. Quando esta solução recebe um retângulo com um "X" no meio como input, sabe que tem de usar um componente de imagem.

Outro aspeto interessante é que cada símbolo corresponde a um componente React, e por isso o código é gerado em HTML/CSS e Javascript.

3.3 Tecnologia relevante

Nesta secção será apresentada ao leitor o contexto histórico das linguagens e *frameworks* que são de relevância para este projeto e que são muito utilizadas pela comunidade para resolver problemas semelhantes ao que se está a tentar resolver.

Algumas das tecnologias aqui apresentadas foram usadas para implementar as soluções apresentadas no capítulo anterior.

3.3.1 R

A linguagem de programação R, foi inicialmente escrita por Ross Ihaka e Robert Gentleman em 1993 e foi projeta para trabalhar com dados estatísticos e elaboração de gráficos. Esta linguagem tem uma grande influência das linguagens S e a Scheme (Hornik, 2017).

Em janeiro de 2018 a linguagem R estava na posição 8 do ranking de linguagens de programação da TIOBE, sendo que no período homólogo de 2017, estava na posição 18, o que significa que a linguagem tem tido muito aderentes (TIOBE, 2018). Esta subida de classificação

também pode ser explicada pelo facto do aumento acentuado do interesse na área *Machine Learning* e *Data Mining*.

3.3.2 Python

A linguagem de programação Python, foi criada em 1991 por Guido Van Rossum. É uma linguagem de alto nível e de propósito geral e que foi criada com o objetivo de melhorar a interpretação do código pelos programadores e com uma sintaxe mais compacta.

Segundo o ranking de linguagens de programação TIOBE, a linguagem Python estava na posição 4, no mês de janeiro de 2018 sendo que no período homólogo estava na posição 5 (TIOBE, 2018).

3.3.3 Frameworks

Nos últimos anos tem vindo a surgir cada vez mais bibliotecas de *Machine Learning*, sendo que as grandes empresas de software internacionais, como a Google e a Microsoft, estão constantemente a competir para criar a sua biblioteca de *Machine Learning*.

Com toda estas variedade de bibliotecas, a questão que se coloca é: "Qual biblioteca é mais adequada a este trabalho?".

Nas secções seguintes será apresentada o atual estado da arte em *frameworks* de *Machine Learning*, sobretudo para a linguagem Python e também apresentada a *framework* interna da Glintt-HS, o Glinttology.

3.3.3.1 Tensorflow

Tensorflow é uma biblioteca *open-source* criado pela Google, lançada no fim do ano de 2015. Esta biblioteca foi inicialmente escrita em C++, no entanto oferece API's nativas para Python, Java e GO (Tensorflow, 2017).

O Tensorflow é atualmente a maior biblioteca no Github com cerca de 88 mil estrelas.

3.3.3.2 Theano

Theano é uma biblioteca open-source desenvolvida pela Universidade de Montreal, no Canada, e foi uma das primeiras bibliotecas de machine learning, sendo primeira versão lançada no ano de 2010.

No entanto, no final do ano de 2017 os responsáveis pelo projeto anunciaram que não seriam implementadas novas funcionalidades e apenas irão garantir o suporte mínimo do projeto (Peng, 2017).

3.3.3.3 Keras

Keras é uma biblioteca de alto nível e também é open-source. Foi desenvolvida pelo engenheiro da Google, François Chollet, e foi publicada pela primeira vez no inicio do ano 2015.

No entanto, esta biblioteca tem como principal objetivo abstrair pormenores de implementação de outras bibliotecas em que esta se pode basear. É possível usar o Keras sob as bibliotecas, Tensorflow, Theano e CNTK, que é a biblioteca de machine learning da Microsoft. Atualmente é possível programar em Keras em Python ou em R (Keras, 2018).

Tensorflow traz, atualmente, uma implementação do Keras o que torna possível usar o Keras a partir do Tensorflow. Isto permite trabalhar em alto nível, com o Keras, em alguns casos e ter a flexibilidade para fazer implementações mais especificas usando o Tensorflow.

Keras é o segundo projeto com mais contribuidores de machine learning no Github com 25 mil estrelas.

3.3.3.4 Glinttology

O Glinttology é uma *framework* interna da Glintt que consiste numa palete com todos os componentes existentes nas aplicações. Esta solução nasceu da necessidade de criar um guia visual, que conseguisse colocar todas as partes envolvidas nos projetos, tais como clientes, desenvolvedores, equipa de design e consultores, a comunicar de igual forma, ou seja, desta forma quando uma das partes pretende colocar um botão de erro na página, todas as partes conseguem visualizar o resultado final no ecrã. Na Figura 11, pode-se observar um exemplo das diferentes cores que os botões podem assumir nas aplicações.

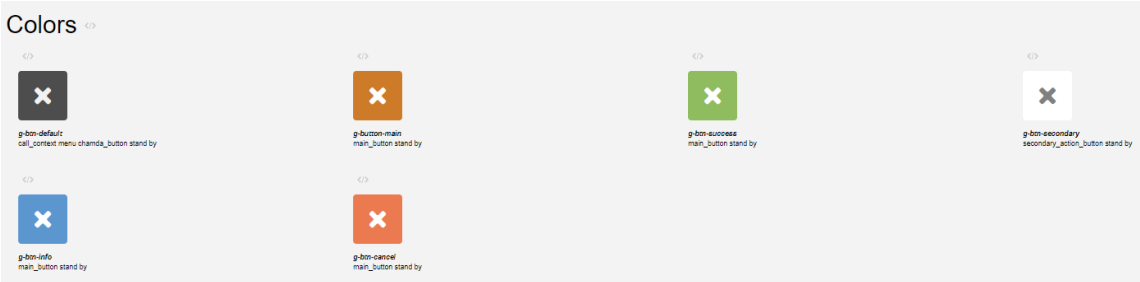


Figura 11 – Representação da secção de botões no Glinttology

Esta *framework* permite também que o design das aplicações seja mais consistente, uma vez que um componente é implementado exatamente da mesma maneira em todas os produtos da Glintt.

4. Avaliação das soluções existentes

Neste capítulo serão compardas as soluções apresentadas no capítulo anterior e apresentar que aspetos das suas arquiteturas podem ser adaptados ao projeto a ser desenvolvido e também que características destas soluções gostaríamos que colocar neste projeto.

Também iremos comparar as tecnologias apresentadas na secção 3.3 do capítulo anterior classificando segundo a adequação ao projeto a ser desenvolvido e segundo a facilidade de implementação e pesquisa de informação, definido assim a nossa *stack* tecnológica para o desenvolvimento do projeto.

4.1 Avaliação das soluções existentes

Na secção 3.2 deste documento será apresentada uma breve introdução das soluções existentes no mercado que se propõe a resolver problemas semelhantes ao do projeto.

O projeto pix2code está muito bem documentado tendo inclusive descritos muitos pormenores de implementação como por exemplo os tipos de redes neuronais utilizadas e a forma como é gerado código após a interpretação da maquete.

No entanto, o projeto sketch2code da Airbnb não tem nenhuma documentação relativa à arquitetura nem aos algoritmos usados, de modo que será descartada a parte técnica relacionada com este projeto.

4.1.1 pix2code

Como se pode observar na Figura 12, este projeto usa três redes neuronais, sendo que duas delas são LSTM e a outra é uma CNN.

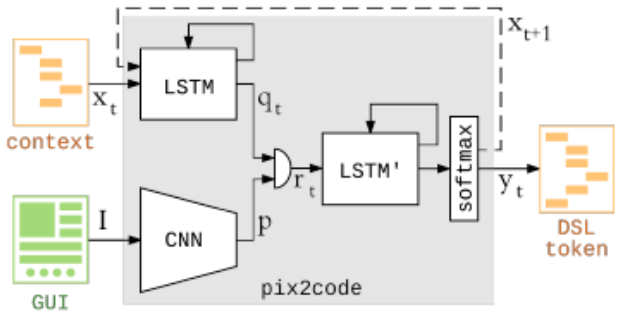


Figura 12 – Arquitetura pix2code.

Na preparação das imagens para introduzir na CNN deste projeto, as imagens foram redimensionadas para a 256 x 256 pixéis, a sua proporção não foi preservada e os valores dos

pixéis foram normalizados. Esta CNN é constituída por 3 camadas, a primeira camada tem 32 pixéis de largura, a segunda tem 64 e a terceira tem 128.

A primeira rede LSTM é constituída por duas camadas de 128 células cada uma. É responsável por transformar a DSL num vetor de valor intermédios. A segunda rede LSTM também apelidada no projeto como "*Decoder*" é constituída por duas camadas de 512 células e o seu objetivo é aprender a relacionar o vetor de valor intermédios gerado pela primeira rede LSTM e os objetos detetados pela rede CNN.

Para treinar a rede neuronal é dado como input uma imagem da maquete representada na Figura 12 como "GUI" e um ficheiro com a descrição dos componentes que estão presentes na imagem e está representado na Figura 12 como "contexto", escrita na DSL própria deste problema. Na Figura 13 é possível observar um exemplo dos dados de entrada usados para treinar o modelo.


```
stack {
   row {
      label, switch
   }
   row {
      label, btn-add
   }
   row {
      label, slider, label
   }
   row {
      img, label
   }
}
footer {
   btn-more, btn-contact, btn-search, btn-download
}
```

Figura 13 – Dados de input do projeto pix2code, do lado esquerdo a maquete e do lado direito a DSL correspondente à maquete.

Após o treino do sistema, a rede começa a converter as maquetes em ficheiros de texto com base na DSL, no entanto, o código HTML ainda não é gerado. Para obter o código é usado um mecanismo de compilação que trata de ler o ficheiro de *ouput* e transformá-lo em código, com base num dicionário em que cada palavra da DSL tem associado o código correspondente.

4.2 Avaliação das tecnologias existentes

Nesta secção serão confrontadas as tecnologias já enunciadas anteriormente na secção 3.1 e 3.2. A escolha das ferramentas adequadas para a execução do trabalho são um passo fundamental.

De modo a facilitar a implementação desta solução procuramos por algumas características que nos permitam fazer um desenvolvimento rápido e adequado, tais como:

- **Familiarização com a tecnologia:** é importante que haja experiência prévia com a tecnologia, para ser mais rápido começar a desenvolver e reduzir o tempo de aprendizagem.
- **Suporte, Comunidade e Documentação:** este fator é um dos mais importantes na escolha de uma tecnologia, uma vez que uma boa documentação e suporte técnico poupa muito tempo na pesquisa de informação.
- **Licenciamento:** o licenciamento de uma ferramenta é importante uma vez que pode influenciar na forma como o produto é vendido ou distribuído.
- **Adequação da tecnologia ao trabalho em questão:** A tecnologia deve responder às necessidades de desenvolvimento, não seria adequado usar neste projeto uma tecnologia que não foi projetada para *machine learning*.

4.2.1 R vs Python

Atualmente, R e Python são as linguagens de programação mais populares na área de estudos de machine learning. A Figura 14 mostra a popularidade pesquisas "python machine learning" a azul e "R machine learning", a vermelho, na pesquisa do Google no período de 19 de fevereiro de 2017 até 19 de fevereiro de 2018. Como se pode verificar a linguagem Python é mais popular e tem vindo até a ganhar popularidade relativamente à linguagem R que vai caindo gradualmente ao longo do tempo.

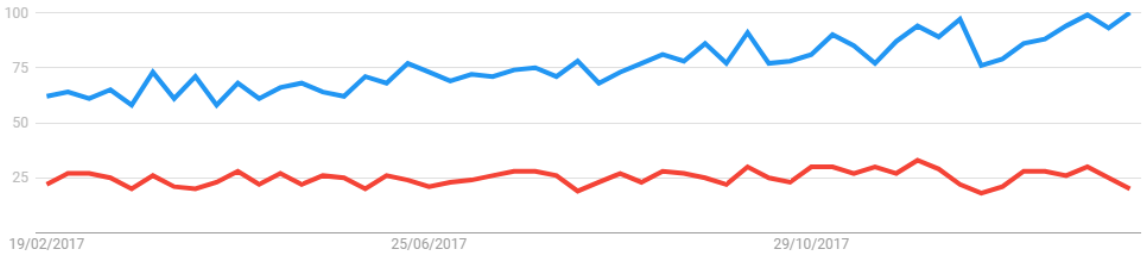


Figura 14 – Comparação das pesquisas entre "python machine learning" e "R machine learning". Fonte: Google Trends

	Linguagem R	**Linguagem Python**
Familiarização	3 meses de experiência Curva de aprendizagem muito inclinada	Nenhuma experiência Fácil de aprender
Suporte, Comunidade e Documentação	Stackoverflow Rdocumentation R-project	Stackoverflow Python doc Devdocs
Licenciamento	GPL	GPL
Adequação da tecnologia	Adequado para estatística e visualização de dados	Multipropósito Adequada para *machine learning.* Muitas *frameworks* de *machine learning.*

Tabela 6 – Comparação entre as linguagens R e Python.

Na Tabela 6 pode-se observar a comparação entre as linguagens R e Python mediante os parâmetros definidos na introdução da secção 4.2. Em termos de licenciamento as duas linguagens têm o mesmo tipo de licença e no fator de suporte e documentação as duas são muito bem documentadas e com várias comunidades dedicadas a estas linguagens.

O que as distingue é o fato que a linguagem Python está mais ligada à área de *machine learning* uma vez que a grande maioria de *frameworks* de machine learing que existem hoje em dia têm interface com Python, mas só algumas têm para R.

Outro fator a ter em conta é o que a curva de aprendizagem de Python é menor do que a da linguagem R, o que quer dizer que a longo prazo, a aprendizagem de Python irá por compensar relativamente à experiência prévia com a linguagem R.

Pelas razões enunciadas anteriormente este projeto será desenvolvido com recurso à linguagem Python.

4.2.2 Tensorflow vs Theano vs Keras

Na secção 4.2.1 decidiu-se que a linguagem mais adequada a este projeto seria Python. Tendo em conta que se irá usar Python é necessário usar uma biblioteca de *machine learning* compatível com esta linguagem.

Atualmente as bibliotecas mais populares para trabalhar na área de *machine learning* são Tensorflow, Theano e Keras.

Library	Rank	Overall	Github	Stack Overflow	Google Results
tensorflow	1	10.87	4.25	4.37	2.24
keras	2	1.93	0.61	0.83	0.48
caffe	3	1.86	1.00	0.30	0.55
theano	4	0.76	-0.16	0.36	0.55
pytorch	5	0.48	-0.20	-0.30	0.98
sonnet	6	0.43	-0.33	-0.36	1.12
mxnet	7	0.10	0.12	-0.31	0.28
torch	8	0.01	-0.15	-0.01	0.17
cntk	9	-0.02	0.10	-0.28	0.17
dlib	10	-0.60	-0.40	-0.22	0.02

Figura 15 – Ranking de popularidade das frameworks de machine learning em Python.
Fonte: Github

Como se pode observar na figura Figura 15, a framework mais popular de todas é, por uma margem muito grande, a *framework* da Google, Tensorflow. Apesar de existir uma *framework* chamada caffe no top 3, essa *framework* não será considerada visto que foi lançada uma nova versão recente, caffe2, e é expectável que com o tempo a sua popularidade venha a descer.

	Keras	**Tensorflow**	**Theano**
Familiarização	Nenhuma experiência	Nenhuma experiência	Nenhuma experiência
Suporte, Comunidade e Documentação	26 mil estrelas no Github. Keras.io.	90 mil estrelas no Github. Tensorflow.org.	Já não existe suporte oficial. 8 mil estrelas no Github. Não tem documentação oficial.
Licenciamento	MIT	Apache 2.0	BSD
Adequação da tecnologia	Convolutional Neural Networks. Recurrent Neural Networks.	Convolutional Neural Networks. Recurrent Neural Networks.	Convolutional Neural Networks. Recurrent Neural Networks.

Tabela 7 - Comparação das frameworks Keras, Tensorflow e Theano

Como se pode ver na Tabela 7, a experiência com a tecnologia não é um fator a ter em conta uma vez que não existe familiarização com nenhuma das ferramentas. No caso da adequação da tecnologia, também verificamos que todas as *frameworks* são competentes e fornecem as ferramentas necessárias ao desenvolvimento do trabalho. O fator licenciamento também não será um problema uma vez que nenhuma delas tem uma licença comercial que implique gastos monetários, o que poderia ser um problema.

O fator que distingue estas *frameworks* é o suporte, comunidade e documentação. Como se pode observar o Keras e o Tensorflow estão à frente do Theano, uma vez que esta *framework*

foi abandonada pelos fundadores e o interesse em desenvolver nesta ferramenta tem caído ao longo do tempo.

O Keras é diferente do Tensorflow, sendo que o objetivo do Keras é diminuir a complexidade da implementação de estruturas complexas como redes neuronais. O Keras não suporta nativamente as operações necessárias à manipulação das estruturas de *deep learning* e por isso é considerada uma abstração de outra framework compatível, como por exemplo Tensorflow e Theano.

Com base nos fatores enunciados anteriormente para o desenvolvimento iremos usar Keras e Tensorflow, uma vez que usar o Keras permite diminuir o tempo de aprendizagem e abstrair os pormenores de implementação de redes neuronais.

4.2.3 Faster R-CNN vs YOLO

Na secção 3.1 apresentamos dois algoritmos de deteção de imagem baseados em Convolutional Neural Networks. Aqui iremos confrontar as duas tecnologias em termos de precisão e de velocidade, e consoante o desempenho destas ferramentas escolher uma para usar no sistema de deteção de objetos a desenvolver.

Figura 16 – Comparação entre Faster RCNN e YOLO em termos de precisão e velocidade.
Fonte: CV-Tricks.com

Como se pode observar na Figura 16, o algoritmo Faster RCNN é mais preciso do que o YOLO apesar da diferença não ser significativa comparativamente com a diferença de velocidade entre estes dois algoritmos. Como já explicado na secção 2.4, a velocidade tem um fator mais importante que o fator precisão uma vez que os objetos a serem detetados não têm muitos pormenores que façam com que o seu reconhecimento seja difícil, ao contrário de uma árvore ou animais que têm muitos pormenores o que obriga a um sistema com maior precisão.

Pelas razões enunciadas anteriormente, o sistema será implementado com recurso ao algoritmo de deteção de objetos YOLO.

5. Design

Neste capítulo serão apresentadas duas arquiteturas, sendo que apenas uma delas será adotada para o desenvolvimento da plataforma, explicando a razão pela qual uma delas será a escolhida.

Também será apresentada uma arquitetura de granularidade baixa, relativamente aos componentes que fazem parte da solução de *Deep Learning*.

5.1.1 Arquitetura

Nesta secção serão apresentadas as duas arquiteturas consideradas para o desenvolvimento do projeto, sendo elas a arquitetura Cliente-Servidor e *Console Application.* Serão apresentadas as vantagens e as desvantagens de cada uma e posteriormente serão tiradas conclusões segundo a adequação de cada uma às necessidades da Glintt-HS.

5.1.2 Console Application

Uma das alternativas para implementar esta solução é desenvolver uma *console application* em que a aplicação corre no terminal do utilizador.

Este tipo de arquitetura é ideal para o desenvolvimento de protótipos, uma vez que o desenvolvimento da aplicação é mais rápido. No entanto como solução final não é adequado uma vez que seria necessário instalar novas versões em todos os computadores que iriam usar a aplicação sempre que houvesse uma atualização.

No entanto este tipo de aplicação tem uma performance superior às aplicações cliente-servidor, e em *Machine Learning* a performance é um aspeto importante a considerar.

5.1.3 Cliente-Servidor

Numa aplicação Cliente-Servidor a aplicação divide o processamento entre o lado do cliente, que normalmente corre no browser, e o lado do servidor.

Este tipo de arquitetura permite um acesso mais fácil, sendo que pode ser acedido em qualquer computador, não sendo necessário estar no computador onde o programa está instalado como no caso da *console application*.

No que diz respeito à performance este tipo de arquitetura não é a melhor opção uma vez que os dados são transmitidos através da rede.

5.1.4 Conclusões

Este projeto deve ser capaz de enviar uma resposta rápida, tal como vimos na secção 1.3, e também deve ser acessível por todos os colaboradores da Glintt-HS para que possam converter as maquetes que lhes são entregues com a última versão existente do projeto.

Primeiro será implementada uma *console application* de modo a ter o projeto funcional no menor espaço de tempo possível. Após a avaliação da solução implementada será colocado o projeto em produção através de uma arquitetura cliente-servidor para que fique disponível para todos os utilizadores na Glintt-HS.

Na Figura 17 está representado como poderá ser feita a instalação do projeto usando uma arquitetura cliente-servidor.

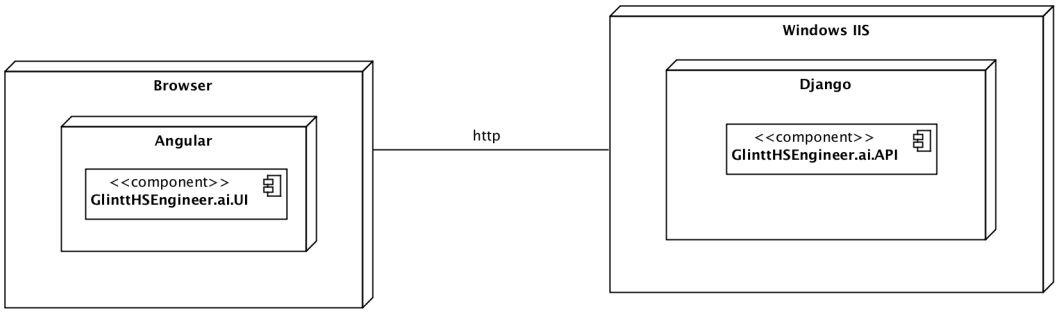


Figura 17 – Diagrama de implantação do projeto

5.2 Arquitetura Pix2Code

Pix2Code foi um dos primeiros projetos a fazer a conversão de maquete para código, segundo o autor, Tony Beltramelli. Como se pode ver na Figura 18, a arquitetura é composta por três componentes principais, o "Image Model", o "Language Model" e o "Decoder". O "Image Model" tem como responsabilidade receber como input uma imagem e enviar as características dessa imagem como input para o "Decoder". Sendo assim, o componente "Image Model" é uma CNN dentro da arquitetura pix2code. O "Language Model" tem como responsabilidade receber a descrição textual da imagem no momento e prever qual será o próximo token na sequência. O "Decoder" depois relaciona as *features* provenientes do "Image Model" e da "Language Model" e posteriormente irá calcular qual dos tokens presentes no dicionário será o mais acertado, através da função de softmax.

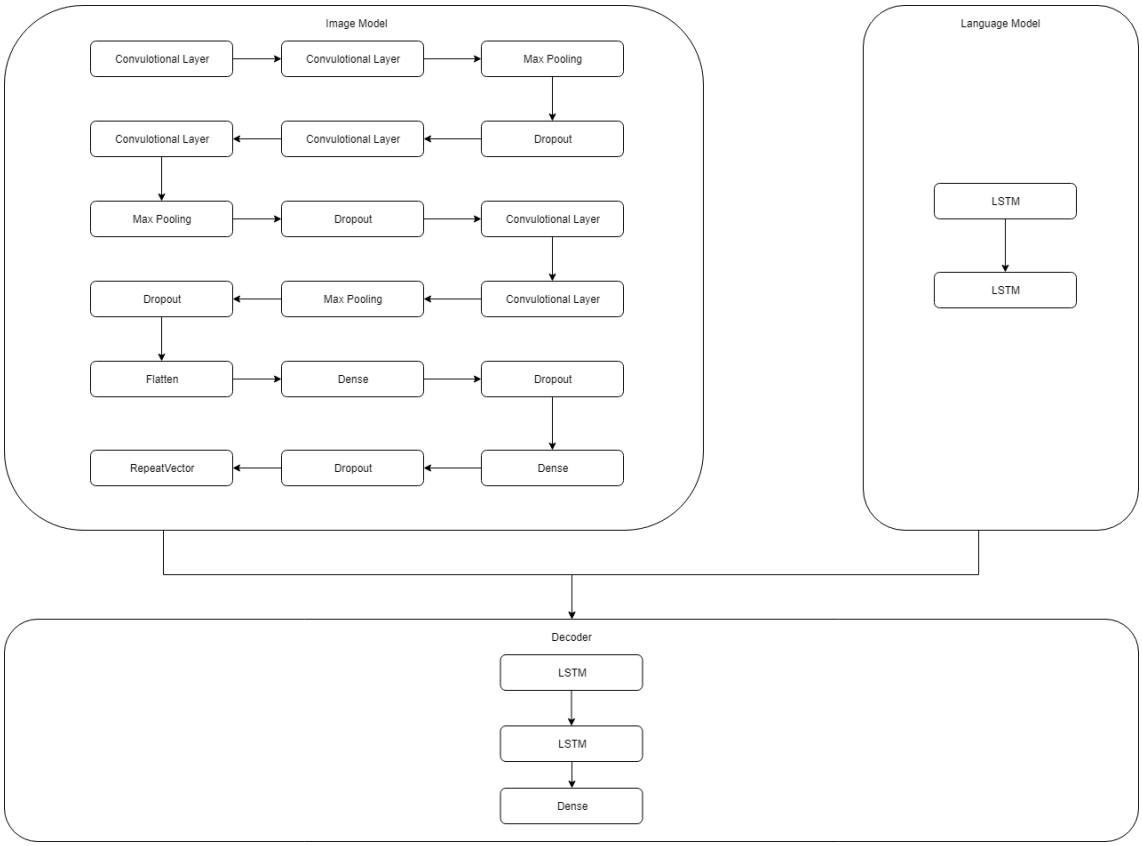


Figura 18 – Arquitetura pix2code (Beltramelli, 2017)

5.3 Arquiteturas Screenshot2code

Após a publicação de Tony Beltramelli, outros investigadores começaram a replicar o projeto e a tentar melhorá-lo, como foi o caso de Emil Wallner. Emil Wallner, no entanto, sugeriu duas arquiteturas para resolver este problema (Wallner, 2018). Em seguida serão brevemente descritas cada uma destas arquiteturas uma vez que o design final será influenciado por estas arquiteturas.

5.3.1 Versão HTML

Na versão HTML, Emil Wallner construiu um modelo em que dado uma imagem como input é dado como output imediatamente o código HTML. Isto difere da solução apresentada por Beltramelli no sentido em que deixa de existir o passo intermédio da construção de uma DSL e a posterior compilação para código HTML.

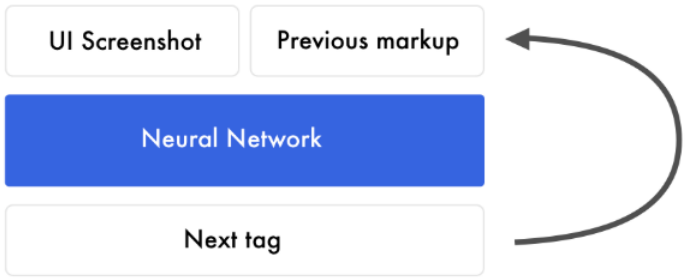


Figura 19 – Inputs e outputs da versão HTML de Wallner (Wallner, 2018).

No entanto, esta solução apesar de eliminar um dos processos, tem uma pior performance uma vez que o espaço de pesquisa de palavras é maior, pois tem de reconhecer, tags HTML, atributos, etc.

5.3.2 Versão Bootstrap

Para esta versão foram usadas diversas imagens de componentes de Bootstrap como input na rede, daí o seu nome. Esta solução é praticamente igual à de Beltramelli, no entanto tem uma ligeira mudança na sua arquitetura que se mostrou ser bastante interessante.

Como foi apresentado, a arquitetura pix2code é composta por uma parte denominada "Image Model" que é basicamente uma CNN, no entanto, o input dessa CNN até que a sequência seja toda encontrada, é sempre a mesma imagem, o que se traduz na degradação de performance do modelo. A solução encontrada por Wallner foi usar uma CNN genérica já treinada e usar as *features* extraídas nessa rede como input para a rede. Sendo assim, e modificado o diagrama da Figura 19, o diagrama para esta solução será algo como o apresentado na Figura 20.

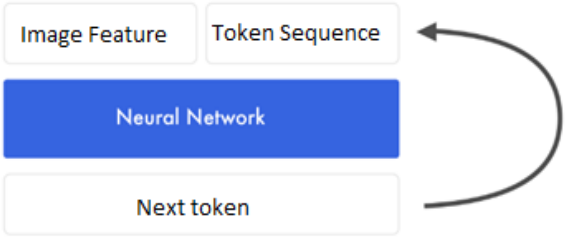


Figura 20 – Input e outputs da versão Bootstrap de Wallner.

Wallner após a construção desta solução comparou a sua accuracy com a que foi obtida através da arquitetura de Beltramelli e constatou que uma abordagem personalizada como a do pix2code obteve melhores resultados, falando até uma melhoria de 30% (Wallner, 2018).

5.4 Arquitetura Developer AI

Nesta secção será descrito como a solução proposta foi desenhada de forma a conseguir retirar o melhor de cada uma das soluções apresentadas nas secções 5.2 e 5.3. Primeiro será apresentado um diagrama de alta granularidade para descrever as partes constituintes do modelo desenvolvido, tal como foi apresentado anteriormente, depois serão apresentados mais aprofundadamente cada um dos componentes e justificadas cada uma das decisões.

Como se pode ver na Figura 21, a arquitetura proposta é em tudo muito semelhante às arquiteturas apresentadas.

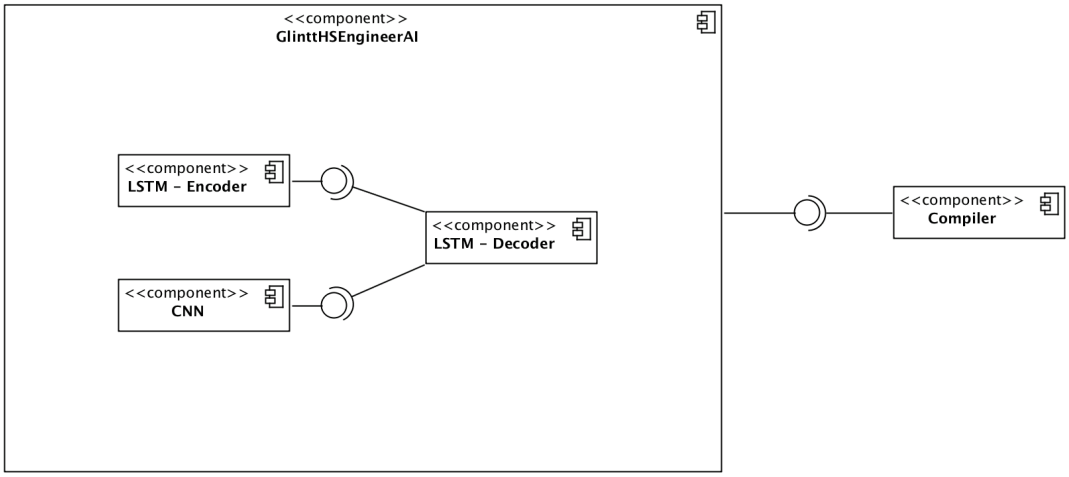


Figura 21 – Componentes constituintes da arquitetura final da solução

Tal como na versão *bootstrap* da arquitetura de screenshot2code, será usada uma CNN já treinada e genérica para que o modelo a desenvolver seja mais rápido de treinar, mesmo sabendo que esta decisão irá implicar uma redução de performance de cerca de 30% (Wallner, 2018). Esta opção prende-se com o fato de o hardware disponível para o desenvolvimento da solução não ter capacidade para treinar uma rede neuronal com um elevado número de nós. Relativamente às camadas de LSTM será adotado o mesmo esquema que foi usado pelas duas arquiteturas anteriores, no entanto, com o aperfeiçoamento dos *hyperparameters* é expectável que o número de camadas e o número de nós mude relativamente ao que será usado inicialmente.

6. Construção

Nesta secção serão apresentados os passos que envolveram a realização desta solução. Primeiro será descrito onde e como foram recolhidos os dados para alimentar o modelo, depois será mostrado de que forma os dados recolhidos foram manipulados para puderem ser inseridos no modelo. Após a recolha e processamento dos dados será construído o modelo, recorrendo a técnicas de *machine learning*, nomeadamente *deep learning*, modelo esse que será treinado dando como input uma parte dos dados e fazendo uma posterior validação do modelo usando uma outra parte dos dados. Finalmente será avaliada a performance do modelo, e ajustados os parâmetros, tais como o número de camadas da rede neuronal, o número de nós na rede, entre outros parâmetros, e repetidamente voltar a treinar a rede até se atingir uma performance ideal de modo a colocar o modelo em produção.

6.1 Hardware

Uma das partes principais da construção de um modelo de *deep learning* é o treino da rede. Este tipo de mecanismo de *machine learning* exige muitos recursos computacionais uma vez que envolve várias operações, como por exemplo multiplicação de matrizes. Assim sendo o hardware utilizado influencia diretamente tanto a precisão dos resultados como o tempo em que esses resultados são alcançados.

Atualmente as *Graphics Process Units* (GPU) são mais utilizadas em detrimento dos *Central Processing Units* (CPU) uma vez que as características da GPU favorecem o tipo de operações utilizadas pelo *deep learning*. A arquitetura das GPUs favorece o processamento em paralelo enquanto os CPUs foram desenhados para realizar tarefas de forma mais rápida.

Para a realização deste projeto foram utilizados os seguintes recursos de hardware:

- Processador Intel i7 – 3537U 2.00GHz
- Placa gráfica Nvidia GeForce GT 740M 2GB

6.1.1 Google Colaboratory

Google Colaboratory é uma plataforma online, que tal como o nome indica foi criada pela Google, e que permite aos utilizadores usarem GPUs com uma grande capacidade comparada à maioria dos GPUs presentes nos computadores pessoais.

O GPU disponível é uma Tesla K80 que tem 24GB de memória. No entanto a utilização desta plataforma tem limitações como por exemplo, apenas é possível usar a GPU no máximo 12 horas de cada vez, o que muitas vezes para tarefas em que exigem mais tempo de processamento, como pesquisa dos melhores *hyperparameters*, se torna limitado. A sessão é suspensa ao fim de 90 minutos sem que haja algum processamento.

6.2 Recolha de dados

Os dados usados no treino e teste dos modelos foram recolhidos do repositório de maquetes da Glintt, e consistiram em todos os ecrãs já implementados. As maquetes têm de obedecer à condição de estarem implementadas, uma vez que será necessário verificar de que forma foi codificado o ecrã, bem como ter a certeza que componentes foram utilizados para que o ecrã tivesse a aparência aprovada pela equipa de design.

Apesar do repositório ter milhares de maquetes nem todas podem ser aproveitadas, uma vez que algumas têm as *guidelines* de implementação, como representado na Figura 22, ou seja, têm texto e outro tipo de indicações a sobrepor a maquete. Na Figura 23 pode-se visualizar como são as maquetes sem *guidelines* e que foram usadas como *input* para o modelo.

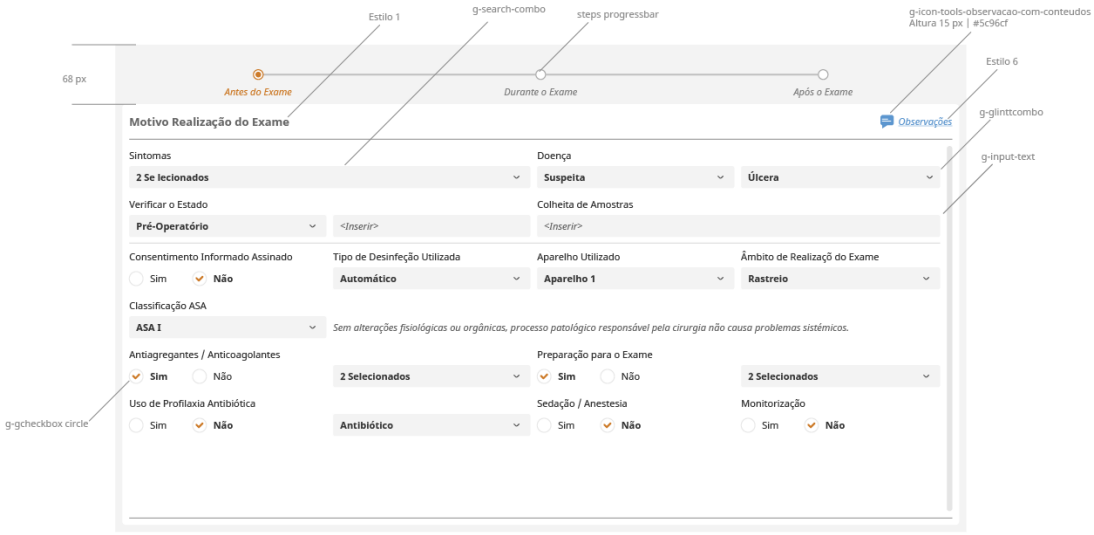


Figura 22 – Exemplo de uma maquete com *guidelines*

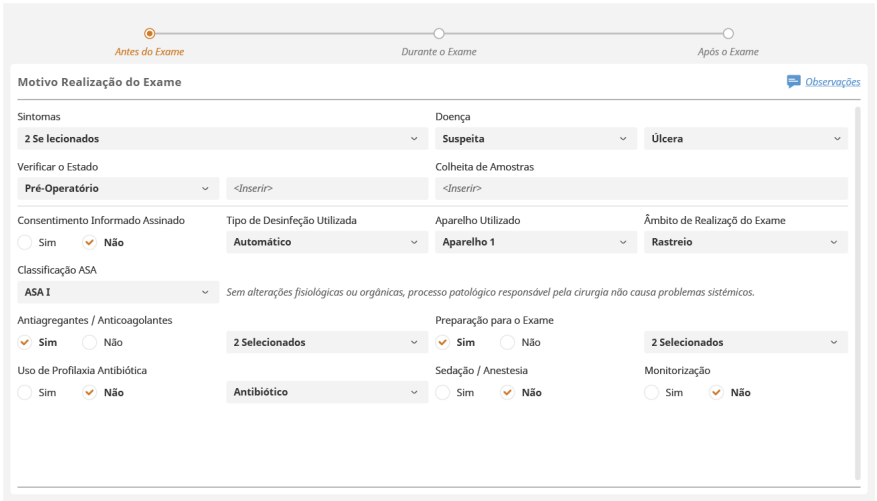


Figura 23 – Exemplo de uma maquete sem guidelines

A implementação de um ecrã, normalmente, é apenas o núcleo do que está na maquete, sendo que o que está a volta, como por exemplo, menus, barras de navegação e cabeçalhos, já está implementado.

Durante este processo, de recolha das maquetes em condições de serem lidas pelo modelo a implementar, para cada uma das maquetes foi feita a descrição textual, à medida que novos componentes iam aparecendo pela primeira vez. Estas descrições foram sendo colocadas na DSL, que posteriormente irá ser utilizada para compilar as descrições textuais geradas pelo modelo. Um exemplo de uma descrição textual de uma maquete pode ser observada na Figura 24.

```
g-layout-filters {
    g-filter-left {
        g-filter {
            g-first { }
            g-property { }
            g-property-selected { }
        }
    }
    g-filter-right {
        filter {
            g-filter {
                g-first { }
            }
        }
        filter {
            filter-label { }
            g-glinttcombo-filter-multi { }
        }
    }
}
g-layout-search {
    g-chips { }
}
g-layout-body {
    iggrid { }
}
```

Figura 24 – Exemplo de uma descrição textual de uma maquete

No final do processo de recolha de dados foram recolhidas 45 maquetes. Este número é baixo no que diz respeito a maquetes, no entanto o número de componentes existente em média para cada maquete é de 81 componentes, o que faz com que o número total de componentes seja à volta de 3645, o que já é um número considerável.

6.3 Pré-processamento dos dados

Um dos passos mais importantes na construção de modelos de machine learning, é o pré-processamento dos dados, uma vez que o formato como estes são lidos pelo modelo influencia diretamente os resultados obtidos no treino.

Durante esta etapa, foi necessário processar tanto as imagens, como as descrições textuais inseridas no modelo. Tal como referido na secção 5.4, usou-se uma rede pré-treinada de forma a abstrair a complexidade da construção de uma CNN de raiz. A rede pré-treinada, no

entanto, tem como input imagens com dimensões de 224x224, por isso foi necessário mudar as dimensões originais das maquetes.

Figura 25 – Maquete redimensionada para o formato 244 pixéis de altura e 244 pixéis de largura.

Como se pode verificar na Figura 25, o aspeto da maquete é preservado sendo possível identificar facilmente os componentes. A imagem original tem em média a dimensão de 1300 pixéis de largura por 600 pixéis de altura, sendo que cada pixel tem a informação para a quantidade de vermelho, verde e azul, isto significa que a imagem original pode ser traduzida através de 2 340 000 valores (1300 x 600 x 3). Pela mesma lógica a imagem redimensionada pode ser traduzida através de 178 608 valores (244 x 244 x 3), o que significa que reduzimos o número de valores na proporção de 1 para 13. Outra estratégia que poderia ser adotada era utilizar maquetes a preto e branco e reduzir ainda mais o número de valores, no entanto, as cores dos componentes desempenham um papel importante na identificação de alguns componentes, como é o caso dos botões.

Relativamente às descrições, foi necessário reduzir a complexidade das descrições, como por exemplo, eliminando as indentações e quebras de linha. Para quem está a escrever a descrição textual é muito mais fácil quando existem quebras de linhas e indentações para perceber a hierarquia dos componentes e de que forma se relacionam entre si, no entanto, para a interpretação do modelo, isso é apenas ruído, uma vez que esta mesma hierarquia pode ser traduzida através de parenteses, “{“ e “}”. Para além disso é necessário adicionar à descrição os *tokens* especiais “<START>” e “<END>”. Esta abordagem também foi feita por Beltramelli e Wallner, a adição destes *tokens* especiais que indicam o começo e o fim da descrição.

6.4 Treinar

Após a construção de uma arquitetura é necessário alimentá-la com dados para que esta possa começar a ajustar os seus pesos, e assim aprender a partir dos dados e gerar um modelo capaz de prever os resultados. No ramo de *machine learning* este processo é denominado por treino.

Os dados que serão usados para o treino da rede, são os dados recolhidos no primeiro passo deste processo que está descrito na secção 6.2. No entanto, é prática normal dividir os dados em duas partes, uma parte para treinar o modelo e a outra parte para testar a performance do modelo. Existem várias formas de poder dividir os dados em teste e validação, sendo que os mais populares são *holdout* e *cross-validation*.

O método de *holdout,* Figura 26*,* é talvez o mais conhecido e o mais simples, este método consiste em dividir os dados em 1/3 para teste e 2/3 para treino (Kohavi, 1995). No entanto pode haver outras formas de separar os dados entre teste e treino, como por exemplo, 50% a 70% para a parte de treino e 30% a 50% para o teste.

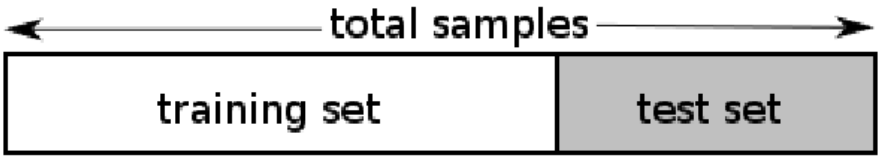


Figura 26 – Exemplo da separação dos dados usando a método de holdout (Chlis, 2013).

No entanto, este método não dá garantias que o modelo é suficientemente genérico uma vez que a performance neste conjunto de dados de teste não garante a qualidade do modelo quando apresentado a um conjunto de dados novos. Para resolver este problema é usado o método de Cross Validation (Kohavi, 1995). Cross Validation é um método que divide os dados em várias partes, o número mais usado é 10, e depois o modelo é treinado com 9 dessas partes e validado com a parte não usada no treino, isto para cada uma das divisões realizadas. Este processo está representado na Figura 27. No final a performance do modelo é a média da performance em cada iteração obtendo-se assim um indicador mais fidedigno do modelo.

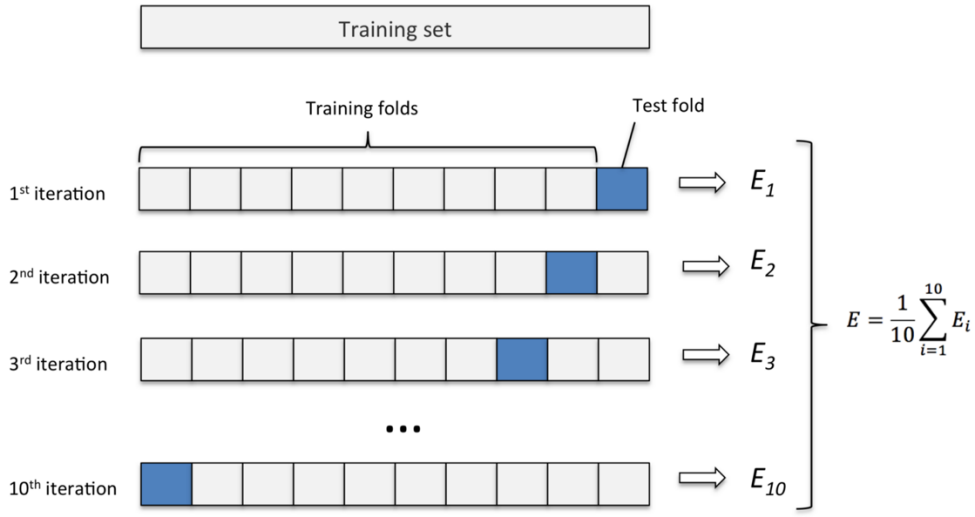


Figura 27 – Divisão dos dados segundo o método de Cross Validation (Rosaen, 2016).

Durante o processo de treino do modelo foram usados os dois métodos de divisão dos dados. Quando é necessário fazer uma pesquisa de hyperparameters ou obter uma noção de performance de forma mais rápida é usado o método Holdout. O método de Cross Validation é essencialmente usado para determinar a performance do modelo final de forma mais precisa.

6.5 Avaliação

Quando estamos a treinar o modelo, é necessário ter uma, ou mais métricas que nos permitam verificar se o modelo está ou não a melhorar durante o treino. Normalmente são utilizadas funções para avaliar o estado do modelo, tais como *Loss Function* e *Accuracy.* Existe uma grande variedade deste tipo de funções, como por exemplo:

1. Loss Function
    a. Mean Squared Error
    b. Categorical Cross Entropy
2. Accuracy
    a. Binary Acuracy
    b. Categorical Accuracy

Outra métrica importante para a avaliação do modelo é o BLEU score (Papineni, 2002). Esta métrica é utilizada para validar de que forma a descrição atual é semelhante à descrição gerada pelo modelo. A forma mais usada deste método é o 4-ngram, que verifica as palavras uma a uma, depois duas a duas, depois três a três e finalmente quatro a quatro e por último multiplica por 0.25 cada uma das ocorrências certas pelo número de ocorrências acertadas. Como por exemplo, vamos considerar a seguinte frase,

Deep Learning é engraçado

No entanto, o modelo gerou a seguinte frase,

Deep Learning não é difícil

De forma a obter o bleu score 4-ngram, é necessário dividir a frase agrupando as palavras por um, dois, três e quatro, sendo assim obtém-se a Tabela 8.

N-gram	Sequências
1	Deep, Learning, não, é, difícil
2	Deep Learning, Learning não, não é, é difícil
3	Deep Learning não, Learning não é, não é difícil
4	Deep Learning não é, Learning não é dificil

Tabela 8 – Sequências por n-gram da frase gerada

Agora para calcularmos o BLEU score, é necessário verificar para cada uma das sequências quantas aparecem na frase original. Os resultados estão na Tabela 9.

N-gram	Sequências	Classificação
1	Deep, Learning, não, é, difícil	3/5
2	Deep Learning, Learning não, não é, é difícil	1/4
3	Deep Learning não, Learning não é, não é difícil	0/3
4	Deep Learning não é, Learning não é dificil	0/2

Tabela 9 – Contagem da ocorrência das sequências na frase original

Após recolher a contagem será calculado o BLEU score, ou seja

$$0.25*(3/5) + 0.25*(1/4) + 0.25(0/3) + 0.25*(0/2) = 0.2125$$

O resultado do BLEU score para a frase gerado foi de 0.2125, que tal como era de esperar é um mau resultado.

6.6 Configuração dos *hyperparameters*

Hyperpameteres são parâmetros que permitem mudar a forma como a rede neuronal se comporta. Existem dois tipos de *hyperparameters*, do tipo estrutural, ou seja, que permitem mudar a estrutura da rede, como por exemplo o número de camadas e número de nós na rede e também existem *hyperparameteres* que influenciam de que forma é feito o treino na rede neuronal.

6.6.1 Hyperparameters do tipo estrutural

Os *hyperparameters* do tipo estrutural, tal como o nome indica, determinam a estrutura da rede. Nas secções seguintes serão apresentados alguns *hyperparameters* que permitem ajustar a estrutura da rede de forma a melhorar a performance do modelo.

6.6.1.1 Número de camadas e nós na rede

Ao adicionar mais camadas à rede é possível que o modelo melhore a *accuracy*, no entanto existe um limite em que a *accuracy* do modelo estagna, ou a rede se torna muito complexa e seja demasiado difícil de treinar, ou ainda ocorra *overfitting*, ou seja, o modelo gerado pela rede ajusta-se demasiado aos dados de treino e apresenta um elevado erro com os dados de teste. Por outro lado, um número reduzido de nós e camadas pode causar *underfitting*, ou seja, não se consegue obter um modelo.

6.6.1.2 Dropout

É uma técnica para combater o *overfitting*. O *dropout* indica a probabilidade de os nós existentes na camada serem desativados e assim fazer com que alguns dos nós não fiquem demasiado dependentes do input. Estes valores variam entre 0% e 100%, sendo que o intervalo mais usado é entre 20% a 50%. A Figura 28 mostra visualmente este processo em ação.

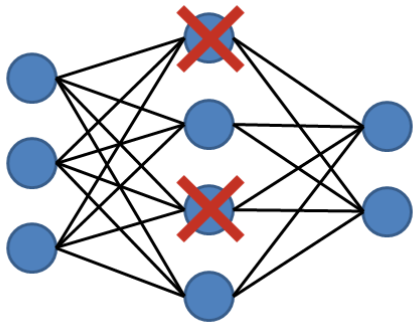


Figura 28 – Nós da rede cancelados devido ao efeito do técnica de *dropout* (Radhakrishnan, 2017).

6.6.1.3 Função de ativação

A função de ativação calcula se o sinal será transmitido na rede e com que valor ele será transmitido. As funções de ativação mais populares são: sigmoid (Figura 29), thanh (Hyperbolic tangent) e relu (Rectified linear units).

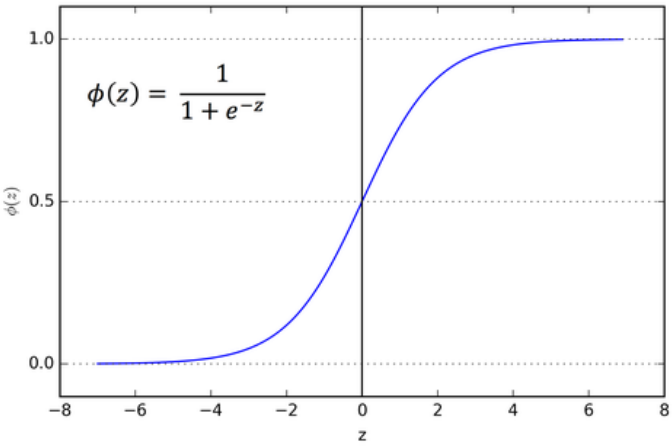


Figura 29 – Sigmoid function (Sharma, 2017)

6.6.2 Hyperparameters de treino

Este tipo de *hyperparameters* determina de que modo o modelo será treinado, ou seja, de que modo os pesos da rede serão ajustados para encontrar a melhor performance. Nas secções seguintes serão apresentados alguns deste *hyperparameters*.

6.6.2.1 Learning rate

O parâmetro de learning rate define o quão rápido a rede renova os pesos da rede. Ou seja, quanto maior o learning rate menos é tido em conta o peso atual da rede e quanto menor o learning rate mais impacto tem os pesos anteriores para o calculo dos novos pesos.

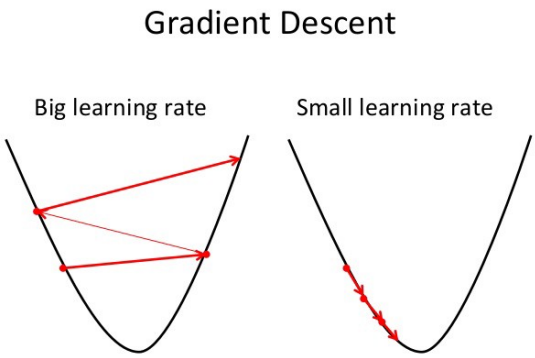


Figura 30 – Diferença entre número alto e baixo de learning rate (Radhakrishnan, 2017).

6.6.2.2 Batch size e número de epochs

O batch size, é a quantidade de registos dos dados de treino que são necessários para atualizar os pesos da rede, ou seja, se o batch size tiver o valor 1, então a cada nova passagem de registos pela rede a rede atualiza os seus pesos, no entanto se o batch size tiver o mesmo

valor que o número de registos existentes nos dados de treino, então os pesos da rede apenas são atualizados depois de todos os registos passarem pela rede.

O número de *epochs* corresponde ao número de vezes que todos os registos passaram pela rede. Por exemplo se o *batch size* for igual ao número de registos, então um ciclo de *batch* corresponde a um *epoch*, no entanto se o *batch size* for metade do número total dos registos existentes são necessários dois ciclos para completar um *epoch*.

6.6.3 Automatização da pesquisa de *hyperparameters*

A pesquisa dos *hyperparameters* que correspondem à melhor configuração da rede é um dos passos mais importantes na construção de um modelo de *Deep Learning*. No entanto, encontrar os *hyperparameters* para o modelo é uma tarefa difícil, uma vez que é necessário ter alguma experiência para saber quais os *hyperparameters* que devem ser alterados.

Durante o processo de construção do modelo desenvolvido foi adotado essencialmente a método de *Grid Search*, quando havia um conjunto de valores bem definido que se queria testar, como por exemplo testar a performance de uma arquitetura com o número de nós de 128, 256 e 512. O método *Bayesian Optimization* foi usado quando o espaço de pesquisa era maior e não existia um conjunto de valores bem definido que se pretendia testar, como por exemplo a combinação ideal entre o número de camadas de uma rede e o número de nós.

Para além dos métodos mencionados anteriormente, serão apresentados nas seções seguintes outros métodos de pesquisa de *hyperparameters*.

6.6.3.1 Grid Search

Um dos métodos para encontrar os *hyperparameters* que melhor se ajustam aos dados é denominado por *Grid Search*. Quando se utiliza um *Grid Search*, basicamente reúne-se um conjunto de hipóteses que voltem a melhorar o modelo e geram-se todas as combinações possíveis dessas hipóteses para depois verificar qual delas teve melhor performance.

Por exemplo, assumindo que se pretende testar o número de camadas 1, 2, 3 e 4 e testar o número de nós 128, 256, 512 e 1024. O número de vezes que o modelo será construído e avaliado será 16, e se para além disso também se testar o número de *epochs* ideal, como por exemplo, 100, 200, 300 e 400 é necessário construir o modelo e avaliá-lo cerca de 64 vezes.

Como se pode verificar este método não é o melhor quando se pretende verificar um grande número de variáveis, porque se torna muito custoso computacionalmente.

6.6.3.2 Random Search

Com o método de *Grid Search*, existem casos que não são explorados e pode ser que um desses casos seja a solução ótima para o problema. Para resolver este problema foi criado o método *Random Search* (Bergstra & Bengio, 2012) que propõe a geração de parâmetros

aleatórios a ser explorados no modelo. O problema descrito anteriormente pode ser visualizado na Figura 31.

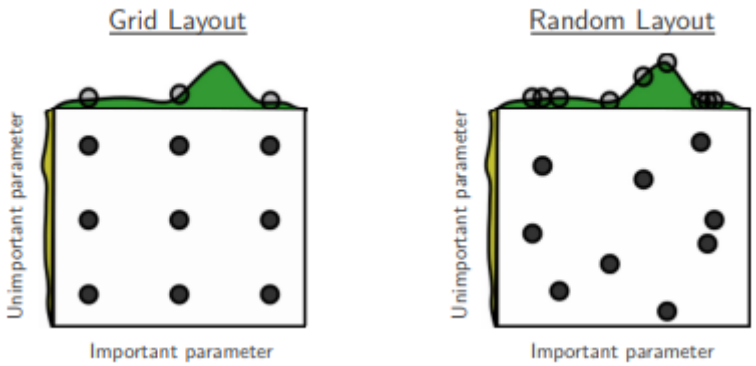


Figura 31 – Comparação de nove pesquisas entre o método de Grid Search e Random Search (Bergstra & Bengio, 2012).

6.6.3.3 Bayesian Optimization

Segundo (Snoek & Larochelle, 2012), o que torna o método *Bayesian Optimization* diferente de outros procedimentos é que este constrói um modelo probabilístico f(x) e, em seguida, explora esse modelo para tomar decisões para que valor de x deve avaliar a função, enquanto integra a incerteza. Com isto à medida que o algoritmo vai explorando o espaço vai reduzindo a incerteza e encontrando a função com a qual o modelo se rege. Na Figura 32 pode-se verificar um gráfico que ilustra este processo.

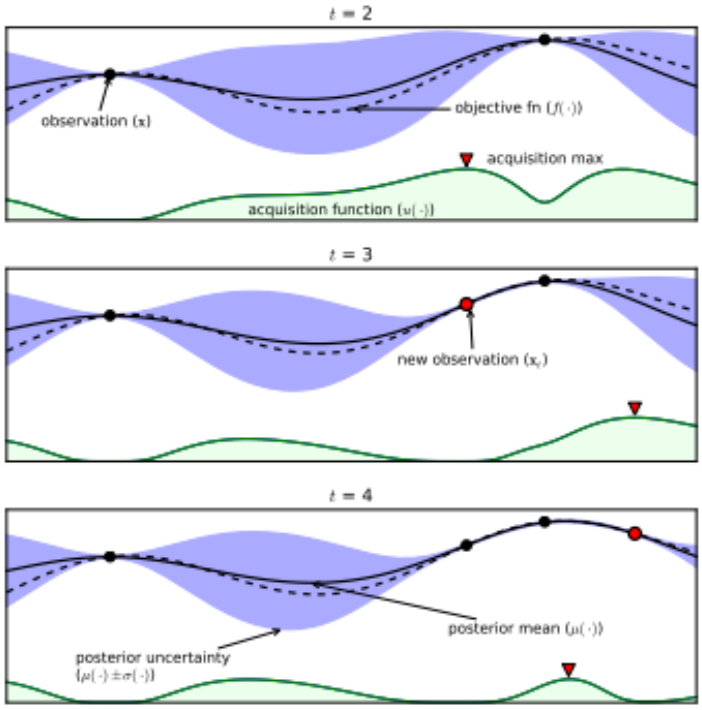


Figura 32 – Exemplo da exploração do espaço com a utilização do método Bayesian Optimization (Brochu et al., 2010).

No entanto, na prática este algoritmo procura o mínimo global da função. Se for utilizado o parâmetro de *accuracy* como valor a ser minimizado, o método *Bayesian Optimization* irá procurar aquele que tem a menor *accuracy*, ou seja, irá procurar o pior modelo. De modo a usar a minimização no sentido de encontrar a melhor *accuracy* possível, é preciso fazer com que a maior *accuracy* seja o valor mais baixo, e por causa deste comportamento, é usado o valor simétrico da *accuracy*, como por exemplo, um valor de 0.7 ficaria -0.7.

6.6.3.4 Otimização de Learning Rate e Momentum usando Bayesian Optimization

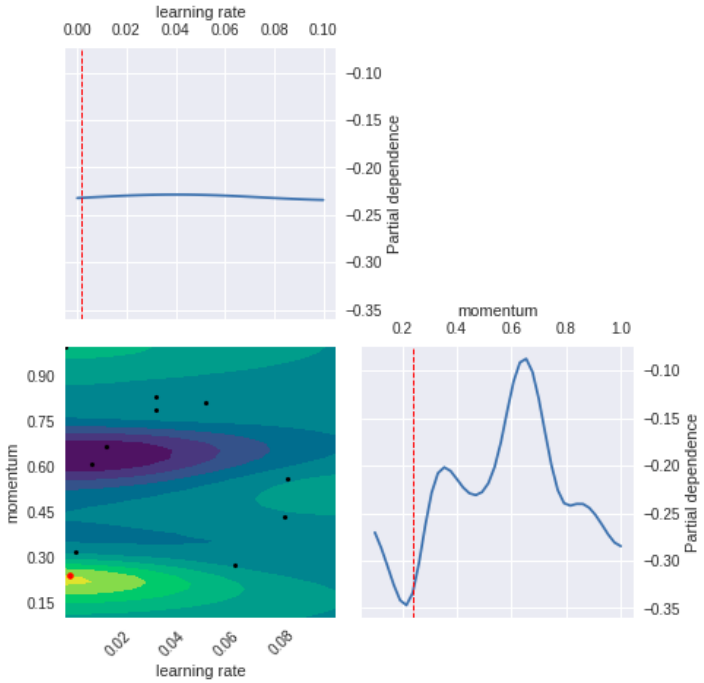


Figura 33 – Primeiro levantamento da influência do *learning rate* e *momentum* para a accuracy do modelo

A Figura 33 apresenta a relação entre o modelo e a configuração do *optimizer RMSprop* através dos parâmetros *learning rate* e *momentum*. Como se pode verificar neste primeiro levantamento existem algumas zonas onde o modelo teve uma boa *accuracy*, no entanto, não se pode afirmar por exemplo, que a configuração dos parâmetros devolvidos pela função de minimização, *learning rate* a 0.0017 e o *momentum* a 0.24, são a configuração ótima para o modelo. De modo a ter uma ideia de quais são os melhores valores para estes parâmetros é necessário voltar a fazer outra análise com o espaço de pesquisa menor, dentro da área em que se obteve melhores resultados. Um intervalo candidato seria talvez entre [0.01, 0.35] para o *learning rate* e [0.0005, 0.02].

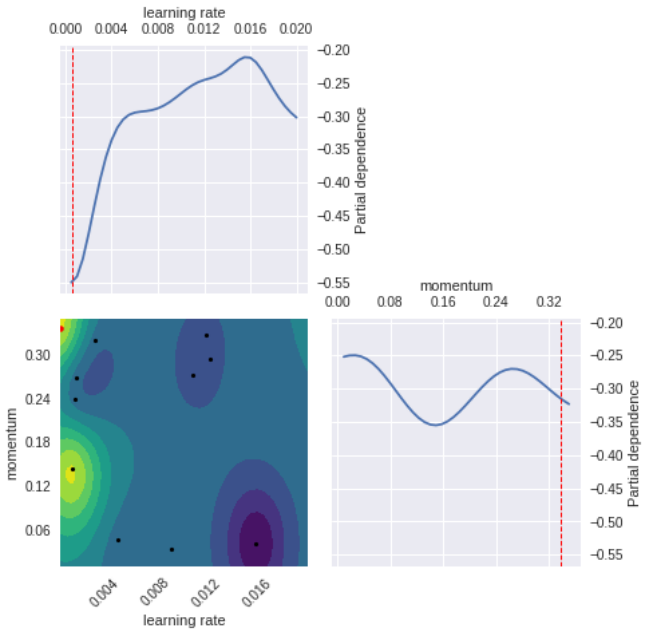


Figura 34 - Segundo levantamento da influência do *learning rate* e *momentum* para a accuracy do modelo

No segundo levantamento os melhores resultados, Figura 34, pode-se constatar que quando o parâmetro de *learning rate* está muito próximo de zero é quando se obtém os melhores resultados, sendo que o parâmetro de *momentum* parece não ter tanta influência nos resultados.

6.7 Construção

Após a recolha dos dados e o pré-processamento, o próximo passo é construir o modelo. A construção do modelo é um processo iterativo, em que se testam diversas arquiteturas para o modelo e várias formas diferentes de treinar o modelo.

Como ponto de partida será adotada a arquitetura de Wallner, que posteriormente será treinada com os dados recolhidos. A performance do modelo perante os novos dados irá determinar os ajustes que serão feitos à arquitetura ou modo de treino da rede.

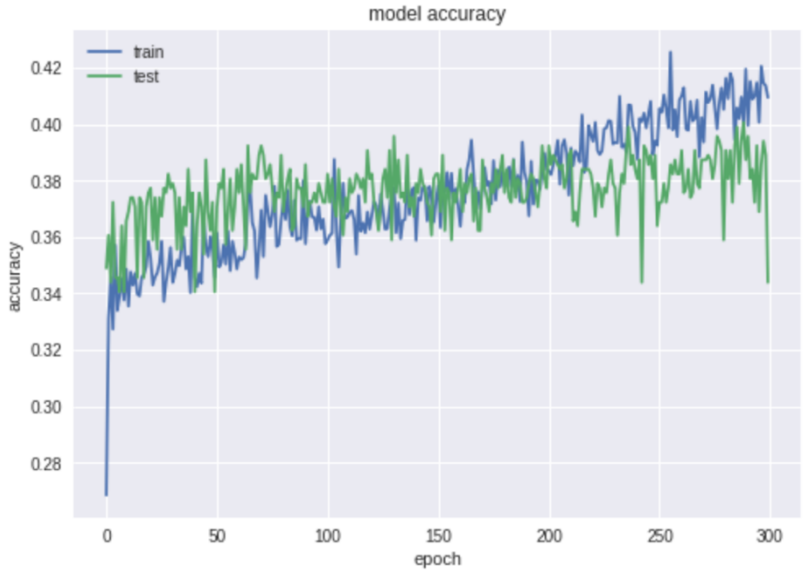


Figura 35 – Performance do modelo

Como se pode verificar através da Figura 35, a performance do modelo baseado na arquitetura de Wallner não é satisfatória, uma vez que a *accuracy* com os dados de teste estagnou entre 0.35 e 0.4 e para além disso, a *accuracy* dos dados de treino continua a aumentar o que significa que o modelo está a começar a ajustar-se em demasia aos dados de treino, ou seja, está a ocorrer *overfitting*.

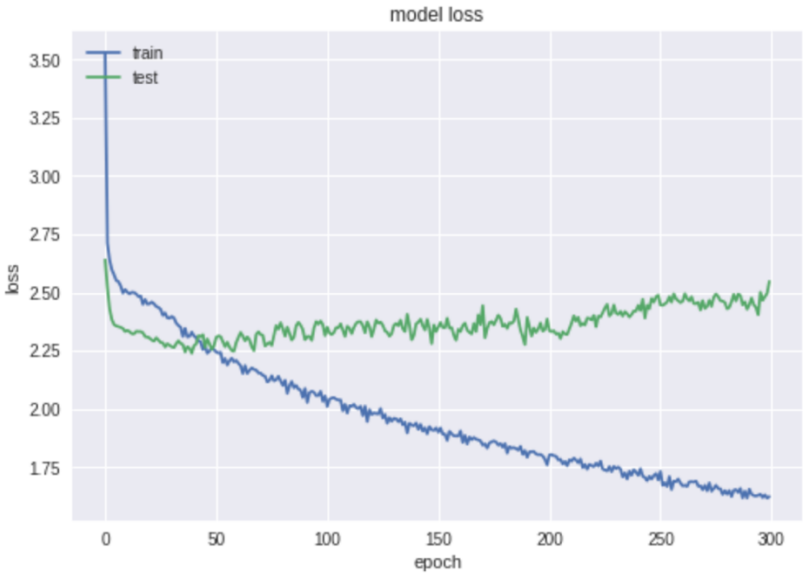


Figura 36 – Função erro para o modelo

Relativamente à função de erro do modelo, que pode ser observada através da Figura 36, pode-se confirmar a análise que fizemos para o gráfico anterior. A partir de 50 *epochs* os dados de teste mantêm o mesmo nível de erro durante o resto do processo de treino, enquanto o erro para os dados de treino continua a descer.

Como referido anteriormente o modelo está em *overfitting*. Isto pode acontecer quando o número de nós existente na rede é muito grande relativamente ao número de dados que existem a circular na rede. Para combater este problema é necessário diminuir a complexidade da rede. Mas para além deste problema o modelo está a demorar demasiado tempo a aumentar a sua performance, por isso será também pertinente testar outros algoritmos de otimização.

Optimizer	Accuracy	Loss	Validation Accuracy	Validation Loss	Acc – Val. Acc
SGD	0.3600	2.0567	0.3655	2.1646	-0.0055
Adam	**0.9975**	**0.0119**	0.7809	1.5462	0.2166
Rmsprop	**0.9972**	**0.0076**	0.7632	2.0924	0.2340
Adagrad	**0.9938**	0.0610	**0.8687**	**0.6616**	0.1251
Adadelta	0.5287	1.2087	0.4627	2.0007	**0.0660**
Adamax	0.7092	0.7842	0.6507	1.6392	**0.0585**
Nadam	0.8619	0.4145	0.5367	3.2203	0.3252

Tabela 10 – Desempenho dos algoritmos de otimização

A Tabela 10 apresenta o resultados dos vários testes feitos para cada um dos algoritmos de otimização presentes na *framework* Keras. Cada um dos testes foram feitos com 300 *epochs* e *batch size* de 3.

Como se pode verificar os algoritmos Adam, Rmsprop e Adagrad foram os que tiveram a melhor performance no conjunto de dados de treino, no entanto este indicador apesar de ser importante não desempenha um papel crucial, uma vez que o modelo pode estar em *overfitting*. Os melhores indicadores são o Validation Accuracy e Validation Loss, uma vez que representam o desempenho do modelo em dados diferentes dos quais ele foi treinado. Também é importante verificar a diferença entre a Accuracy e Validation Accuracy, uma vez que se este valor for elevado quer dizer que o modelo está em *overfitting*.

Tendo em conta os indicadores que foram referenciados anteriormente como mais relevantes, pode-se afirmar que o algoritmo que melhor se adapta ao modelo é o Adagrad, uma vez que tem a maior Validation Accuracy (0.8687), o Validation Loss mais baixo (0.6616), mas apesar de não ter o melhor indicador de diferença entre Accuracy e Validation Accuracy, foi o que teve melhor resultado neste indicador no grupo dos algoritmos que tiveram melhor performance no indicador de Validation Accuracy.

Com os parâmetros de treino já encontrados, será necessário agora modificar a arquitetura da rede de modo a encontrar a melhor combinação e atingir o maior valor possível de Validation Accuracy e com o menor *overfitting* possível. Para encontrar a arquitetura ideal será utilizado o método de pesquisa por *hyperparameters*, *Bayesian Optimization*.

Identifier	CNN Dense Layers	CNN Dense Neurons	LSTM Layers	LSTM Neurons	Encoder LSTM Layers	Encoder LSTM Neurons
1	3	84	1	200	2	449
2	2	116	2	426	3	177
3	1	127	2	354	1	168
4	2	66	1	208	2	315
5	2	116	2	194	3	211
6	1	67	2	468	2	218
7	2	125	2	175	2	466
8	3	78	2	183	2	333
9	2	69	2	189	2	481
10	2	102	3	284	2	265
11	2	118	1	330	1	136
12	2	107	1	504	1	363

Tabela 11 – Arquiteturas testadas para o modelo

As arquiteturas testadas representadas na Tabela 11, foram testadas usando 125 *epochs* e 3 de *batch size*. Cada *epoch* demora em média 45 segundos usando o Google Colaboratory, para testar estes 12 exemplos demorou cerca de 19 horas, para o mesmo conjunto de testes usando o computador pessoal, em que demora em média 300 segundos por cada *epoch*, este valor iria ascender a cerca de 5 dias e 5 horas.

Identifier	Accuracy	Loss	Validation Accuracy	Validation Loss	Acc – Val. Acc
1	**0.9626**	**0.1368**	**0.7883**	**0.9092**	**0.1743**
2	0.4094	1.8340	0.3903	2.0883	0.0191
3	0.3381	2.2004	0.3300	2.3214	0.0081
4	**0.9046**	**0.3078**	**0.7874**	**0.9345**	**0.1172**
5	0.4120	1.8986	0.3893	2.0850	0.0227
6	0.4263	1.6540	0.4199	2.0576	0.0064
7	0.4455	1.7403	0.4336	1.9733	0.0119
8	0.5011	1.4996	0.5068	1.6360	-0.0057
9	0.4811	1.5033	0.4561	1.6622	0.0250
10	0.4895	1.5701	0.5000	1.6463	0.0105
11	0.3220	2.3443	0.3232	2.2775	-0.0012
12	**0.9584**	**0.1609**	**0.8592**	**0.5397**	**0.0992**

Tabela 12 – Resultados obtidos para cada uma das estruturas testadas

Na Tabela 12, pode-se observar a performance para cada uma das arquiteturas referidas na Tabela 11. Deste conjunto de arquiteturas será escolhido o conjunto com a melhor performance, idealmente com o maior Validation Accuracy, para ser posteriormente analisada com mais pormenor e comparada com a estrutura inicial. Uma destas arquiteturas será a arquitetura final a usar na solução a ser desenvolvida.

6.7.1 Arquitetura 1

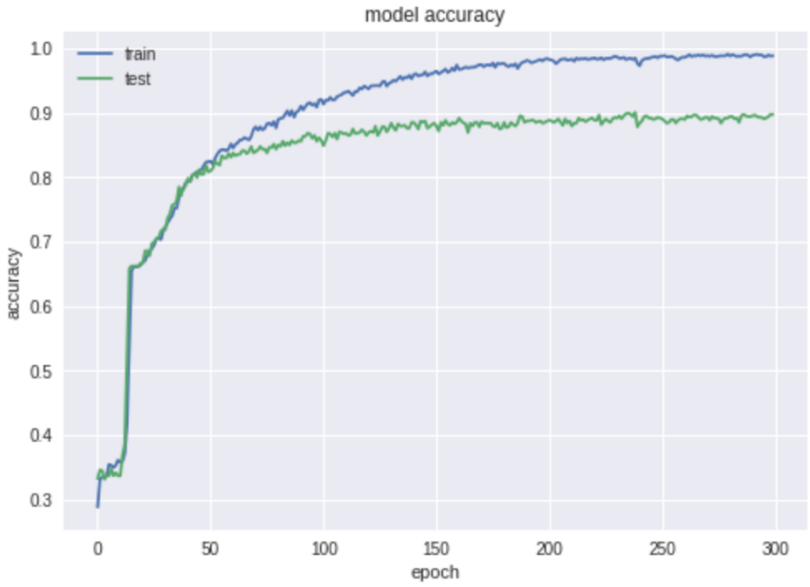


Figura 37 – Accuracy da arquitetura 1

Na Figura 37 pode-se observar a *accuracy* para os dados de teste e os dados de treino para a arquitetura identificada como 1 na Tabela 11 e na Tabela 12. Ao fim de 300 *epochs* esta arquitetura obteve a *accuracy* para os dados de teste de cerca de 0.8975, superando assim a performance obtida pela arquitetura original. Relativamente ao BLEU score esta arquitetura teve o desempenho de 0.7848.

6.7.2 Arquitetura 2

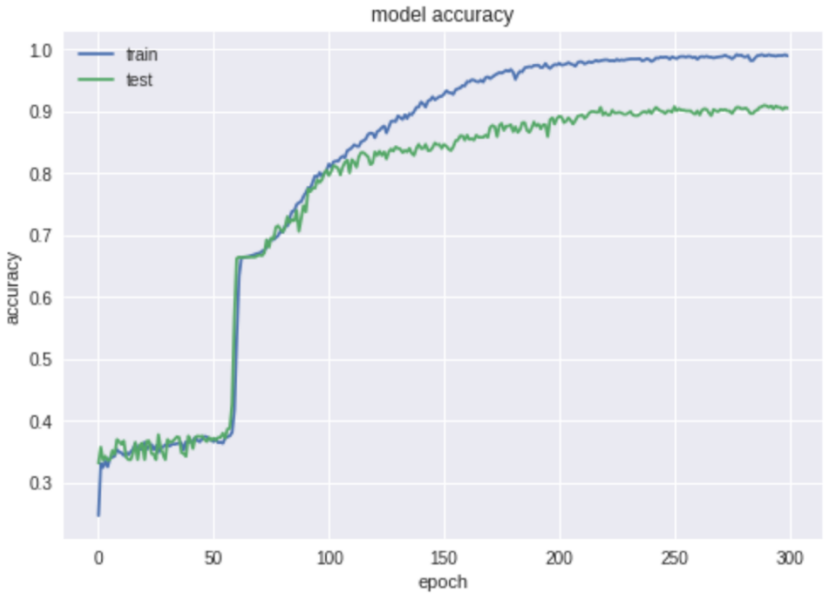


Figura 38 – Accuracy estrutura 2

Na Figura 38 é possível observar a *accuracy* para os dados de treino e dados de teste para arquitetura 2 definida anteriormente na Tabela 11 e Tabela 12. Esta arquitetura teve a *accuracy* para os dados de teste de cerca de 0.9054, superando assim a performance tanto da arquitetura original como a performance da arquitetura 1. No entanto relativamente ao BLEU score a arquitetura 2 não teve melhor resultado que a arquitetura 1, tendo atingido a marca de 0.7395.

6.7.3 Arquitetura 3

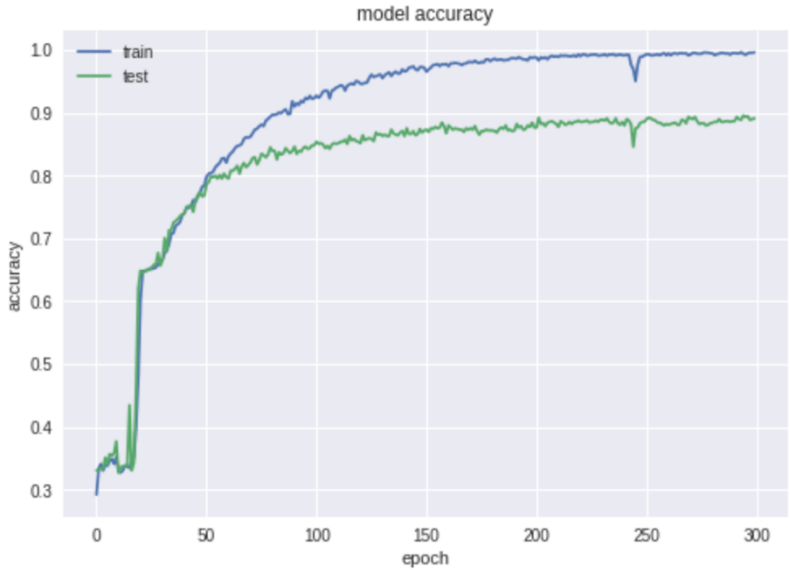


Figura 39 – Accuracy da estrutura 3

Na Figura 39, pode-se verificar a *accuracy* para os dados de teste e para os dados de treino da arquitetura representada com o identificador 13 na Tabela 11 e Tabela 12. A *accuracy* de teste para esta arquitetura é de 0.8908 que apesar de ser superior à performance da arquitetura inicial não tem melhor performance que a arquitetura 1 e 2.

6.7.4 Arquitetura final

Analisando todas as arquiteturas candidatas à arquitetura final para a solução a ser desenvolvida, pode-se verificar que as arquiteturas 1 e 2 são as que tem maior *accuracy* e maior BLEU score. No entanto a arquitetura 2 tem maior *accuracy* do que a arquitetura 1, mas por outro lado a arquitetura 1 tem maior BLEU score do que a arquitetura 2.

Para determinar qual das arquiteturas adotar, será usado o método de *Cross Validation* para as duas arquiteturas, usando 100 *epochs* e dividindo os dados em 5 partes.

Fold	Validation Accuracy	BLEU score
1	0.7749	0.4128
2	0.7373	0.4256
3	0.3669	0.1175
4	0.9219	0.7759
5	0.7962	0.3748
	0.7194	0.4213

Tabela 13 – Resultados Cross-Validation para a arquitetura 1

Fold	Validation Accuracy	BLEU score
1	0.7240	0.3789
2	0.7107	0.2667
3	0.6948	0.3389
4	0.9060	0.4524
5	0.8225	0.6016
	0.7716	0.4077

Tabela 14 – Resultados Cross-Validation para a arquitetura 2

Olhando para os resultados obtidos na Tabela 13 e na Tabela 14 usando o método de *Cross Validation*, comprova-se o que já se tinha verificado anteriormente sobre a Validation Accuracy e BLEU score para estas arquiteturas. No entanto, como o BLEU score é um indicador que permite classificar o texto gerado pelo modelo e este resultado final é o que mais importa, para posteriormente se compilar a descrição em código, a arquitetura final é a arquitetura 1 que pode ser analisada com mais detalhe através da Figura 40.

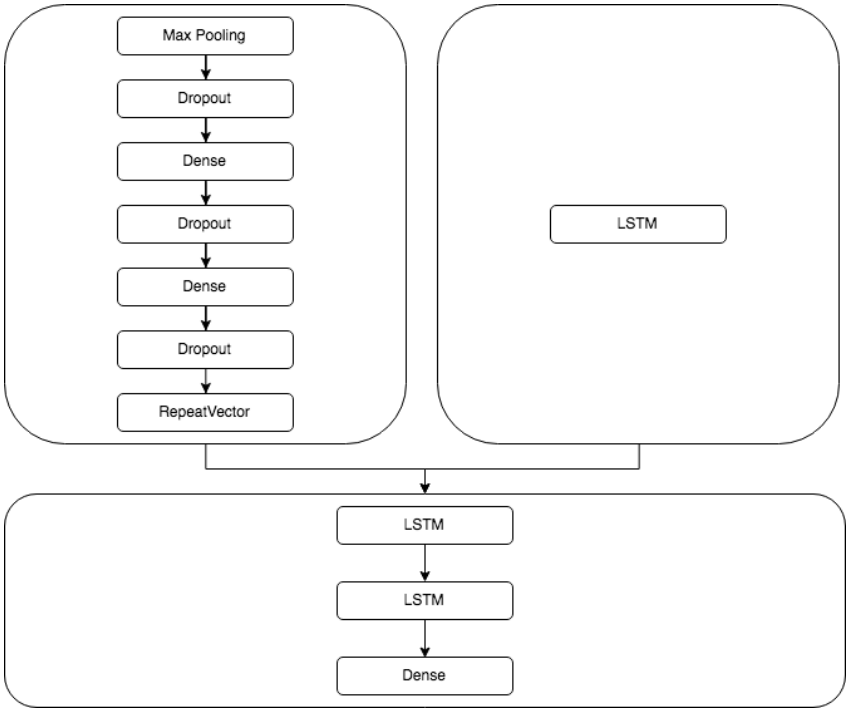


Figura 40 – Arquitetura final do modelo

6.8 Produção

Nesta secção será descrito o processo de colocação do modelo em produção. O projeto está num repositório Gitlab privado, em que sempre que existe um *commit* este executa o pipeline e coloca o modelo em produção através do Heroku Cloud. Cada uma destas etapas serão descritas nas secções seguintes.

6.8.1 Heroku

Heroku é um serviço de Cloud, fundado em 2007, e que foi posteriormente comprada pela empresa Salesforce. Atualmente o Heroku permite receber aplicações em Ruby, Java, Clojure, Python, Scala e Node. O Heroku tem várias modalidades de preço sendo que uma delas é grátis que foi a modalidade usada para a execução deste trabalho. No entanto, o serviço gratuito apenas tem algumas limitações, sendo uma delas a desativação da nossa aplicação após trinta minutos de inatividade. Isto faz com que a abertura da aplicação demore mais que o normal.

6.8.2 Gitlab

Gitlab é um gestor de repositórios Git criado em 2011. Atualmente é um dos gestores de repositórios mais populares do mundo a par de Github e Bitbucket. Esta plataforma foi escolhida porque é a que permite o maior número de repositórios privados e o maior número de colaboradores na sua modalidade grátis, como pode ser observado na Figura 41.

Free Plans	Public Repos	Private Repos	Collaborators	Storage Space	Hosting	Support
GitHub Public	Unlimited	0	Unlimited	N / A	Cloud	Email / Forum
Bitbucket Small teams	Unlimited	Unlimited (1Gb /project)	5	N / A	Cloud	Email / Forum
GitLab Cloud Hosted	Unlimited	Unlimited (10Gb / Project)	Unlimited	Unlimited	Cloud	Forum

Figura 41 – Comparação do plano gratuito entre Github, Bitbucket e Gitlab (flow.ci, 2016).

Para além disso, o Gitlab também tem um sistema de pipelines o que permite automatizar algumas tarefas, tal como a implantação da solução desenvolvida.

6.8.3 DeveloperAI

DeveloperAI é o nome do portal onde o presente projeto está alocado, e pode ser acedido em https://developerai.herokuapp.com. Como se pode verificar na Figura 42, é um portal

bastante simples constituído apenas por um controlo para enviar a maquete para o site e outro botão para receber o ficheiro com o código correspondente à maquete.

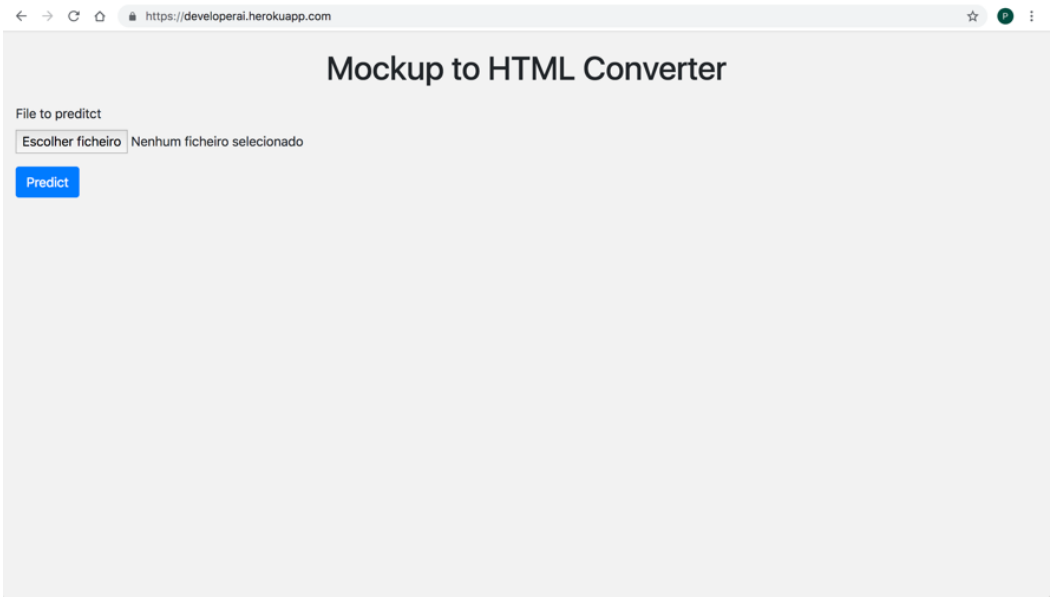


Figura 42 – Portal DeveloperAI

6.8.3.1 Manual de utilização

Para converter a maquete para código primeiro é necessário aceder ao portal DeveloperAI, https://developerai.herokuapp.com. Para começar a fazer a conversão é necessário fazer *upload* da maquete, para isso é necessário clicar na região vermelha representada na Figura 43.

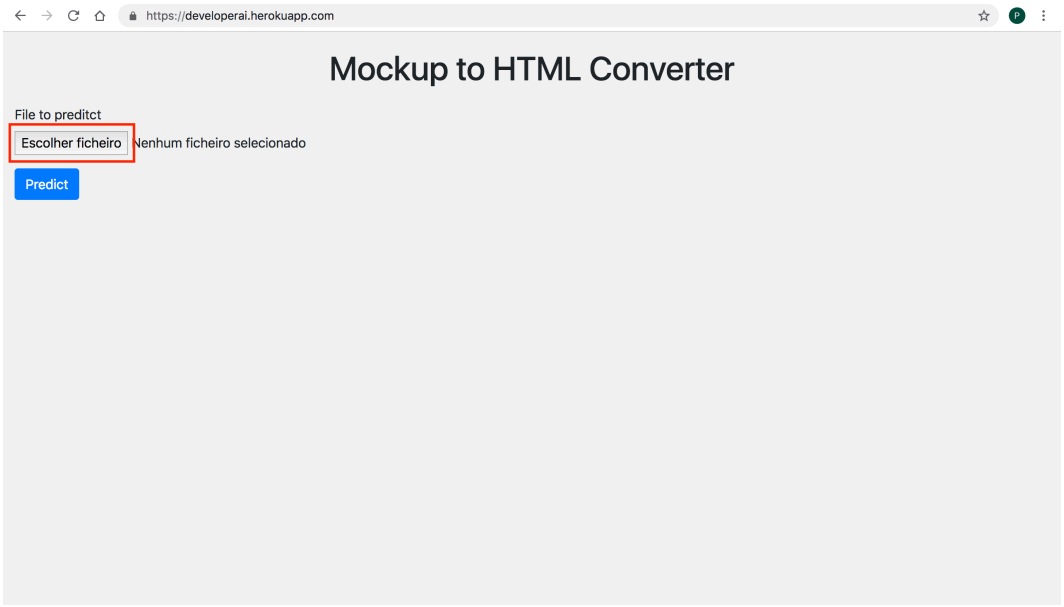


Figura 43 – Upload da maquete

Após seguir os passos anteriores, a aplicação irá perguntar qual é o ficheiro que deseja fazer upload, é necessário então navegar para o local onde se encontra a maquete, selecionar a imagem e de seguida carregar na região vermelhar como representado na Figura 44.

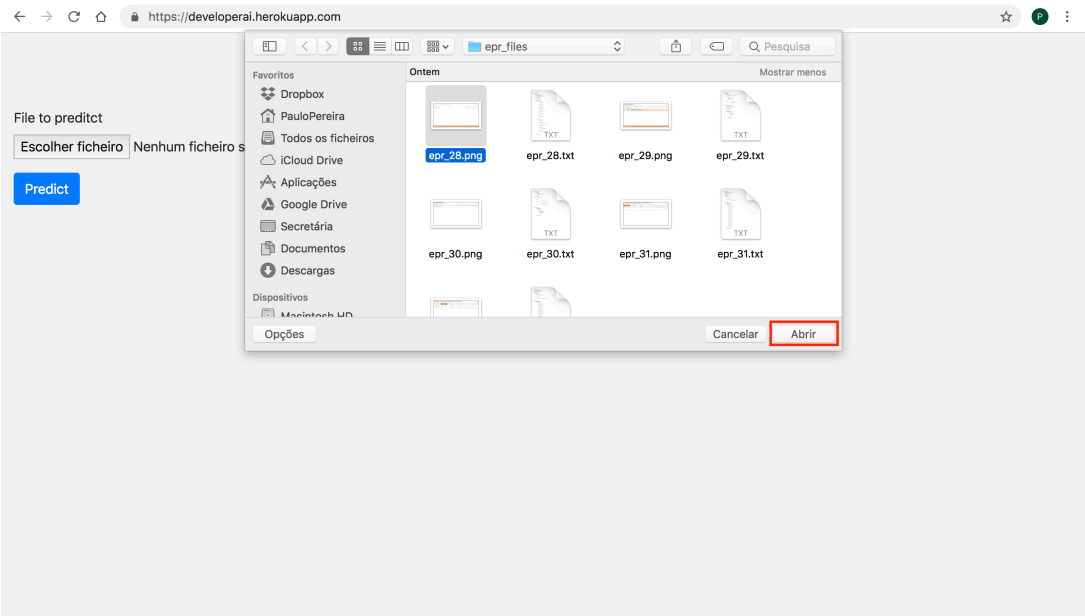


Figura 44 – Seleção da imagem para fazer upload

Se o ficheiro foi carregado com sucesso para o website, então o nome do ficheiro deverá aparecer na região a laranja representada na Figura 45. Depois do ficheiro ter sido carregado com sucesso é necessário carregar na região vermelha representada na Figura 45, para fazer a conversão da maquete.

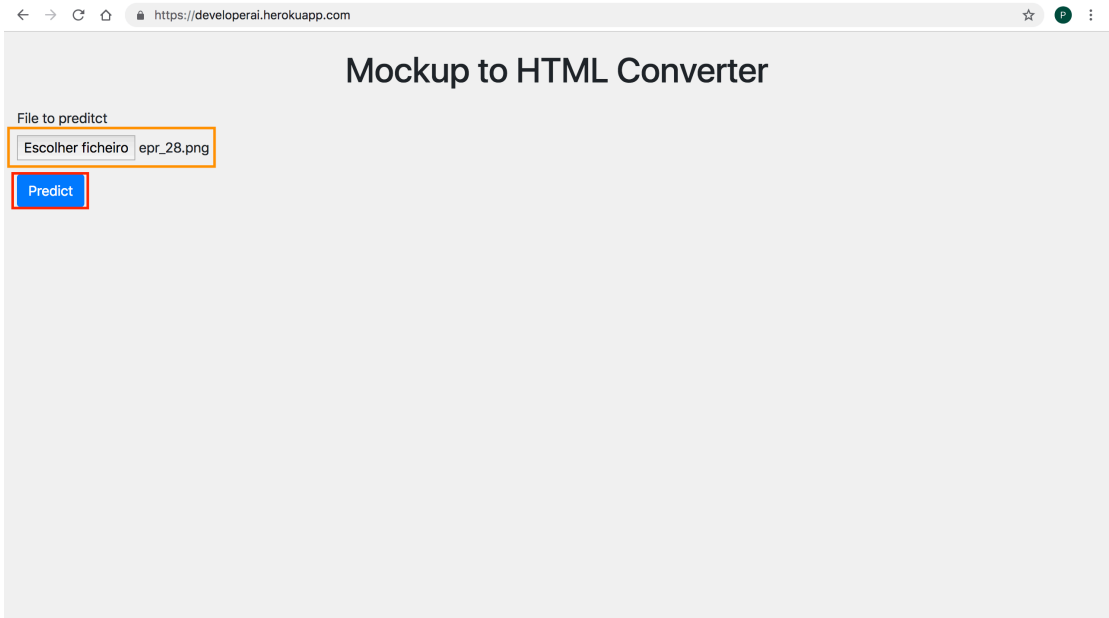


Figura 45 – Despoletar a conversão da maquete para código

Após carregar no botão, é necessário esperar alguns segundos para que que a maquete seja convertida. Se a conversão for bem sucedida então, iniciar-se–á o download do ficheiro com o código da conversão, como representado na Figura 46.

Figura 46 – Download do ficheiro com o código correspondente à maquete

Após acabar o download do ficheiro é necessário descomprimir o ficheiro, que contém uma pasta e um ficheiro HTML, como representado na Figura 47. A pasta "assets" contém as dependências para conferir a aparência e comportamento semelhante à maquete. Entre estas dependências pode-se encontrar as bibliotecas JQuery, JQuery-UI, Bootstrap, Glinttology, entre outras.

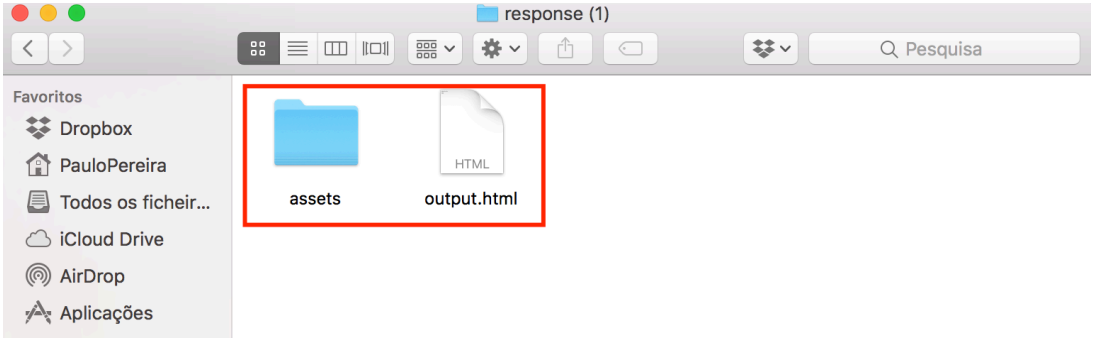


Figura 47 – Ficheiros após a descompressão

O ficheiro output.html é o ficheiro que irá permitir visualizar o resultado final no browser. Um exemplo desse resultado pode ser consultado na Figura 48. Na zona vermelha da Figura 48 pode-se também verificar que se trata do ficheiro HTML e não da imagem que se fez *upload* anteriormente.

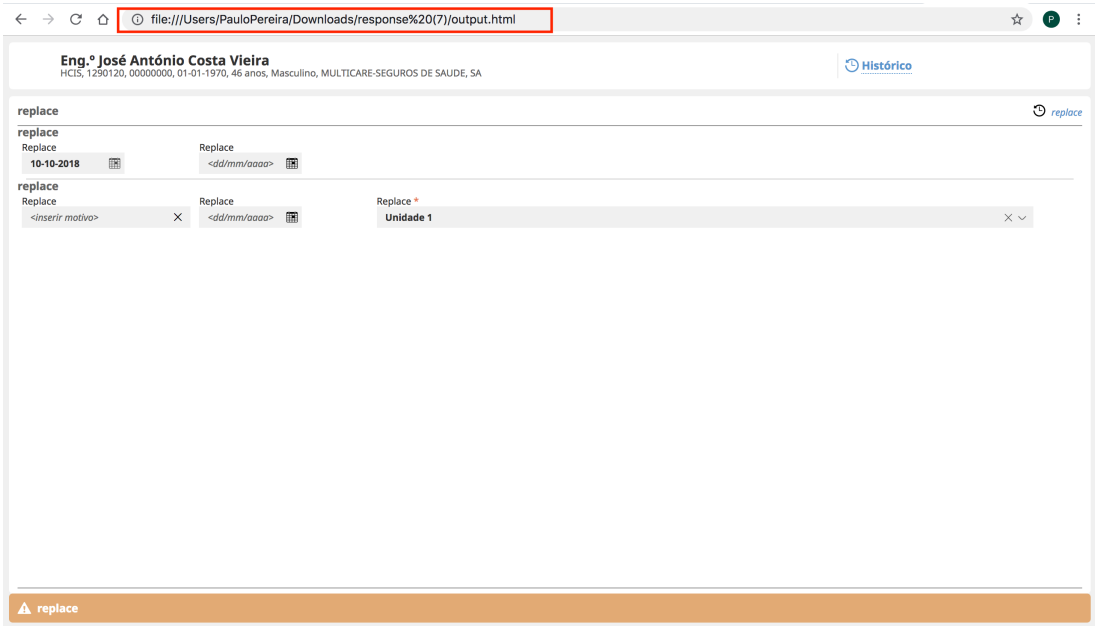


Figura 48 – Código final representado no browser

7. Avaliação da solução

Após a solução ter sido desenvolvida é necessário verificar se os objetivos que foram traçados no início do projeto são cumpridos. Desta forma, pode-se recorrer a testes de hipóteses e a testes estatísticos de modo a demonstrar o cumprimento dos objetivos.

Para realizar a avaliação à solução é necessário definir o que irá ser avaliado, recolher informação relevante que permita avaliar cada parâmetro de avaliação, após este passo é necessário fazer os testes estatísticos com os dados recolhidos e finalmente tirar conclusões e salientar pontos de melhoria.

Para avaliar algo é necessário saber que características a avaliar. No caso deste projeto o objetivo era ter uma taxa de erro inferior a vinte por cento, tempo de geração automática de código inferior a um minuto e aumentar o grau de satisfação dos utilizadores da plataforma.

7.1 Metodologia de avaliação

Para avaliar a solução é necessário recolher dados que nos permitam tirar conclusões após uma análise cuidada desses mesmos dados. De modo a saber se os desenvolvedores estão mais satisfeitos com o novo sistema, será necessário primeiro elaborar inquéritos.

O inquérito será constituído essencialmente por três perguntas que comparam o processo atual com o novo processo, o questionário pode ser observado na secção Anexos.

As respostas a estas questões são de escolha múltipla, sendo as possibilidades totalmente desadequado, desadequado, normal, adequado e totalmente adequado.

Para avaliar a taxa de erros é necessário recolher informação sobre os objetos totais identificados e os objetos detetados incorretamente para cada maquete inserida na plataforma. A taxa de erros é dada pela fórmula seguinte

$$Taxa\ de\ erros = \frac{Nº\ de\ objetos\ detetados\ incorretamente}{Nº\ de\ objetos\ detetados} \times 100\%$$

De forma semelhante à taxa de erros, também é necessário para cada maquete inserida no sistema, guardar o tempo que demorou a gerar o código correspondente.

7.2 Hipóteses a testar

As hipóteses que serão testadas de modo a garantir que o projeto a ser desenvolvido cumpriu os objetivos propostos são:

- Taxa de erro inferior a 20%

- Tempo de geração de código inferior a 1 minuto
- Aumento da satisfação dos desenvolvedores com o novo processo

7.3 Testes estatísticos

Teste de hipóteses é um procedimento estatístico que permite rejeitar ou aceitar uma hipótese sobre uma população com base numa amostra com um certo grau de confiança. Para realizar este tipo de teste é aconselhado a ter uma dimensão da amostra superior a trinta.

A hipótese nula H_0 é a hipótese assumida como verdadeira para a construção dos testes e a hipótese alternativa H_1 é a hipótese que se pretende concluir que é verdadeira com base nos dados recolhidos da amostra.

Após o cálculo das hipóteses, usa-se o p-value de modo a rejeitar ou não rejeitar a hipótese nula. No caso de o p-value ser menor que o nível de significância então pode-se rejeitar a hipóteses nula com um determinado grau de confiança. Quando rejeitamos a hipótese nula, obtem-se a prova estatística de que a alternativa é verdadeira. Também é comum usar o grau de significância de 0.05.

Uma vez que os testes são baseados em probabilidades pode-se cometer erros ao tomar as decisões, mesmo que seja com um grau de confiança elevado. Quando ocorre um erro num teste estatístico pode ser de dois tipos, Tipo I e Tipo II. Na Tabela 15 pode-se verificar como as decisões tomadas podem levar aos diferentes tipos de erros.

Tabela 15 – Decisões que podem ser tomadas num teste de hipóteses

	Verdade da população	
Decisão	**H_0 é verdade**	**H_0 é falso**
Não rejeitar H_0	correto	Erro tipo II
Rejeitar H_0	Erro tipo I	correto

Para comparar a média da amostra com um determinado valor, como é o exemplo da taxa de erros e o tempo de geração de código, será utilizado o teste de hipóteses, *one-sample t-teste* ou o teste de Wilcoxon. No entanto, para comparar a média de satisfação do processo transformação de maquetes em código que existe atualmente com o novo processo, será usado o teste *paired t-test*. A utilização de um tipo de teste ou de outro irá depender se a amostra respeita os pressupostos do teste t-test que são:

1. Segue uma distribuição normal
2. A amostra não deve ter *outliers*

Para avaliar se as amostras seguem uma distribuição normal, será usado o teste de Shapiro e para garantir que não existem *outliers* serão verificados os dados antes de os testar para procurar possíveis *outliers* e removê-los da nossa amostra.

Para realizar os testes referidos anteriormente será utilizada a ferramenta R Studio.

7.3.1 Taxa de erros

Definindo as hipóteses para a taxa de erro tem-se:

$$H_0: \mu = 20\%$$

$$H_1: \mu < 20\%$$

Para testar se a taxa de erros está abaixo de 20% será aplicado o método *one sample t-test*.

7.3.2 Tempo de geração de código

Definindo as hipóteses para o tempo de geração de código tem-se:

$$H_0: \mu = 60 \text{ sec}$$

$$H_1: \mu < 60 \text{ sec}$$

Para testar se o tempo de geração de código é menor que 1 minuto será aplicado o método *one sample t-test*.

7.3.3 Satisfação dos desenvolvedores

Para analisar os resultados dos inquéritos será atribuído um valor numérico a cada resposta de modo a ser possível criar uma média das repostas obtidas. A correspondência entre a escolha do inquérito e o seu valor numérico pode ser observado na Tabela 16.

Valor no Inquérito	Valor numérico
Totalmente desadequado	1
Desadequado	2
Normal	3
Adequado	4
Totalmente adequado	5

Tabela 16 - Correspondência entre a escolha do inquérito e o seu valor numérico.

Definindo as hipóteses para avaliar a satisfação dos desenvolvedores tem-se:

$$H_0: \mu_{pa} = \mu_{np}$$

$$H_1: \mu_{pa} < \mu_{np}$$

Sendo que μ_{pa} representa a média das respostas relativamente ao processo atual com a implementação do presente projeto e μ_{np} representa a média das respostas relativamente ao novo processo.

7.4 Resultados

Nesta secção será verificado se realmente a solução implementada cumpre os objetivos propostos recorrendo a testes de hipóteses para comprovar.

7.4.1 Taxa de erros

Para determinar a taxa de erros, foram geradas dez maquetes através do modelo em produção, e posteriormente comparado com os *tokens* que este era suposto gerar. A Tabela 17 apresenta os dados recolhidos.

Taxa de erros	Nº componentes errados	Nº componentes
40.81%	20	49
0.00%	0	26
52.27%	29	55
38.46%	10	26
6.49%	5	77
3.84%	1	26
0.00%	0	26
0.00%	0	14
16.67%	6	36
0.00%	0	27

Tabela 17 – Taxa de erros para as maquetes testadas

A média da taxa de erros para os dados recolhidos é de 15.85%, ao que indica a solução desenvolvida tem uma taxa de erros inferior ao objetivo de 20%. Mas para comprovar esta observação será feito um teste de hipóteses. Sendo assim será necessário verificar primeiro se a amostra segue uma distribuição normal. Sendo assim as hipóteses são as seguintes:

H_0: segue uma distribuição normal

H_1: não segue uma distribuição normal

Usando o teste de Shapiro, é obtido o valor de p-value de 0.009004. Como p-value é inferior ao nível de significância de 0.05, rejeita-se a hipótese nula, H_0, pelo que se pode afirmar com 95% de confiança que a amostra não segue uma distribuição normal.

Como a amostra não segue uma distribuição normal, será necessário usar o teste de Wilcoxon para comparar a média da amostra com a taxa de erros definida como objetivo. As hipóteses são:

$$H_0: \mu = 20\%$$

$$H_1: \mu < 20\%$$

Aplicando o teste de Wilcoxon, o valor de p-value é de 0.3408. Como p-value < 0.05 não se rejeita a hipótese nula, H_0, pelo que não se pode afirmar que a média de taxa de erros para a amostra não é significativamente inferior a 20%.

7.4.2 Tempo de geração

De modo a verificar se o tempo de geração do código é inferior a 60 segundos como definido nos objetivos, fizemos a conversão de dez maquetes de modo a recolher informação sobre o tempo de geração. No fim da conversão das maquetes os resultados obtidos são os que estão representados na Tabela 18.

16.17	16.76	18.23	16.73	15.20	21.87	11.73	18.41	15.88	16.70

Tabela 18 – Tempos da codificação automática de maquetes

Analisando os tempos recolhidos, parece que o resultado foi substancialmente abaixo do objetivo de 60 segundos uma vez que a média dos tempos recolhidos foi de 16.77 segundos. De modo a confirmar a suspeita anterior, será feito um teste de hipótese.

Antes de realizar o teste para verificar se realmente a média é inferior a 60 segundos, é necessário realizar o teste que verifica se os dados têm uma distribuição normal. Sendo assim as hipóteses são as seguintes:

H_0: segue uma distribuição normal

H_1: não segue uma distribuição normal

Aplicando o teste de Shapiro pode-se verificar que o p-value é 0.4235. Como o p-value é superior ao nível de significância de 0.05, p-value > 0.05, não se rejeita a hipótese nula, H_0, pelo que se pode afirmar com um grau de confiança de 95% que a amostra segue uma distribuição normal.

Com o pressuposto que a amostra segue uma distribuição normal confirmado, será realizado o teste de médias, usando o teste t-test. Formulando as hipóteses tem-se:

$$H_0: \mu = 60 \text{ sec}$$

$$H_1: \mu < 60 \text{ sec}$$

Após realizar o teste verifica-se que o p-value é de 7.6e-13. Como p-value < 0.05, rejeita-se a hipóteses nula, pelo que se pode afirmar com um grau de confiança de 95% que o tempo de geração do código é inferior a 60 segundos, tal como se queria provar.

7.4.3 Satisfação dos desenvolvedores

Tal como descrito na secção 7.3.3, foi realizado um inquérito de forma a apurar a satisfação dos desenvolvedores com o processo existente atualmente na Glintt-HS de codificação a partir de maquete e o processo desenvolvido de conversão automática de maquetes para código.

De modo a recolher esta informação foi elaborado um questionário através da ferramenta Google Forms e enviado para 16 colaboradores através de email, que durante a resposta ao questionário foram acompanhados para que pudessem pedir explicações sobre algum dos pontos referidos no questionário. Após a recolha e tratamento das respostas os resultados foram os seguintes:

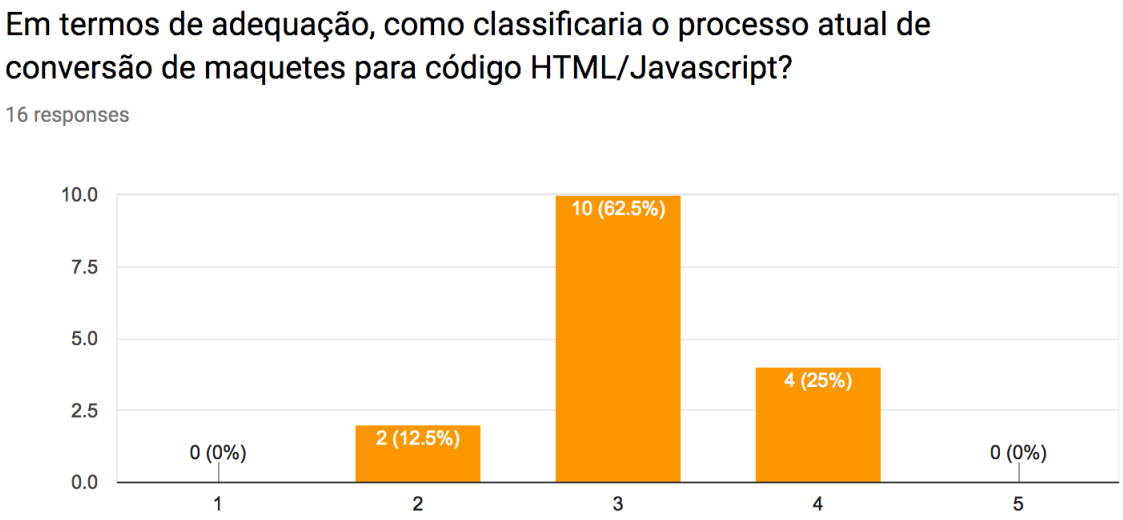


Figura 49 – Pertinência do processo atual da Glintt-HS

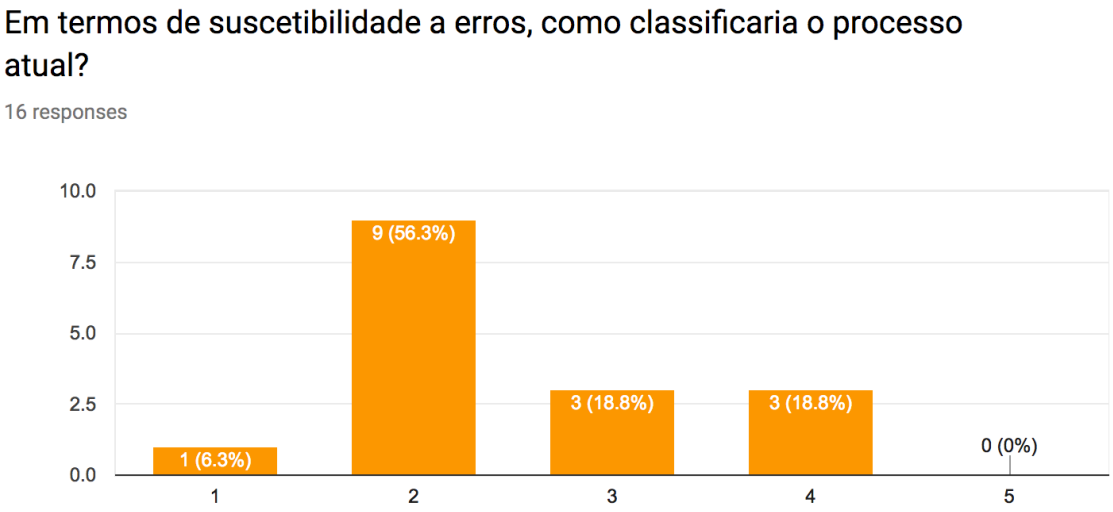


Figura 50 – Opinião sobre a suscetibilidade a erros do processo atual da Glintt-HS

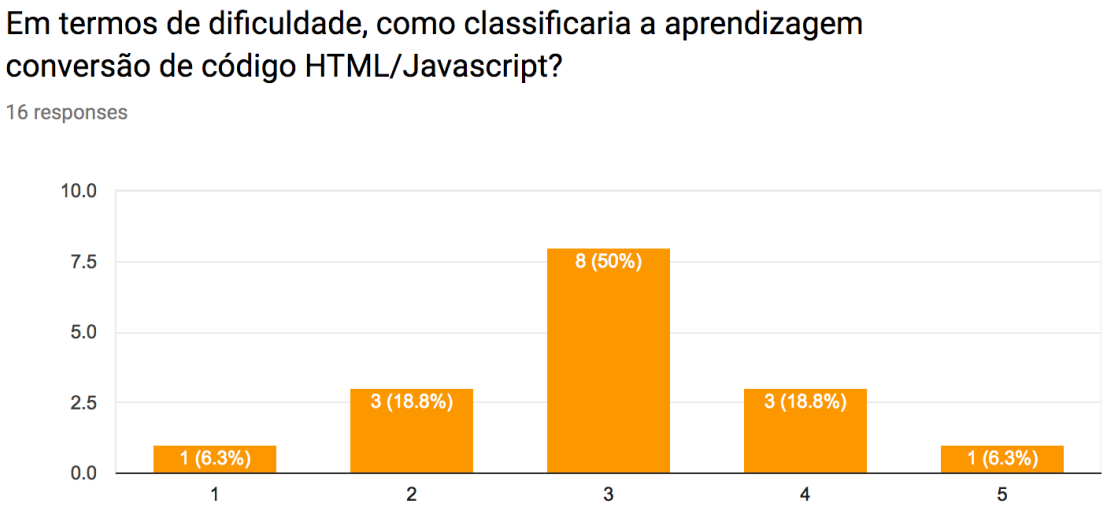


Figura 51 – Dificuldade de aprendizagem do processo atual da Glintt-HS

Como se pode verificar pela Figura 49, os inquiridos revelam que o processo de conversão de maquetes para código atualmente existente na Glintt-HS é indiferente em termos de adequação. No que diz respeito à suscetibilidade a erros do processo atual, Figura 50, os inquiridos parecem estar de acordo que este processo é mais suscetível a erros que o normal. Ainda relativamente ao processo implementado atualmente, os inquiridos revelam que a dificuldade de aprendizagem deste processo é normal, como se pode verificar através da Figura 51.

Em termos de adequação, como classificaria o novo processo de conversão de maquetes para código HTML/Javascript?

16 responses

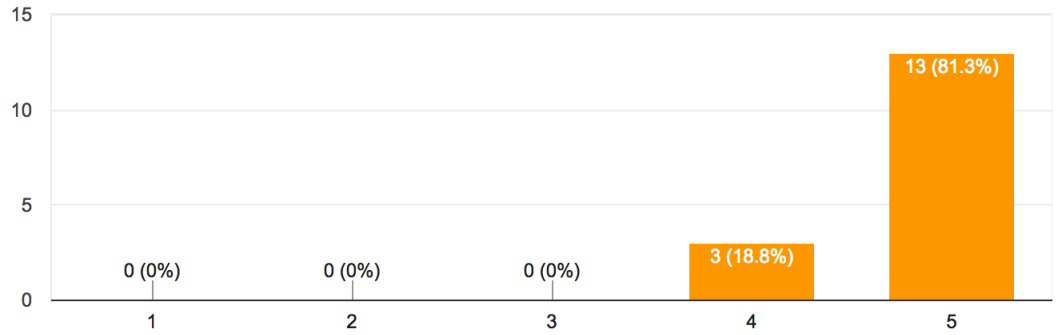


Figura 52 – Adequação do novo processo

Em termos de suscetibilidade a erros, como classificaria o novo processo?

16 responses

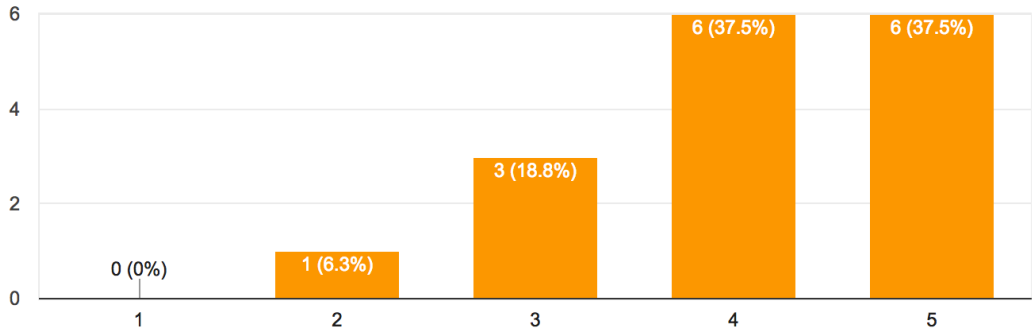


Figura 53 – Opinião sobre as suscetibilidade a erros do novo processo

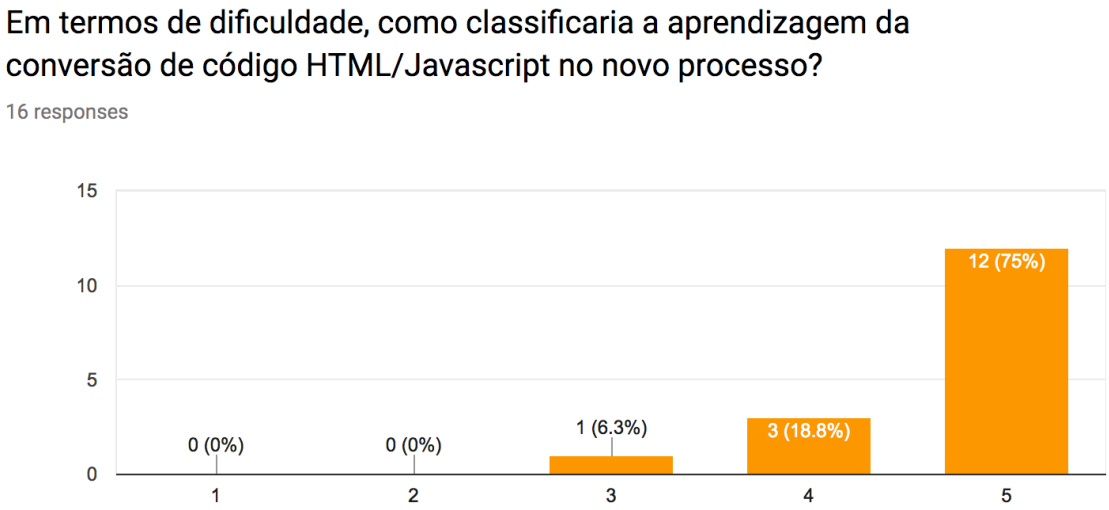


Figura 54 – Dificuldade de aprendizagem do novo processo

Relativamente ao novo processo de conversão automática de maquetes para código, a maioria dos inquiridos revela que acha este processo é muito adequado, como se pode observar pela Figura 52. Segundo a opinião dos inquiridos relativamente à suscetibilidade de erros deste novo processo, a maioria parece concordar que é menos suscetível a erros do que o normal ou então que é pouco suscetível. Finalmente no parâmetro de dificuldade de aprendizagem do novo processo, Figura 53, a maioria dos inquiridos respondeu que é muito fácil o processo de aprendizagem.

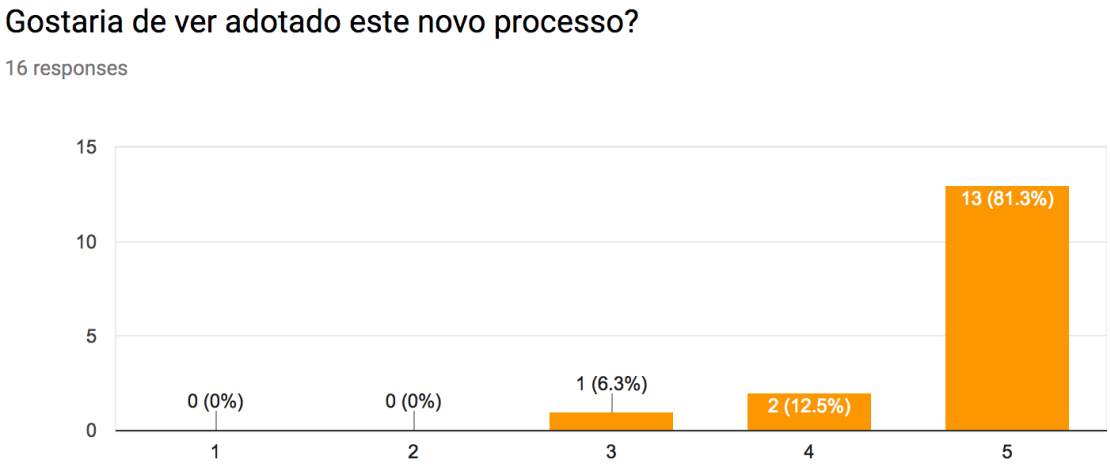


Figura 55 – Desejo de ver o novo processo implementado na Glintt-HS

Para além das questões que permitem avaliar os processos em termos de dificuldade de aprendizagem, suscetibilidade a erros e adequação ao trabalho que está a ser realizado, também foi questionado se os inquiridos gostariam de ver este novo processo implementado na Glintt-HS. Como se pode verificar pela Figura 55, é quase unânime que os inquiridos gostariam de ver este novo processo adotado pela Glintt-HS, sendo que nenhum dos

inquiridos manifestou a vontade de continuar com o processo existente atualmente e apenas um se mostrou indiferente quando ao processo a ser utilizado.

De modo a comprovar qual dos processos obteve melhor resultados para cada um dos parâmetros de avaliação, iremos realizar alguns testes estatísticos.

7.4.3.1 Adequação

As hipóteses para o teste de Shapiro são iguais tanto para as respostas sobre a adequação do processo atual como para as mesmas respostas sobre o processo de conversão automática. Sendo assim as hipóteses são as seguintes:

H_0: segue uma distribuição normal

H_1: não segue uma distribuição normal

Para as respostas sobre a adequação do processo atual, o p-value do teste de Shapiro foi 0.001421. Usando o nível de significância de 0.05, como p-value < 0.05, rejeita-se a hipótese nula, pelo que se pode afirmar com 95% de confiança que a população não segue uma distribuição normal.

Como uma das amostras já não segue uma distribuição normal, invalida a utilização do *t-test*. Como as populações são dependentes, uma vez que foram respondidas pelas mesmas pessoas e as variáveis são do tipo nominal, pode-se usar o teste de Wilcoxon, que permite comparar a média de duas populações sem o pressuposto que estas sigam uma distribuição normal.

Definindo as hipóteses para a adequação do processo tem-se:

H_0: $\mu_{pa} = \mu_{np}$

H_1: $\mu_{pa} < \mu_{np}$

Recorrendo ao teste de Wilcoxon, foi obtido o p-value de 6.271e-07, para o nível de significância de 0.05, verifica-se que p-value < 0.05, pelo se rejeita a hipótese nula, H_0, e por isso pode-se afirmar com 95% de confiança que os inquiridos consideram o processo de conversão automática mais adequado do que o processo usado atualmente na Glintt-HS.

7.4.3.2 Suscetibilidade a erros

As hipóteses para o teste de Shapiro são iguais tanto para as respostas sobre a suscetibilidade a erros do processo atual como para as mesmas respostas sobre o processo de conversão automática. Sendo assim as hipóteses são as seguintes:

H_0: segue uma distribuição normal

H_1: não segue uma distribuição normal

Para as respostas sobre a suscetibilidade a erros do processo atual, o p-value do teste de Shapiro foi 0.003725. Usando o nível de significância de 0.05, como p-value < 0.05, rejeita-se a hipótese nula, pelo que se pode afirmar com 95% de confiança que a população não segue uma distribuição normal.

Como uma das amostras já não segue uma distribuição normal, invalida a utilização do t-test. Como as populações são dependentes, uma vez que foram respondidas pelas mesmas pessoas e as variáveis são do tipo nominal, pode ser usado o teste de Wilcoxon, que permite comparar a média de duas populações sem o pressuposto que estas sigam uma distribuição normal.

Definindo as hipóteses para a suscetibilidade a erros tem-se:

$$H_0: \mu_{pa} = \mu_{np}$$

$$H_1: \mu_{pa} < \mu_{np}$$

Recorrendo ao teste de Wilcoxon, foi obtido o p-value de 0.000115, para o nível de significância de 0.05, verifica-se que p-value < 0.05, pelo se rejeita a hipótese nula, H_0, e por isso pode-se afirmar com 95% de confiança que os inquiridos consideram o processo de conversão automática menos suscetível a erros do que o processo usado atualmente na Glintt-HS.

7.4.3.3 Facilidade de aprendizagem

As hipóteses para o teste de Shapiro são iguais tanto para as respostas sobre a facilidade de aprendizagem do processo atual como para as mesmas respostas sobre o processo de conversão automática. Sendo assim as hipóteses são as seguintes:

H_0: segue uma distribuição normal

H_1: não segue uma distribuição normal

Para as respostas sobre a facilidade de aprendizagem do processo de conversão automática, o p-value do teste de Shapiro foi 0.1221. Usando o nível de significância de 0.05, como p-value > 0.05, não se rejeita a hipótese nula, pelo que se pode afirmar com 95% de confiança que a população segue uma distribuição normal.

Para as respostas sobre a facilidade de aprendizagem do processo de conversão automática, o p-value do teste de Shapiro foi 1.213e-05. Usando o nível de significância de 0.05, como p-value < 0.05, rejeita-se a hipótese nula, pelo que se pode afirmar com 95% de confiança que a população não segue uma distribuição normal.

Como a amostra sobre o processo de conversão automática não segue uma distribuição normal, invalida a utilização do t-test. Como as populações são dependentes, uma vez que foram respondidas pelas mesmas pessoas e as variáveis são do tipo nominal, pode ser usado o

teste de Wilcoxon, que permite comparar a média de duas populações sem o pressuposto que estas sigam uma distribuição normal.

Definindo as hipóteses para a facilidade de aprendizagem tem-se:

$$H_0: \mu_{pa} = \mu_{np}$$

$$H_1: \mu_{pa} < \mu_{np}$$

Recorrendo ao teste de Wilcoxon, foi obtido o p-value de 1.326e-05, para o nível de significância de 0.05, verifica-se que p-value < 0.05, pelo se rejeita a hipótese nula, H_0, e por isso pode-se afirmar com 95% de confiança que os inquiridos consideram o processo de conversão automática mais fácil de aprender do que o processo usado atualmente na Glintt-HS.

8. Conclusão

Este trabalho permitiu explorar uma área que ainda não tinha tido a oportunidade de estudar o *Deep Learning*. O estudo realizado em Deep Learning permitiu-me aumentar o grau de competências e despertar ainda mais o meu interesse nesta área.

Também foram adquiridos conhecimentos em novas metodologias de análise de valor e processo de inovação que serão certamente úteis para o processo de desenvolvimento de novos projetos.

Os resultados da avaliação mostram que os desenvolvedores gostariam de ver esta solução aplicada à Glintt-HS e consideram-na melhor em termos de aprendizagem, adequação e fiabilidade. A solução permite ao desenvolvedor ter o código base de uma maquete implementada em menos de 60 segundos, o que reduzirá muito o tempo de desenvolvimento, apesar de ser sempre necessário fazer alterações no código, como por exemplo, mudar as descrições textuais. A taxa de erros não é significativamente inferior a 20%, no entanto, com mais maquetes recolhidas, com mais estruturas testadas e mais tempo de treino, é possível atingir este objetivo.

A falta de hardware adequado acabou por ser um fator determinante para o desenvolvimento do projeto, uma vez que os recursos limitados acabaram por atrasar a obtenção dos resultados e também não permitiram o uso adequado de algumas técnicas mais eficientes, como a pesquisa de hyperparameters.

Grande parte do tempo foi também alocado à pesquisa e à aprendizagem na área de *Machine Learning*, mais concretamente na área de *Deep Learning*, se este conhecimento já tivesse sido adquirido anteriormente teria sido possível alocar mais tempo a outras áreas. Todo o conhecimento foi adquirido de forma autónoma, através da leitura de artigos científicos, cursos online e também com ajuda dos orientadores.

Apesar do trabalho desenvolvido apresentar um desempenho razoável, seria interessante, após o investimento de hardware adequado, experimentar outras arquiteturas, continuando a testar novos hyperparameters através do método Bayesian Optimization como no exemplo da secção 6.6.3.4. De forma a conseguir ter melhor resultados no futuro seria, também, fundamental a recolha de mais maquetes.

Também seria interessante verificar se a utilização de uma CNN nativa, como por exemplo uma das opções avaliadas na secção 2.4, iria realmente alavancar a performance do modelo, tal como previsto por Wallner (Wallner, 2018).

Após a adoção deste processo e já consolidado o processo de conversão de maquetes para código, também faria todo o sentido fazer a conversão direta de wireframes para código, reduzindo ainda mais o tempo entre a ideia e a construção da mesma.

Cada vez mais empresas de renome mundial apostam nesta área, como por exemplo a Airbnb e mais recentemente a Microsoft, por isso a Glintt-HS com a adoção desta solução está também a ser pioneira nesta área não só a nível nacional como internacional.

Bibliografia

Beltramelli, T. (2017). pix2code: Generating Code from a Graphical User Interface Screenshot. 2.

Bergstra, J., & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization.

Britz, D. (17 de Setembro de 2015). *Recurrent Neural Networks Tutorial*. Obtido em 13 de Fevereiro de 2018, de WILDML: http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

Britz, D. (7 de Novembro de 2015). *Understanding Convolutional Neural Networks for NLP*. Obtido em 11 de Fevereiro de 2018, de WILDML: http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

Brochu, E., Cora, V., & Freitas, N. d. (2010). A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning.

Chlis, N. K. (2013). *Comparison of Statistical Methods for Genomic Signature Extraction.*

E.Rumelhart, D., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagationg errors. *Nature* , 533.

flow.ci. (2016). *GitHub vs. Bitbucket vs. GitLab vs. Coding*. Obtido de Medium: https://medium.com/flow-ci/github-vs-bitbucket-vs-gitlab-vs-coding-7cf2b43888a1

Girshick, R. (2015). Fast R-CNN.

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2013). Rich feature hierarchies for accurate object detection and semantic segmentation.

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask R-CNN.

Hornik, K. (4 de Outubro de 2017). *Frequently Asked Questions on R*. Obtido em 3 de Fevereiro de 2018, de R FAQ: https://cran.r-project.org/doc/FAQ/R-FAQ.html#What-is-R_003f

Keras. (12 de Janeiro de 2018). *Keras Documentation*. Obtido em 5 de Fevereiro de 2018, de Keras: https://keras.io/

Koen, P. (2001). Providing clarity and a common language to the “Fuzzy Fron End”. *Taylor & Francis* , 47.

Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional.

Ng, A. (24 de Setembro de 2017). *CS229: Machine Learning*. Obtido em 8 de Fevereiro de 2018, de Stanford: http://cs229.stanford.edu/materials/CS229-DeepLearning.pdf

Olah, C. (27 de Agosto de 2015). *Understanding LSTM Networks*. Obtido em 13 de Fevereiro de 2018, de colah's blog: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Osterwalder, A., & Pigneur, Y. (2010). *Business Model Generation.* JOHN WILEY AND SONS.

Osterwalder, A., & Pigneur, Y. (2003). Modelling value propositions in e-business.

Papineni, K. (2002). BLEU: a Method for Automatic Evaluation of Machine Translation.

Peng, T. (29 de Setembro de 2017). *RIP Theano*. Obtido em 5 de Fevereiro de 2018, de Synced: https://syncedreview.com/2017/09/29/rip-theano/

Radhakrishnan, P. (2017). *What are Hyperparameters ? and How to tune the Hyperparameters in a Deep Neural Network?* Obtido de Towards Data Science: https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (9 de Maio de 2016). You Only Look Once: Unified, Real-Time Object Detection. *Arxiv* .

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

Ren, S., He, K., Girshick, R., & Sun, J. (6 de Janeiro de 2016). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *Arxiv* .

Rosaen, K. (2016). Obtido de Karl Rosaen: http://karlrosaen.com/ml/learning-log/2016-06-20/

Saaty, T. (2008). Decision making with the analytic hierarchy process. *Int. J. Services Sciences* , 84.

SAVE. (2016). *About Value Engineering*. Obtido de SAVE: http://www.value-eng.org/page/AboutVE

Sharma, S. (2017). *Activation Functions: Neural Networks*. Obtido de Towards Data Science: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Snoek, J., & Larochelle, H. (2012). Practical Bayesian Optimization of Machine.

Tensorflow. (17 de Agosto de 2017). *API Documentation*. Obtido em 5 de Fevereiro de 2018, de Tensorflow: https://www.tensorflow.org/api_docs/

TIOBE. (Janeiro de 2018). *TIOBE Index*. Obtido em 3 de Fevereiro de 2018, de TIOBE Index for January 2018: https://www.tiobe.com/tiobe-index/

Wallner, E. (9 de Janeiro de 2018). *Turning Design Mockups Into Code With Deep Learning*. Obtido de https://blog.floydhub.com/turning-design-mockups-into-code-with-deep-learning/

Wirth, R., & Hipp, J. (2000). CRISP-DM: Towards a Standard Process Model for Data Mining.

Woodall, T. (2003). Conceptualising 'Value for the Customer': An Attributional, Structural.

Yann LeCun, Yoshua Bengio, & Geoffrey Hinton. (28 de Maio de 2015). Deep learning. *Nature* .

Anexos

Inquérito de satisfação

Conversão de maquetes para código

* Required

1. **Em termos de adequação, como classificaria o processo atual de conversão de maquetes para código HTML/Javascript? ***
   Mark only one oval.

	1	2	3	4	5	
Totalmente desadequado	◯	◯	◯	◯	◯	Totalmente adequado

2. **Em termos de suscetibilidade a erros, como classificaria o processo atual? ***
   Mark only one oval.

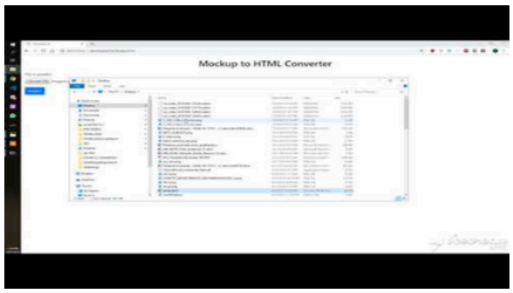
	1	2	3	4	5	
Muito suscetível	◯	◯	◯	◯	◯	Pouco suscetível

3. **Em termos de dificuldade, como classificaria a aprendizagem conversão de código HTML/Javascript? ***
   Mark only one oval.

	1	2	3	4	5	
Muito dificil	◯	◯	◯	◯	◯	Muito fácil

DeveloperAI

Demo do novo processo



http://youtube.com/watch?v=f5m5bFREMiA

4. **Em termos de adequação, como classificaria o novo processo de conversão de maquetes para código HTML/Javascript? ***
   Mark only one oval.

	1	2	3	4	5	
Totalmente desadequado	◯	◯	◯	◯	◯	Totalmente adequado

5. **Em termos de suscetibilidade a erros, como classificaria o novo processo?** *
Mark only one oval.

	1	2	3	4	5	
Muito suscetível	◯	◯	◯	◯	◯	Pouco suscetível

6. **Em termos de dificuldade, como classificaria a aprendizagem da conversão de código HTML/Javascript no novo processo?** *
Mark only one oval.

	1	2	3	4	5	
Muito dificil	◯	◯	◯	◯	◯	Muito fácil

7. **Gostaria de ver adotado este novo processo?** *
Mark only one oval.

	1	2	3	4	5	
Não gostaria	◯	◯	◯	◯	◯	Gostaria imenso