RecipeS – Sistema para Receitas Culinárias

Projeto II

João Daniel Malés Louro, 20210931

Orientadores

Filipe Miguel Bispo Fidalgo

Ângela Cristina Marques de Oliveira

Trabalho de Projeto apresentado à Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco para cumprimento dos requisitos necessários à obtenção do grau de Licenciado em Engenharia Informática, realizada sob a orientação científica do Professor Doutor Filipe Miguel Bispo Fidalgo e coorientação da Professora Doutora Ângela Cristina Marques de Oliveira, do Instituto Politécnico de Castelo Branco.

Janeiro de 2024

Composição do júri

Presidente do júri

Doutor, Osvaldo Arede dos Santos

Vogais

Doutor, Osvaldo Arede dos Santos

Professor Adjunto, Escola Superior de Tecnologia de Castelo Branco

Doutor, Eurico Ribeiro Lopes

Professor Coordenador, Escola Superior de Tecnologia de Castelo Branco

Agradecimentos

Para o desenvolvimento da segunda fase do projeto final de curso foi necessário muito esforço e dedicação, de forma a conseguir concluir todos os objetivos a que me tinha proposto.

Gostaria de agradecer à minha família por todo o apoio e por tornarem possível o término da minha licenciatura em Engenharia Informática.

Agradecer também aos meus orientadores, o Professor Doutor Filipe Fidalgo e a Professora Doutora Ângela Costa, por me encaminharem pelo caminho correto durante o desenvolvimento do projeto.

Resumo

O presente relatório foi realizado no âmbito da Unidade Curricular de Projeto II, pertencente ao 2º semestre do 3º ano da Licenciatura de Engenharia Informática da Escola Superior de Tecnologia.

O desperdício alimentar é há várias décadas uma preocupação mundial, mas só nos últimos anos se percebeu o verdadeiro impacto que causa na sociedade, uma vez que, tem impactos tanto a nível social e económico como ambiental. Todos nós sabemos que o acesso aos alimentos é muito díspar entre a população, se por um lado há pessoas que não sabem o que têm em casa porque têm uma vasta quantidade dos mais diversos alimentos, há outras pessoas que não tem mesmo nada para comer.

Por estes motivos é apresentada uma solução para combater o desperdício alimentar através da criação de uma aplicação mobile que implementa 2 *CNN's*, a 1ª é o modelo ResNet-50 treinado no *dataset* criado em Projeto I, sendo capaz de identificar 30 alimentos distintos, mas apenas um de cada vez, visto que é um modelo de classificação. A 2ª *CNN* permite que o utilizador detete vários alimentos simultaneamente, porém foi treinado apenas para as classes Maça e Ovo.

Caso o utilizador pretenda utilizar o modelo de classificação a aplicação permite-lhe fazer upload de uma fotografia ou utilizar a câmera do smartphone para tirar uma foto, por outro lado, se desejar utilizar o modelo de deteção a câmera é iniciada e os alimentos começam a ser detetados automaticamente. De seguida, após a identificação dos alimentos, são lhe sugeridas uma série de receitas recorrendo à *API* da OpenAI.

A aplicação possui ainda uma 2ª zona onde o utilizador pode procurar por alimentos e ser-lhe-ão apresentados os preços desse produto em diferentes supermercados, permitindo-lhe adicionar ao carrinho para fazer uma estimativa de custo.

Ao longo deste relatório serão explicados ambos os modelos utilizados e os resultados obtidos. Para além disso é descrito em detalhe o mecanismo utilizado para obter os preços dos vários supermercados e ainda será documentado todo o trabalho referente ao desenvolvimento da aplicação que teve como base uma investigação extensiva de modo a proporcionar uma experiência agradável ao utilizador.

Palavras-chave

Rede Neuronal Convolucional

Reconhecimento de Alimentos

Recomendação de Receitas

Abstract

This study was conducted as a component of the Project II curriculum unit, which is part of the second semester of the third year of the Computer Engineering degree program at the School of Technology.

Food waste has been a pervasive global issue for many years, but its true consequences on society have only been acknowledged in recent times, as it has significant social, economic, and environmental ramifications. It is widely acknowledged that there is a significant disparity in food accessibility among different segments of the population. Some individuals possess an abundance of various food items in their homes, although remain unaware of their inventory. Conversely, some individuals lack any sustenance at all.

To address this issue, a proposed solution involves developing a mobile application that incorporates two Convolutional Neural Networks (CNNs) to tackle food waste. The ResNet-50 model, trained on the dataset from Project I, is capable of classifying 30 distinct food items. However, it can only recognize one food item at a time due to its categorization nature. The second convolutional neural network (CNN) enables the user to simultaneously recognize many food items, albeit it has just undergone training for the apple and egg categories.

If the user desires to utilize the classification model, the program provides the option to either upload a photo or utilize the smartphone's camera to capture an image. Conversely, if the user wishes to employ the detection model, the camera is activated and automatically detects the food. After the food is detected, the OpenAI API generates a sequence of recipes as suggestions.

The application also features a second part where the user may search for food and check the pricing of that goods in other supermarkets, allowing them to add it to their shopping basket to estimate the cost.

This report outlines the two models employed and the outcomes achieved. Additionally, it provides a comprehensive explanation of the methodology employed to gather pricing from various supermarkets and thoroughly chronicles the entire process of designing the program, which was built upon significant research to ensure a user-friendly interface.

Keywords

Convolutional Neural Network

Food Recognition

Recipe Recommendations

Índice Geral

Índice de figuras

Lista de abreviaturas, siglas e acrónimos

API - Application Programming Interface

APK - Android Package Kit

BERT - Bidirectional Encoder Representations from Transformers

CNN - Convolutional Neural Network

GPT - Generative Pre-trained Transformer

GPU - Graphics Processing Units

IA – Inteligência Artificial

iOS - iPhone Operating System

LLM - Large Language Model

mAP - mean Average Precision

ML – Machine Learning

NLP - Natural Language Processing

SOTA – State of the art

UI – User Interface

UML - Unified Modeling Language

1. Introdução

Atualmente existem diversas aplicações móveis e especialmente *websites* que contêm milhares de receitas, a sugestão das mesmas baseia-se em palavras-chave onde o utilizador apenas tem de selecionar os alimentos que deseja incluir e são lhe apresentados um conjunto de receitas. Muitas das vezes o problema destas aplicações está no facto das receitas sugeridas conterem alimentos que o utilizador não selecionou, e, portanto, não contribuem para o combate ao desperdício alimentar, mas sim para o consumismo, uma vez que, na maioria das vezes o utilizador acaba por comprar os ingredientes em falta.

Para ajudar os utilizadores a evitar estes ajustes, a seleção dos alimentos pode ser feita em tempo real identificando apenas os ingredientes que o utilizador tem à sua disposição no momento, e posteriormente recomendar diversas receitas.

A principal vantagem deste sistema de recomendação de receitas através de um modelo *GPT (Generative Pre-trained Transformer)* da *OpenAI* é o facto que as sugestões têm em conta as características do utilizador (peso, idade), sendo um enorme diferencial para a maioria das *API's* no campo da culinária que apenas fazem uma procura básica na base de dados tendo em conta os alimentos selecionados. Isto pode não só facilitar o processo de criação de uma refeição, que pode ser bastante exaustivo pois é feita todos os dias, como incentivar o utilizador a mudar de hábitos alimentares.

Para além disso foi decidido incorporar na app uma funcionalidade que permite ao utilizador visualizar os preços do supermercado recorrendo a uma *API* para o efeito.

1.1 Enquadramento

O seguinte relatório foi realizado no âmbito da Unidade Curricular Projeto II da licenciatura de Engenharia Informática, inicialmente em Projeto I, foi feita uma pesquisa e análise de vários artigos científicos encontrados através do acesso a vários repositórios. A partir dessa pesquisa foi possível entender as características dos modelos mais utilizados para o reconhecimento de alimentos, bem como o tipo de abordagens utilizadas para realizar a geração de receitas com base no alimento reconhecido.

Serão apresentados ambos os modelos, classificação e deteção, bem como os testes realizados sobre o *dataset* criado em Projeto I para comprovar que esta abordagem poderá ser implementada numa aplicação móvel com excelente desempenho. Fazer-se-á ainda a descrição das funcionalidades da aplicação e o processo necessário para implementar as mesmas.

1.2 Objetivos

O principal objetivo deste projeto é a criação de uma aplicação móvel que permita o reconhecimento de alimentos seguida da recomendação de receitas, e ainda uma funcionalidade extra para que o utilizador possa fazer uma estimativa de custos de uma lista de compras.

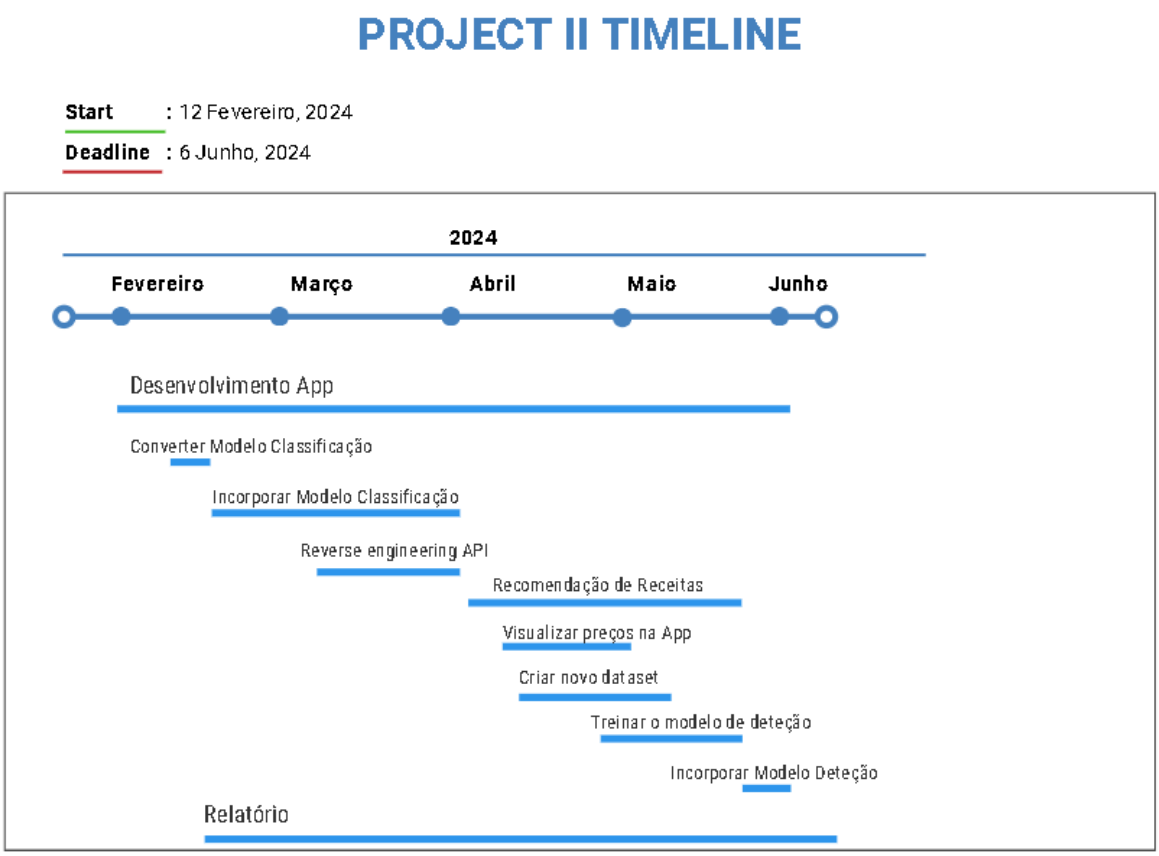
Portanto os objetivos específicos são:

- Treinar o modelo de classificação no *dataset* criado em Projeto I;
- Treinar o modelo de deteção em tempo real num novo *dataset*;
- Verificar a eficácia dos modelos;
- Criar uma aplicação *mobile*;
- Implementar a recomendação de receitas;
- Implementar a funcionalidade de consulta dos preços;
- Documentar o desenvolvimento do projeto;

1.3 Planeamento

O projeto II teve como data de início Fevereiro de 2024 e como data de finalização Junho de 2024.

Tal como sugere a Figura 1 o desenvolvimento da aplicação, assim como do relatório foi efetuado durante todo o prazo do projeto, ou seja, entre os meses de Fevereiro e Junho. De forma paralela foram efetuadas várias tarefas para que a aplicação cumprisse com todos os objetivos propostos. Assim nos meses de Fevereiro e Março, o modelo que tinha sido previamente treinado em Projeto I foi convertido para o formato *tflite*, de forma que pudesse ser integrado numa aplicação móvel. Neste intervalo de tempo o modelo foi incorporado na aplicação e programaram-se os métodos necessários para realizar a classificação dos alimentos. No mês de março foi realizado uma pesquisa com o objetivo de conseguir obter os preços dos produtos nos diversos supermercados, sendo atingida com sucesso a partir de um método de *reverse engineering* feito a uma *API* privada que será detalhado na Capítulo 3. No mês de Abril fez-se novamente uma pesquisa, mas desta vez para tentar implementar a melhorar abordagem que recomendasse receitas personalizadas, neste mesmo mês implementou-se na aplicação a chamada à *API* que obtém os preços do supermercado e criou-se um novo *dataset*, através da anotação de imagens para que a aplicação, para além de classificar alimentos conseguisse detetá-los em tempo real. Já no mês de Maio foi treinado um modelo de deteção de alimentos e foram finalizadas as tarefas de criação do *dataset* e a abordagem para a recomendação de receitas. Por fim a última tarefa foi incorporar o modelo de deteção na aplicação tendo acabado no início de Junho.



LABEL	TASK	START	END
Desenvolvimento App	Desenvolvimento da aplicação móvel	12/02/2024	01/06/2024
Relatório	Desenvolvimento do Relatório	18/02/2024	05/06/2024
Converter Modelo Classificação	Converter o modelo para TFLite	16/02/2024	18/02/2024
Incorporar Modelo Classificação	Integrar o modelo na app	19/02/2024	31/03/2024
Reverse engineering API	Reverse engineering da API que obtém os preços do supermercado	07/03/2024	01/04/2024
Recomendação de Receitas	Implementar metodologia para Sugestão de Receitas Personalizadas	02/04/2024	15/05/2024
Visualizar preços na App	Integrar a chamada à API de consulta dos preços na app	08/04/2024	30/04/2024
Criar novo dataset	Anotar imagens de alimentos para a deteção em tempo real	15/04/2024	02/05/2024
Treinar o modelo de deteção	Treinar o modelo de deteção e verificar a sua eficácia	29/04/2024	15/05/2024
Incorporar modelo deteção	Converter o modelo para TFLite e integrá-lo na app	15/05/2024	01/06/2024

Figura 1 - Cronograma Projeto II

1.4 Estrutura do Relatório

O presente relatório é composto por 9 capítulos

- No primeiro capítulo apresenta-se uma breve introdução, com a intenção de explicar de forma geral a aplicação móvel que será desenvolvida, seguido dos objetivos, planeamento e estrutura do relatório.
- No segundo capítulo são listadas todas as ferramentas utilizadas no desenvolvimento deste projeto, bem como uma breve explicação da sua aplicabilidade no projeto.

- No terceiro capítulo é detalhado todo o processo para obter os preços dos produtos nos supermercados através do uso de uma *API* privada.
- No quarto capítulo é explicado minuciosamente a implementação do modelo *ResNet-50* aplicado ao *dataset* criado em Projeto I, bem como de que forma é feita a integração na aplicação desenvolvida em *Android Studio*.
- No quinto capítulo é demonstrado cada etapa para a criação do dataset que será utilizado para o treino de um modelo de deteção. Nesta secção, para além de ser explicado o método de anotação de imagens utilizado é detalhado também o modelo aplicado ao *dataset*.
- No sexto capítulo são abordados todos os métodos que poderiam ser utilizados para a recomendação personalizada de receitas, e a justificação para ter sido selecionado o método recorrendo à *API* da *OpenAI*.
- No sétimo capítulo é detalhado de que forma as funcionalidades principais da aplicação foram implementadas em *Android Studio* e a forma como interagem com o sistema.
- O oitavo capítulo diz respeito às conclusões e sugestões de trabalho futuro.
- No nono e último capítulo estão as referências bibliográficas.

2. Ferramentas Utilizadas

Neste capítulo vão ser abordadas as ferramentas que serão utilizadas para o desenvolvimento do projeto. O critério de escolha das ferramentas utilizadas teve em consideração a informação presente na *web* e os estudos anteriormente realizados nesta área.

Kaggle

O *Kaggle* [1] é uma plataforma que faz parte do *Google Cloud* e pode ser utilizada para diversos fins, sendo que as suas principais ofertas são: competições de *data science*, *datasets open-source*, *notebooks* sobre problemas de *machine learning* e ainda tutoriais para ajudar os utilizadores a melhorar as suas competências. Ou seja, é uma plataforma extremamente versátil que pode ser usada por pessoas com diferentes níveis de conhecimento nesta área [2].

Visto que a plataforma oferece um ambiente de desenvolvimento em nuvem que inclui *GPUs (Graphics Processing Units)* para acelerar o treino de modelos, neste projeto utilizámo-la para treinar o ResNet-50 e também para obter alguns *datasets* com imagens de diversos tipos de alimentos.

Figura 2 – Logotipo do *Kaggle* (adaptado de [3])

Google Colab

O *Google Colab* [4] é um produto semelhante ao *Kaggle*, porém destaca-se em algumas funcionalidades, nomeadamente na integração com o *Google Drive* e *GitHub* e ainda pela facilidade de acesso aos *notebooks* na *cloud* [5].

Optou-se por esta plataforma por 2 razões: o *dataset* que foi necessário criar com imagens de Maças e Ovos e as respetivas anotações foi colocado no *Google Drive*, logo o acesso será facilitado e ainda porque para o treino do modelo de deteção de alimentos será utilizado *transfer learning* a partir dos modelos do *tensorflow* disponíveis no *github*, o que torna a integração bastante mais eficaz. Portanto, esta plataforma foi utilizada para treinar o modelo de deteção de alimentos *SSD MobileNet V2 FPNLite 320x320*, que será explicado mais à frente.

Figura 3 - Logotipo do *Google Colab* (adaptado de [6])

TensorFlow

O *TensorFlow* [7] foi desenvolvido pela *Google* e é a ferramenta mais popular atualmente para problemas de *machine* e *deep learning*, sendo também multi-plataforma, podendo ser executado no Windows, MacOS ou Linux, para além disso possui uma vasta comunidade que contribui diariamente para o seu repositório [8].

Neste projeto foi utilizado para construir ambos os modelo uma vez que temos acesso a diversas *APIs* que estão integradas nesta biblioteca.

Figura 4 - Logotipo do *TensorFlow* (adaptado de [9])

Python

Python está atualmente entre as linguagens de programação mais populares dado que é uma linguagem poderosa, flexível e fácil de usar. Algumas das principais vantagens são: *third-party* modules (vasta gama de bibliotecas criadas pela comunidade), suporte para vários paradigmas de programação e também a sua eficiência dado que efetua a gestão automática da memória [10]. O *Python* é utilizado em diversas aplicações nomeadamente: desenvolvimento de *software*, *Machine Learning*/Inteligência Artificial, *Big Data* e *Web Scraping* [11].

Neste projeto foi utilizado *Python* para o treino do modelo de classificação e do modelo de deteção, para além disso recorreu-se ao uso desta linguagem para a construção do script de *Web Scraping* responsável por obter imagens, que resultou no *dataset* proposto em Projeto I. Além do mais em Projeto II utilizou-se esta linguagem para realizar o *script* que faz uma solicitação *POST* para a *API* que retorna os preços do supermercado.

Figura 5 - Logotipo *Python* (adaptado de [12])

Android Studio

O *Android Studio* [13] é o IDE oficial concebido especificamente para o desenvolvimento de aplicações *Android*, para além disso permite aos programadores testarem as aplicações em diferentes dispositivos como telemóveis ou tablets, e ainda a facilidade para as lançarem na *Google Play Store*, através da compilação num ficheiro APK (Android Package Kit) ou num *Android App Bundle* [14].

Neste projeto foi utilizado para o desenvolvimento da aplicação *mobile*.

Figura 6 - Logotipo *Android Studio* (adaptado de [15])

Java

Java é uma das linguagem de programação mais utilizadas pelas empresas especialmente pelo facto de ser independente da plataforma, além do mais é uma linguagem orientada a objetos e possui recursos importantes ao nível da segurança [16].

Neste projeto foi a linguagem escolhida para desenvolver a aplicação *mobile*.

Figura 7 - Logotipo *Java* (adaptado de[17])

Firebase

A *Firebase* [18] é uma base de dados *NoSQL* baseada na infraestrutura da *Google*, fornecendo uma série de serviços aos desenvolvedores. Segundo [19], é o melhor *backend* para aplicações móveis por fornecer diversos serviços para executar apps *iOS* ou *Android*.

Neste projeto foi utilizado para armazenar a informação dos utilizadores e assim permitir tanto a adição de novos *users* como o login por *users* já registados.

Figura 8 - Logotipo *Firebase* (adaptado de [20])

OpenLabeling

Os modelos de deteção são, em geral, baseados em aprendizagem supervisionada, isto significa que o modelo será treinado num *dataset* em que as imagens estão anotadas, durante o treino o modelo aprende a mapear as imagens para as saídas corretas (localização e classes dos objetos).

Neste projeto foi utilizado a ferramenta *OpenLabeling* [21], disponível no *GitHub*, para anotar o *dataset* por ser uma ferramenta bastante prática e permitir exportar as anotações em formatos como *PASCAL VOC* ou *YOLO darknet*.

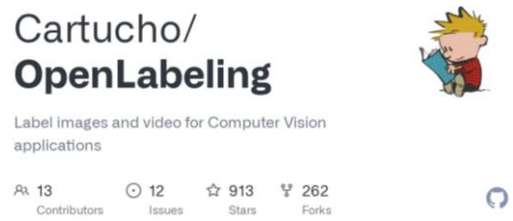


Figura 9 - Logotipo *OpenLabeling* (adaptado de [21])

Roboflow

O *Roboflow* [22] é uma ferramenta que facilita tarefas de visão computacional, onde é possível realizar: pré-processamento do *dataset*, treinar modelos, anotar imagens etc.

Neste projeto, como o *dataset* era composto por imagens de diferentes resoluções e o modelo de deteção espera imagens com resolução de 320x320 *pixels*, foi utilizado para redimensionar as imagens e ainda para obter variações de imagens de maças e ovos através de *data augmentation*.

Figura 10 - Logotipo *Roboflow* (adaptado de [22])

OpenAI API

A *OpenAI* [23] é uma empresa de investigação e desenvolvimento de modelos de inteligência artificial estando disponível através de ferramentas como o *ChatGPT*. Em termos práticos utilizar uma das *API's* disponíveis permite-nos tirar partido destes poderosos modelos de IA e beneficiar das suas capacidades de fornecer respostas baseadas em dados e garantir o uso de tecnologia de ponta na nossa aplicação [24].

Neste projeto foi utilizado a *API* da *OpenAI GPT-3.5 Turbo Instruct*, para recomendar receitas através de um *prompt* que contém as informações selecionadas pelo utilizador.

Figura 11 - Logotipo *OpenAI* (adaptado de [25])

Figma

O *Figma* [26] é uma plataforma para construção de interfaces e protótipos bastante utilizado por *designers* dado a sua capacidade de construir o *design* de *sites* ou aplicações para dispositivos móveis como *tablets*, *smartphones* e *smartwatches* [27].

Neste projeto foi utilizado para ajudar a visualizar como a interface da aplicação ficaria e que passos seriam necessários implementar para o seu desenvolvimento, nas Figura 12 e Figura 13 é possível observar como a interface foi imaginada.

Figura 12 - Página Inicial protótipo

Figura 13 - Página Compras protótipo

Postman

O *Postman* [28] é uma das ferramentas mais populares para testar *APIs*, visto que, fornece diversos recursos, nomeadamente: testes automáticos, compatibilidade entre sistemas operativos, integração com o *GitHub* e ainda a sua *UI*, fornecendo tudo isto de forma gratuita [29].

Neste projeto foi utilizado o *Postman* para testar a comunicação à *API* que obtém os preços do supermercado disponibilizados no site *Prices Crawler*.

Figura 14 - Logotipo *Postman* (adaptado de[30])

Visual Studio Code

O *VS Code* [31] é uma plataforma extremamente versátil e leve, que pode ser útil em diversas tarefas como desenvolvimento *web* e *data science*. As suas principais características são: velocidade, flexibilidade e uma vasta lista de extensões [32].

Neste projeto o *VS Code* foi utilizado para escrever o código que faz o *download* de imagens automaticamente através da biblioteca *Selenium* e do *ChromeDriver*, resultando no *dataset* proposto em Projeto I.

Figura 15 – Logotipo *VsCode* (adaptado de[33])

3. Processo para obter os preços do supermercado

Na Figura 16 está representada a arquitetura que foi necessária executar de forma a conseguir comunicar com a *API* que retorna os preços do supermercado. Uma vez que esta não está disponível publicamente, foi necessário fazer o processo de *reverse engineering* para enviar solicitações à *API*.

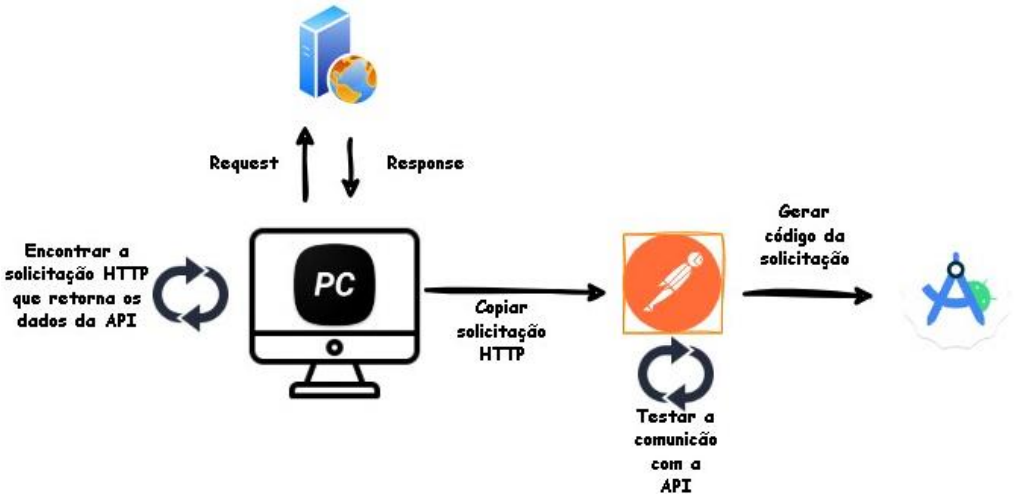


Figura 16 - *Workflow* para comunicar com a *API*

Inicialmente pesquisou-se pelo site *Prices Crawler* [34] e inspecionou-se o mesmo de forma a encontrar a solicitação que retorna os dados desejados, como mostra a Figura 17. Após isso copiou-se esta solicitação *HTTP* como *cURL* de forma a incluir o *URL* os *headers* e o corpo da solicitação. De seguida, no *Postman*, fez-se o *import* desta solicitação e testou-se a comunicação com o servidor da *API*.

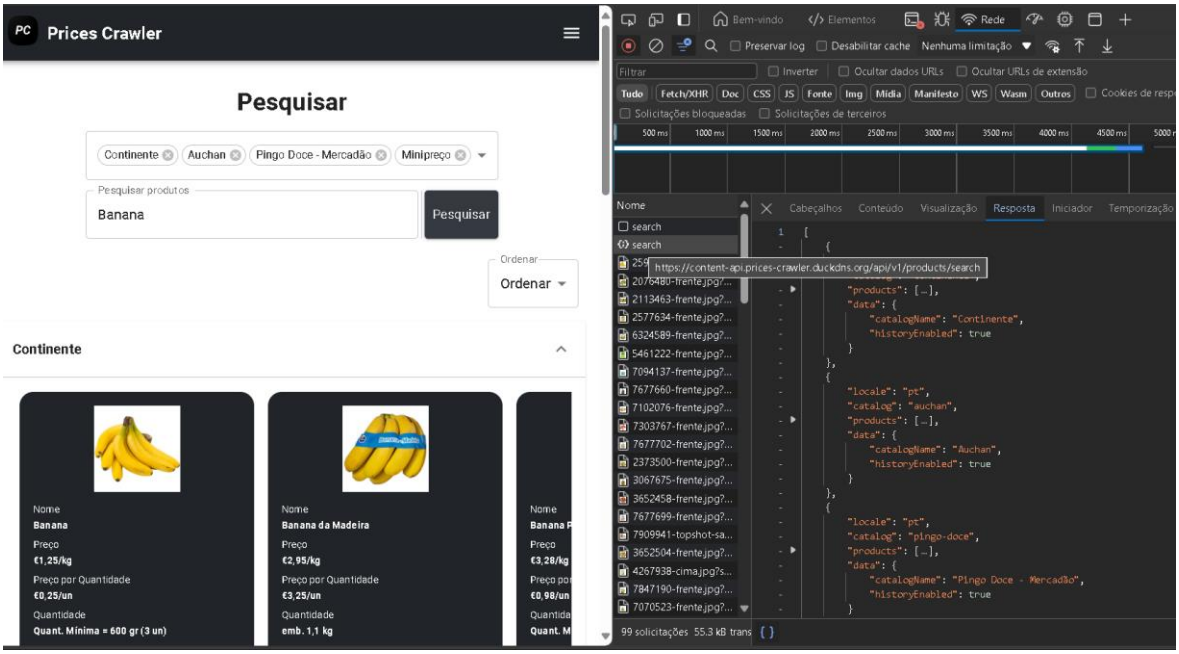


Figura 17 - Encontrar a solicitação que retorna os dados

A Figura 18 está a exibir a resposta da *API*, onde o *Postman* para além de permitir testar a comunicação com a *API* de forma bastante eficiente, ainda tem a capacidade de gerar o código da solicitação para inúmeras linguagens. Neste caso foi escolhido *Java* com a biblioteca *OkHttp*, uma vez que está a ser utilizado *Android Studio*, e esta biblioteca permite uma interação eficiente com *API's* num ambiente android [35].

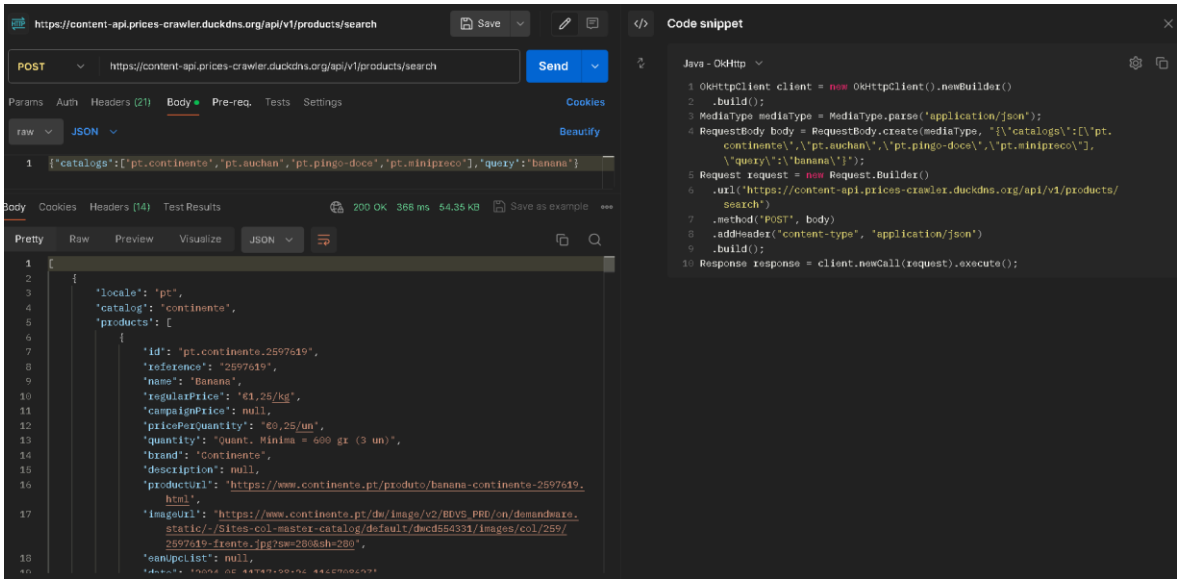


Figura 18 - Resposta da *API* e código da solicitação

Foi necessário a execução de todo este processo, porque as grandes cadeias de supermercados não disponibilizam *APIs* publicamente que partilhem a informação dos seus produtos, isto acontece, principalmente para evitar que a concorrência aumente.

O site escolhido foi o *Prices Crawler*, uma vez que contém informação dos supermercados mais populares (*Auchan*, Continente, Pingo Doce…). Este site foi feito em *React* recorrendo a *web crawlers* que fazem a extração dos dados dos supermercados em tempo real e a alguns mecanismos de *chaching* como o próprio autor revelou numa discussão na plataforma *Reddit*. Estes mecanismos de *caching* são utilizados para melhorar a eficiência e desempenho do *website*, sendo extremamente úteis neste caso pois são feitos frequentemente acessos aos mesmos dados, tornando possível recuperá-los a partir da memória em vez de serem recolhidos novamente da fonte original. Desta forma a parte mais complexa da extração dos dados dos vários supermercados é feita no *backend* do *site Prices Crawler* e para a sua utilização no *Android Studio* terá de se fazer apenas o processamento dos dados retornados pela *API*.

Apesar dos mecanismos de *Web Scraping* parecerem relativamente simples o grau de complexidade é exponencial consoante a informação que se quer extrair, como é referido em [36] há biliões de páginas *web* e estão em constante transformação o que

aumenta o desafio na implementação de algoritmos e conceção de sistemas. Este facto levanta uma questão bastante interessante, qual dos mecanismos é melhor para extrair dados *Web Scraping* ou *APIs*? Ambas as técnicas têm pontos positivos e negativos, do lado do *Web Scraping* as principais vantagens são [37]:

- Disponibilidade 24/7 ao contrário das *APIs* que podem ser encerradas/alteradas.

- Dados mais precisos uma vez que as *APIs* muitas das vezes não são a prioridade dos *sites* e podem fornecer informação obsoleta.

- Não há limite para o número de consultas como acontece em muitas *APIs* que obrigam ao pagamento de *royalties.*

- Dados melhor estruturados comparativamente a diversas *APIs* que são mal desenvolvidas exigindo pré-processamento de dados.

Até aqui o *Web Scraping* seria um claro vencedor se os desenvolvedores das empresas aos quais os dados estão a ser extraídos não arranjassem mecanismos para tentar evitar estes *crawlers*. As técnicas mais populares são: analisar a atividade dos *IPs*; colocar links invisíveis nas páginas banindo os *IPs* dos pedidos a esses *honeypots*; ou então mudar a organização da página para o *scraping* falhar. É claro que os próprios autores dos web crawlers também encontraram estratégias para evitar isto nomeadamente simular o comportamento de um utilizador (ex: pausas aleatórias entre solicitações), alterar os *IPs* utilizados pelos *web crawlers* ou projetá-los para analisar o conteúdo das páginas.

4. Modelo de Classificação e *Dataset*

Neste capítulo será apresento qual modelo e *dataset* vão ser utilizados para a tarefa de identificação dos alimentos e de que forma o modelo será exportado para ser utilizado no *Android Studio*. A Figura 19 ilustra este processo onde, inicialmente será utilizado o *dataset* criado recorrendo a mecanismos de *web scraping* e depois serão aplicadas técnicas de *data augmentation*, para que o modelo seja treinado num conjunto maior de imagens diversificadas. Por fim, após o treino do modelo estar concluído é necessário exportá-lo para *TensorFlow Lite*, que é uma *framework* de *deep learning* concebida especificamente para a eficiência em dispositivos móveis, de forma a consumir menos recursos e a reduzir o tempo de inferência do modelo [38]. Decidiu-se ainda aplicar uma técnica de quantização ao modelo *tflite*, visto que este mecanismo permite que o modelo consuma menos memória e energia bem como otimiza a velocidade do mesmo ao representar os pesos com tipos de dados mais pequenos [39].

Posteriormente serão realizados testes que comprovem a eficácia do modelo num contexto onde os recursos computacionais são limitados.

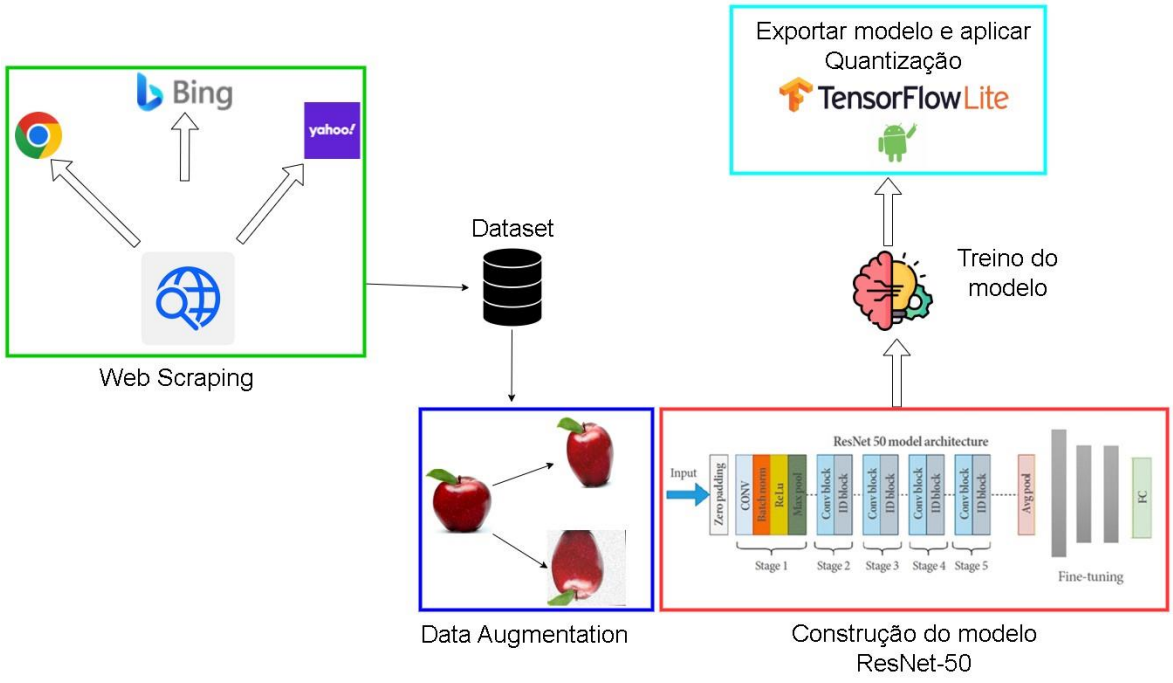


Figura 19 - Processo desde a criação do *dataset* até ao treino do modelo

4.1 Modelo

Depois de realizado o Projeto I onde foram estudadas e analisadas várias abordagens para o objetivo final, foi decidida a utilização da *CNN* ResNet-50, uma vez que os resultados comprovaram a eficácia deste modelo e, portanto, será utilizado para o desenvolvimento da aplicação *mobile*.

4.2 *Dataset*

Tal como foi referido em Projeto I, o *dataset* Food-101 não poderia ser utilizado, nomeadamente por limitações ao nível do *hardware*, por isso optou-se pela criação de um *dataset* através de *Web Scraping*, permitindo assim a seleção personalizada de alimentos.

O *dataset* resultante é constituído por 30 classes, totalizando 32020 imagens, sendo 26709 imagens para treino e 5311 para validação. Isto significa que cerca de 80% das imagens é reservada para o treino e 20% para o teste, esta divisão é extremamente utilizada num contexto de aprendizagem supervisionada e segundo [40] a utilização deste rácio é motivada pelo Princípio de Pareto, este princípio afirma que 80% do efeito é causado por 20% das causas.

4.3 Implementação do modelo ResNet-50

A construção do modelo ResNet-50, bem como o respetivo treino, foi realizado na plataforma *Kaggle*, este foi um processo iterativo que levou o modelo a ser sucessivamente ajustado com o objetivo de obter uma precisão mais elevada.

A primeira das técnicas aplicadas foi adicionar *data augmentation* ao modelo, demonstrado na Figura 20, desta forma todas as imagens serão replicadas durante o treino o que melhora a generalização do modelo e a redução do *overfitting*.

```
data_augmentation = tf.keras.Sequential([
    layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
    layers.experimental.preprocessing.RandomRotation(0.2),
    layers.experimental.preprocessing.RandomZoom(0.2),
    layers.experimental.preprocessing.RandomContrast(0.2),
    layers.experimental.preprocessing.RandomTranslation(0.2, 0.2),
    layers.experimental.preprocessing.RandomHeight(0.2),
    layers.experimental.preprocessing.RandomWidth(0.2)
])


resnet_model=Sequential()

resnet_model.add(layers.Input(shape=FIXED_SIZE))
resnet_model.add(data_augmentation)
```

Figura 20 - *Data Augmentation*

Posteriormente é adicionado ao modelo, o ResNet-50 pré-treinado no *ImageNet* (*Transfer Learning*), e de seguida são adicionadas as seguintes camadas em sequência: *Flatten*, *Dense*, *Dropout*, *Dense*; a camada *Flatten* transforma o vetor de características 3D produzido pelo modelo ResNet-50 pré-treinado, num vetor 1D, uma vez que as próximas camadas necessitam entradas deste tipo. A camada Dense utiliza a função de ativação *ReLU* e ambas as regularizações L1 e L2 *(Elastic Net)*, desta forma L1 seleciona as características mais relevantes e L2 previne o *ovefitting* [41]. A camada de *Dropout* desliga aleatoriamente os neurônios da camada anterior e, por fim, a última camada *Dense* tem um número de neurônios igual ao número de classes do *dataset*, apesar de manter a regularização L1 e L2, desta vez utiliza a função *Softmax*, que é apropriada para problemas de classificação multiclasse. O

modelo é treinado durante 40 *epochs* com uma *learning rate* de 0.0001, como demonstra a Figura 21.

```
pretrained_model=tf.keras.applications.ResNet50(
    include_top=False,
    input_shape=FIXED_SIZE,
    pooling='avg',
    classes=NUM_CLASSES,
    weights='imagenet'
)

for layer in pretrained_model.layers:
    layer.trainable=False


resnet_model.add(pretrained_model)
resnet_model.add(Flatten())
resnet_model.add(Dense(512, activation='relu',kernel_regularizer=tf.keras.regularizers.L1L2(l1=1e-5, l2=1e-4)))
resnet_model.add(Dropout(0.5))
resnet_model.add(Dense(NUM_CLASSES, activation='softmax',kernel_regularizer=tf.keras.regularizers.L1L2(l1=1e-5, l2=1e-4)))


resnet_model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
    loss=tf.keras.losses.CategoricalCrossentropy(),
    metrics = ["accuracy"]
)

history = resnet_model.fit(train_ds,validation_data=val_ds, epochs=40)
```

Figura 21 - Construção do modelo resnet-50 com *Transfer Learning*

Por fim as últimas 10 camadas do modelo ResNet-50 são definidas como treináveis *(Fine-Tuning)*, e é configurado o *callback LearningRateScheduler*, isto significa que em cada *epoch* a *learning rate* é atualizada conforme a função *Schedule* que definimos [42], a Figura 22 ilustra este processo.

```
for layer in pretrained_model.layers[-10:]:
    layer.trainable = True

resnet_model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.00001),
    loss=tf.keras.losses.CategoricalCrossentropy(),
    metrics = ["accuracy"]
)

start_lr = 0.001
exp_decay = 0.1


def schedule(epoch):
  def lr(epoch, start_lr, exp_decay):
    return start_lr * math.exp(-exp_decay*epoch)
  return lr(epoch, start_lr, exp_decay)


lr_callback = tf.keras.callbacks.LearningRateScheduler(schedule)

history_fine_tuning = resnet_model.fit(train_ds,validation_data=val_ds, epochs=EPOCHS, callbacks=[lr_callback])
```

Figura 22 - ResNet-50 *fine-tuning* e *LearningRateScheduler*

4.4 Quantização

A quantização é uma técnica essencial para que modelos de AI possam correr em dispositivos móveis de forma eficiente, como os modelos estão com dimensões cada vez maiores isto exige mais memória e poder computacional tanto para o treino como para a inferência do modelo (momento em que é utilizado para realizar a tarefa que aprendeu). Esta técnica resume-se a um processo que permite converter valores de entrada infinitos e contínuos de um grande *dataset* em valores de saída finitos e discretos num *dataset* mais pequeno [43].

As *ANN (Artificial Neural Networks)* são constituídas por centenas de nós e ligações entre eles, sendo que cada uma destas tem um peso associado o que pode resultar em milhões de operações matemáticas. A Figura 23 ilustra a redução da precisão no valor dos pesos, ou seja, a informação que era armazenada para representar números de 32 bits de ponto flutuante é agora representada num inteiro de 8 bits.

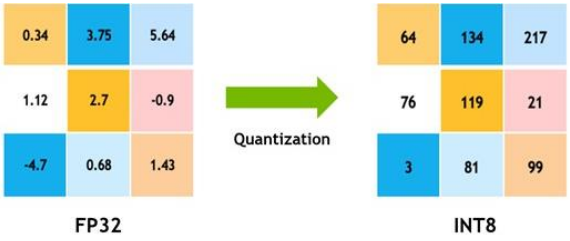


Figura 23 - Redução da precisão dos pesos (adaptado de [44])

A Figura 24 ilustra as vantagens de utilizar técnicas de quantização, apesar do cálculo dos pesos durante o treino do modelo não ser tão exato, a precisão dos modelos muitas das vezes não é sacrificada o que torna o uso desta técnica uma clara mais-valia num ambiente *mobile*.

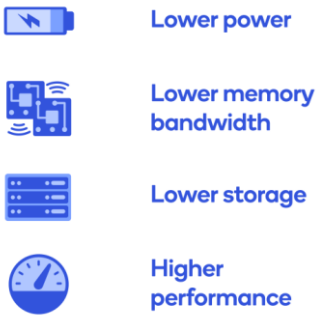


Figura 24 - Vantagens da Quantização (adaptado de [43])

A Figura 25 contém o código necessário para a conversão e 2 linhas adicionais que aplicam a técnica de quantização *float16*, ou seja, os pesos do modelo passaram a ser representados em números de 16 *bits*.

```
converter = tf.lite.TFLiteConverter.from_keras_model(resnet_model)
#Quantização
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.target_spec.supported_types = [tf.float16]
##############
tflite_model = converter.convert()
```

Figura 25 - Converter o modelo para *tflite*

As Figura 26 e Figura 27 e comprovam a diferença no peso do modelo quando exportado para o formato *tflite*, uma vez que a quantização *float16* reduz exatamente para metade o espaço que este ocupa em memória. Para além do tamanho do modelo a execução é bastante mais rápida comparativamente aos cálculos feitos com *float32* [45].

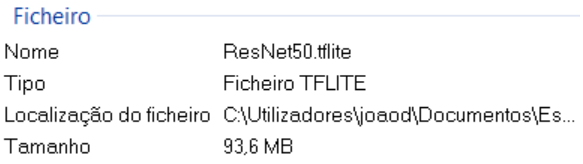


Figura 26 - Tamanho do modelo em *FP32*

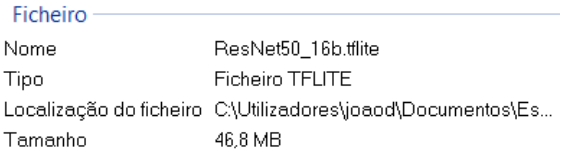


Figura 27 - Tamanho do modelo em *FP16*

4.5 Resultados

Uma vez que já tinham sido obtidos os resultados do modelo ResNet-50 e comprovado que era bastante preciso, faltava apenas garantir que a conversão do modelo para *tflite* tinha sido bem feita e ocorrido sem nenhum imprevisto. Para isso realizou-se um pequeno script *python* que carrega o modelo e gera como output a classe real e a classe prevista pelo mesmo. A Figura 28 demonstra que o modelo acertou em todas as imagens, o que permite avançar para a fase seguinte que é carregá-lo para o *Android Studio*.

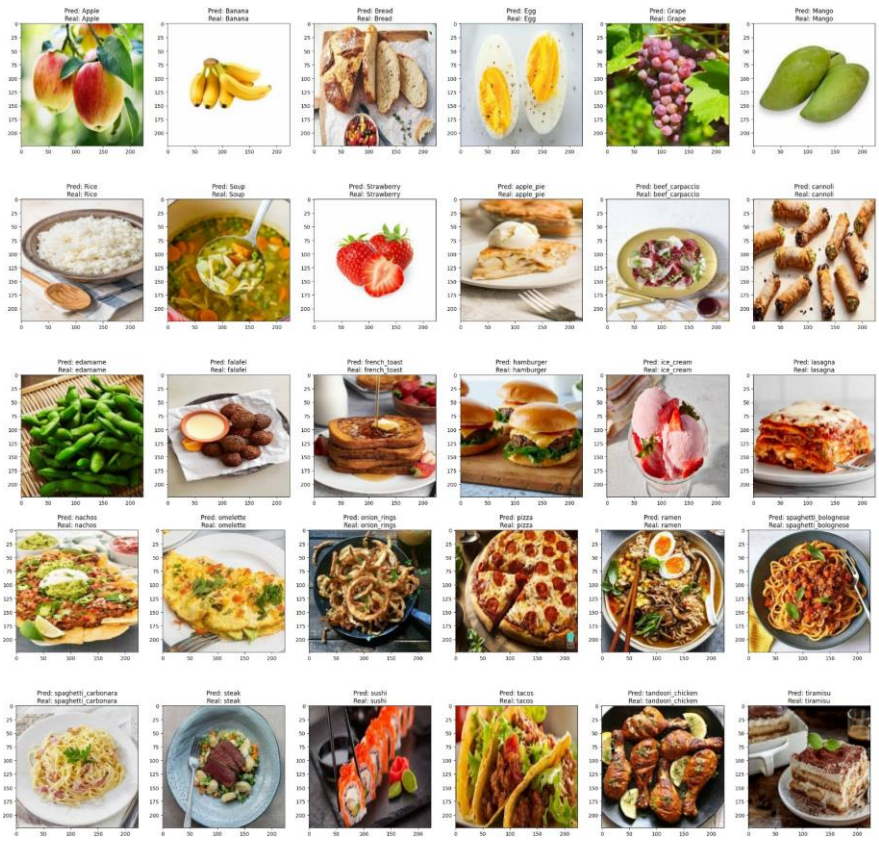


Figura 28 – Resultados

5. Modelo de Deteção e *Dataset*

Neste capítulo será apresento qual modelo e *dataset* vão ser utilizados para a tarefa de deteção dos alimentos. Posteriormente serão realizados testes que comprovem a eficácia do modelo num contexto onde os recursos computacionais são limitados.

5.1 Modelo

Para implementar o modelo foi necessário clonar o repositório do *TensorFlow Model Garden* disponível no *GitHub*, este é um repositório com várias implementações de modelos *SOTA* (*state-of-the-art)* [46]. Optou-se pelo modelo *SSD MobileNet V2 FPNLite 320x320* porque tem uma boa velocidade de inferência e um bom valor na métrica *COCO mAP (mean Average Precision)*, esta métrica é a mais utilizada para problemas de deteção de objetos, uma vez que não basta classificar os objetos, mas também a localização dos mesmos [47].

Este modelo é pré-treinado no *dataset* COCO 2017 que é composto por 164000 imagens e 2.1 milhões de rótulos pertencentes a 80 classes, este *dataset* pode ser utilizado para problemas de segmentação, pelo que as imagens também estão segmentadas. Na Figura 29 é possível observar dois exemplos onde cada imagem pode ter vários objetos associados, por exemplo pessoa e pizza, porém como este *dataset* tem como objetivo ser bastante abrangente para problemas do mundo real é composto por diversos tipos, como alimentos (banana, brócolos, laranja…), animais (gato, urso, girafa…) ou outro tipo de objetos como carro, garrafa, cama etc.

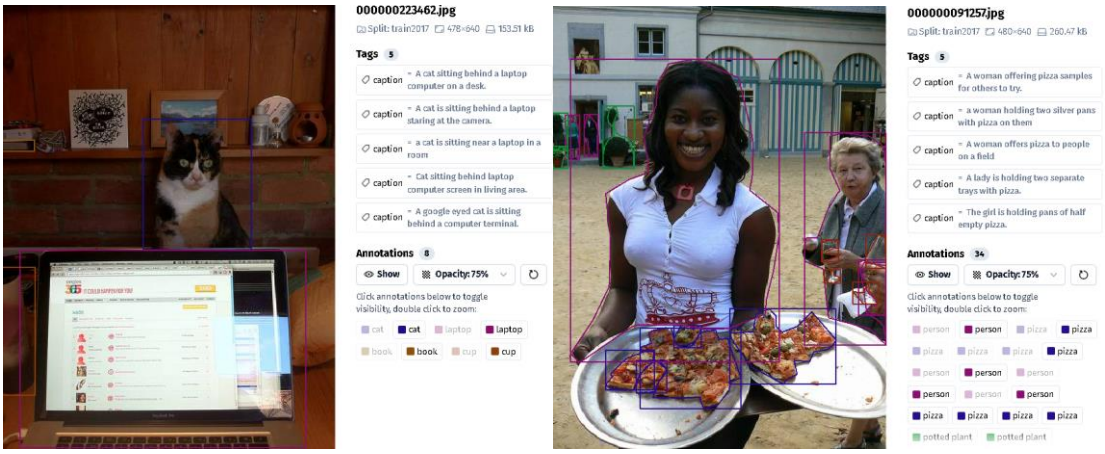


Figura 29 - Anotação das imagens no *COCO* 2017

5.2 *Dataset*

Para a construção do *dataset* foi utilizado a ferramenta de anotação de imagens disponível no *GitHub OpenLabeling*.

Dado a dificuldade que é anotar imagens manualmente, inicialmente a anotação foi feita apenas para 200 imagens de maça e 200 imagens de Ovo, o processo de anotação está exemplificado na Figura 30.

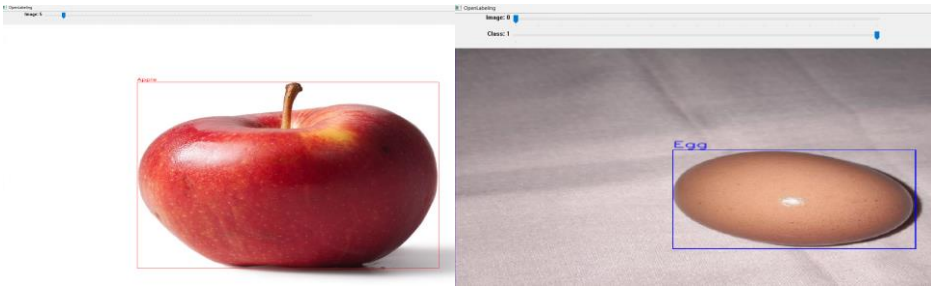


Figura 30 - Anotação das imagens no *OpenLabeling*

Os resultados obtidos eram extremamente maus, porque o modelo identificava quase sempre ovos, como ilustra a Figura 31. Inicialmente não se tinha percebido o porquê, visto que o número de imagens era o mesmo, no entanto, depois descobriu-se que grande parte das imagens de ovos continham vários ovos, por exemplo uma caixa, e por isso o modelo continha o triplo de anotações deste alimento.

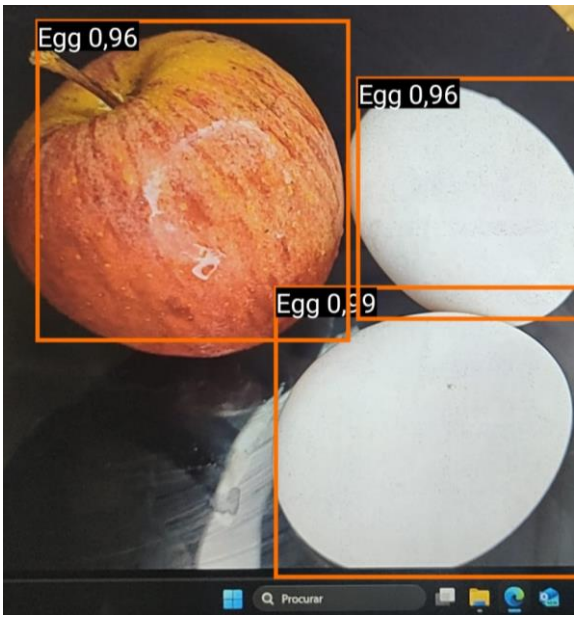


Figura 31 - Deteção de alimentos incorreta

Para resolver esse problema foram removidas algumas imagens de ovos, que não estavam a contribuir para o treino do modelo, como demonstra a Figura 32. De

seguida foram descarregadas mais imagens de maças para tentar equilibrar o número de anotações.

Figura 32 - Imagem de ovos difíceis de anotar

Porém mesmo assim os resultados não eram bons especialmente quando câmera estava a ser utilizada em ambientes do mundo real e não em imagens já com os alimentos, como demonstra a Figura 33.

Figura 33 - Problemas na deteção

A solução encontrada para estes problemas foi aumentar o número de imagens de ambas as classes e adicionar imagens em que ambos os alimentos estivessem presentes como representa a Figura 34.

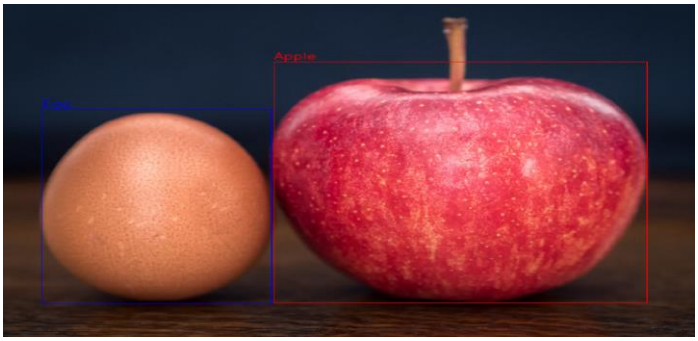


Figura 34 - Anotação de ambas as classes

Como as imagens onde ambos os alimentos estejam presentes são poucas, utilizou-se novamente o *Roboflow* mas desta vez para aumentar o número de imagens através de *data augmentation*, por exemplo alterar o brilho e rotação como ilustra a Figura 35.

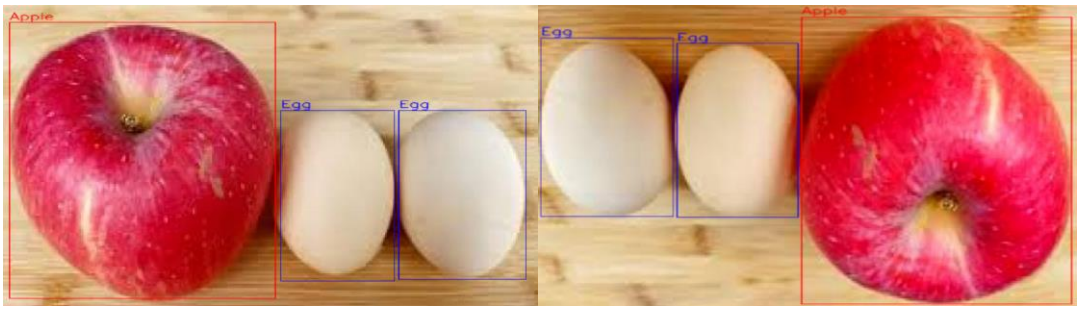


Figura 35 - Novas imagens geradas pelo *Roboflow*

O *dataset* resultante contém 2 classes, sendo constituído por 830 imagens e 1494 anotações no total, onde 799 pertencem à classe *Apple* e 695 à classe *Egg*.

5.3 Implementação do modelo SSD MobileNetV2 FPN

O treino do modelo realizado no *Google Coloab* exigiu várias etapas, neste caso, destacam-se a ligação ao *Google Drive*, clonar o repositório do *TensorFlow Model Garden*, baixar o modelo pré-treinado e por fim prosseguir para o treino do modelo. A Figura 36 ilustra algumas etapas desse processo onde é extremamente importante, antes de copiar o arquivo de configuração do modelo para a diretoria dos dados, fazer alterações consoante as necessidades, neste projeto foram utilizados 2000 steps e um batch size de 16, visto que aumentar este último valor resultaria em erros ao nível da alocação de memória porque está a ser utilizado o plano grátis do google colab e eles impõem algumas limitações no uso das VM's.

```
# Montar a ligação ao Drive
from google.colab import drive
drive.mount('/content/gdrive')

!ln -s /content/gdrive/My\ Drive/ /mydrive
!ls /mydrive

# Clonar o repositório com os modelos TensorFlow
!git clone --q https://github.com/tensorflow/models.git

# Baixar e descompactar o modelo pré-treinado
!wget http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz
!tar -xzvf ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz

# Copiar o arquivo de configuração para a diretoria dos dados
!cp /content/models/research/object_detection/configs/tf2/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.config /mydrive/customTF2/data

# Treino do modelo
!python model_main_tf2.py --pipeline_config_path=/mydrive/customTF2/data/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.config --model_dir=/mydrive/customTF2/training
```

Figura 36 - Implementação do modelo de deteção

5.4 Resultados

Para avaliar o desempenho do modelo, foi analisado o valor de *classification_loss* e os resultados obtidos para uma imagem que não pertencia ao *dataset* como mostram as Figura 37 e Figura 38 . Como os resultados eram bastante positivos e todos os erros que o modelo cometia anteriormente tinham sido resolvidos basta converter novamente o modelo para *TensorFlow Lite* para que possa ser incorporado no *Android Studio*.

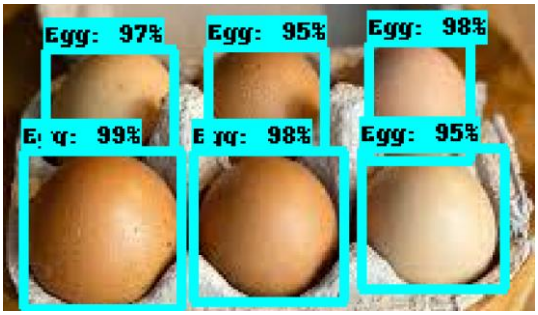


Figura 37 - Resultados

```
INFO:tensorflow:Step 2000 per-step time 8.124s
I0424 20:17:29.943000 132586404638720 model_lib_v2.py:705] Step 2000 per-step time 8.124s
INFO:tensorflow:{'Loss/classification_loss': 0.0570212,
 'Loss/localization_loss': 0.018712915,
 'Loss/regularization_loss': 0.14358045,
 'Loss/total_loss': 0.21931458,
 'learning_rate': 0.07991781}
I0424 20:17:29.943480 132586404638720 model_lib_v2.py:708] {'Loss/classification_loss': 0.0570212,
 'Loss/localization_loss': 0.018712915,
 'Loss/regularization_loss': 0.14358045,
 'Loss/total_loss': 0.21931458,
 'learning_rate': 0.07991781}
```

Figura 38 - Métricas do modelo

6. Sistema de Recomendação de Receitas

Neste capítulo o objetivo é explicar todas as abordagens que foram tentadas implementar na aplicação para a recomendação de receitas. Esta descrição pretende identificar as limitações de cada método, destacando as vantagens que o método selecionado tem sobre os outros.

6.1 Primeira Abordagem *(API)*

Inicialmente para a recomendação de receitas, baseada nos ingredientes reconhecidos, implementou-se uma abordagem utilizando a *API Spoonacular* [48]. Porém o objetivo seria recomendar receitas com base nas características dos utilizadores e esta implementação apenas faz uma consulta à base de dados e retorna as receitas que contêm os ingredientes selecionados, como ilustrado nas Figura 39 e Figura 40.

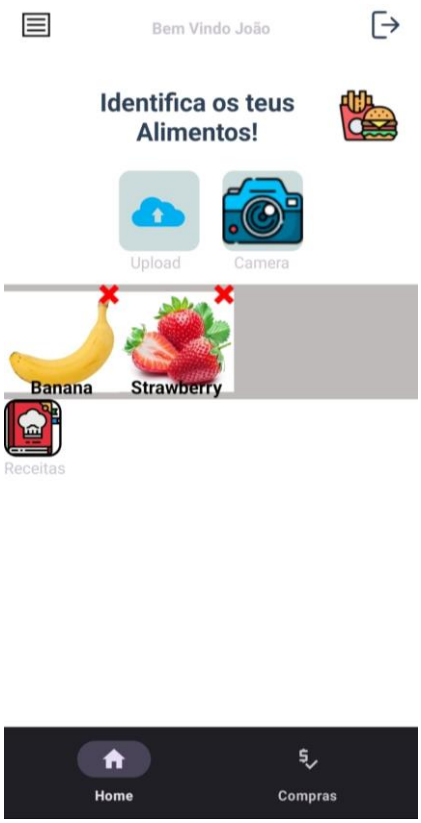


Figura 39 - Página Inicial

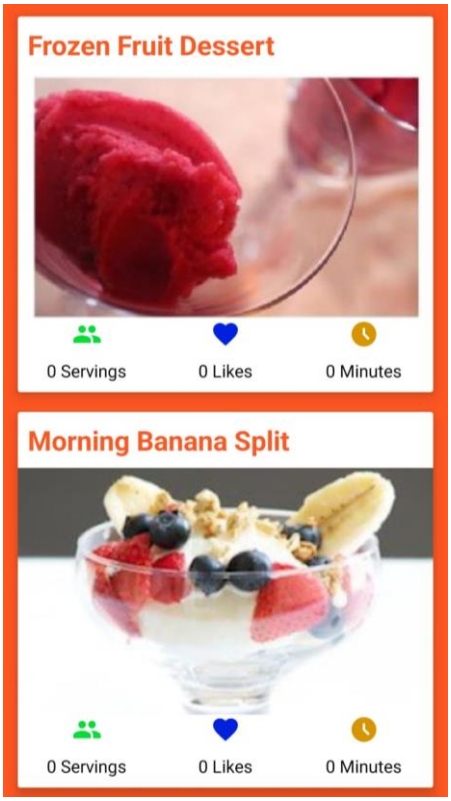


Figura 40 - Receitas da *API*

6.2 Segunda Abordagem (Modelo de *ML*)

Com vista a implementar uma solução mais complexa e versátil para o utilizador foram estudadas diversas abordagens, sendo que a grande maioria implementava modelos de *machine learning* utilizando informações ou baseadas no utilizador, ou baseadas nos itens. A Figura 41 ilustra esta técnica de recomendação designada

filtragem colaborativa, onde no exemplo a) são recomendados alimentos ao utilizador com base nas características de outros utilizadores que têm gostos semelhantes, já no exemplo b) os alimentos recomendados baseiam-se nos itens, ou seja, se um utilizador gosta de gelado ser-lhe-ão recomendados outros alimentos de utilizadores que gostaram de gelado.

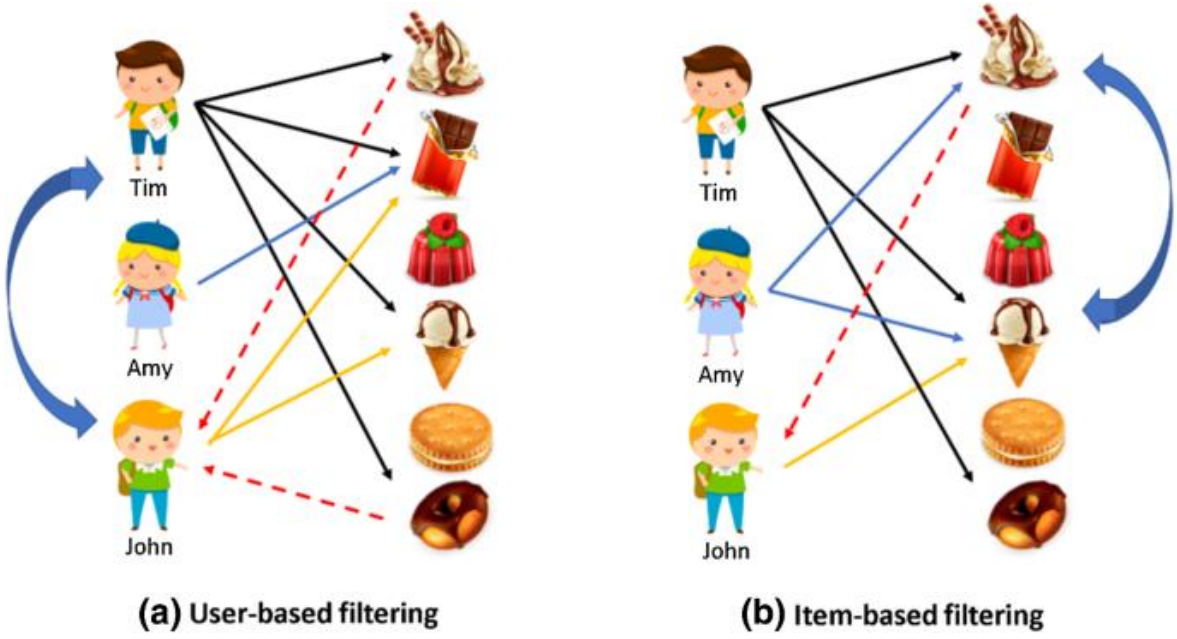


Figura 41 – Filtragem Colaborativa baseada nos utilizadores vs baseada nos Itens (adaptado de [49])

Apesar de ser um dos métodos mais utilizados quando o assunto é sistemas de recomendação, o principal problema desta abordagem é a necessidade de um *dataset* gigante com milhares/milhões de informação. Foi observado que em implementações de sistemas de recomendação de filmes, os datasets utilizados continham sempre milhares de filmes e milhões de avaliações para que a rede neural use esta informação e consiga relacionar os filmes e os utilizadores. Para além do grande problema que seria encontrar um *dataset* com esta quantidade de informação, em [49] é alertado outro problema deste método, uma vez que estes modelos não se adaptam às variações nos interesses dos utilizadores, visto que foram treinados com determinados dados. Ou seja, estes sistemas muito provavelmente podem levar a recomendações desatualizadas e não permitem ao utilizador mudar de opinião ou interesses.

6.3 Terceira Abordagem *(Firebase Analytics + BigQuery)*

Para evitar esse problema foi tentado implementar um método que utilizava o *Firebase Analytics* e o *Google BigQuery*, que podem ser integradas para permitir a análise de dados e tomada de decisão em tempo real. Como demonstrado na Figura 42, o *Firebase Analytics* seria utilizado para recolher informações dos utilizadores e o

Google BigQuery seria responsável por guardar e segmentar informação. De seguida era necessário treinar um modelo de *machine learning* e exportá-lo para *TensorFlow Lite*, porém surgiu novamente o problema da falta de dados, uma vez que a nossa app não tem utilizadores suficientes para recolher a informação necessária. Apesar de ser uma solução interessante que utiliza dados em tempo real fornecidos pela própria aplicação, outro dos problemas, é a necessidade de fazer a adesão ao plano pago do *Google Cloud Platform*.

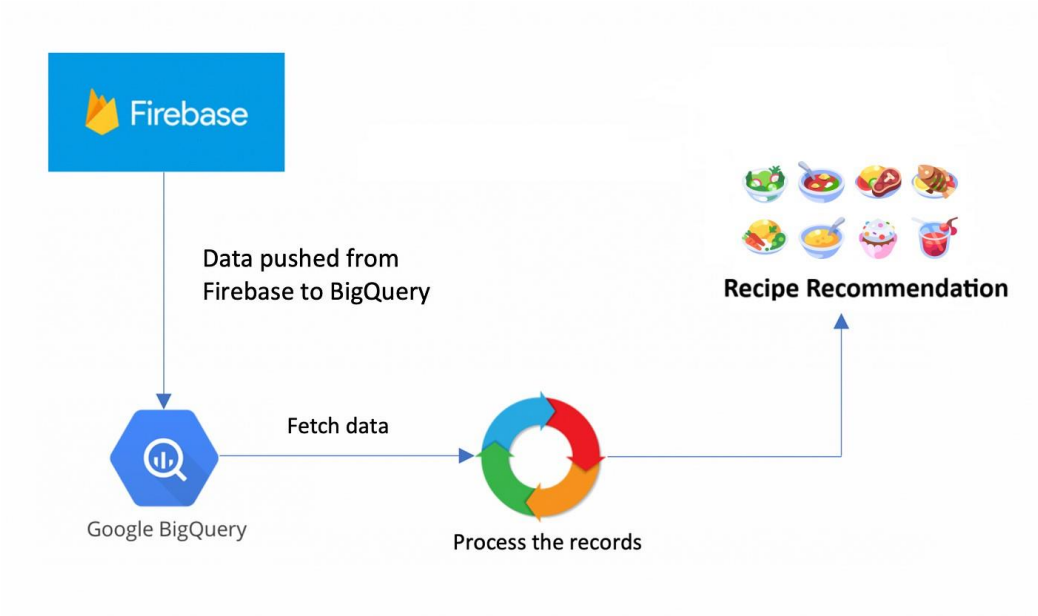


Figura 42 - *FireBase Analytics + BigQuery + ML* (adaptado de [50])

6.4 Método Escolhido *(LLM)*

Felizmente, foi encontrado, um método que se encaixava perfeitamente nos objetivos pretendidos, este método utiliza o *GPT-3.5 Turbo Instruct* que é um *LLM (large language model)* baseado nas capacidades do *GPT-3.5 Turbo*, destacando-se por fornecer respostas diretas e precisas às instruções fornecidas [51]. Este é um dos diversos modelos disponibilizados pela *OpenAI*, empresa criadora do *ChatGPT*, que é possível encontrar na sua página de modelos [52]. Alguns dos modelos que se destacam são o *DALL·E*, o *TTS* e o *Whisper*, o primeiro é capaz de gerar imagens através de um *prompt* escrito em linguagem natural, o segundo consegue converter texto em áudio e o último é capaz de converter áudio em texto. Devido à enorme diversidade de modelos a seleção do mesmo exigiu primeiramente consultar a página de depreciações da *OpenAI* [53], onde é possível consultar as recomendações dadas pela própria e assim evitar a utilização de um modelo que será descontinuado no futuro. Após consulta da documentação foi concluído que o modelo *GPT-3.5 Turbo Instruct* seria a melhor opção uma vez que é o recomendado para modelos de *Text*

generation. A estrutura do sistema de recomendação de receitas está demonstrada na Figura 43 de forma sequencial assim:

1) O utilizador requisita o reconhecimento de alimentos;

2) Na aplicação, o utilizador seleciona a identificação ou a deteção dos alimentos;

3) Os alimentos são classificados e enviados para a atividade que fará a recomendação de receitas;

4) A aplicação liga-se à *Firebase* que contém a informação dos utilizadores;

5) As características do utilizador com sessão iniciada são enviadas para o *prompt*;

6) O utilizador seleciona informação adicional para incluir no *prompt*;

7) Ligação à *API* da *OpenAI*;

8) Colocar a resposta na interface do utilizador;

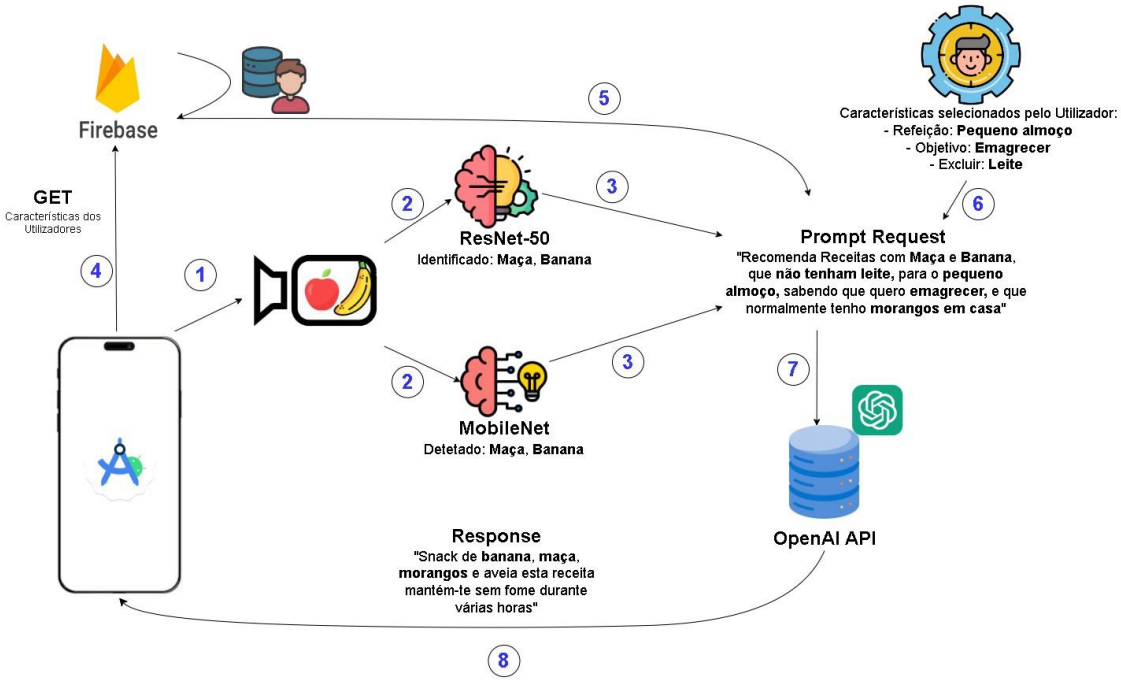


Figura 43 - *Workflow* do sistema apresentado no projeto

6.5 *GPT-3.5 Turbo Instruct*

Este modelo é do tipo *Text To Text*, ou seja, tanto o input como o output são texto e possui uma arquitetura baseada em *Transformers*. Os *Transformers* são modelos de *deep learning* baseados num mecanismo de auto-atenção e foi inicialmente desenvolvido pela equipa da *Google Brain*, sendo proposto num artigo em 2017 com o nome *"Attention Is All You Need"* [54].

Os transformadores foram concebidos para processar dados de entrada sequenciais, e por isso tornam-se especialmente eficazes no processamento de

linguagem natural *(NLP)*, sendo capazes de processar toda a entrada de uma só vez graças ao mecanismo de atenção que permite ao modelo focar-se nas partes mais relevantes, permitindo o desenvolvimento de sistemas como: *BERT (Bidirectional Encoder Representations from Transformers)* e *GPT (Generalized Pre-trained Transformers)* [55].

O que distingue os *Transformers* de outras arquiteturas é a sua capacidade de paralelização, e a utilização de mecanismos de atenção. Os mecanismos de *self-attention* permitem que o modelo identifique a importância de diferentes partes da sequência de entrada [56].

No caso do *GPT-3* o transformador possui uma arquitetura com 175 mil milhões de parâmetros, o que apenas é possível dada a eficiência do modelo. Primeiramente o texto de entrada é *tokenizado* (dividir em partes menores), de seguida o mecanismo de *self-attention* transforma cada input numa sequência de entrada em três vetores: *Queries*, *Keys* e *Values*. Como, na Figura 44 o input tem dimensão 4, se quisermos cada um dos vetores com dimensão 3, teremos de utilizar pesos de formato 4x3 como demonstra a Equação 1, utilizando pesos aleatórios. Este processo teria de ser repetido para obter *value* e *query* modificando apenas os pesos utilizados.

Equação 1 - Obter valores de *Key* com pesos aleatórios

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ 4 & 4 & 0 \\ 2 & 3 & 1 \end{bmatrix}$$

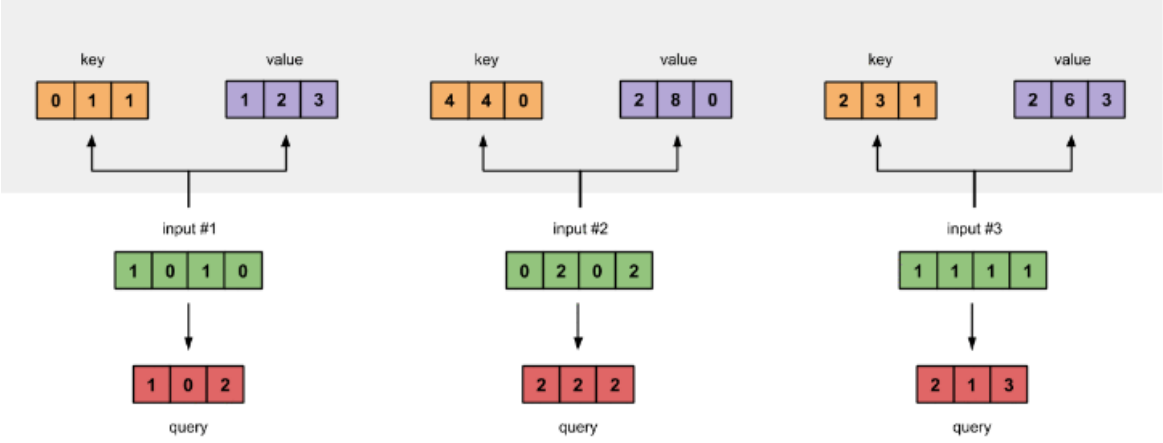


Figura 44 - Representação de *key*, *value* e *query* para cada *input* (adaptado de [57])

Para obter o valor de atenção para o input 1 é necessário multiplicar o valor de *query* por uma matriz que é composta pelos valores de *key*, só que trocando as linhas por colunas como demonstra a Equação 2.

Equação 2 - Attention Score

$$[1 \quad 0 \quad 2] \times \begin{bmatrix} 0 & 4 & 2 \\ 1 & 4 & 3 \\ 1 & 0 & 1 \end{bmatrix} = [2 \quad 4 \quad 4]$$

De seguida é aplicada a função *softmax* ao vetor contendo o valor de atenção, sendo posteriormente multiplicado pelo *value*, por fim cada um destes vetores é somado resultando no valor de atenção final [57]. Para obter, os output 2 e 3 teriam de ser repetidos os mesmos passos, a Figura 45 ilustra apenas o processo para o output 1.

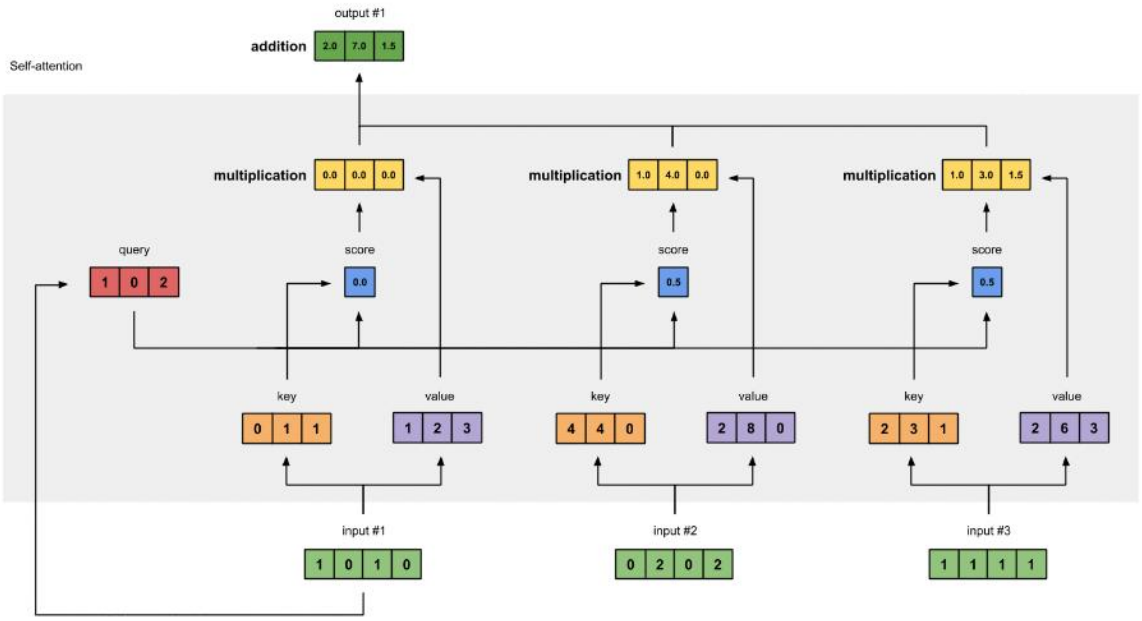


Figura 45 - Obtenção do valor de atenção final (adaptado de [57])

A Figura 46 ilustra a arquitetura do modelo *Transformer*, é composto por um *Encoder* (bloco esquerdo) e um *Decoder* (bloco direito). O *Encoder* utiliza o mecanismo de *self-attention* explicado anteriormente, com uma ligeira diferença visto que é utilizado *Multi Head Attention*, ou seja, ao invés de apenas um vetor para *key*, *value* e *query* são utilizados vários vetores com pesos independentes, treinados de forma paralela de forma a obter mais significado das frases [58]. Após a camada de *self-attention* os resultados são alimentados numa rede neural *Feedforward* que ajuda a capturar padrões complexos nos dados. O descodificador é responsável por gerar uma saída com base nas informações processadas pelo codificador através de uma pilha de várias camadas que pode ser dividida em 3 componentes principais: **Autoatenção com Máscara** (semelhante ao *Multi Head Attention*, mas utiliza uma máscara para que as palavras futuras não sejam acedidas durante a saída),

Autoatenção Cruzada (cada palavra na saída tem em conta as palavras do *Encoder*) e ***Feedforward*** (Rede neural para processar as informações) [59].

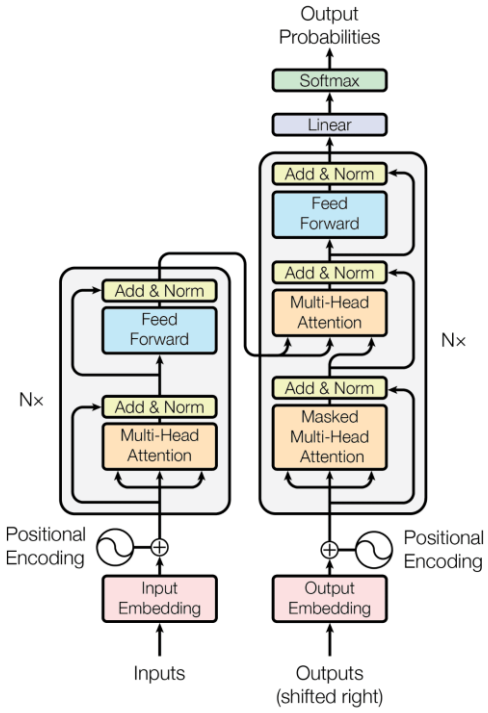


Figura 46 - Arquitetura do modelo *Transformer* (adaptado de [59])

Apesar de tanto o modelo *GPT-3.5* como o *GPT-3.5 Turbo Instruct* possuírem uma arquitetura deste género, existem grandes diferenças entre eles, o 1º tem como funcionalidade principal simular conversas enquanto o 2º está otimizado para responder diretamente a perguntas, para além disso é uma enorme melhoria em termos de geração de conteúdos de qualidade, fornecendo resultados mais precisos e diretos, uma vez que não é um modelo de chat como o próprio nome indica [60]. Este modelo possui uma *Context Window* de *4K tokens*, isto é referente à quantidade máxima de texto que o modelo considera para gerar uma resposta, incluindo tanto o pedido como a resposta gerada pelo mesmo, caso o pedido exceda este limite o modelo perde o acesso às partes mais antigas da conversa [61].

Este tipo de modelos, e especificamente o *GPT-3.5 Turbo Instruct*, pode transformar vários setores como o marketing, gerando campanhas adaptadas a dados demográficos específicos ou a Educação através da geração de perguntas sobre um tópico, tendo um enorme potencial de revolucionar diversos *workflows* [62].

7. Implementação

Neste capítulo o objetivo é demonstrar de que forma a aplicação com o nome *"Recipe Rescue"* foi construída, descrevendo detalhadamente o *design* e as funcionalidades principais (Reconhecimento de ingredientes; Recomendação de Receitas; Consulta dos preços do supermercado). Esta *app* poderá ser utilizada por todo o tipo de pessoas interessadas em receber receitas personalizadas ou poupar dinheiro nas compras do supermercado, porém foi concebida especialmente para recomendar receitas com base nos alimentos que o utilizador tem em casa de forma a evitar o desperdício alimentar.

Com o objetivo de identificar os requisitos funcionais da aplicação, foi realizado um diagrama de casos de uso, apresentado na Figura 47, em notação *Unified Modeling Language (UML)*. Os casos de uso representam as interações entre o utilizador e a aplicação e permitem identificar as funcionalidades às quais tem acesso.

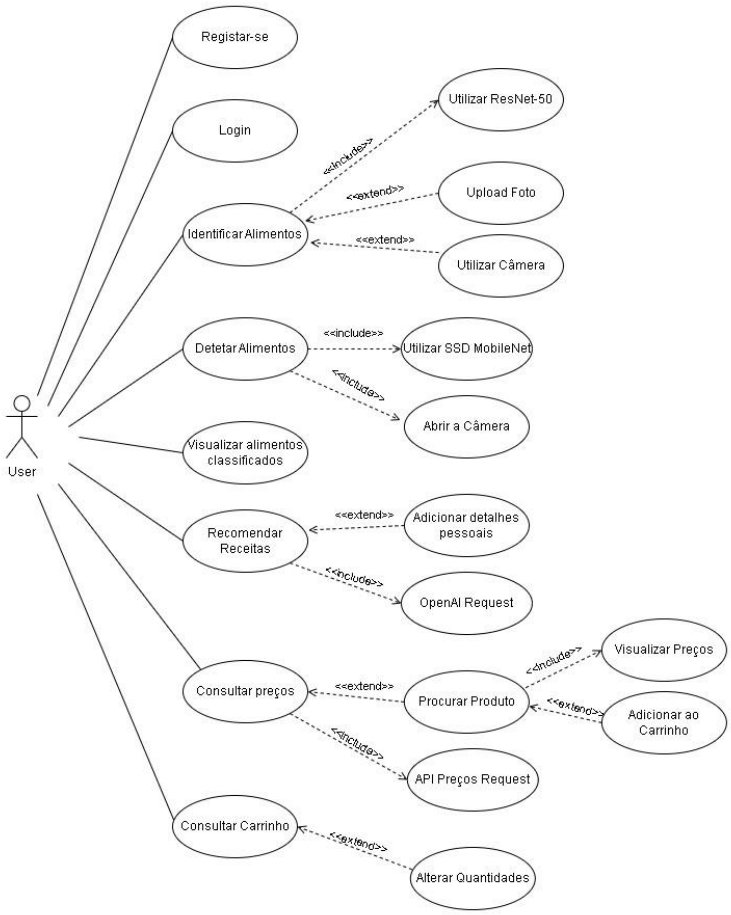


Figura 47 - Diagrama de casos de uso da app

Após o login a página inicial permite ao utilizador aceder a todas as funcionalidades da aplicação, como representa a Figura 48.

Figura 48 - Página Inicial

7.1 Reconhecimento de Ingredientes

Como foi referido anteriormente, implementaram-se na aplicação 2 modelos, um deles faz a identificação dos alimentos e o outro possui a capacidade de deteção em tempo real de múltiplos ingredientes.

Para classificação dos alimentos ser bem-sucedida é necessário carregar um arquivo *txt* para o *Android Studio* com todas as classes que os modelos são capazes de identificar. Além disso é possível verificar na Figura 49 que as imagens vão ser redimensionadas para 224x224 *pixels*, uma vez que é o tamanho aceite pelo modelo de identificação (ResNet-50).

```
//Obtem as possiveis labels dos alimentos
labels = new ArrayList<>();
try {
    labels.addAll(new BufferedReader(new InputStreamReader(getAssets().open( fileName: "labelmap.txt"))).lines().collect(Collectors.toList()));
} catch (IOException e) {
    e.printStackTrace();
}
imageProcessor = new ImageProcessor.Builder()
        .add(new ResizeOp( targetHeight: 224,  targetWidth: 224, ResizeOp.ResizeMethod.BILINEAR))
        .build();
```

Figura 49 - Bloco de código para redimensionar as imagens e carregar as *labels*

A Figura 50 contém a interface apresentada ao utilizador após o upload de uma foto ou a utilização da câmera para tirar uma fotografia. A partir do momento em que o alimento é detetado, o resultado é apresentado na interface, permitindo ao utilizador retirar alimentos identificados e passá-los para a atividade que faz a recomendação das receitas com base nessa informação.

Figura 50 - Identificação de Alimentos

A identificação dos alimentos é feita recorrendo a um método designado *classifyImage*, cujo código está representado na Figura 51. Este método é responsável por carregar o modelo ResNet-50, sendo que a classificação é feita com base no alimento que obteve maior pontuação de confiança. Além disso o alimento identificado é carregado para a base de dados do utilizador, uma vez que esta informação será utilizada no *prompt* para a recomendação de receitas, com o objetivo de estas serem baseadas nos alimentos que o utilizador tem frequentemente em casa e assim evitar o desperdício alimentar.

```java
public void classifyImage(Bitmap image) throws IOException {
    DatabaseReference userRef = FirebaseDatabase.getInstance().getReference( path: "Utilizador");

    ResNet50 model = ResNet50.newInstance(getApplicationContext());

    TensorBuffer inputFeature0 = TensorBuffer.createFixedSize(new int[]{1, 224, 224, 3}, DataType.FLOAT32);
    TensorImage tensorImage = new TensorImage(inputFeature0.getDataType());
    tensorImage.load(image);
    tensorImage = imageProcessor.process(tensorImage);

    inputFeature0.loadBuffer(tensorImage.getBuffer());

    ResNet50.Outputs outputs = model.process(inputFeature0);
    TensorBuffer outputFeature0 = outputs.getOutputFeature0AsTensorBuffer();

    float[] confidences = outputFeature0.getFloatArray();

    int maxPos = 0;
    float maxConfidence = 0;
    for(int i = 0; i < confidences.length; i++){
        if(confidences[i] > maxConfidence){
            maxConfidence = confidences[i];
            maxPos = i;
        }
    }

    foodName = labels.get(maxPos);

    userRef.child(userKey).child( pathString: "Alimentos").push().setValue(foodName);

    onFoodIdentified(foodName, bitmap);

    model.close();
}
```

Figura 51 - Método para classificação dos alimentos

Para a deteção de alimentos será automaticamente iniciada a câmera do telemóvel e a deteção começará a acontecer em tempo real como mostra a Figura 52.

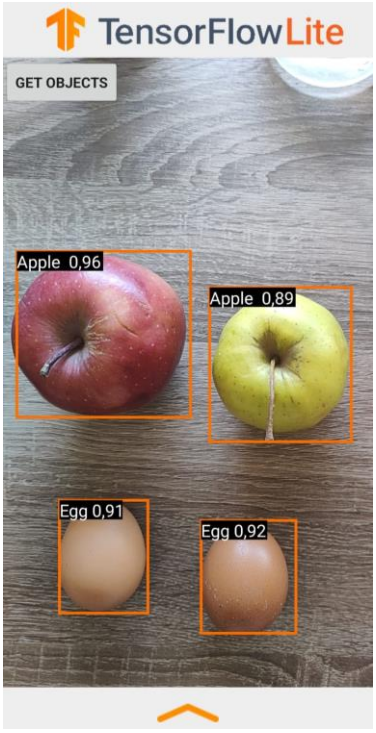


Figura 52 - Deteção em tempo real

O processo de deteção de alimentos é bastante mais complexo, porém vale ressaltar o momento onde são definidas 2 variáveis, *threshold* e *maxResults*, como ilustra a Figura 53. O *threshold* representa o limite de confiança, ou seja, quanto mais baixo for este valor mais objetos o modelo irá detetar e possivelmente mais erros cometerá, Já o *maxResults* define o número máximo de objetos que o modelo irá conseguir detetar simultaneamente.

```
public class ObjectDetectorHelper {

    3 usages
    private float threshold = 0.7f;
    3 usages
    private int maxResults = 3;
```

Figura 53 - Variáveis personalizáveis

O valor destas variáveis pode ser alterado pelo utilizador no momento da deteção, a Figura 54 ilustra o resultado dessa alteração, onde apenas são detetados os 2 alimentos em que o modelo está a obter uma maior pontuação.

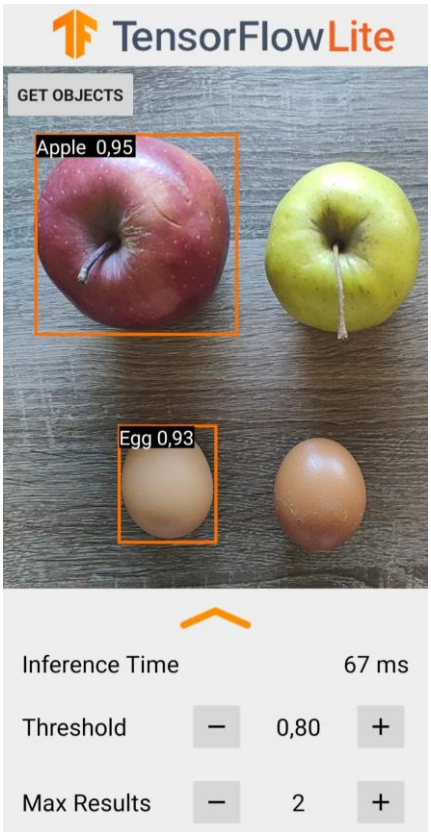


Figura 54 - Alteração do valor das variáveis

Para realizar a deteção são necessários essencialmente 2 métodos: *setupObjectDetector* e *Detect*. O primeiro método ilustrado na Figura 55, é responsável por definir o limite de confiança e o número máximo de resultados, bem como por carregar o modelo *TensorFlow Lite* previamente exportado para o *Android Studio*.

```
public void setupObjectDetector() throws IOException {
    ObjectDetector.ObjectDetectorOptions.Builder optionsBuilder =
            ObjectDetector.ObjectDetectorOptions.builder()
                    .setScoreThreshold(threshold)
                    .setMaxResults(maxResults);
    BaseOptions.Builder baseOptionsBuilder = BaseOptions.builder().setNumThreads(numThreads);

    optionsBuilder.setBaseOptions(baseOptionsBuilder.build());

    String modelName = "detect.tflite"; // Default model

    objectDetector = ObjectDetector.createFromFileAndOptions(context, modelName, optionsBuilder.build());

}
```

Figura 55 - Método para carregar o modelo de deteção

A Figura 56 contém o código do método *detect* que tem como funções processar a imagem e detetar os objetos contidos na mesma.

```
public void detect(Bitmap image, int imageRotation) throws IOException {

    setupObjectDetector();

    long inferenceTime = SystemClock.uptimeMillis();

    ImageProcessor imageProcessor =
            new ImageProcessor.Builder()
                    .add(new Rot90Op( k: -imageRotation / 90))
                    .build();

    TensorImage tensorImage = imageProcessor.process(TensorImage.fromBitmap(image));
    List<Detection> results = objectDetector.detect(tensorImage);

    inferenceTime = SystemClock.uptimeMillis() - inferenceTime;

    if (objectDetectorListener != null) {
        objectDetectorListener.onResults(
                results,
                inferenceTime,
                tensorImage.getHeight(),
                tensorImage.getWidth()
        );
    }
}
```

Figura 56 - Método para deteção em tempo real

A principal vantagem deste modelo é o facto de conseguir distinguir múltiplos alimentos no mesmo sítio. As Figura 57 e Figura 58contêm o *output* do modelo para uma mesma imagem composta por maças e ovos, onde o modelo de identificação tem como *output* Apple, visto que apenas consegue classificar um alimento, já o modelo de deteção classifica ambos os alimentos e o número de vezes que estão na imagem.

Figura 57 - *Output* modelo de identificação

Figura 58 - *Output* modelo de deteção

7.2 Recomendação de Receitas

Os alimentos previamente identificados serão passados à atividade responsável pela recomendação de receitas. Esta recomendação é baseada na *API* da *OpenAI* como explicado anteriormente, e o utilizador poderá alterar uma série de opções mediante a sua necessidade, como mostra a Figura 59.

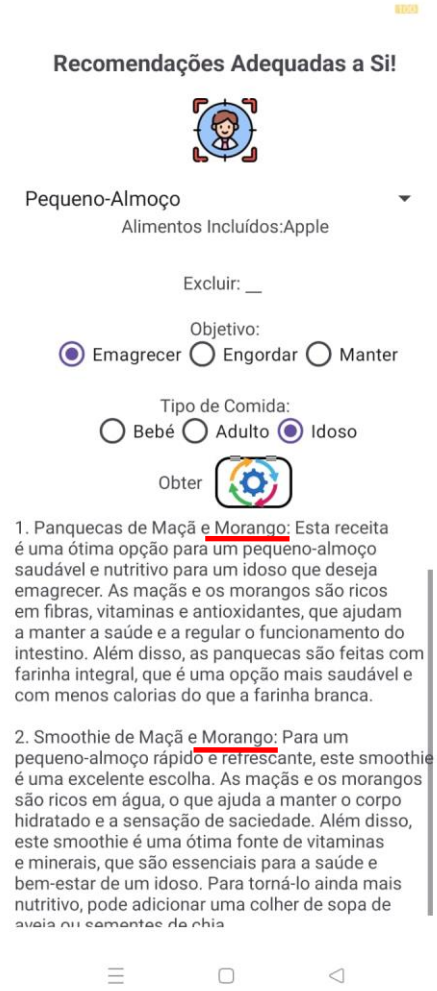


Figura 59 - Recomendação de Receitas

Figura 60 - Alimentos na Base de Dados

É possível comprovar, com base nas, Figura 59 e Figura 60 ,que as recomendações são baseadas nas opções selecionadas pelo utilizador, uma vez que as 2 receitas incluem maça e são adequadas para pequeno almoço de um idoso que deseja emagrecer, mas também têm por base os alimentos que o utilizador identificou recentemente, dado que as recomendações contêm morango, que é um alimento que o utilizador possui frequentemente.

A Figura 61 contém o código que faz a chamada à *API* da *OpenAI*, onde inicialmente é definido um objeto *json* para armazenar os diversos parâmetros que serão enviados. O *prompt* foi construído de maneira iterativa por forma a obter a melhor resposta da *API* com base no maior número de informação possível. Para além da informação selecionada pelo utilizador é importante definir os parâmetros *max_tokens* e *temperature*, onde o 1º representa o comprimento máximo do texto gerado e o 2º controla a aleatoriedade da geração de texto, ou seja, valores mais altos (perto de 1)

resultam em texto mais aleatório e criativo e valores mais baixos (perto de 0) resultam em texto mais focado e consistente [63].

```
ImageView button = findViewById(R.id.getRecommendationsButton);
button.setOnClickListener(v -> {
    JSONObject json = new JSONObject();
    try {

        String alergias = alergiasInput.getText().toString();
        json.put( name: "prompt",  value: "Por favor, recomende duas receitas para "+selectedMeal+" que incluam " +alimentosDetetados+
                "As recomendações devem ter em conta os alimentos que o utilizador normalmente tem em casa, " + alimentosPercStr+
                "As recomendações devem ser para " + objetivo+ "e adequadas para um"+ tipoComida+
                ".Por favor, evite ingredientes que contenham " + alergias);

        json.put( name: "max_tokens",  value: 400);
        json.put( name: "temperature",  value: 0.1);
    } catch (JSONException e) {
        e.printStackTrace();
    }

    new Thread(() -> {
        try {
            String result = runOpenAI(json.toString());

            JSONObject jsonResponse = new JSONObject(result);
            String text = jsonResponse.getJSONArray( name: "choices").getJSONObject( index: 0).getString( name: "text");

            String[] lines = text.split( regex: "\n");

            String dishNamesStr = TextUtils.join( delimiter: "\n", lines);
            runOnUiThread(() -> resultView.setText(dishNamesStr));
        } catch (IOException | JSONException e) {
            e.printStackTrace();
        }
    }).start();
});
```

Figura 61 - *Prompt* enviado à *API* da *OpenAI*

7.3 Consulta dos preços dos supermercados

Como foi referido anteriormente, a aplicação permite que o utilizador consulte os preços do supermercado através de uma *API*. Primeiramente é necessário que pesquise por um produto como mostra a Figura 62.

Figura 62 - Pesquisar Produto

Depois a resposta da *API* do site *Prices Crawler* é processada de forma a apresentar o conteúdo relevante na interface, como é possível observar nas Figura 63 e Figura 64, o preço está a ser obtido corretamente.

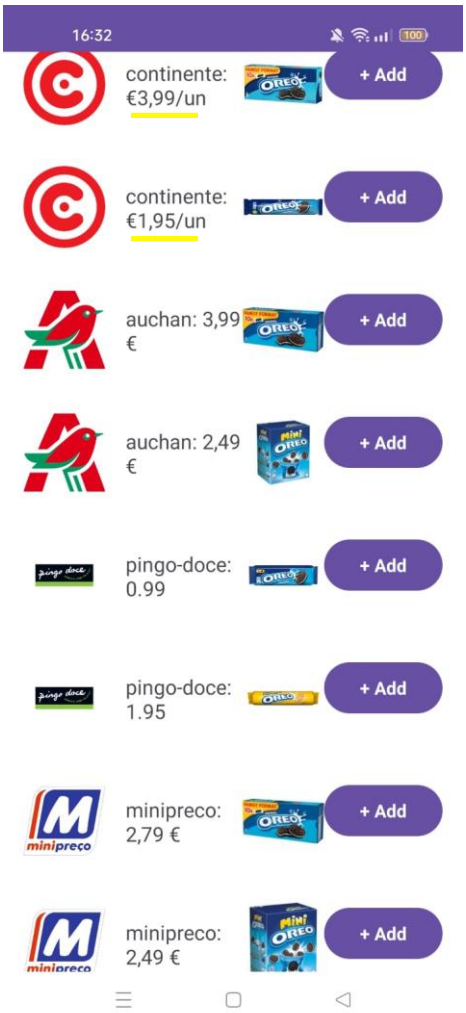


Figura 63 - Preços obtidos na aplicação

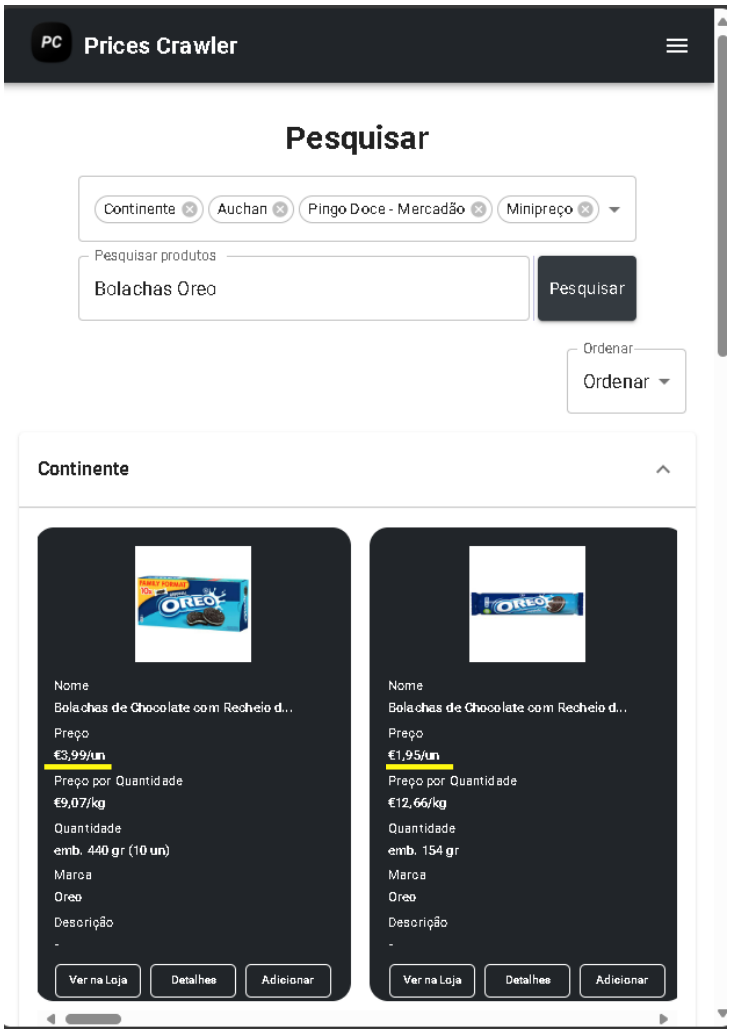


Figura 64 - Preços no site *Prices Crawler*

Para implementar esta funcionalidade no *Android Studio* é necessário fazer uma solicitação *HTTP POST* para a *API* através da biblioteca *OkHttpClient*. Esta biblioteca é utilizada para fazer solicitações de rede e contém a *URL* da *API* e o corpo do pedido como mostra a Figura 65, e tem de ser feita numa nova *thread*, visto que em aplicações android é necessário fazê-lo para não bloquear a *thread* principal da interface do utilizador *(UI)*.

```java
new Thread(() -> {
    try {
        OkHttpClient client = new OkHttpClient(); //Faz a solicitação HTTP POST com um corpo JSON

        String url = "https://content-api.prices-crawler.duckdns.org/api/v1/products/search";

        JSONObject json = new JSONObject();
        json.put( name: "query", query);
        JSONArray catalogs = new JSONArray();
        catalogs.put( value: "pt.continente");
        catalogs.put( value: "pt.auchan");
        catalogs.put( value: "pt.pingo-doce");
        catalogs.put( value: "pt.minipreco");
        json.put( name: "catalogs", catalogs);

        RequestBody body = RequestBody.create(json.toString(), MediaType.parse( $this$toMediaTypeOrNull: "application/json; charset=utf-8"));

        Request request = new Request.Builder()
                .url(url)
                .post(body)
                .addHeader( name: "accept",  value: "application/json, text/plain, */*")
                .addHeader( name: "content-type",  value: "application/json")
                .build();

        Response response = client.newCall(request).execute();
        //Resposta da API
        final String responseStr = response.body().string();
```

Figura 65 - Chamada à *API* do *Prices Crawler*

O utilizador poderá ainda adicionar um item ao carrinho e alterar a sua quantidade, de forma a fazer uma estimativa de preço, como mostra a Figura 66.

Figura 66 - Interface do carrinho

8. Conclusão e Trabalho Futuro

Para o desenvolvimento deste trabalho, foi necessário definir vários objetivos de forma conseguir definir uma linha temporal para a implementação das várias funcionalidades da aplicação. Dessa forma, foi documentado todo o trabalho realizado para implementar a app em *Android Studio*, e foi detalhado o processo das principais capacidades da aplicação, referindo-se nomeadamente à classificação e deteção dos alimentos, bem como a recomendação de receitas personalizadas ao utilizador. Além disso a aplicação permite também a consulta dos preços dos produtos em diversos supermercados e adicionar os mesmos a um carrinho fictício de forma a fazer uma estimativa do preço.

Inicialmente são detalhadas todas as ferramentas utilizadas para o desenvolvimento deste projeto e de que forma foram aplicadas. De seguida, como em Projeto I foi construído o modelo ResNet-50 aplicado ao *dataset* criado através de *web scraping*, começou-se por treinar novamente este modelo e convertê-lo para um formato que seja eficiente numa aplicações mobile.

Na tarefa de recomendação, o uso da *API* da *OpenAI* tornou-se uma grande mais-valia em relação a outros métodos, uma vez que permitiu realizar a recomendação de receitas tendo em conta as características e inputs colocados pelo utilizador.

Dado as limitações que os modelos de classificação têm no reconhecimento simultâneo de ingredientes, foi proposta um novo modelo bem como outro *dataset*, com todas as imagens anotadas. O *dataset* resultante é composto por 2 classes de alimentos (*Apple* e *Egg*) onde estão associadas 830 imagens e 1494 anotações no total.

Assim todos os objetivos propostos foram cumpridos e as componentes principais da aplicação foram desenvolvidas com sucesso. Ainda assim, há sempre formas de melhorar e tornar a aplicação mais comercial, por isso, em termos de trabalho futuro seria importante estender ambos os *datasets* criados para que o utilizador tenha oportunidade tanto de classificar como detetar mais alimentos. Seria também relevante possibilitar o utilizador alterar o seu perfil como nome, peso e gostos pessoais, porém isto seriam apenas funcionalidades extras, caso a aplicação fosse lançada publicamente.

9. Bibliografia

[1] "Kaggle: Your Home for Data Science." Accessed: Feb. 28, 2024. [Online]. Available: https://www.kaggle.com/

[2] "What Is Kaggle and What Is It Used For? | Coursera." Accessed: Feb. 28, 2024. [Online]. Available: https://www.coursera.org/articles/kaggle

[3] "How to find data science talent on Kaggle." Accessed: Feb. 28, 2024. [Online]. Available: https://www.herohunt.ai/blog/how-to-find-data-science-talent-on-kaggle

[4] "colab.google." Accessed: May 03, 2024. [Online]. Available: https://colab.google/

[5] "A comparison between Google Colab and Kaggle | by Ahmadsabry | Medium." Accessed: May 03, 2024. [Online]. Available: https://medium.com/@ahmadsabry678/a-comparison-between-google-colab-and-kaggle-2ee3a5f65e

[6] "Google Colab — Você conhece?. Se você se interessa por ciência de… | by Arthur Marchi | Medium." Accessed: May 03, 2024. [Online]. Available: https://medium.com/@arthur.marchi/google-colab-voc%C3%AA-conhece-f6ee9dd1e88d

[7] "TensorFlow." Accessed: Feb. 28, 2024. [Online]. Available: https://www.tensorflow.org/?hl=pt

[8] "Tudo o que você queria saber sobre o TensorFlow." Accessed: Feb. 28, 2024. [Online]. Available: https://www.databricks.com/br/glossary/tensorflow-guide

[9] "Building a Deep Learning Model with TensorFlow and Keras: Easier Than You Think | by Guglielmo Cerri | Level Up Coding." Accessed: Feb. 28, 2024. [Online]. Available: https://levelup.gitconnected.com/building-a-deep-learning-model-with-tensorflow-and-keras-easier-than-you-think-62d0cb7945ff

[10] "Python Language advantages and applications - GeeksforGeeks." Accessed: Feb. 28, 2024. [Online]. Available: https://www.geeksforgeeks.org/python-language-advantages-applications/

[11] "Exploration of the 7 real-world applications of Python programming." Accessed: Feb. 28, 2024. [Online]. Available: https://www.bocasay.com/7-applications-python-programming/

[12] "IT12A01: FUNDAMENTALS OF PYTHON PROGRAMMING (SF) (SYNCHRONOUS E-LEARNING) - NTUC LearningHub." Accessed: Feb. 28, 2024. [Online]. Available: https://www.ntuclearninghub.com/en-gb/-/course/fundamentals-of-python-programming-sf

[13] “Fazer o download do Android Studio e das ferramentas de apps: desenvolvedores Android | Android Developers.” Accessed: Mar. 01, 2024. [Online]. Available: https://developer.android.com/studio?hl=pt-br

[14] “Core features of Android Studio.” Accessed: Mar. 01, 2024. [Online]. Available: https://www.educative.io/answers/core-features-of-android-studio

[15] “Android Studio - Wikipedia.” Accessed: Mar. 01, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Android_Studio

[16] “Advantages of Java - IBM Documentation.” Accessed: Mar. 01, 2024. [Online]. Available: https://www.ibm.com/docs/en/aix/7.3?topic=monitoring-advantages-java

[17] “28,344 Java Icons - Free in SVG, PNG, ICO - IconScout.” Accessed: Mar. 01, 2024. [Online]. Available: https://iconscout.com/icons/java

[18] “Firebase | Google’s Mobile and Web App Development Platform.” Accessed: May 03, 2024. [Online]. Available: https://firebase.google.com/?hl=pt-br

[19] “Top 10 Benefits of having Firebase for Mobile App Development.” Accessed: May 03, 2024. [Online]. Available: https://www.amarinfotech.com/top-10-benefits-of-having-firebase-for-mobile-app-development.html

[20] “Código Google: O Firebase foi expandido para se tornar uma plataforma de aplicativos unificada.” Accessed: May 03, 2024. [Online]. Available: https://developers-br.googleblog.com/2016/05/firebase-foi-expandido-para-se-tornar-plataforma-unificada.html

[21] “GitHub - Cartucho/OpenLabeling: Label images and video for Computer Vision applications.” Accessed: May 04, 2024. [Online]. Available: https://github.com/Cartucho/OpenLabeling

[22] “Roboflow: Computer vision tools for developers and enterprises.” Accessed: May 04, 2024. [Online]. Available: https://roboflow.com/

[23] “OpenAI.” Accessed: May 04, 2024. [Online]. Available: https://openai.com/

[24] “What is OpenAI’s API? [+ How to Start Using It].” Accessed: May 04, 2024. [Online]. Available: https://blog.hubspot.com/website/what-is-open-ai-api

[25] “‘openai’ Icon - Download for free – Iconduck.” Accessed: May 04, 2024. [Online]. Available: https://iconduck.com/icons/1213/openai

[26] “Figma Downloads | Web Design App for Desktops & Mobile.” Accessed: May 03, 2024. [Online]. Available: https://www.figma.com/downloads/

[27] “Figma: o que é a ferramenta, design e como usar | Alura.” Accessed: May 03, 2024. [Online]. Available: https://www.alura.com.br/artigos/figma

[28] "Postman API Platform." Accessed: Mar. 01, 2024. [Online]. Available: https://www.postman.com/

[29] "Por que você deve usar o Postman para testes de API?" Accessed: Mar. 01, 2024. [Online]. Available: https://www.linkedin.com/advice/1/why-should-you-use-postman-api-testing-skills-software-testing-a6hbe

[30] "1,598 Postman Icons, Logos, Symbols - Free in SVG, PNG, GIF | IconScout." Accessed: May 12, 2024. [Online]. Available: https://iconscout.com/icons/postman

[31] "Visual Studio Code - Code Editing. Redefined." Accessed: Mar. 01, 2024. [Online]. Available: https://code.visualstudio.com/

[32] "Visual Studio Code Review 2023 | A Comprehensive Look." Accessed: Mar. 01, 2024. [Online]. Available: https://www.developer.com/languages/visual-studio-code-review/

[33] "File:Visual Studio Code 1.35 icon.svg - Wikipedia." Accessed: Mar. 01, 2024. [Online]. Available: https://en.m.wikipedia.org/wiki/File:Visual_Studio_Code_1.35_icon.svg

[34] "Prices Crawler." Accessed: Mar. 06, 2024. [Online]. Available: https://prices-crawler.vercel.app/

[35] "OkHttp in Android. OkHttp is a popular open-source HTTP… | by Manu Aravind | Medium." Accessed: May 11, 2024. [Online]. Available: https://medium.com/@manuaravindpta/okhttp-in-android-4c2771141f79

[36] A. Z. Broder, M. Najork, and J. L. Wiener, "Efficient URL caching for world wide web crawling," *Proceedings of the 12th International Conference on World Wide Web, WWW 2003*, pp. 679–689, 2003, doi: 10.1145/775152.775247.

[37] "Why Don't All Websites Have an API? And What Can You Do About It? – Diffblog." Accessed: Mar. 06, 2024. [Online]. Available: https://blog.diffbot.com/why-dont-all-websites-have-an-api-and-what-can-you-do-about-it/

[38] "TensorFlow Lite - Computer Vision on Edge Devices [2024 Guide] - viso.ai." Accessed: May 12, 2024. [Online]. Available: https://viso.ai/edge-ai/tensorflow-lite/

[39] "Quantization." Accessed: May 12, 2024. [Online]. Available: https://huggingface.co/docs/optimum/concept_guides/quantization

[40] "The Motivation for Train-Test Split | by Ahmed | Medium." Accessed: May 12, 2024. [Online]. Available: https://medium.com/@nahmed3536/the-motivation-for-train-test-split-2b1837f596c3

[41] "Regularization — Understanding L1 and L2 regularization for Deep Learning | by Ujwal Tewari | Analytics Vidhya | Medium." Accessed: Mar. 05, 2024. [Online]. Available: https://medium.com/analytics-vidhya/regularization-understanding-l1-and-l2-regularization-for-deep-learning-a7b9e4a409bf

[42] "LearningRateScheduler | Tensorflow LearningRateScheduler." Accessed: Mar. 05, 2024. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/06/decide-best-learning-rate-with-learningratescheduler-in-tensorflow/

[43] "Fitting AI models in your pocket with quantization - Stack Overflow." Accessed: Mar. 06, 2024. [Online]. Available: https://stackoverflow.blog/2023/08/23/fitting-ai-models-in-your-pocket-with-quantization/

[44] "Achieving FP32 Accuracy for INT8 Inference Using Quantization Aware Training with NVIDIA TensorRT | NVIDIA Technical Blog." Accessed: Mar. 06, 2024. [Online]. Available: https://developer.nvidia.com/blog/achieving-fp32-accuracy-for-int8-inference-using-quantization-aware-training-with-tensorrt/

[45] "Quantização pós-treinamento | TensorFlow Lite." Accessed: Mar. 07, 2024. [Online]. Available: https://www.tensorflow.org/lite/performance/post_training_quantization?hl=pt

[46] "GitHub - tensorflow/models: Models and examples built with TensorFlow." Accessed: May 04, 2024. [Online]. Available: https://github.com/tensorflow/models/tree/master

[47] "Mean Average Precision (mAP) Using the COCO Evaluator - PyImageSearch." Accessed: May 04, 2024. [Online]. Available: https://pyimagesearch.com/2022/05/02/mean-average-precision-map-using-the-coco-evaluator/

[48] "spoonacular recipe and food API." Accessed: Apr. 04, 2024. [Online]. Available: https://spoonacular.com/food-api/docs#Search-Recipes-by-Ingredients

[49] Y. C. Chen, L. Hui, and T. Thaipisutikul, "A collaborative filtering recommendation system with dynamic time decay," *Journal of Supercomputing*, vol. 77, no. 1, pp. 244–262, Jan. 2021, doi: 10.1007/S11227-020-03266-2/FIGURES/12.

[50] "Mobile Application Analytics — MakeMyTrip | by Srijan Rana | MakeMyTrip-Engineering." Accessed: Apr. 04, 2024. [Online]. Available: https://tech.makemytrip.com/mobile-application-analytics-makemytrip-4e5b7366f7ce

[51] "gpt-3_5-turbo-instruct model | Clarifai - The World's AI." Accessed: Apr. 04, 2024. [Online]. Available: https://clarifai.com/openai/completion/models/gpt-3_5-turbo-instruct

[52] "Models - OpenAI API." Accessed: Apr. 04, 2024. [Online]. Available: https://platform.openai.com/docs/models/overview

[53] "Deprecations - OpenAI API." Accessed: Apr. 04, 2024. [Online]. Available: https://platform.openai.com/docs/deprecations

[54] "Evolution of Transformers — Part 1 | by Sanchit Goel | Medium." Accessed: Apr. 04, 2024. [Online]. Available: https://sanchman21.medium.com/evolution-of-transformers-part-1-faac3f19d780

[55] "What Is the Transformer Architecture and How Does It Work?" Accessed: Apr. 04, 2024. [Online]. Available: https://datagen.tech/guides/computer-vision/transformer-architecture/#

[56] "What is Self-attention?" Accessed: Apr. 04, 2024. [Online]. Available: https://h2o.ai/wiki/self-attention/

[57] "Illustrated: Self-Attention. A step-by-step guide to self-attention… | by Raimi Karim | Towards Data Science." Accessed: Apr. 04, 2024. [Online]. Available: https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a#bypass

[58] "All you need to know about 'Attention' and 'Transformers' — In-depth Understanding — Part 1 | by Arjun Sarkar | Towards Data Science." Accessed: Apr. 04, 2024. [Online]. Available: https://towardsdatascience.com/all-you-need-to-know-about-attention-and-transformers-in-depth-understanding-part-1-552f0b41d021

[59] "Transformers: An Overview of the Most Novel AI Architecture | by Diego Unzueta | Towards Data Science." Accessed: Apr. 04, 2024. [Online]. Available: https://towardsdatascience.com/transformers-an-overview-of-the-most-novel-ai-architecture-cdd7961eef84#bypass

[60] "What is ChatGPT Turbo 3.5 Instruct?" Accessed: Apr. 04, 2024. [Online]. Available: https://anakin.ai/blog/what-is-gpt-3-5-turbo-instruct/

[61] "What is your Context Window?. The context window for a large language… | by Amir Aryani | Medium." Accessed: Apr. 05, 2024. [Online]. Available: https://medium.com/@amiraryani/what-is-your-context-window-772b1c65b881

[62] "GPT-3.5 Turbo Instruct: A Powerful New Tool for Professionals | by Tarun Singh | Medium." Accessed: Apr. 05, 2024. [Online]. Available:

https://medium.com/@krtarunsingh/gpt-3-5-turbo-instruct-a-powerful-new-tool-for-professionals-2e931f5e5874

[63] “How to use OpenAI GPT-3 temperature • Gptforwork.com.” Accessed: May 08, 2024. [Online]. Available: https://gptforwork.com/guides/openai-gpt3-temperature