UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO DE ENGENHARIA ELETRÔNICA

NADER DE ARAUJO FARHAT
VINICIUS DE SOUZA KLINGENFUS

ALIMENTADOR PARA PEQUENOS ANIMAIS CONTROLADO POR APLICATIVO MÓVEL

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2019

NADER DE ARAUJO FARHAT
VINICIUS DE SOUZA KLINGENFUS

ALIMENTADOR PARA PEQUENOS ANIMAIS CONTROLADO POR APLICATIVO MÓVEL

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná, como requisito parcial para a obtenção do título de Engenheiro Eletrônico.

Orientador: Prof. Dr. Daniel Fernando Pigatto

CURITIBA
2019

NADER DE ARAUJO FARHAT
VINICIUS DE SOUZA KLINGENFUS

ALIMENTADOR PARA PEQUENOS ANIMAIS CONTROLADO POR APLICATIVO MÓVEL

Este Trabalho de Conclusão de Curso de Graduação foi apresentado como requisito parcial para obtenção do título de Engenheiro Eletrônico, do curso de Engenharia Eletrônica do Departamento Acadêmico de Eletrônica (DAELN) outorgado pela Universidade Tecnológica Federal do Paraná (UTFPR). Os alunos foram arguidos pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Curitiba, 05 de julho de 2019.

_____
Prof. Dr. Robinson Vida Noronha
Coordenador de Curso
Engenharia Eletrônica

_____
Profª. Drª. Carmen Caroline Rasera
Responsável pelos Trabalhos de Conclusão de Curso
de Engenharia Eletrônica do DAELN

BANCA EXAMINADORA

_____
Prof. Dr. Daniel Fernando Pigatto
Universidade Tecnológica Federal do Paraná
Orientador

_____
Prof. Dr. Gustavo Benvenutti Borba
Universidade Tecnológica Federal do Paraná

_____
Profª. Drª. Ana Cristina B. Kochem Vendramin
Universidade Tecnológica Federal do Paraná

A folha de aprovação assinada encontra-se na Coordenação do Curso de Engenharia Eletrônica.

AGRADECIMENTOS

Aos meus pais, Sami Jamil Farhat e Dayse Regina de Araujo Farhat, por todo amor, carinho e apoio que me proporcionaram.

Ao meu irmão, Jamil de Araujo Farhat, pela ajuda e incentivo durante todo o curso.

Aos amigos, por todo o apoio durante a trajetória do curso.

Aos professores, por todo o aprendizado.

Nader Farhat

Aos meus pais, Gilson Klingenfus e Maria Jaqueline Rodrigues de Souza Klingenfus, por todo o amor, apoio e incentivo que me deram durante todo o curso.

À minha namorada, Isadora de Brida Santi, por todo o carinho e amor, por sempre acreditar em meu potencial e estar ao meu lado, seja em momentos bons ou ruins.

À minha irmã, Andressa de Souza Klingenfus, que sempre esteve ao meu lado para qualquer situação.

Aos meus amigos, por me ajudarem ao longo desta caminhada, seja ajudando a superar problemas ou trazendo momentos de alegria.

Ao corpo docente do curso de Engenharia Eletrônica da UTFPR, pelos incontáveis ensinamentos que adquiri durante o curso.

Ao Prof. Daniel Fernando Pigatto, nosso orientador, por sua disposição em nos auxiliar durante a realização do projeto.

Vinicius Klingenfus

It is a mistake to think you can solve any major problems just with potatoes. (ADAMS, Douglas, 1982).

RESUMO

FARHAT, Nader; KLINGENFUS,Vinicius. Alimentador para Pequenos Animais Controlado por Aplicativo Móvel. 2019. 65 f. Trabalho de Conclusão de Curso – Curso de Engenharia Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2019.

Neste projeto foi desenvolvido um comedouro para pets automático. Para a realização deste projeto foram utilizados: uma Raspberry Pi, uma câmera e um aplicativo Android. O comedouro é acionado no horário previamente escolhido, liberando uma porção de ração. Tanto o horário quanto a quantidade de ração da porção são definidos pelo usuário. Um algoritmo de *deep learning* foi desenvolvido para identificar se não existe ração disponível no pote para o animal, fazendo com que o usuário seja alertado no aplicativo Android. É possível que o comedouro seja acionado remotamente através de um dispositivo Android, por meio do protocolo MQTT. O dispositivo foi elaborado para ser o mais *user-friendly* e seguro possível.

Palavras-chave: Android. Comedouro eletrônico. *Deep Learning*. MQTT. Raspberry Pi.

ABSTRACT

FARHAT, Nader; KLINGENFUS,Vinicius. Pet Feeder Controlled by Mobile Application. 2019. 65 f. Trabalho de Conclusão de Curso – Curso de Engenharia Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2019.

In this project an automatic pet feeder was developed. For the realization of this project were used: a Raspberry Pi, a camera and an Android application. The feeder is triggered at the previously chosen time, releasing a portion of feed. Both the time and the amount of feed a portion has are defined by the user. A deep learning algorithm was developed to identify if there is no food available in the pot for the animal, causing the user to be alerted in the Android application. It is possible for the feeder to be triggered remotely through an Android device via the MQTT protocol. The device is designed to be as user-friendly and reliable as possible.

Keywords: Android. Automatic pet feeder. Deep Learning. MQTT. Raspberry Pi.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

ABINPET	Associação Brasileira da Indústria de Produtos para Animais de Estimação
ADC	*Analog to Digital Converter*
API	*Application Program Interface*
ARM	*Advanced RISC Machine*
CNN	*Convolutional neural network*
CNTK	*Microsoft Cognitive Toolkit*
CPU	*Central Processing Unit*
GPIO	*General Purpose Input/Output*
GPU	*Graphics Processing Unit*
IA	Inteligência Artificial
IBGE	Instituto Brasileiro de Geografia e Estatística
IDE	*Integrated Development Environment*
IOT	*Internet of things*
IP	*Internet Protocol*
JSON	*JavaScript Object Notation*
M2M	*Machine-to-Machine*
MNIST	*Modified National Institute of Standards and Technology*
MQTT	*Message Queuing Telemetry Transport*
PNL	Processamento de Linguagem Neural
QoS	*Quality of Service*
RAM	*Random Access Memory*
SSH	*Secure Shell*
TCP	*Transmission Control Protocol*
TLS	*Transport Layer Security*

UI	*User Interface*
USB	*Universal Serial Bus*
WEB	*World Wide Web*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

A domesticação de animais pelo homem teve início no período pré-histórico, onde ambos foram beneficiados com essa aproximação. O adestramento se iniciou quando os homens começaram a viver em determinadas regiões do mundo e passaram a usar a criação de animais para auxiliar tanto na agricultura quanto no transporte de pessoas.

No Brasil, 44,3% dos 65 milhões de domicílios possuem pelo menos um cachorro e 17,7% ao menos um gato, de acordo com dados do Instituto Brasileiro de Geografia e Estatística(IBGE). Atualmente, há aproximadamente 52,2 milhões de cães e 22,1 milhões de gatos no País (IBGE, 2015).

De acordo com uma pesquisa feita pela Associação Brasileira da Indústria de Produtos para Animais de Estimação (Abinpet), entre as classes B e C (de 10 a 20 salários mínimos, e de quatro a 10 salários mínimos, respectivamente), o custo médio mensal para cães com produtos padrão é de 3,2% e 7% da renda familiar, ou seja, R$ 302. Famílias que optam por gatos, nessa faixa salarial, gastam respectivamente 1,3% e 2,8% da renda/mês com a manutenção do animal, o que corresponde a R$121,39. Famílias da classe A (acima de 20 salários mínimos) podem optar por produtos padrão comprometendo somente 2,4% da renda total, no caso de cães e 1% no caso de gatos (ABINPET, 2016).

Com os dados apresentados é possível perceber a importância que os animais de estimação tem para as famílias do Brasil. Consequentemente, um produto que traga uma facilidade, auxiliando numa alimentação correta do animal, acaba se tornando vantajoso. Com o objetivo de implementar medidas para o monitoramento e controle de alimentação remoto, este projeto visa desenvolver uma solução barata e eficaz.

1.1 Objetivos

1.1.1 Objetivo geral

O objetivo geral deste projeto é desenvolver um sistema para alimentar remotamente os animais de estimação, com foco na detecção de pouca ração disponível ao animal através de processamento de imagens utilizando uma rede neural convolucional, uma câmera e um aplicativo móvel para definir um alarme para gerar notificação e para a liberação de comida remotamente. Para o funcionamento do sistema, exige-se um computador Raspberry Pi, uma câmera e um dispositivo móvel.

1.1.2 Objetivos específicos

A seguir são listados todos os objetivos específicos do projeto:

- Implementar um algoritmo inteligente, que visa detectar a falta de comida disponível ao animal com o objetivo de melhorar a confiabilidade do sistema como um todo;
- Realizar a comunicação entre Raspberry Pi e o dispositivo Android através do protocolo MQTT;
- Desenvolver uma aplicação fácil de usar e intuitiva, e um aplicativo com uma interface amigável, simples e clara;
- Desenvolver um algoritmo que envie notificações ao usuário quando a comida estiver acabando ou quando estiver sendo liberada;
- Implementar um sistema de medição de massa que apresente alta confiabilidade;
- Desenvolver um aplicativo de modo que ele não ocupe muito espaço em memória.

1.2 Justificativa

Com o aumento do número de animais de estimação na casa das famílias brasileiras e em uma sociedade em que as pessoas passam muito tempo fora de casa, é cada vez mais necessário um produto de baixo custo e de fácil operação para o cuidado e bem estar do animal.

O sistema de alimentação desenvolvido propõe integrar-se à vida digital do usuário, participando da sua rotina em conjunto com outros dispositivos (como *smartphones* e *tablets*), trazendo mais tranquilidade no cuidado de seus animais.

O uso de Deep Learning, além de permitir a detecção de pouco alimento disponível, torna possível a predição de outros aspectos do cotidiano do animal em trabalhos futuros.

1.3 Organização

Este documento inicia apontando o aspecto geral do sistema em questão e salientando os requisitos do projeto. Posteriormente indica a metodologia adotada, especificando cada ferramenta utilizada no desenvolvimento. Por fim, informa todo o processo de implementacão

de hardware, software, da implantacão do sistema em detalhes e uma análise de resultados juntamente à conclusão do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são descritos os diversos conceitos teóricos e princípios técnicos que foram utilizados para desenvolver o projeto.

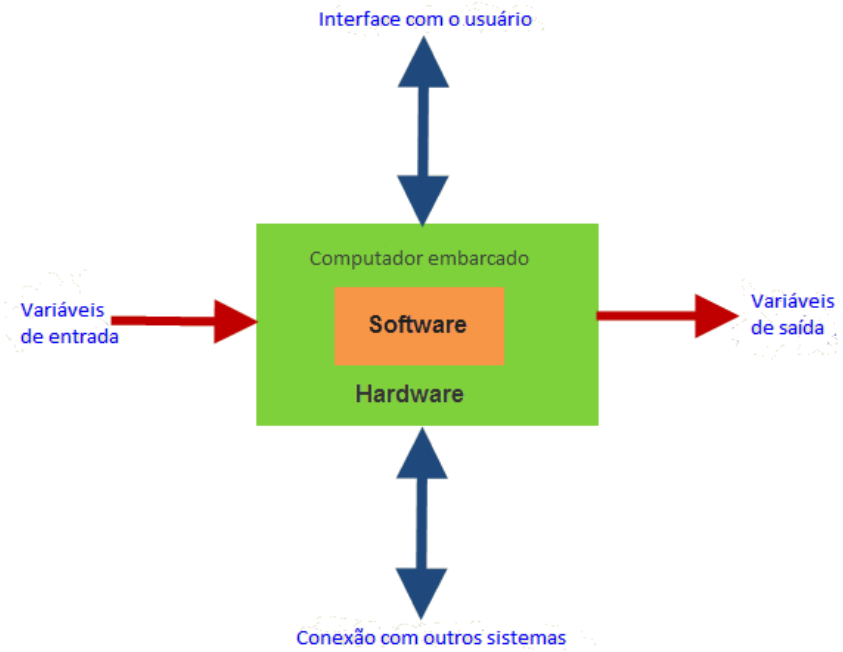
2.1 Sistemas Embarcados

Sistemas embarcados são uma combinação de hardware e software com severas restrições, que desempenham uma tarefa específica e devem ter alta confiabilidade e qualidade (NOERGAARD, 2005).

Um sistema embarcado usa uma plataforma de hardware para executar a operação. O hardware do sistema embarcado é montado com um microcontrolador. Possui elementos como interfaces de entrada/saída, memória e interface com o usuário.

A Figura 1 ilustra o diagrama de blocos do funcionamento de um sistema embarcado.

Figura 1 – Diagrama do funcionamento de um sistema embarcado.



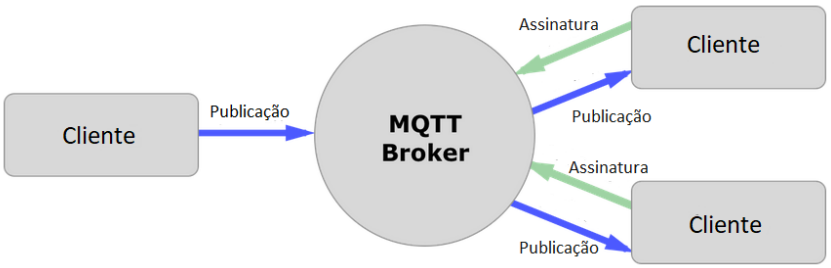
Fonte: Adaptado de (ELPROCUS, 2017)

Como os sistemas embarcados geralmente possuem apenas uma função, eles são capazes de operar com baixo consumo de energia e podem caber em um espaço pequeno em comparação com outros componentes. Eles também são baratos, tornando-os uma maneira acessível de controlar dispositivos.

2.2 MQTT

Para os dispositivos de Internet das Coisas (IoT), a conexão com a Internet é um requisito. Essa conexão permite que os dispositivos se comuniquem entre si e com serviços de *backend*. Desenvolvido com base na pilha de protocolos TCP/IP, o MQTT (*Message Queue Telemetry Transport*) tornou-se um dos principais padrões de comunicações IoT (YUAN, 2017). Na Figura 2 é possível visualizar um diagrama funcional do MQTT.

Figura 2 – Diagrama funcional do MQTT



Fonte: Adaptado de (HAMSTRA, 2017)

O padrão de troca de mensagens no MQTT é o *publish/subscriber*. Cada mensagem de publicação e assinatura contém o tópico que é usado pelo *broker* para filtragem de mensagens. A mensagem no publicador (*Publish*) contém a carga útil junto com o tópico, a qual consiste no conteúdo real a ser transmitido. Pode conter dados binários, dados textuais e até dados *JavaScript Object Notation* (JSON) ou *Extensible Markup Language* (XML). A mensagem no subscritor (*Subscriber*) contém a lista de nomes de tópicos nos quais o assinante está interessado. Por isso, o assinante pode assinar um ou mais tópicos de interesse (MANANDHAR, 2017). O *broker* é responsável por receber todas as mensagens, filtrá-las, determinar quem está inscrito em cada mensagem e, por fim, enviar para esses clientes inscritos.

A popularidade das mensagens baseadas no MQTT deriva da maneira simples pela qual as informações podem ser publicadas ou assinadas, sem a necessidade de saber quem ou o que está enviando ou recebendo as informações. Essa simplicidade permite que cada mensagem seja muito pequena em tamanho, reduzindo as demandas na rede e nos dispositivos de monitoramento remoto dos quais muitas mensagens MQTT emanam (LAMPKIN et al., 2012).

O MQTT é um protocolo de conectividade máquina-a-máquina (M2M) /"Internet das Coisas". Foi projetado como um transporte de mensagens de publicação / assinatura extremamente leve. É útil para conexões em locais remotos onde é necessário pouca largura de banda.

Entre seus principais benefícios estão (YUAN, 2017):

- Distribuir informações de maneira mais eficiente;
- Aumentar a escalabilidade;

- Reduzir drasticamente o consumo de largura de banda de rede;
- Maximiza a largura de banda disponível.

Este protocolo de mensagem usa segurança da camada de transporte (TLS), os dados são criptografados e sua integridade é validada. Da mesma forma, a maioria das implementações do MQTT usa funcionalidades de autorização no servidor para controlar o acesso (YUAN, 2017).

O nível de Qualidade de Serviço (QoS) é a qualidade da publicação entre o remetente de uma mensagem e o destinatário que define a garantia de entrega para uma mensagem específica. Existem 3 níveis de QoS no MQTT (THE HIVEMQ TEAM, 2015):

- No máximo uma vez (0);
- Pelo menos uma vez (1);
- Exatamente uma vez (2).

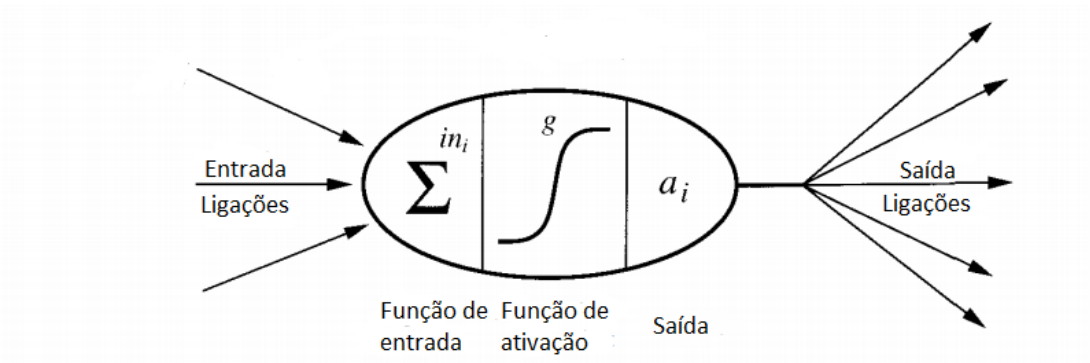
2.3 Redes neurais artificiais

Os elementos simples da computação aritmética correspondem aos neurônios - as células que realizam o processamento de informações no cérebro - e a rede como um todo corresponde a uma rede neural, coleção de neurônios interconectados.

Uma rede neural é composta de um número de nós, ou unidades, conectados por links. Cada link tem um peso numérico associado a ele. Os pesos são o principal meio de armazenamento de longo prazo em redes neurais, e o aprendizado geralmente ocorre atualizando os pesos. Algumas das unidades são conectadas ao ambiente externo e podem ser designadas como unidades de entrada ou saída. Os pesos são modificados para tentar alinhar o comportamento de entrada/saída da rede com o ambiente que fornece as entradas (RUSSEL; NORVIG, 2009).

Na Figura 3 é possível visualizar a representação do modelo matemático de uma rede neural.

Figura 3 – Representação de um nó.



Fonte: Adaptado de (RUSSEL; NORVIG, 2009)

Redes neurais multicamadas contêm mais de uma camada computacional. O nó contém

uma camada de entrada e saída, da qual a camada de saída é a única camada de processamento computacional. A camada de entrada transmite os dados para a camada de saída e todos os cálculos são completamente visíveis para o usuário. As camadas intermediárias adicionais (entre entrada e saída) são chamadas de camadas ocultas porque os cálculos executados não são visíveis para o usuário.

A arquitetura específica de redes neurais de multicamadas é chamada de redes *feed-forward*, porque camadas sucessivas se alimentam umas das outras na direção da entrada para saída. A arquitetura padrão das redes de *feed-forward* pressupõe que todos os nós em um camada são conectados aos da próxima camada. Portanto, a arquitetura da rede neural é quase totalmente definida, uma vez que o número de camadas e o número/tipo de nós em cada camada foram definidos. O único detalhe restante é a função de perda que é otimizada na camada de saída. É extremamente comum usar saídas *softmax* com perda de entropia cruzada para previsão discreta e saídas lineares com perda quadrática para previsão com valor real (AGGARWAL, 2018).

2.3.1 Aprendizagem profunda

O *Deep Learning* permite modelos computacionais compostos de múltiplas camadas de processamento para aprender representações de dados com múltiplos níveis de abstração. Esses métodos melhoraram drasticamente o estado da arte em reconhecimento de fala, reconhecimento de objetos visuais, detecção de objetos e muitos outros domínios. O *Deep Learning* descobre uma estrutura complexa em grandes conjuntos de dados usando o algoritmo de retropropagação para indicar como uma máquina deve alterar seus parâmetros internos que são usados para calcular a representação em uma camada anterior (LECUN; BENGIO; HINTON, 2015).

2.3.2 Redes neurais convolucionais

Uma rede neural convolucional (CNN) é um tipo de rede neural artificial usada no reconhecimento e processamento de imagens que é especificamente projetada para processar dados de pixel.

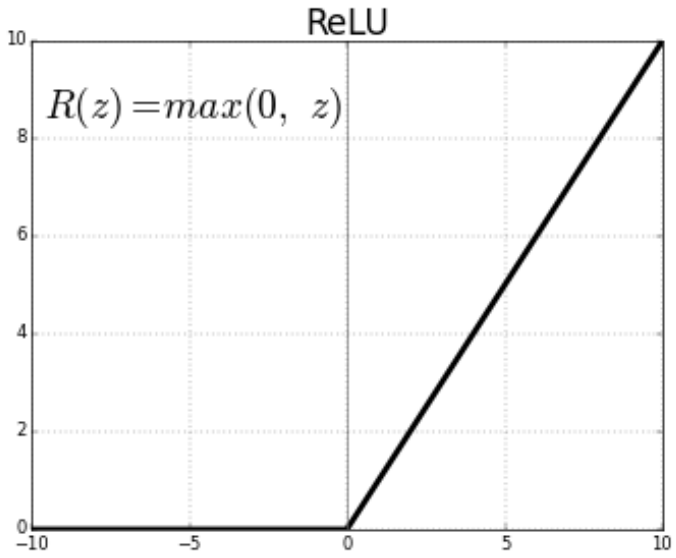
Camadas de redes neurais tradicionais usam a multiplicação de matrizes por uma matriz de parâmetros com um parâmetro separado descrevendo a interação entre cada unidade de entrada e cada unidade de saída. Isso significa que cada unidade de saída interage com cada unidade de entrada. As redes de convolução, no entanto, normalmente têm interações separadas. Por exemplo, ao processar uma imagem, a imagem de entrada pode ter milhares ou milhões de pixels, mas pode-se detectar recursos pequenos e significativos, como bordas com núcleos que ocupam apenas dezenas ou centenas de pixels. Isso significa que é preciso armazenar menos parâmetros, o que reduz os requisitos de memória do modelo e melhora sua eficiência estatística (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.3.3 Função de ativação

A função de ativação de um nó define a saída desse nó, ou "neurônio", dado uma entrada ou conjunto de entradas. Esta saída é então usada como entrada para o próximo nó e assim por diante até que uma solução desejada para o problema original seja encontrada. ReLU é linear para todos os valores positivos e zero para todos os valores negativos. Em comparação com os neurônios que utilizam tanh/sigmóide que envolvem operações caras (exponenciais, etc.), a ReLU pode ser implementada simplesmente limitando uma matriz de ativações em zero.

A Figura 4 mostra a representação gráfica da função de ativação utilizada no projeto, onde apenas o primeiro quadrante é considerado e o restante é descartado.

Figura 4 – Representação gráfica da função de ativação ReLU.



Fonte: (RELU, 2018)

2.3.4 LeNet

A primeira aplicação bem conhecida e bem sucedida de redes neurais convolucionais foi a LeNet-5, descrita por LeCun et al. (1998).

O sistema foi desenvolvido para uso em um problema de reconhecimento de caracteres manuscritos e demonstrado no conjunto de dados padrão MNIST, alcançando uma precisão de classificação de aproximadamente 99,2% (ou uma taxa de erro de 0,8%). A rede foi então descrita como a técnica central em um sistema mais amplo, denominado Graph Transformer Networks LeCun et al. (1998).

A entrada para a arquitetura LeNet-5 é uma imagem em escala de cinza com tamanho de 32×32 pixels. O modelo propõe um padrão de uma camada convolucional seguida por uma camada de *pool* média, conhecida como camada de subamostragem. Esse padrão é repetido

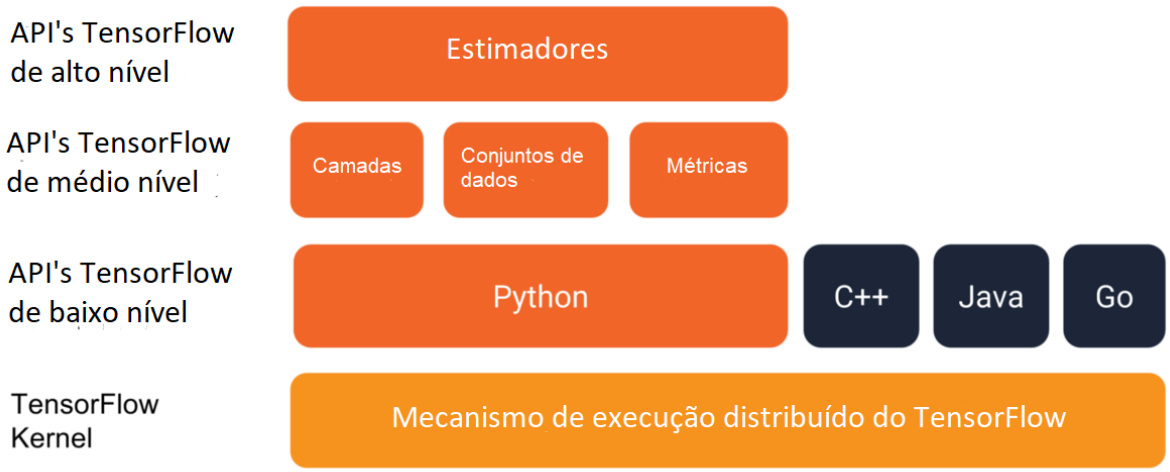
duas vezes e meia antes de os mapas de recursos de saída serem achatados e alimentados a um número de camadas totalmente conectadas para interpretação e previsão final.

2.4 Tensorflow

O TensorFlow é uma biblioteca de código aberto utilizado para aprendizagem de máquina. Esta biblioteca foi inicialmente desenvolvida pela equipe de inteligência artificial do Google. O TensorFlow permite o desenvolvimento em Python e C++ e pode ser executado em CPU's e GPU's.

Na Figura 5 é possível visualizar os diversos níveis das API's do TensorFlow.

Figura 5 – Representação do TensorFlow



Fonte: Adaptado de (TENSORFLOW, 2019)

O Tensorflow é uma estrutura computacional para construir modelos de aprendizado de máquina. Esta biblioteca fornece uma variedade de kits de ferramentas diferentes que permitem construir modelos em diversos níveis de abstração. É possível usar APIs de nível inferior para criar modelos definindo uma série de operações matemáticas. Como alternativa, é possível usar APIs de nível superior (como *tf.estimator*) para especificar arquiteturas predefinidas, como regressores lineares ou redes neurais.

2.5 Keras

Keras é uma API de redes neurais de alto nível, escrita em Python e capaz de rodar em cima do TensorFlow, *Microsoft Cognitive Toolkit* (CNTK) ou Theano. Foi desenvolvido com foco em permitir o desenvolvimento rápido, sendo possível suportar redes convolucionais e redes recorrentes, bem como a combinação das duas. Funciona em *Graphics Processing Unit* (GPU) e textitCentral Processing Unit (CPU) (CHOLLET et al., 2015).

O Keras foi criado através dos seguintes princípios (CHOLLET et al., 2015):

- **Facilidade de uso**. A Keras oferece APIs consistentes e simples, minimiza o número de ações do usuário necessárias para casos de uso comuns e fornece um *feedback* claro e acionável sobre o erro do usuário;
- **Modularidade**. Um modelo é entendido como uma sequência ou um gráfico de módulos independentes, totalmente configuráveis, que podem ser conectados com o menor número de restrições possível;
- **Fácil extensibilidade**. Novos módulos são simples de adicionar (como novas classes e funções) e os módulos existentes fornecem exemplos amplos;
- **Funciona com o Python**. Não há arquivos de configuração de modelos separados em um formato declarativo. Os modelos são escritos em código Python, que é compacto, fácil de depurar e permite a facilidade de extensibilidade.

2.6 Android

Android é o nome do sistema operacional para dispositivos móveis baseado no Linux.

2.6.1 Android Studio

O Android Studio é o ambiente de desenvolvimento integrado (IDE) oficial para o desenvolvimento de aplicativos Android e é baseado no IntelliJ IDEA.

Para suportar o desenvolvimento de aplicativos no sistema operacional Android, o Android Studio usa um sistema de criação, emulador, modelos de código e integração do Github baseados em *Gradle*. Todos os projetos no Android Studio tem uma ou mais modalidades com código-fonte e arquivos de recursos. Essas modalidades incluem módulos de aplicativos para Android, módulos de biblioteca e módulos do Google App Engine.

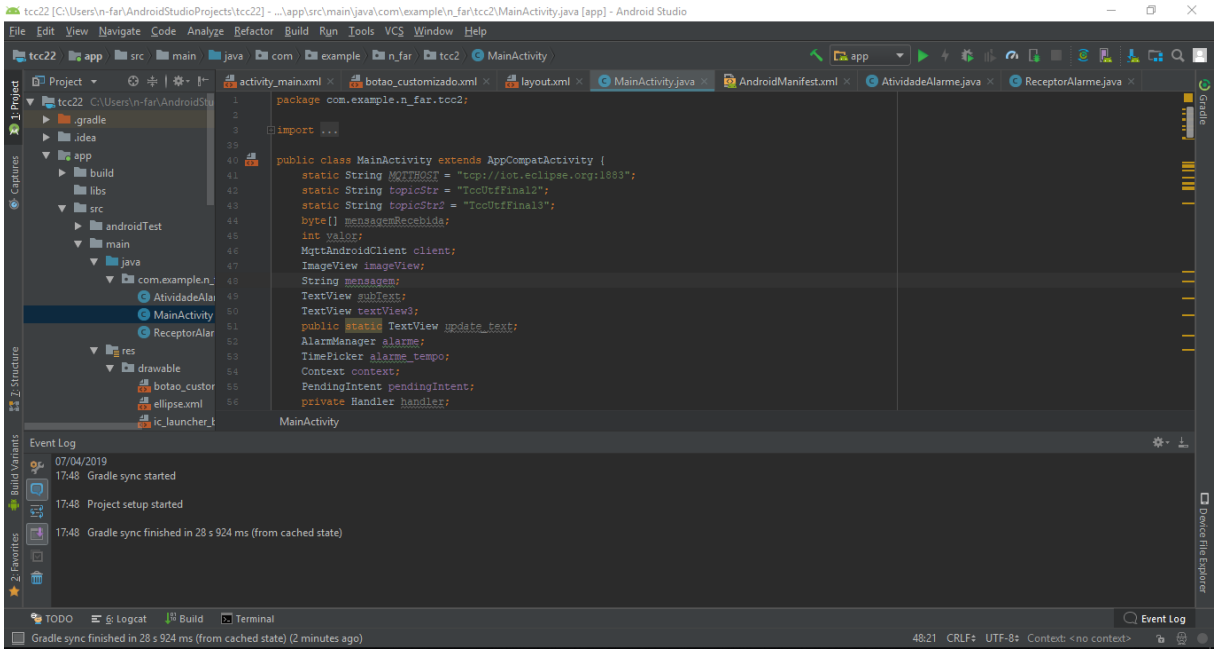
A Figura 6 mostra a interface com o usuário do ambiente de desenvolvimento Android Studio.

2.6.2 Arquitetura

O Android é uma pilha de software com base em Linux, de código aberto e criada para diversos dispositivos. A principal plataforma suportada pelo sistema Android é a arquitetura ARM com arquitetura x86 (DEVELOPERS, 2019).

Os componentes da arquitetura do Android são uma coleção de bibliotecas que ajudam a criar aplicativos robustos, testáveis e de fácil manutenção. Com classes para gerenciar o ciclo de vida de seu componente de UI e lidar com a persistência de dados (DEVELOPERS, 2019).

Figura 6 – Ambiente de Desenvolvimento Android Studio.



Fonte: Autoria própria (2019)

2.6.3 Trabalho relacionado

Nesta seção será apresentado um trabalho relacionado ao tema abordado. A proposta é a mesma, desenvolver um alimentador automático para animais domésticos, porém foram utilizadas algumas tecnologias diferentes que serão explicitadas a seguir.

No trabalho de Pazini e Pereira (2016), foi desenvolvido um dispenser automático de alimentos, onde o sistema utiliza uma rosca helicoidal acoplada a um motor DC para o despejamento da ração. Esta funcionalidade faz com que o controle da quantidade de ração que é servida seja feita através do tempo em que o motor fica acionado.

Na interface com o usuário, foi utilizado um sistema *WEB* para programar os horários de servimento, a quantidade de ração e acionar o dispositivo para servir. O protocolo utilizado na comunicação foi o *SSH*.

Para o sistema desenvolvido neste relatório, foi desenvolvida uma pá de 4 quadrantes, responsável pelo servimento da ração, que também é acionada por um motor DC. Foi desenvolvido um aplicativo Android para servir de interface para o sistema e o protocolo de comunicação utilizado foi o MQTT. O sistema desenvolvido ainda conta com um algoritmo de *Deep Learning*, para a detecção de pouca ração disponível.

3 METODOLOGIA

Neste capítulo são apresentados a metodologia e os procedimentos utilizados para a realização deste projeto.

3.1 Raspberry Pi

O Raspberry Pi 3 Model B+ é um pequeno computador de baixo-custo, com a possibilidade de ser utilizado em projetos eletrônicos. Este modelo possui um processador *quadcore* de 64 bits com *clock* de 1,4 GHz, 1 GB de memória RAM, 4 portas USB 2.0, GPIO de 40 pinos e Adaptador Wi-Fi 802.11 b/g/n/AC 2.4GHz e 5GHz integrado.

A Figura 7 exibe uma imagem da placa de desenvolvimento Raspberry Pi 3 model B+.

Figura 7 – Raspberry Pi 3 model B+

Fonte: (RASPBERRY PI, 2019)

O sistema operacional empregado na placa foi o Raspbian[1] e a linguagem utilizada foi o Python. Foi decidido utilizar esta placa por apresentar capacidade de processamento suficiente para a implementação deste projeto.

3.2 *Strain Gauge*

Um *Strain gauge* é um sensor cuja resistência varia com a força aplicada. Converte força, pressão, tensão e peso em uma mudança na resistência elétrica que pode ser medida.

[1] https://www.raspbian.org

Quando forças externas são aplicadas a um objeto estacionário, o estresse e a tensão são o resultado. O estresse é definido como as forças de resistência internas do objeto.

3.2.1 HX711

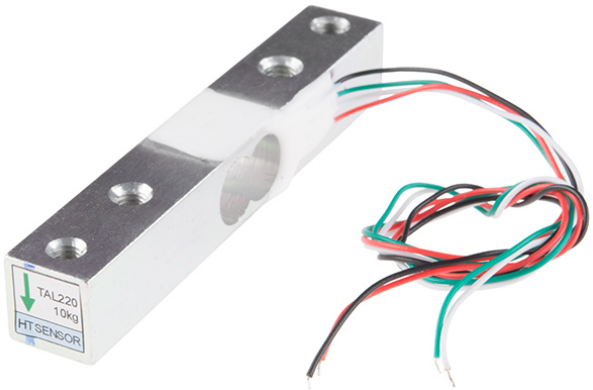
O HX711 é um conversor analógico-digital de precisão de 24 bits (ADC) projetado para balanças e aplicações de controle industrial com interface direta a uma ponte para sensores (AVIA, 2019).

O multiplexador de entrada seleciona o canal A ou B como entrada, possui um amplificador de ganho programável (PGA) para o baixo ruído. O Canal A pode ser programado com um ganho de 128 ou 64, correspondendo a uma tensão total de +- 20mV ou +- 40mV respectivamente, quando uma fonte de 5V é conectada ao pino de alimentação analógico AVDD. O canal B tem um ganho fixo de 32 (AVIA, 2019).

3.2.2 Célula de carga

Esta célula de carga de barra reta, ilustrada na Figura 8, pode traduzir até 10KG de pressão (força) em um sinal elétrico. As células de carga são projetadas para medir uma força específica e ignorar outras forças sendo aplicadas. O sinal elétrico emitido pela célula de carga é muito pequeno e requer um amplificador de célula de carga, como o HX711, utilizado neste projeto.

Figura 8 – Célula de carga.

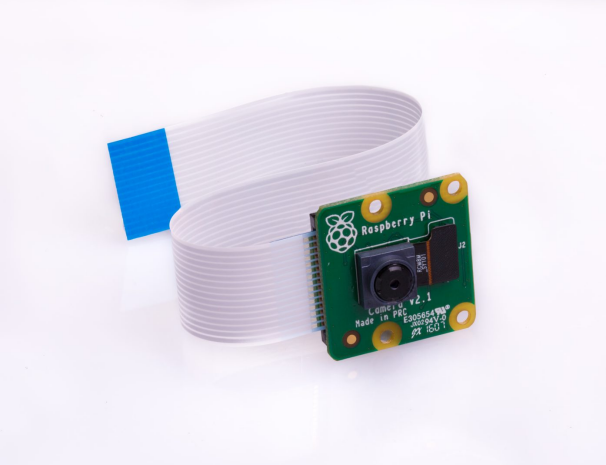


Fonte: (SPARKFUN, 2019)

3.3 Câmera

A Câmera utilizada, mostrada na Figura 9, usa como sensor o OV5647, permitindo imagens com alta resolução 1080p. O ajuste de foco permite regular a distância ao objeto sem que seja preciso tirar a câmera do lugar, facilitando o manuseio.

Figura 9 – Câmera Raspberry Pi v2



Fonte: (RASPBERRY PI, 2019)

A câmera possui resolução de 5MP, Abertura (F) de 1,8, Comprimento focal ajustável de 3,6mm e Ângulo de visão diagonal de 75,7 graus.

3.4 *Deep Learning*

Deep Learning é uma área de pesquisa em Aprendizado de Máquina, que foi introduzida com o objetivo de se aproximar da Inteligência Artificial. *Deep Learning* é sobre aprender vários níveis de representação e abstração que ajudam a dar sentido aos dados como imagens, som e texto (DENG; YU, 2014).

3.4.1 Conjunto de dados

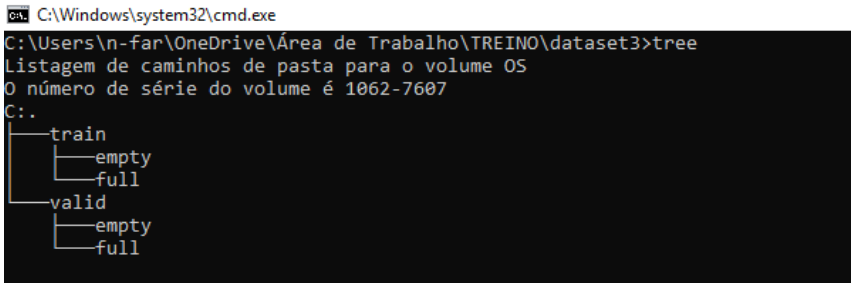
Inicialmente, foi criado um diretório de dados de treinamento (*train*) e um diretório de dados de validação (*valid*) contendo um subdiretório por classe de imagem, preenchido com imagens .jpg.

Neste projeto, foram utilizados dois conjuntos de imagens gerados pela equipe: 195 imagens de pote vazio e 265 imagens de pote parcialmente cheio ou totalmente cheio.

Foram utilizadas 100 imagens para a validação do modelo, sendo 50 imagens localizadas na classe *empty* e 50 imagens situadas na classe *full*. A estrutura dos diretórios que contém todas as imagens utilizadas está representada na Figura 10.

As redes neurais convolucionais - um algoritmo pilar de *deep learning* - são um dos melhores modelos disponíveis para a maioria dos problemas "perceptivos" (como a classificação de imagens), mesmo com poucos dados para a aprendizagem.

Figura 10 – Estrutura dos diretórios contendo as imagens do conjunto de dados.



Fonte: Autoria própria (2019)

3.4.2 Pré-processamento de dados

Para aproveitar os nossos exemplos de treinamento, eles foram "aumentados" através de várias transformações aleatórias, para que o modelo implementado nunca "veja" duas vezes a mesma imagem. Isso ajuda a prevenir o *overfitting* e ajuda o modelo a se adaptar melhor. O *overfitting* ocorre quando o modelo se ajusta muito bem aos dados com os quais está sendo treinado, porém, não se adapta bem a novos dados. A Figura 11 traz uma parte do código que faz este pré-processamento.

Figura 11 – Código para pré-processamento das imagens.

```
68  from keras.preprocessing.image import ImageDataGenerator
69
70  train_datagen = ImageDataGenerator(
71      rescale=1. / 255,
72      shear_range=0.2,
73      zoom_range=0.2,
74      horizontal_flip=True)
75
```

Fonte: Autoria própria (2019)

Os seguintes parâmetros são configurados no pré-processamento das imagens:

- *rescale* é um valor pelo qual os dados foram multiplicados antes de qualquer outro processamento. As imagens originais consistem em coeficientes RGB com valores de 0 a 255, mas tais valores seriam altos demais para os modelos processarem (considerando uma taxa de aprendizado típica), então, buscando valores entre 0 e 1, a imagem foi dimensionada com o valor 1/255;
- *zoom_range* serve para obter um zoom aleatório nas imagens;

- *horizontal_flip* serve para inverter de modo aleatório metade das imagens horizontalmente - relevante quando não há suposição de assimetria horizontal.

É possível verificar como as imagens são transformadas, vizualizando a Figura 12, com o exemplo de uma foto de um gato.

Figura 12 – Como as imagens se parecem após a geração dos dados.

Fonte: Autoria própria (2019)

3.4.3 Modelo de rede neural

Para a elaboração do modelo de rede neural foi utilizado o modelo sequencial do Keras, visto que a API *sequencial* permite criar uma pilha linear de camadas. O modelo da rede neural foi feito da seguinte maneira:

- Camada Convolucional (1): aplica 32 filtros 3x3 (extraindo sub-regiões de 3x3 pixels), com função de ativação ReLU;
- Camada de *Pooling* (1): executa o *pool* máximo com um filtro 2x2;
- Camada Convolucional (2): aplica 32 filtros 3x3, com função de ativação ReLU;
- Camada de *Pooling* (2): novamente, realiza o *pool* máximo com um filtro 2x2;
- Camada Convolucional (3): aplica 64 filtros 3x3, com função de ativação ReLU;
- Camada de *Pooling* (3): novamente, realiza o *pool* máximo com um filtro 2x2;
- Camada Densa (1): 64 neurônios;
- Camada Densa (2): 2 neurônios, um para cada classe alvo, '*empty*' e '*full*', com função de ativação *softmax*.

Os métodos utilizados na criação dos três tipos de camadas necessárias para a rede neural foram os seguintes:

- conv2d () - Constrói uma camada convolucional bidimensional. Obtém o número de filtros, o tamanho do kernel do filtro, o preenchimento e a função de ativação como argumentos;
- max pooling2d () - Constrói uma camada de *pool* bidimensional usando o algoritmo de *pool* máximo. Leva o tamanho do filtro de *pool* e *stride* como argumentos;

- dense() - Constrói uma camada densa. Leva número de neurônios e função de ativacão como argumentos.

O modelo é constituído de uma pilha simples de 3 camadas de convolução com uma ativação de ReLU e seguida por camadas de *pool* máximo. Sendo semelhante às arquiteturas que Yann LeCun defendeu na década de 1990, como a LeNet - 5, para a classificação de imagens (com a exceção de ReLU). Isto pode ser observado analisando a Figura 13.

Figura 13 – Código responsável pela construção do modelo de rede neural.

```
39  model = Sequential()
40  model.add(Conv2D(32, (3, 3) , input_shape=input_shape))
41  model.add(Activation('relu'))
42  model.add(MaxPooling2D(pool_size=(2, 2)))
43
44  model.add(Conv2D(32, (3, 3)))
45  model.add(Activation('relu'))
46  model.add(MaxPooling2D(pool_size=(2, 2)))
47
48  model.add(Conv2D(64, (3, 3)))
49  model.add(Activation('relu'))
50  model.add(MaxPooling2D(pool_size=(2, 2)))
51
52  model.add(Flatten())
53  model.add(Dense(64))
54  model.add(Activation('relu'))
55  model.add(Dense(2))
56  model.add(Activation('softmax'))
```

Fonte: Autoria própria (2019)

3.4.4 Treinamento do modelo da rede neural

Para a criação do modelo foram definidas algumas variáveis, como por exemplo, *batch* e *epochs*. *Batch* é a quantidade de imagens utilizadas que serão propagadas na nossa rede neural, neste caso, nosso tamanho do *batch* foi 16. Já a variável *epoch* determina quantas vezes nossas imagens de treinamento irão passar pela rede neural, ou seja, neste caso, iremos passar 50 vezes. Todos os parâmetros citados são conhecidos como hiper-parâmetros, portanto seus valores são definidos através de testes e ajustes finos.

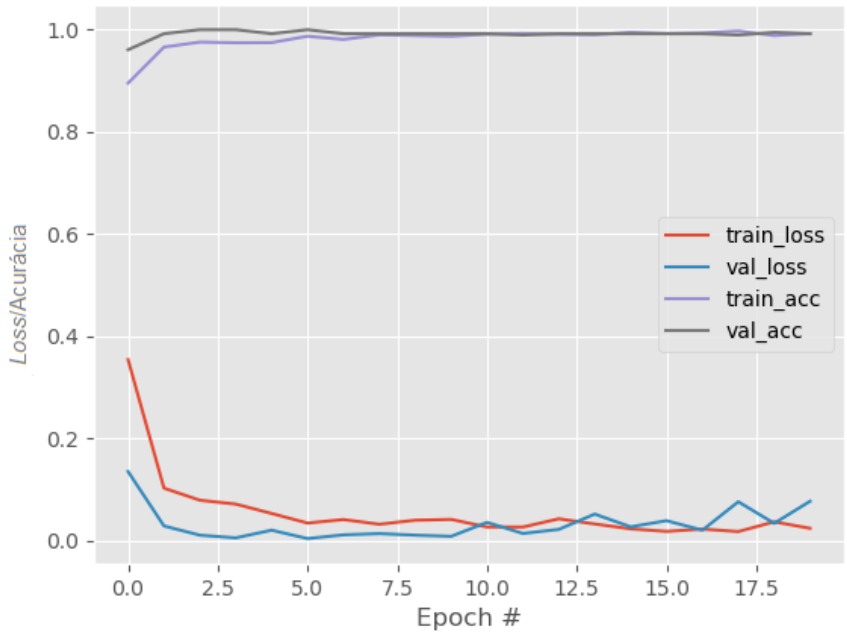
O otimizador utilizado foi o RMSprop e como resultado, foi obtido um valor de 100 % de acurácia para as imagens de treinamento, como é possível observar nas figuras 14 e 15.

Figura 14 – Treinamento da rede neural.

```
Epoch 13/20
50/50 [==============================] - 99s 2s/step - loss: 0.0568 - acc: 0.9887 - val_loss: 3.6559e-05 - val_acc: 1.0000
Epoch 14/20
50/50 [==============================] - 102s 2s/step - loss: 0.1194 - acc: 0.9750 - val_loss: 0.0035 - val_acc: 1.0000
Epoch 15/20
50/50 [==============================] - 97s 2s/step - loss: 0.0316 - acc: 0.9900 - val_loss: 0.0148 - val_acc: 0.9922
Epoch 16/20
50/50 [==============================] - 101s 2s/step - loss: 0.0560 - acc: 0.9825 - val_loss: 2.7898e-04 - val_acc: 1.0000
Epoch 17/20
50/50 [==============================] - 99s 2s/step - loss: 0.0327 - acc: 0.9875 - val_loss: 1.1136e-05 - val_acc: 1.0000
Epoch 18/20
50/50 [==============================] - 97s 2s/step - loss: 0.1118 - acc: 0.9787 - val_loss: 4.4589e-04 - val_acc: 1.0000
Epoch 19/20
50/50 [==============================] - 96s 2s/step - loss: 0.0402 - acc: 0.9900 - val_loss: 2.1059e-04 - val_acc: 1.0000
Epoch 20/20
50/50 [==============================] - 103s 2s/step - loss: 0.0627 - acc: 0.9875 - val_loss: 6.4915e-05 - val_acc: 1.0000
```

Fonte: Autoria própria (2019)

Figura 15 – *Loss* e acurácia durante os *epochs* da rede neural.



Fonte: Autoria própria (2019)

3.4.5 Modelo Alternativo

Outro modelo foi desenvolvido paralelamente ao utilizado no projeto, porém, apesar de apresentar acurácia semelhante, o uso do processador e da memória da placa de desenvolvimento foram maiores que no caso do modelo anterior.

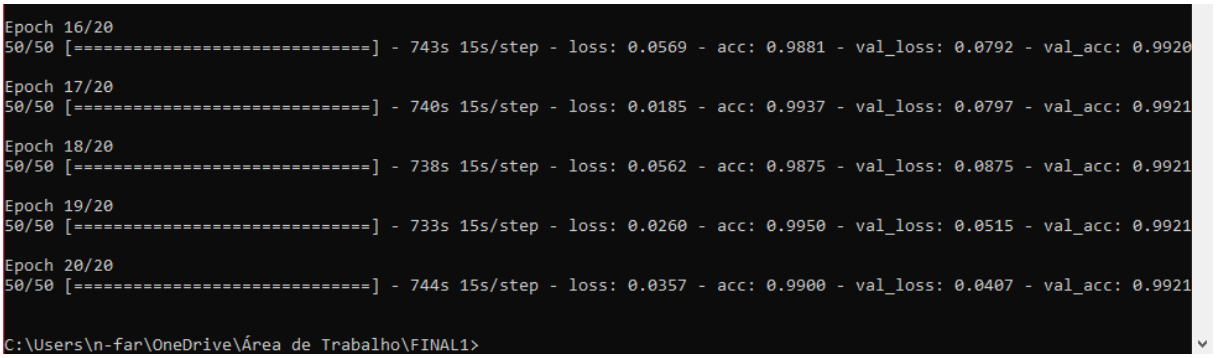
As características do modelo alternativo construído são apresentadas a seguir.

- Camada Convolucional (1): aplica 32 filtros 3x3 (extraindo sub-regiões de 3x3 pixels), com função de ativação ReLU;
- Camada Convolucional (2): aplica 32 filtros 3x3 (extraindo sub-regiões de 3x3 pixels), com função de ativação ReLU;

- Camada de *Pooling* (1): executa o *pool* máximo com um filtro 2x2;
- Camada Convolucional (3): aplica 64 filtros 3x3 (extraindo sub-regiões de 3x3 pixels), com função de ativação ReLU;
- Camada Convolucional (4): aplica 64 filtros 3x3 (extraindo sub-regiões de 3x3 pixels), com função de ativação ReLU;
- Camada de *Pooling* (2): executa o *pool* máximo com um filtro 2x2;
- Camada Convolucional (5): aplica 128 filtros 3x3 (extraindo sub-regiões de 3x3 pixels), com função de ativação ReLU;
- Camada Convolucional (6): aplica 128 filtros 3x3 (extraindo sub-regiões de 3x3 pixels), com função de ativação ReLU;
- Camada de *Pooling* (3): executa o *pool* máximo com um filtro 2x2;
- Camada Convolucional (7): aplica 256 filtros 3x3 (extraindo sub-regiões de 3x3 pixels), com função de ativação ReLU;
- Camada Convolucional (8): aplica 256 filtros 3x3 (extraindo sub-regiões de 3x3 pixels), com função de ativação ReLU;
- Camada de *Pooling* (4): executa o *pool* máximo com um filtro 2x2;
- Camada Densa (1): 256 neurônios;
- Camada Densa (2) : 2 neurônios, um para cada classe alvo, '*empty*' e '*full*', com função de ativação "sigmoid".

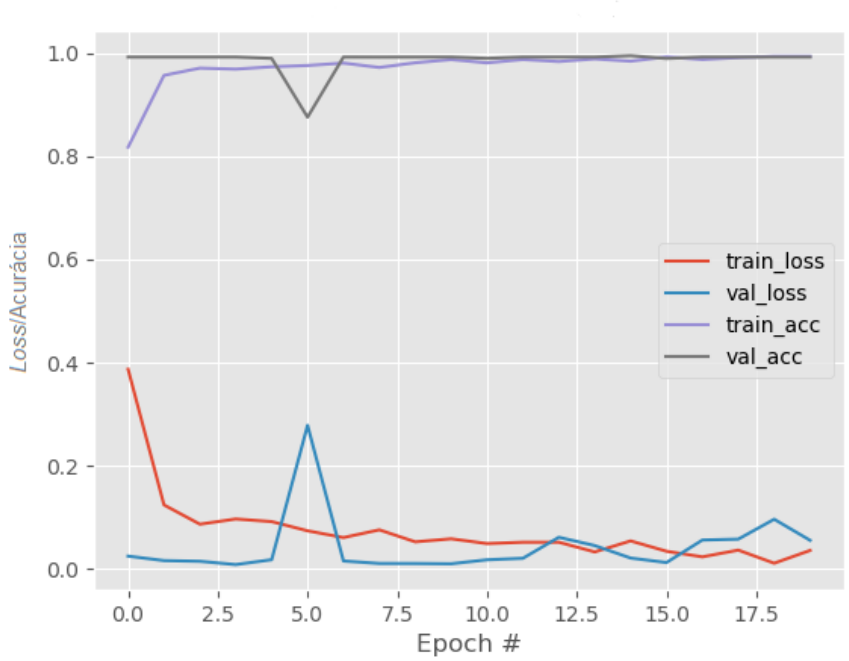
As figuras 16 e 17 apresentam os resultados obtidos com a rede neural alternativa, onde os valores obtidos foram semelhantes ao modelo utilizado no projeto, porém foi descartado por ser mais complexo e apresentar tempos maiores para realizar a predição.

Figura 16 – Treinamento da rede neural alternativa.



Fonte: Autoria própria (2019)

Figura 17 – *Loss* e acurácia durante os *epochs* da rede neural alternativa.



Fonte: Autoria própria (2019)

3.5 Aplicativo Android

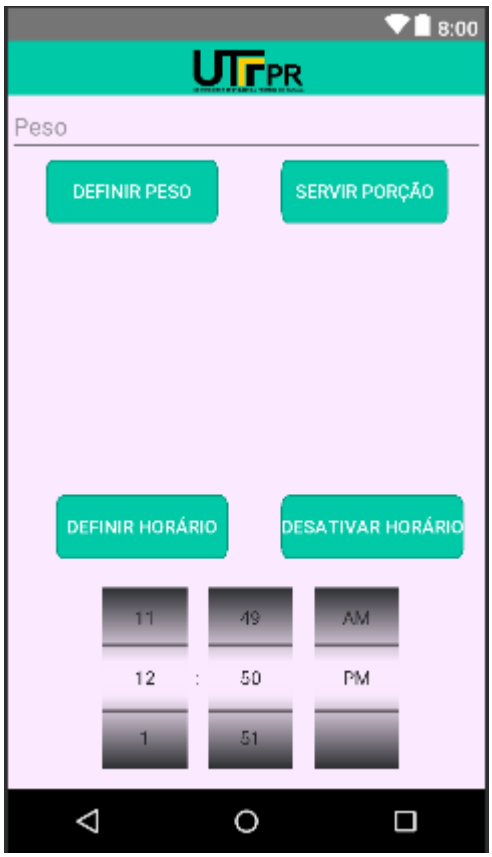
O aplicativo foi desenvolvido em Java e *Extensible Markup Language* (XML). A linguagem XML é utilizada para o desenvolvimento de uma interface com o usuário, mas também dá acesso a todos os aspectos do celular que é executado. A linguagem de programação Java é utilizada para o desenvolvimento das funções das aplicações.

O aplicativo, ilustrado na Figura 18, consiste em duas classes:

- MainActivity - É responsável por realizar a conexão com o servidor do MQTT, enviar e receber dados do MQTT e definir o horário para a liberação do alimento ;
- AtividadeAlarme - Tem a função de enviar as notificações para o usuário quando a comida é liberada e quando a comida está acabando.

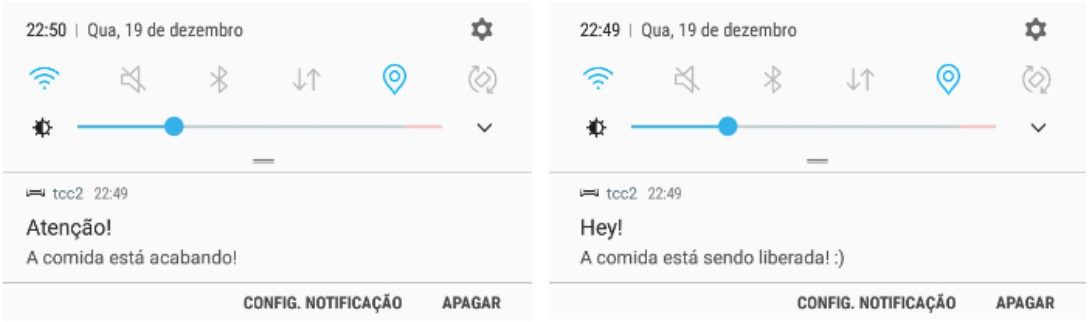
A Figura 19 apresenta as duas situações de notificação que o aplicativo é capaz de enviar.

Figura 18 – Interface com o usuário do aplicativo desenvolvido.



Fonte: Autoria própria

Figura 19 – Os dois casos de notificação do aplicativo.



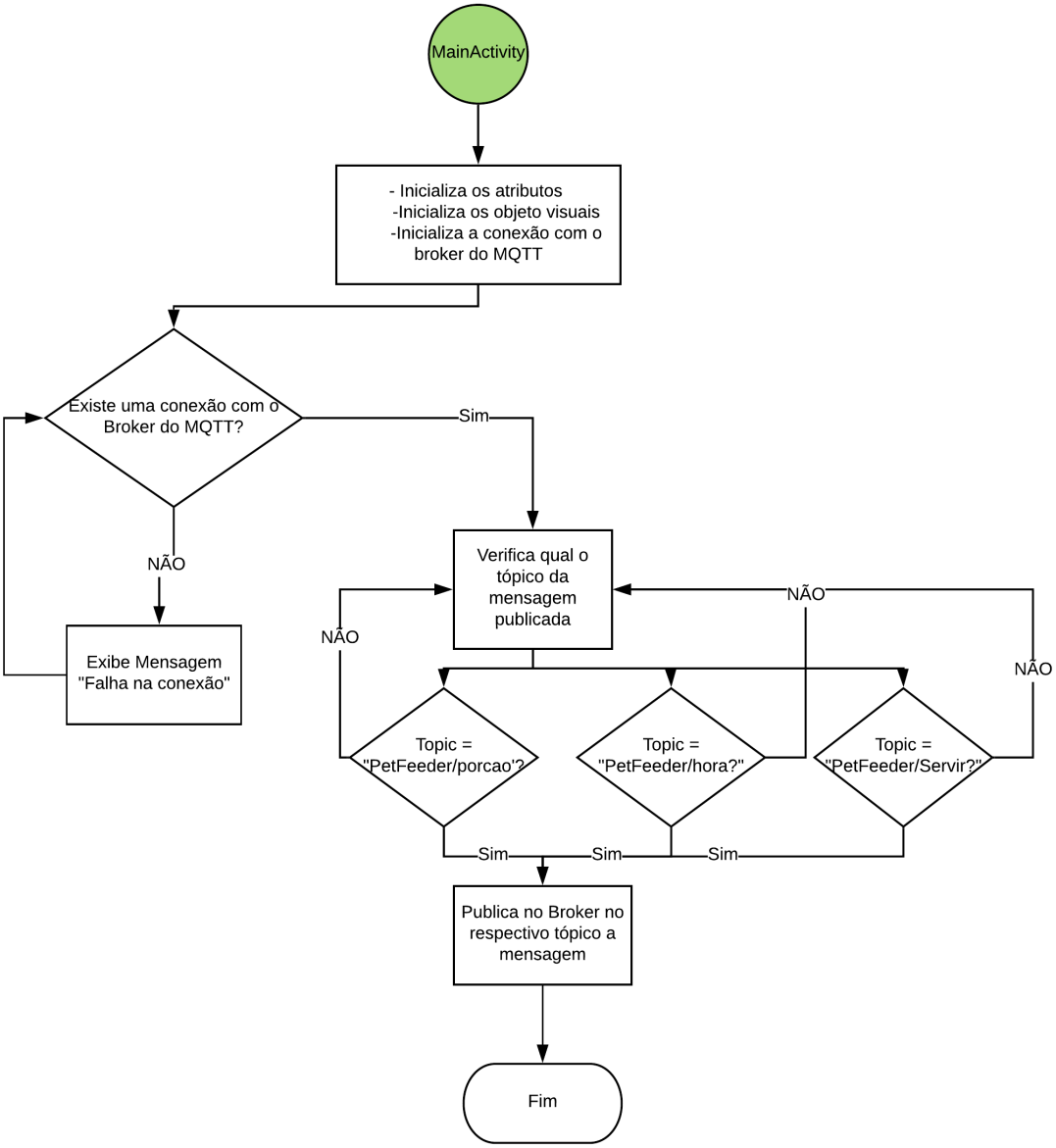
Fonte: Autoria própria (2019)

No caso de a comida estar acabando, o modelo de *Deep learning* detecta a falta de alimento disponível, consequentemente uma notificação é enviada para o usuário, informando que é necessário efetuar uma reposição. Acontecendo o evento de liberar a comida remotamente, uma nova notificação é enviada ao usuário, informando a liberação.

Os fluxogramas das figuras 20 e 21 explicam de maneira mais simplificada como a `MainActivity` é executada, qual a sua lógica de programação e quais são os seus principais

processos e métodos utilizados para as duas situações possíveis de enviar e receber uma mensagem do *broker* do MQTT.

Figura 20 – Fluxograma simplificado da MainActivity para a situação de publicar mensagens no *broker*.

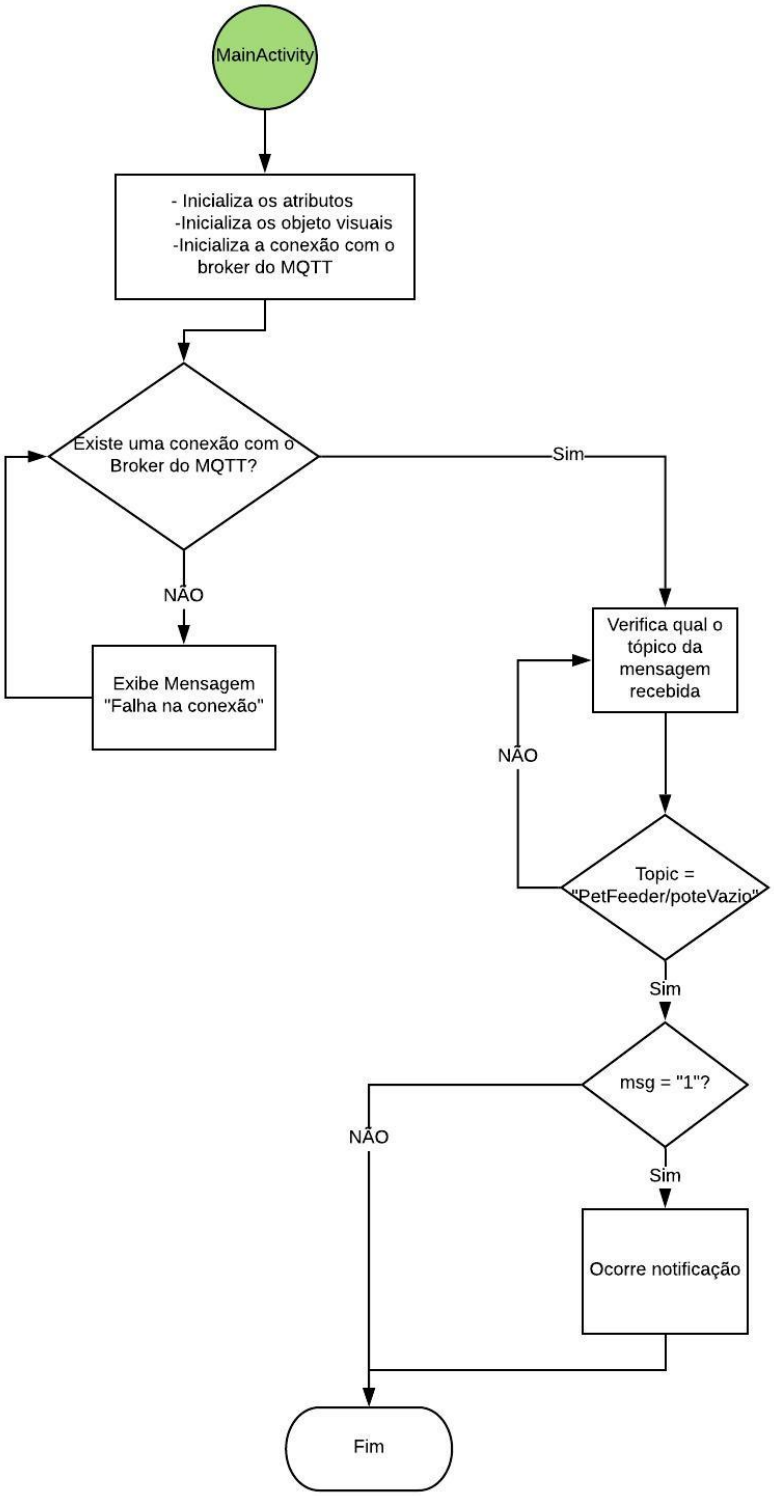


Fonte: Autoria própria (2019)

Caso o tópico seja "PetFeeder/porcao", o valor enviado vai designar o peso desejado para ficar disponível ao animal, na situação do tópico "PetFeeder/hora", o valor vai definir o horário da liberação da ração, e no último caso, em que o tópico é "PetFeeder/servir", a Raspberry Pi libera imediatamente a ração.

No caso de receber a mensagem "1"no tópico "PetFeeder/poteVazio", o aplicativo interpreta como pouca ração disponível ao animal, através do modelo de *Deep Learning*, e

Figura 21 – Fluxograma simplificado da MainActivity para a situação de recebimento de mensagens do *broker*.



Fonte: Autoria própria (2019)

notifica o usuário.

3.6 Circuitos

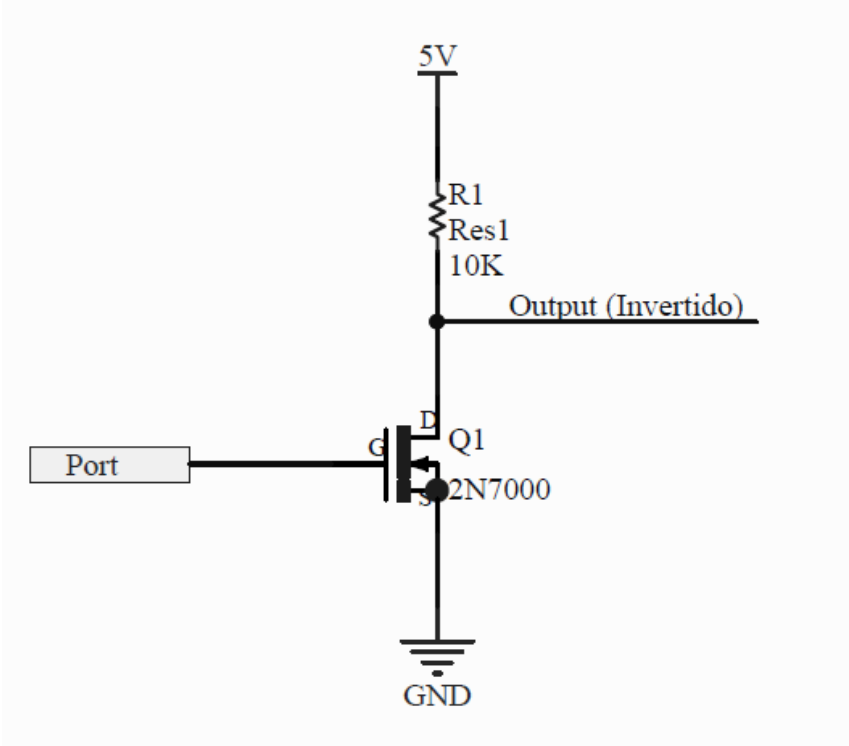
Um circuito é um caminho fechado que permite que a eletricidade flua de um ponto para outro. Pode incluir vários componentes elétricos, como transistores, resistores e capacitores.

3.6.1 *Step-up level shifters*

Na Raspberry Pi 3 model B+ o nível lógico alto é de 3,3V, sendo necessário um circuito de *step up* para o acionamento do relé.

Para o projeto foi utilizado um MOSFET 2N7000 e um resistor de 10KΩ, vide Figura 22. Um transistor MOSFET é usado como uma chave, acionando a porta com lógica 3,3V e obtendo a saída lógica de 5V do dreno. Este é um circuito simples e confiável, porém inverte o nível lógico de 3,3V.

Figura 22 – Diagrama do circuito com MOSFET para o *Step-up level shifters*.

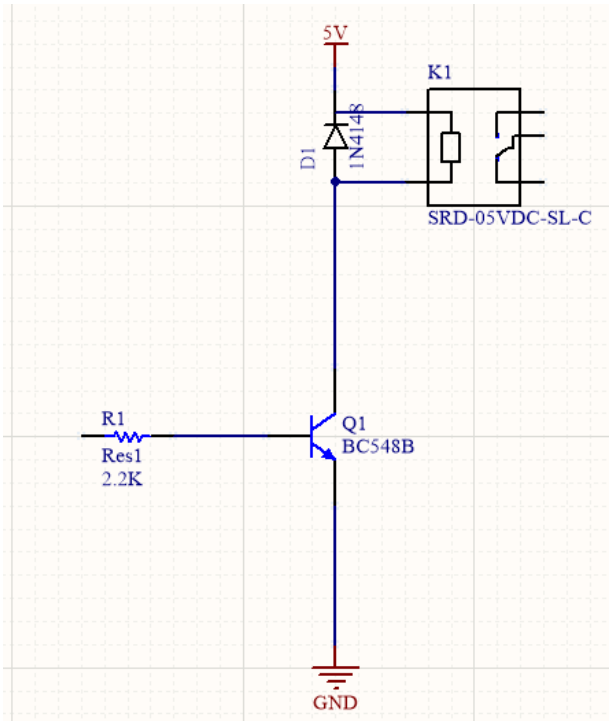


Fonte: Autoria própria (2019)

3.6.2 Circuito para o acionamento do relé

Na prática, a fonte de sinal de 5V que controla o relé pode não suportar a corrente necessária para a sua ativação, fazendo o dispositivo queimar. Foi utilizado o circuito da Figura 23 para superar tal problema. O circuito é constituído de um resistor de 2,2KΩ, um Diodo 1N4148, um Transistor BC548 e um relé.

Figura 23 – Diagrama do circuito para o acionamento do relé.

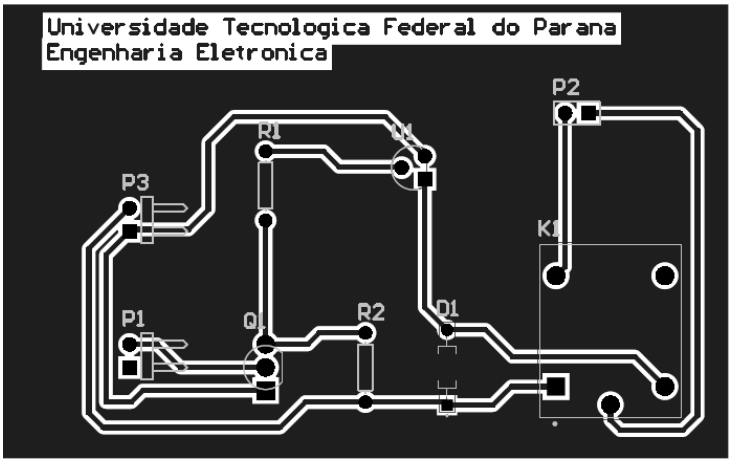


Fonte: Autoria própria (2019)

3.7 Circuito impresso

Para desenhar o *layout* da circuito impresso foi utilizado o software Altium designer. O Altium Designer é um dos mais populares softwares de design PCB de ponta no mercado atualmente, sendo desenvolvido e comercializado pela Altium Limited. A Figura 24 mostra o *layout* do *layout*.

Figura 24 – *Layout* da PCB utilizada para o acionamento do relé.



Fonte: Autoria própria (2019)

3.8 Desenvolvimento do *firmware*

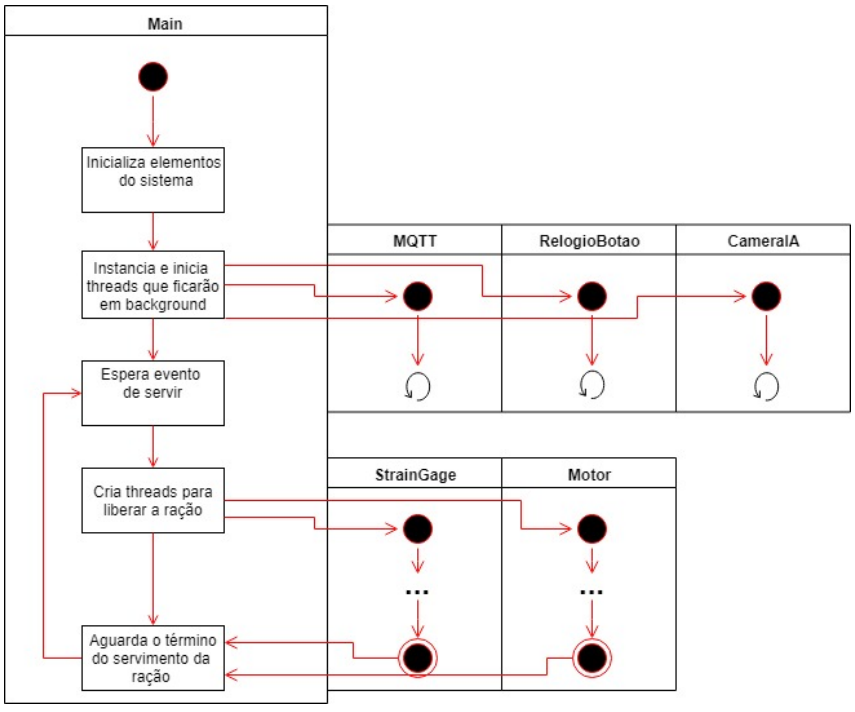
Como foi explicado no Capítulo 2, sistemas embarcados desempenham tarefas específicas. No caso deste projeto, algumas tarefas específicas precisam ser executadas ao mesmo tempo para que o sistema funcione corretamente. Para isso foi criado um ambiente multitarefas, utilizando-se do módulo *threading* do Python. Com ele, é possivel criar várias tarefas e programá-las para serem simultâneas.

Nesta seção são apresentados os principais algoritmos de cada arquivo utilizados para a construção do sistema embarcado do projeto e uma breve explicação de cada tarefa criada utilizando *threads*. Os códigos completos utilizados estão nos apêndices.

3.8.1 Módulo principal (main)

Como o nome já diz, esse é o módulo principal do sistema embarcado. É nele que todos os outros módulos são criados e instanciados. Ele é responsável pela inicialização geral do sistema. Nele também se encontra o *loop* principal do programa, que fica esperando um sinal para ativar as tarefas necessárias somente quando for a hora de servir a porção de ração. Na Figura 25 é possível visualizar o fluxograma desta tarefa.

Figura 25 – Fluxograma simplificado da tarefa principal do sistema.



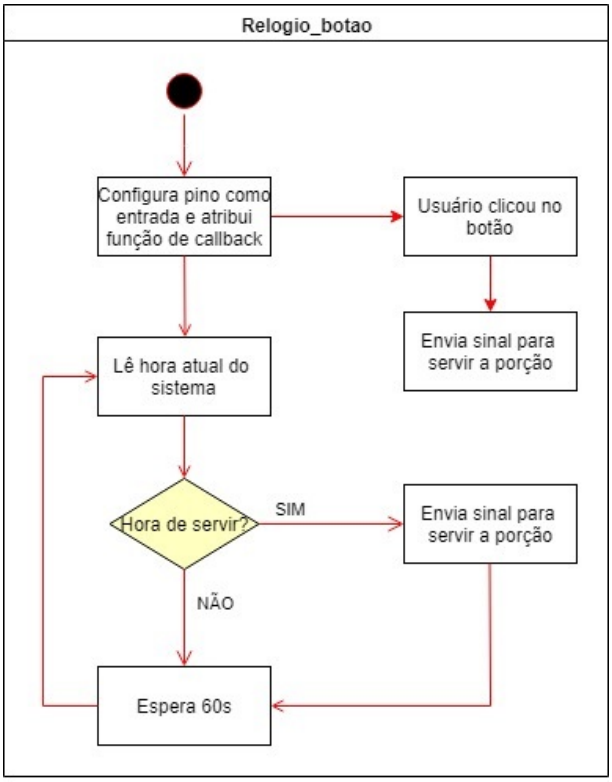
Fonte: Autoria própria (2019)

3.8.2 Verificação da hora de alimentar

Este módulo é responsável por verificar minuto a minuto se está na hora certa de servir uma nova porção de ração e também de verificar se o botão físico de servir foi clicado. Ela é uma das tarefas que ficam constantemente ativas. Caso seja a hora de servir, ele ativa um sinal que será lido pelo módulo principal para proceder com o processo de servimento da ração.

O funcionamento desta tarefa está demonstrado no fluxograma da Figura 26.

Figura 26 – Fluxograma da tarefa responsável pela verificação da hora de alimentar.



Fonte: Autoria própria (2019)

3.8.3 Comunicação com o aplicativo

Assim como a tarefa anterior, esta também fica sempre ativa, pois é nela que será realizada a comunicação entre o sistema embarcado e o aplicativo, através do já mencionado protocolo MQTT. Quando é iniciada, seu trabalho é estabelecer uma comunicação com o *Broker*, se inscrever nos tópicos e aguardar o momento de enviar ou receber mensagens.

A própria biblioteca MQTT do Python oferece uma função que inicia o *loop* da tarefa para aguardar mensagens. Logo após chamá-la, foi criado outro *loop*, o qual permite que a tarefa fique aguardando também sinais que as outras tarefas possam enviar. A Figura 27 apresenta o trecho do código que contém os laços desta tarefa.

O módulo MQTT do Python também disponibiliza uma função *callback* de quando o cliente recebe uma mensagem do *Broker*. Dentro desta função, foram implementados

Figura 27 – Trecho de código que mostra os dois *loops* da tarefa sendo implementados.

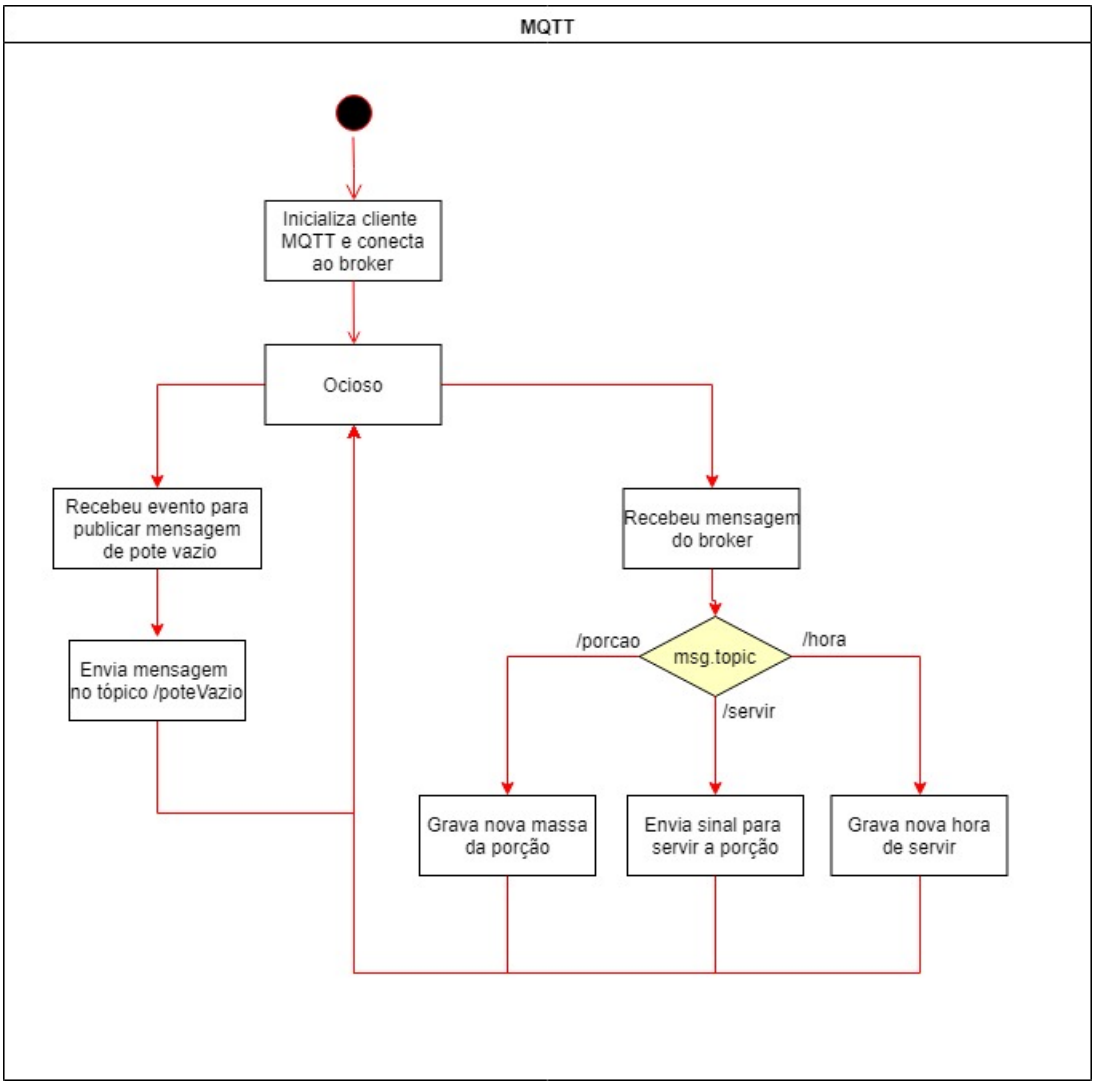
```
65          client.loop_start()
66
67          while True:
68              if globals.eventoNotificarPoteVazio.is_set():
69                  #Prepara e envia a imagem para o aplicativo
70                  print("Notificando para o usuario que o pote esta vazio")
```

Fonte: Autoria própria (2019)

tratamentos para as mensagens de alteração de ração e hora de servir, e de ativação do servimento da ração.

Para simplificar o entendimento da tarefa responsável pela comunicação entre Raspberry e o aplicativo, foi feito um fluxograma, apresentado na Figura 28.

Figura 28 – Fluxograma da tarefa responsável pela comunicação com o aplicativo.



Fonte: Autoria própria (2019)

3.8.4 Câmera e Deep Learning

Como a câmera está intimamente ligada à tarefa de reconhecimento, foi decidido implementar o código dos dois em uma mesma tarefa. Apesar desta tarefa ficar sempre ativa, a câmera e o processo de reconhecimento só são ativados quando uma porção é servida, para consumir menos energia e processamento.

Quando a porção termina de ser servida, a tarefa inicia um *loop* que captura uma imagem, utiliza o modelo para verificar se ainda existe ração no pote e aguarda um período. Esta ação continuará até que seja identificado que a ração do pote acabou. Quando isso ocorrer, é enviado um sinal para a tarefa de comunicação, que por sua vez envia uma notificação para o aplicativo informando que a porção acabou. Este processo está demonstrado no fluxograma da Figura 29.

3.8.5 Acionamento do Motor

Esta é uma das tarefas que somente são ativadas quando for a hora de servir a porção. Ela é responsável pelo acionamento do motor ativando o relé. Este processo está ilustrado na Figura 30. Ao terminar a sua execução, a tarefa será finalizada e só será criada novamente quando for servir uma nova porção, poupando assim energia e processamento.

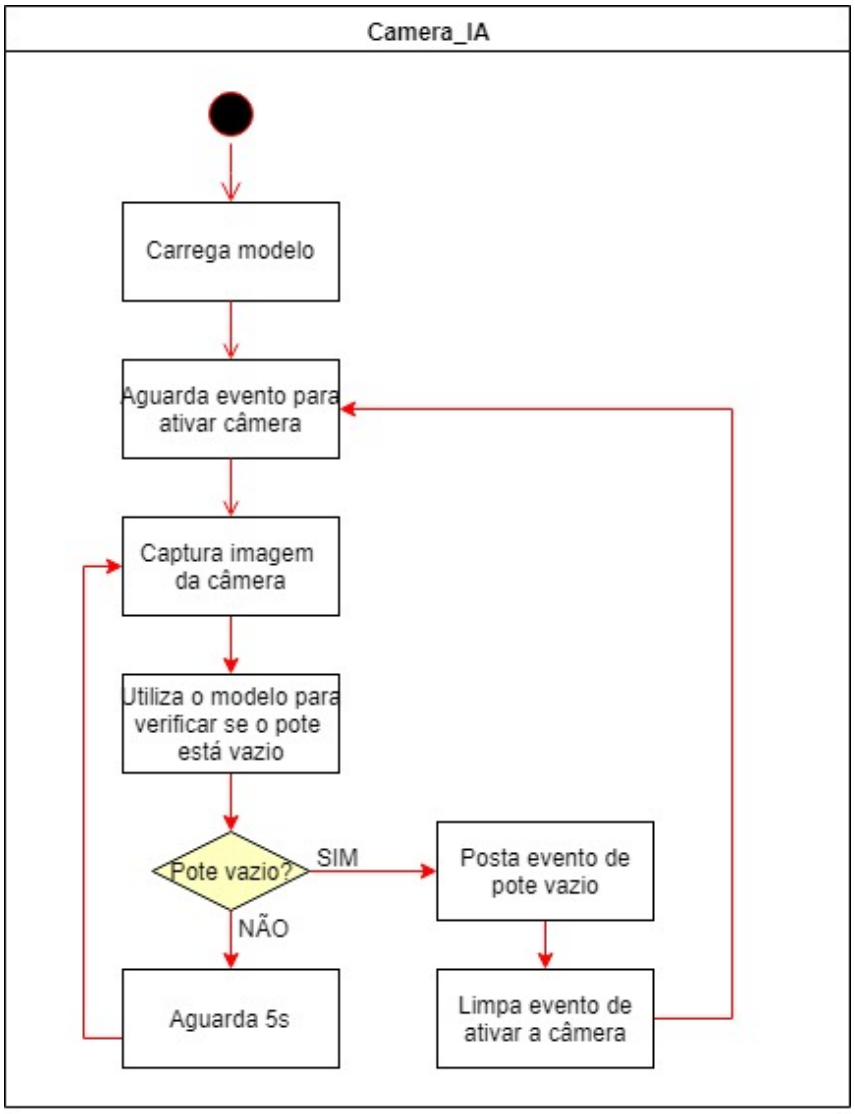
3.8.6 Medição da quantidade de ração

A função desta tarefa é verificar o peso da ração sendo despejada e enviar um sinal para a tarefa responsável pelo motor para desativá-lo quando o peso chegar ao mesmo valor da porção desejada pelo usuário. Ela também apenas será ativada na hora que for servir, e ficando desligada até a próxima execução da tarefa. A Figura 31 apresenta o fluxograma desta tarefa.

3.8.7 Proteção de dados compartilhados entre tarefas

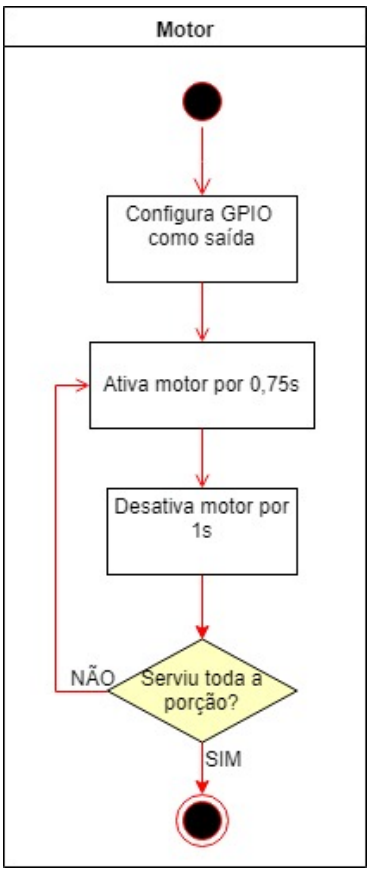
Existem tarefas neste sistema embarcado que compartilham uma mesma variável. Como são tarefas assíncronas, existe a possibilidade de duas ou mais tarefas lerem/escreverem na mesma variável ao mesmo tempo, o que pode causar um grande problema ao sistema. Para garantir que isso não ocorra, o módulo *threading* do Python disponibiliza um elemento chamado *Mutex*, que garante que apenas uma tarefa acesse determinada variável por instante de tempo. No caso deste projeto, foi necessário utilizar *Mutex* em duas variáveis: peso da porção e hora de servir. Essas duas variáveis são utilizadas em muitas das tarefas descritas acima e por isso foi necessário utilizar-se deste artifício para garantir a consistência do sistema.

Figura 29 – Fluxograma da tarefa que identifica quando o pote está vazio.



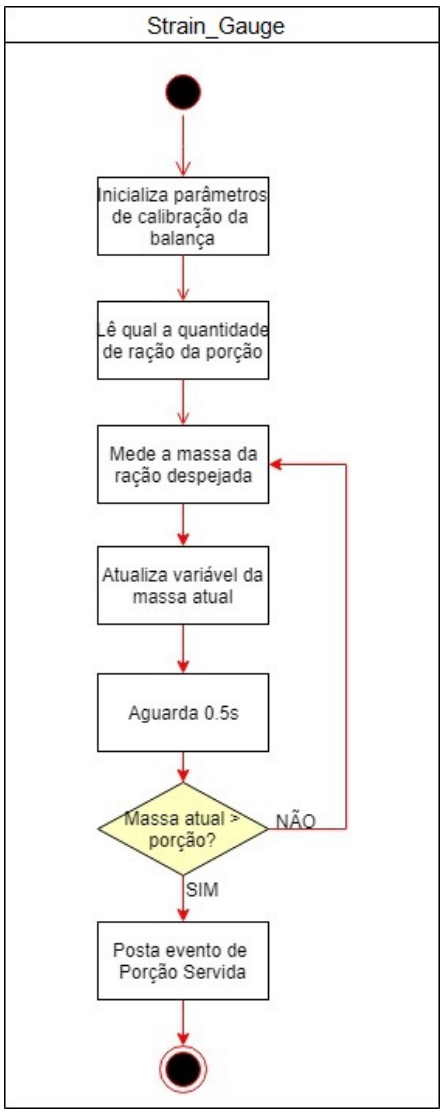
Fonte: Autoria própria (2019)

Figura 30 – Fluxograma da tarefa responsável pelo controle do motor.



Fonte: Autoria própria (2019)

Figura 31 – Fluxograma da tarefa que mede a massa da ração despejada.



Fonte: Autoria própria (2019)

4 ANÁLISE E DISCUSSÃO DOS RESULTADOS

Neste capítulo será explicado o processo de funcionamento do sistema em quatro conjuntos de experimentos. O primeiro conjunto tem como objetivo apurar a capacidade de detecção da falta de comida disponível ao animal. No segundo conjunto de experimentos, verifica-se o funcionamento da comunicação entre o protótipo e o aplicativo Android e no terceiro conjunto de experimentos, o sistema é testado de uma maneira integrada . Por fim, no quarto conjunto de experimentos, o sistema de medição de peso é testada.

4.1 Análise de resultados da rede neural

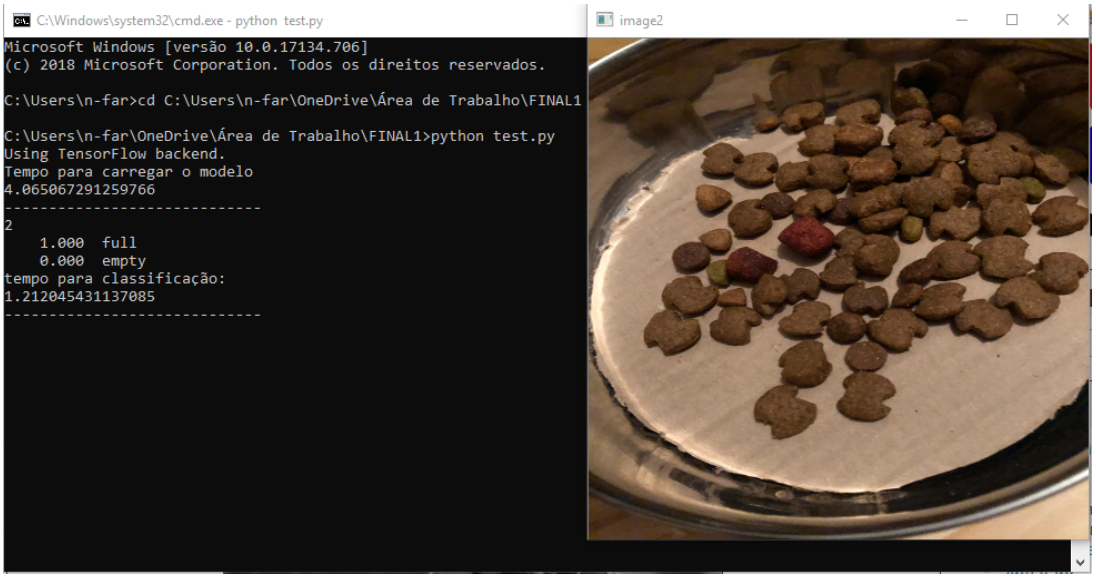
Para verificar a capacidade da rede neural de maneira isolada, ou seja, sem o impacto da capacidade de processamento do Raspberry Pi, foram realizados quatro testes em um computador com maior capacidade computacional que a placa utilizada. Utilizando um *script* em Python, a rede neural foi alimentada com algumas imagens. A classificação da imagem, seja de pote cheio de ração ou não, são exibidos no *prompt* de comando do Windows. Com o teste sendo realizado em diversas situações, foi possível verificar a aplicabilidade do modelo de *deep learning* desenvolvido.

No *script* também foi realizada uma medição do tempo da classificação de uma imagem e do carregamento do modelo, com o objetivo de determinar se é viável a utilização do mesmo no protótipo desenvolvido. Os resultados de acurácia da rede foram apresentados na Seção 3.4. Neste teste de operação, foi verificado o desempenho do modelo para algumas imagens. O resultado, que pode ser visualizado na Figura 32, demonstra a detecção de que a tigela está cheia com uma precisão de 100%.

No segundo teste, a entrada foi uma imagem com uma tigela completamente vazia. Verificando a Figura 33 é possível perceber que o resultado final apresentou com exatidão a detecção da falta de comida no pote.

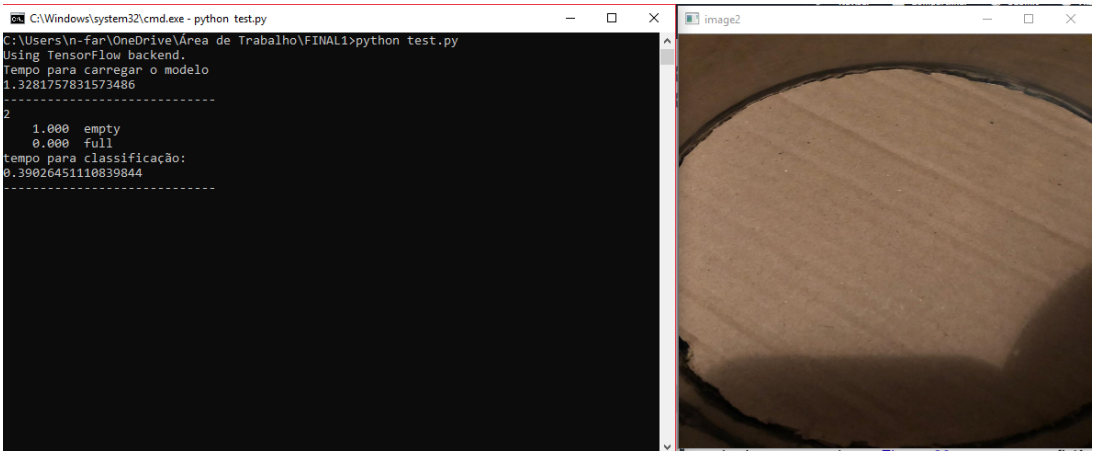
No terceiro teste, a imagem de entrada foi uma tigela completamente cheia. O resultado apresentado na Figura 34 comprova a eficiência do modelo na detecção de pote totalmente cheio de ração.

Figura 32 – Captura de tela durante teste do modelo desenvolvido para o caso de tigela parcialmente cheia.



Fonte: Autoria própria (2019)

Figura 33 – Captura de tela durante teste do modelo desenvolvido para o caso de tigela completamente vazia.



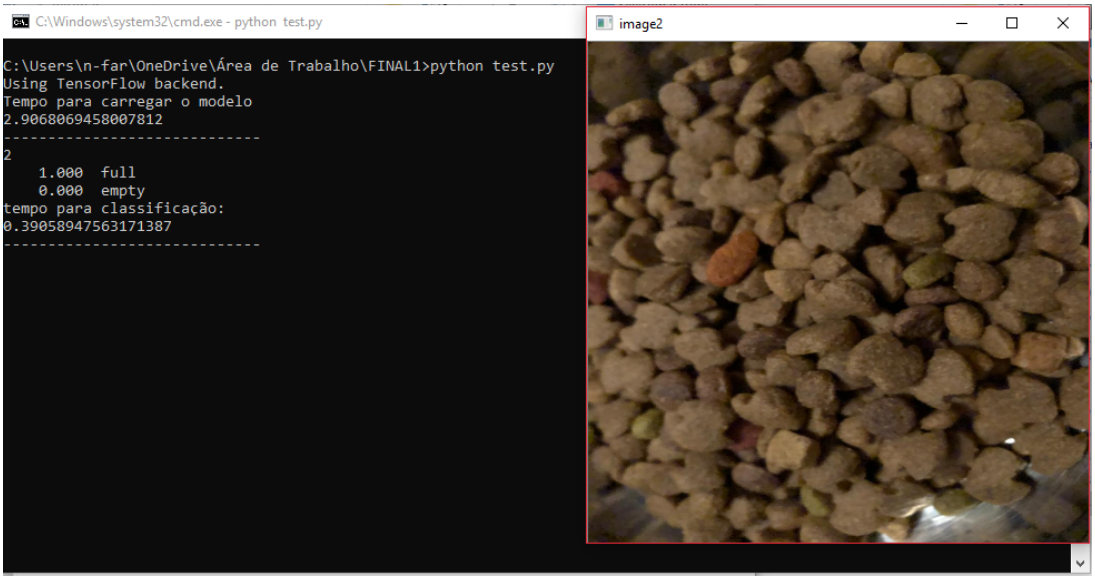
Fonte: Autoria própria (2019)

Em sequência, foi realizado um último teste para verificar se o modelo desenvolvido consegue identificar pouquíssima comida disponível como classe *empty*. Na Figura 35, o resultado é apresentado no *prompt* de comando.

Apesar da imagem do último teste não ter sido classificada como *empty*, o resultado se mostrou eficaz, já que o objetivo final do *deep learning* é classificar pouca comida ao dono do animal e, consequentemente, fornecer alimento suficiente para o bem estar do mesmo.

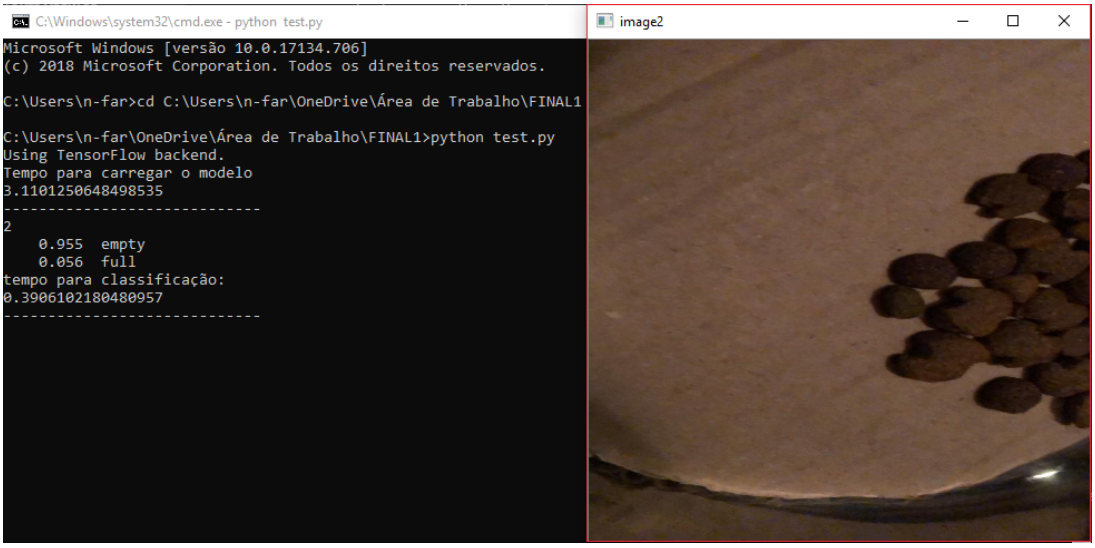
Nos testes realizados na Raspberry Pi, o modelo se comportou como esperado e apresentou valores semelhantes aos testes realizados no computador.

Figura 34 – Captura de tela durante teste do modelo desenvolvido para o caso de tigela completamente cheia.



Fonte: Autoria própria (2019)

Figura 35 – Captura de tela durante teste do modelo desenvolvido para o caso de pouca comida disponível na tigela.



Fonte: Autoria própria (2019)

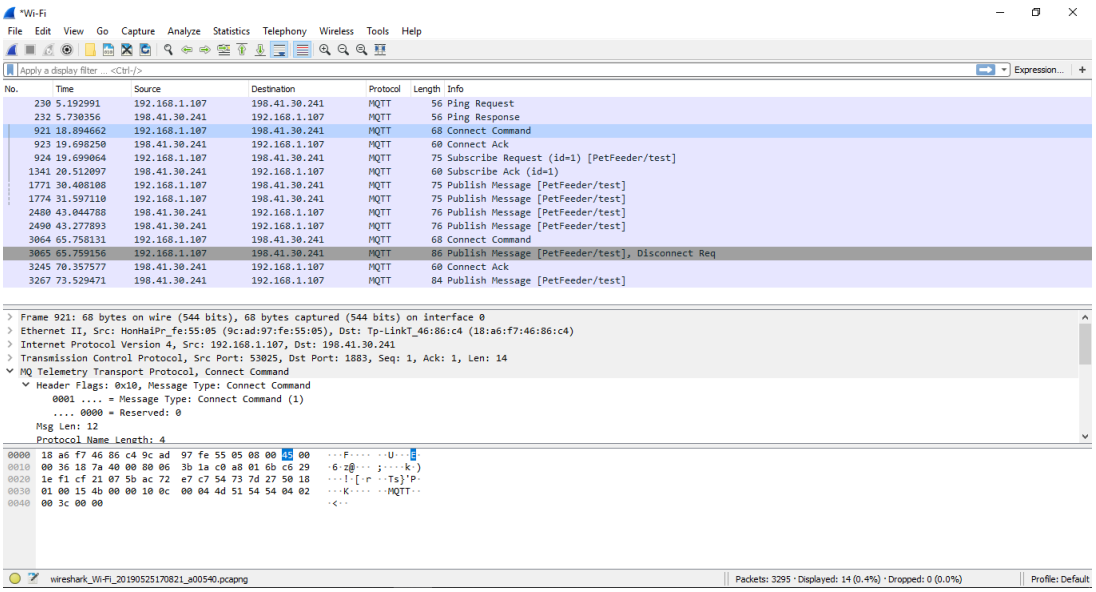
4.2 Análise de resultados do MQTT

A comunicação via protocolo MQTT se mostrou eficaz e de fácil implementação para o objetivo do projeto, sendo necessário apenas que todos os clientes se conectem ao *broker*, que está hospedado na nuvem. Por ser um protocolo de mensagens leve e frequentemente associado à Internet das Coisas, o MQTT não apenas atende às necessidades de comunicação deste projeto, como permite que o protótipo esteja melhor preparado para uma futura integração a

ambientes preparados para IoT.

Através do software Wireshark[1] foi possível analisar os pacotes entre o *broker* do MQTT e um cliente MQTT, visando a identificação de eventuais erros na comunicação (vide Figura 36).

Figura 36 – Análise dos pacotes transmitidos através do software Wireshark.



Fonte: Autoria própria (2019)

Foram analisados os seguintes comandos da conexão:

- *Header Flags*: contém informações sobre o tipo de pacote de controle MQTT.
- *Will QoS*: bit 4 e 3 dos sinalizadores de conexão. Mostra o nível de QoS a ser usado ao publicar a mensagem.
- *Keep alive*: é usado para saber se um cliente MQTT está na rede onde o cliente envia mensagens de solicitação PING regulares para o *Broker*. O *Broker* responde com uma resposta PING.
- *Payload*: contém os campos ID do Cliente, Tópico, Mensagem, Nome do Usuário e Senha cuja presença é determinada pelas *Flags*.

Ao receber uma mensagem CONNECT, o *Broker* responde com uma mensagem CONNACK, que significa mensagem de conexão reconhecida. É possível visualizar essa mensagem de conexão na Figura 37.

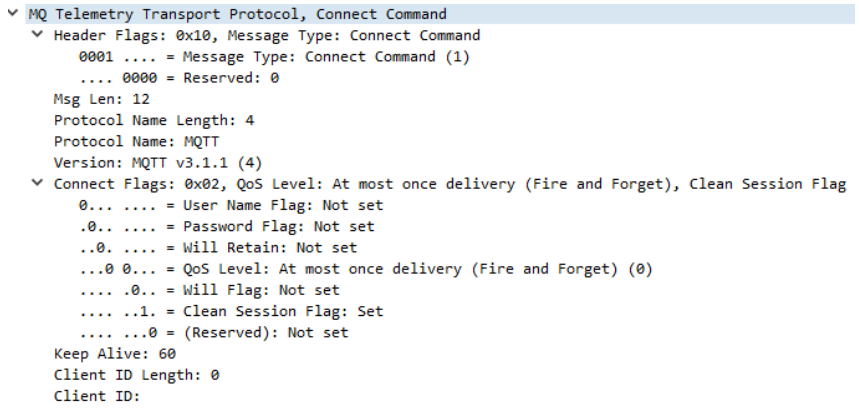
Um cliente envia uma mensagem SUBSCRIBE para um *broker* MQTT para receber mensagens relevantes. A mensagem enviada é visualizada na Figura 38.

O *broker* MQTT confirma a assinatura enviando uma confirmação de volta ao cliente usando uma mensagem SUBACK, como demonstrado na Figura 39.

Quando um cliente MQTT é conectado ao *broker*, ele pode publicar mensagens, como ilustrado na Figura 40.

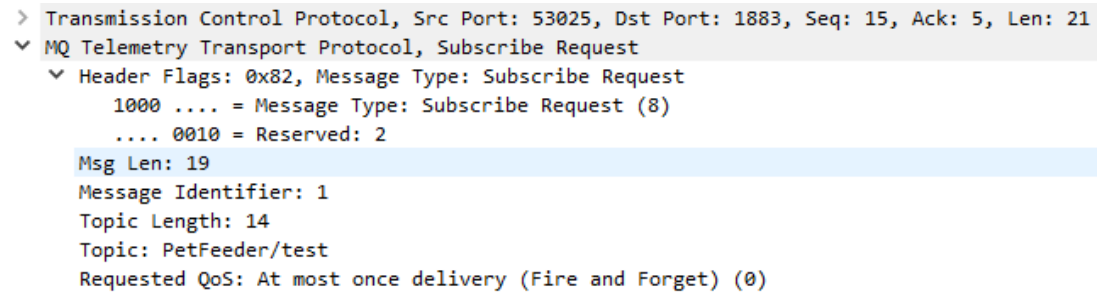
[1]Wireshark é um analisador de tráfego de rede, disponível em https://www.wireshark.org

Figura 37 – Conexão do cliente com o *Broker*.



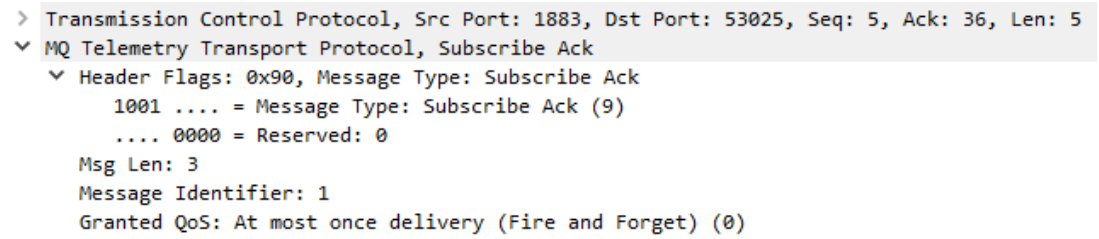
Fonte: Autoria própria (2019)

Figura 38 – Recebimento da mensagem de pedido de assinatura pelo *broker* MQTT.



Fonte: Autoria própria (2019)

Figura 39 – Recebimento da mensagem de confirmação pelo cliente MQTT.



Fonte: Autoria própria (2019)

É possível verificar o desempenho de diversos servidores MQTT através do estudo realizado pelo órgão mqtt.org e pela empresa ScalAgent. Os resultados dos testes mostram o rendimento das mensagens entregues aos assinantes, o uso de CPU pelo servidor MQTT e a latência de transmissão da mensagem, ou seja, a latência causada pela rede e pelo servidor MQTT (MQTT.ORG AND SCALAGENT, 2015).

Neste trabalho, utilizou-se o servidor público do Eclipse[2]. A justificativa para essa

[2]O projeto Eclipse Paho fornece implementações de cliente de software livre do MQTT, disponível em https://iot.eclipse.org

Figura 40 – Publicação de mensagens.

```
> Transmission Control Protocol, Src Port: 1883, Dst Port: 52339, Seq: 24, Ack: 45, Len: 21
v MQ Telemetry Transport Protocol, Publish Message
  v Header Flags: 0x30, Message Type: Publish Message, QoS Level: At most once delivery (Fire and Forget)
      0011 .... = Message Type: Publish Message (3)
      .... 0... = DUP Flag: Not set
      .... .00. = QoS Level: At most once delivery (Fire and Forget) (0)
      .... ...0 = Retain: Not set
    Msg Len: 19
    Topic Length: 14
    Topic: PetFeeder/test
    Message: 313233
```

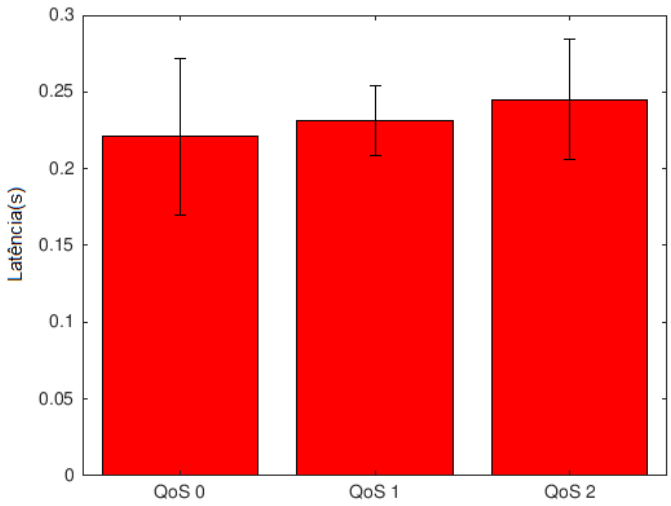
Fonte: Autoria própria (2019)

escolha é a baixa quantidade de Subscribers simultâneos associados ao Broker, o que deve implicar em uma baixa latência, de acordo com estudos publicados na literatura (MQTT.ORG AND SCALAGENT, 2015).

Para verificar a latência da comunicação MQTT em diferentes tipos de Qualidade de serviço (QoS), foi medido o tempo de uma requisição a ser enviada de um cliente até chegar ao destino, que neste ambiente de teste é o próprio cliente que a enviou. A fim de garantir a validade dessas medições, para cada nível de QoS foram medidos 20 vezes o tempo de requisição.

O teste consiste em enviar uma mensagem ao *broker*, momento em que se inicia a contagem de tempo, e retorná-la ao cliente. Todas as medições foram realizadas no mesmo ambiente com o mesmo tamanho de mensagem. É possível conferir o resultado na Figura 41.

Figura 41 – Média e intervalo de confiança das mensagens transmitidas.



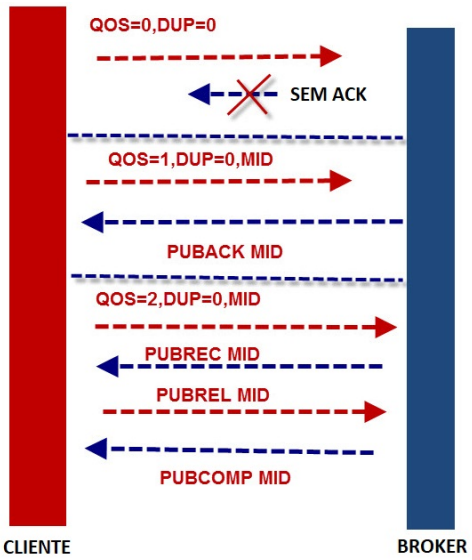
Fonte: Autoria própria (2019)

Através do fluxo de mensagens entre o cliente e o *broker* para mensagens com QoS 0, 1 e 2, apresentado na Figura 42, é possível concluir que quanto maior for o nível de QoS, maior é a garantia de que a mensagem será entregue ao cliente final, aumentando a robustez

do sistema de comunicação.

A partir dos resultados do teste foi constatado que a diferença de latência entre os níveis é mínima, foi decidido utilizar o QoS 2 nas publicações realizadas pelo protótipo e pelo aplicativo, por apresentar mais confiança que a mensagem será entregue ao *broker*.

Figura 42 – Fluxo das mensagem em diferentes Qualidades de serviços(QoS)



Fonte: Adaptado de Paho (2019)

4.3 Análise geral do sistema

Para o teste do sistema completo, primeiramente testou-se cada funcionalidade relevante do projeto isolando-as umas das outras. Foram simuladas diversas situações para verificação de falhas e observado se o comportamento e resposta do protótipo está coerente com os objetivos do projeto.

O protótipo montado está ilustrado na Figura 43.

4.3.1 *Célula de carga*

Com o objetivo de verificar se os dados de massa obtidos com o sistema de medição são coerentes, foram realizadas uma série de comparações entre repetidas medições de várias massas diferentes com uma balança calibrada e os valores obtidos com o sistema. A Figura 44 ilustra os dados relativos a essas medições.

Figura 43 – Imagem do protótipo desenvolvido

Fonte: Autoria própria (2019)

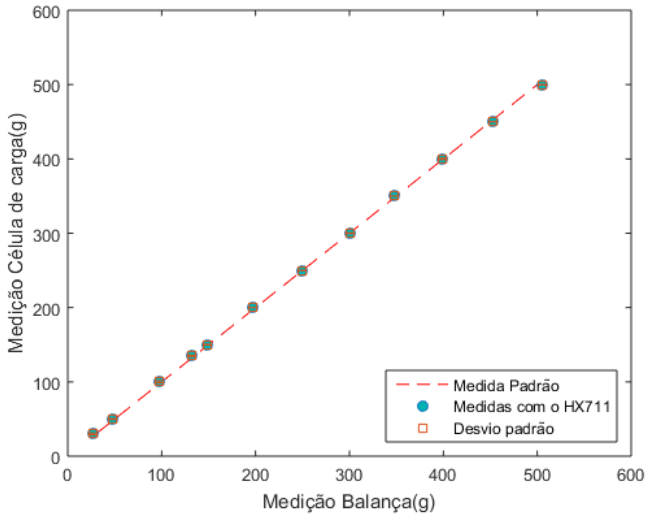
Como esperado, os valores são praticamente iguais, com pequenos desvios quando a massa do objeto se aproximou de zero. Este resultado é fundamental para assegurar o funcionamento correto do projeto, em especial a parte de controle de quantidade de ração.

4.3.2 Servimento da porção

O controle da quantidade de ração despejada é obtido de forma simples: quando a quantidade de ração no pote for superior ao valor que o usuário deseja, o motor é desativado e a pá para de girar. Este método apresentou uma resposta muito rápida no controle do motor, além de ser de simples implementação, por isso foi o escolhido.

Como a pá é dividida em 4 setores, toda a ração que estava em um setor é servida, e isso acaba fazendo com que a quantidade de ração da porção servida seja ligeiramente maior do que a porção desejada pelo usuário. No entanto, durante os testes nenhuma diferença entre

Figura 44 – Gráfico comparativo entre valores medidos com o *Strain Gauge* e balança.

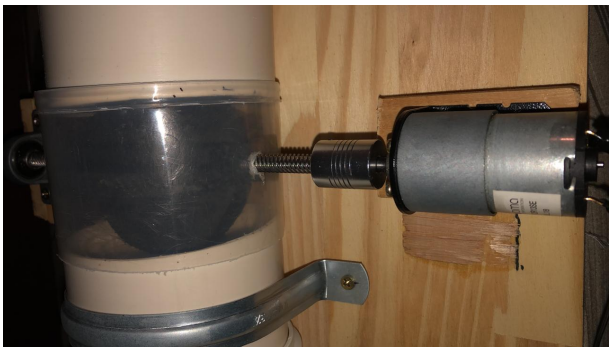


Fonte: Autoria própria (2019)

os valores da porção servida e desejada foram maiores do que 20g, que é um valor aceitável, se tratando de um protótipo.

Na Figura 45, é possível visualizar a pá dividida em 4 setores que foi utilizado neste projeto.

Figura 45 – Imagem da pá utilizada no projeto.



Fonte: Autoria própria (2019)

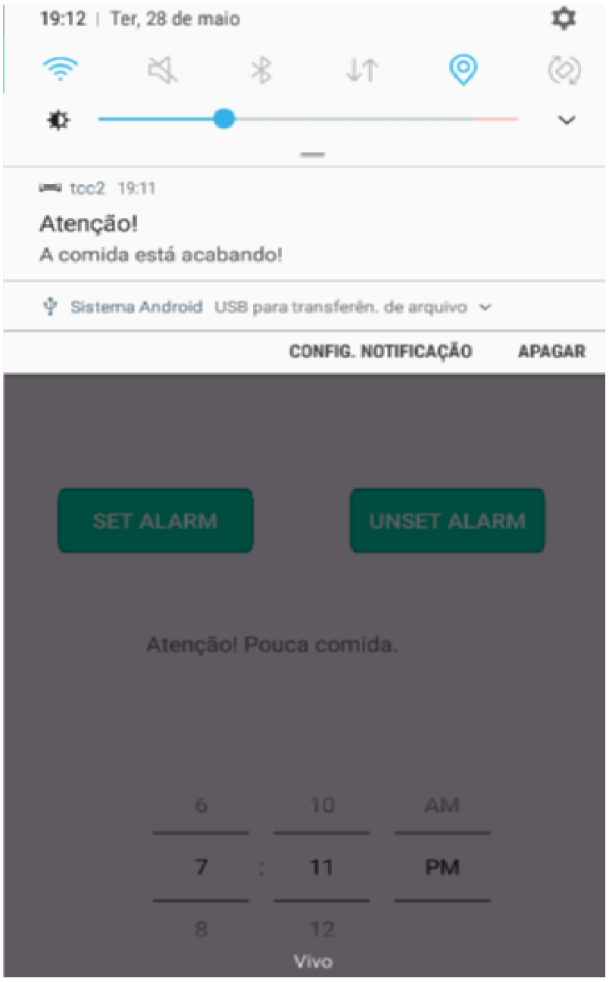
4.3.3 Resultados Gerais

Nos testes, o protótipo desenvolvido respondeu como esperado. A comunicação entre o *smartphone* e a Raspberry Pi ocorreram como previsto, a notificação foi enviada quando o pote possuía pouca comida disponível – ilustrado na Figura 46 – e por fim, a comida foi despejada corretamente na hora certa.

O protocolo MQTT apresentou um desempenho apropriado para a aplicação em questão, principalmente pelo fato de não haver nenhuma execução de tarefas críticas. O uso

deste protocolo permite que o protótipo seja usado em diferentes condições de largura de banda e, também, que seja facilmente integrado a cenários habilitados para a Internet das Coisas.

Figura 46 – Captura de tela do aplicativo em situação de pouca comida.



Fonte: Autoria própria (2019)

Pelo fato de não existir um controle complexo no despejo de ração, houve certa discrepância entre o valor digitado pelo usuário e o valor efetivamente despejado no pote, porém não é um problema crítico por ser um valor insignificante, sendo menor que 20g de diferença do número enviado.

5 PLANO DE NEGÓCIOS

5.1 Sumário Executivo

O sistema desenvolvido procura suprir as necessidades do indivíduo que passa muito tempo fora de casa e deixa o seu animal de estimação sozinho, através de um sistema de alimentação em que é possível ter controle e receber informações de qualquer lugar que possua internet.

O público-alvo será, principalmente, as classes A, B e C, que demanda um produto de alta qualidade e confiabilidade em mercadorias relacionadas ao mercado de animais de estimação, que movimentou cerca de US$ 20,3 bilhões em produtos e serviços para animais em 2017.

A empresa contará com pessoal qualificado na parte técnica. Os funcionários receberão constantemente incentivos para o aprimoramento na formação para que se tenha compromisso com a alta qualidade e confiabilidade do produto.

5.2 Proposta de negócio

O projeto é composto de um sistema físico (hardware e protótipo) e o sistema de serviços (software). A parte física possui sensor *Strain Gauge*, uma câmera, um motor DC e um sistema para o controle de um relé. Todos os sensores e atuadores são agregados a uma Raspberry pi 3. O software é constituído de um aplicativo Android elaborado para o controle do dispositivo, o software embarcado na Raspberry Pi e um serviço de gerenciamento de dados em nuvem para a conexão do MQTT.

As vendas do produto estão previstas para ocorrer em um site próprio da empresa, além de lojas especializadas, podendo ser pagas à vista ou parceladas. O pós-venda vai ser realizado através de redes sociais e por um *chat* online no site da empresa.

5.3 Análise de mercado

O mercado pet no Brasil é um dos maiores do mundo e tem se consolidado como um setor da economia com grande potencial de crescimento para os próximos anos. Em 2018, o setor movimentou mais de R$ 20 bilhões, 9,8% a mais que em 2017. Com isso, o Brasil se tornou o segundo maior mercado global de produtos pet, com 6,4% de participação, ultrapassando o Reino Unido (6,1%) pela primeira vez. Em primeiro lugar estão os Estados Unidos, com 50%.

Uma pesquisa feita pelo SPC Brasil e a CNDL, realizada em todas as capitais com internautas que "possuem" ou são responsáveis financeiros por um animal de estimação, revela que 61% dos entrevistados consideram seus pets como um membro da família. Para cuidar do bem-estar desses companheiros, as famílias brasileiras , gastam em média, R$ 189 todos

os meses – entre consumidores das classes A e B, o valor aumenta para R$ 224. Para quem recebe até dois salários mínimos, esse valor pode representar até 10% da renda familiar.(CNDL; SPC, 2017)

5.3.1 Definição da empresa

A visão é ser referência no desenvolvimento de produtos voltados aos animais.

A missão é fornecer tecnologias inovadoras e de alta qualidade para o mercado.

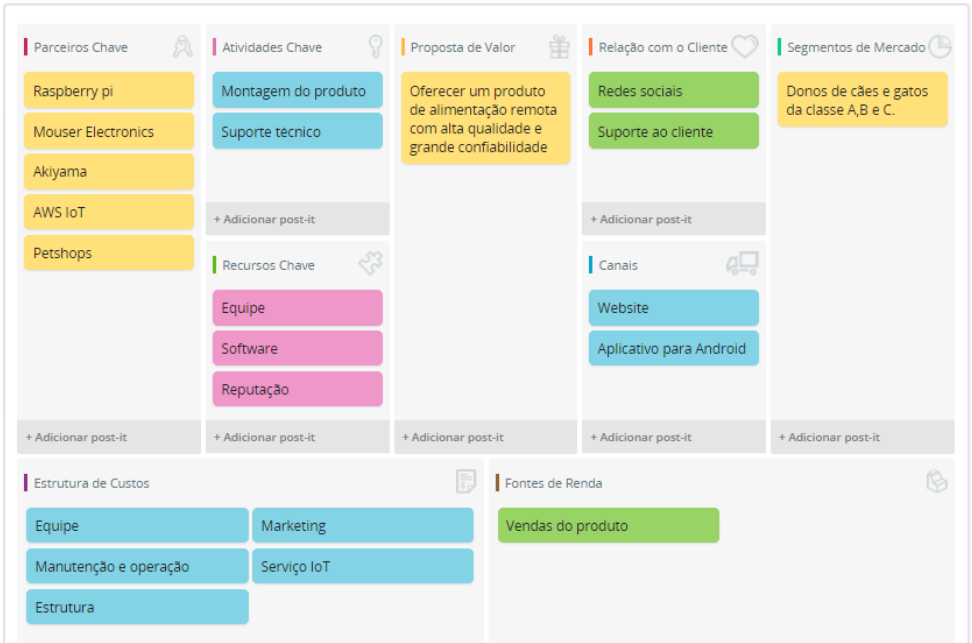
Os valores da empresa são:

- Compromisso com o cliente;
- Busca pela maior qualidade dos produtos e serviços;
- Estar sempre buscando desenvolvimento tecnológico;
- Ética para com o consumidor.

5.4 Modelo Canvas

O modelo Canvas é uma ferramenta estratégica para desenvolver novos modelos de negócios ou documentar e melhorar os já existentes. Na Figura 47 é possível visualizar o modelo canvas do trabalho.

Figura 47 – Canvas desenvolvido para a empresa que vai fabricar o produto.

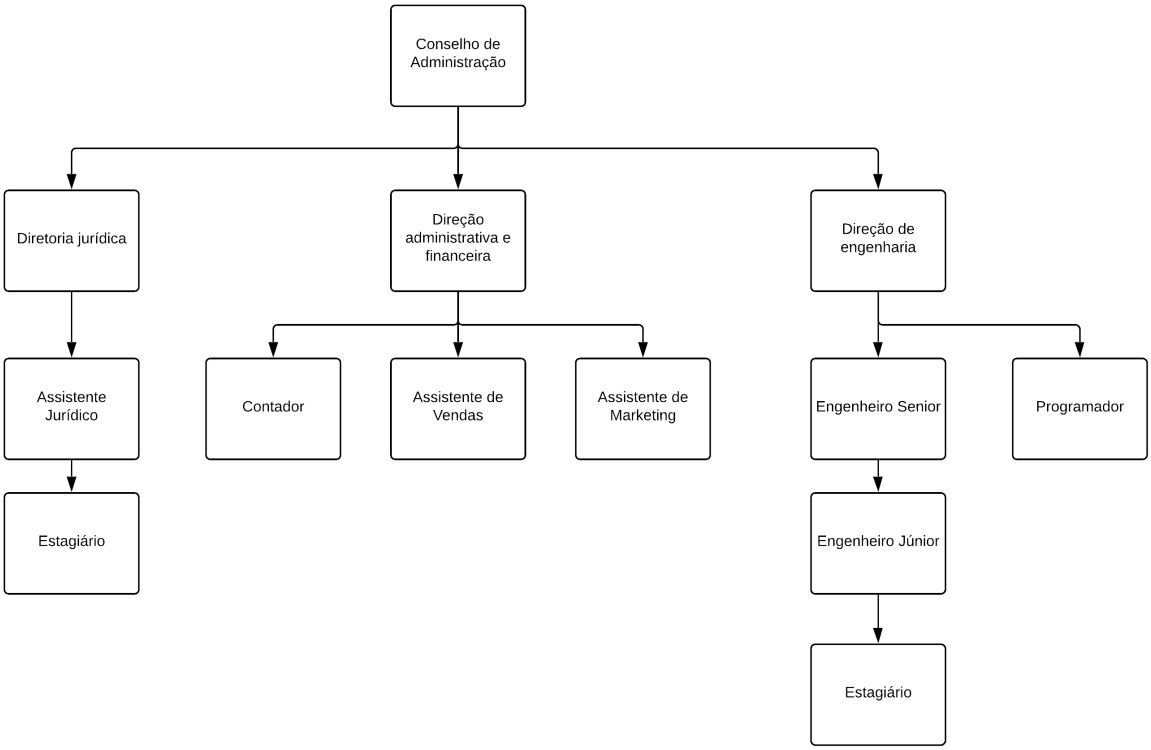


Fonte: Autoria própria (2019)

5.5 Organograma Funcional

Inicialmente a empresa será composta pelos dois sócios fundadores, que irão assumir cargos no conselho de administração. Cada integrante vai ser responsável por uma área específica. Conforme a necessidade, algumas funções serão ocupadas por funcionários contratados. A Figura 48 ilustra o organograma da empresa.

Figura 48 – Estrutura organizacional.



Fonte: Autoria própria (2019)

5.6 Análise SWOT

Uma análise SWOT é uma ferramenta muito simples, mas poderosa, para ajudar a desenvolver uma estratégia de negócios. SWOT significa pontos fortes, pontos fracos, oportunidades e ameaças.

Pontos fortes e fracos são internos da empresa - Sobre os quais é possível exercer algum controle e efetuar mudanças. Os exemplos incluem quem está na equipe, as patentes e propriedade intelectual e, também, sua localização.

Oportunidades e ameaças são externas - Que acontecem fora da empresa, no mercado maior. É possível aproveitar as oportunidades, mas não alterá-las. Exemplos incluem concorrentes, preços de matérias-primas e tendências de compras dos clientes.

Na Figura 49, é possível visualizar a matriz SWOT.

Figura 49 – Análise SWOT.

Fonte: Autoria própria (2019)

5.7 Considerações finais

Com a análise de mercado é possível verificar que o produto pode ser bem aceito pelos consumidores e que este é um mercado em crescimento. Conclui-se que o sistema de alimentação desenvolvido é uma opção apropriada para ser agregada ao negócio de animais de estimação.

6 CONCLUSÃO

O objetivo deste trabalho foi desenvolver um protótipo de um sistema de alimentação para animais, com uma confiabilidade alta e que pudesse ser mais conveniente e com menor custo que outras opções disponíveis no mercado. Os requisitos citados na introdução deste trabalho foram alcançados com êxito, mesmo após algumas dificuldades não previstas no início do projeto. Essas dificuldades foram contornadas e as principais conclusões com relação a elas serão apresentadas na Seção 6.1.

A escolha do protocolo MQTT, além de prover uma comunicação sem elevados atrasos, torna o protótipo final mais adaptado a uma futura integração à Internet das Coisas. Mesmo se aplicado em locais com limitações de acesso à Internet, o uso do protocolo de comunicação MQTT permitirá o funcionamento do alimentador de pets.

Para a vizualização dos dados e controle do dispositivo, foi desenvolvido um aplicativo Android. Os requisitos, que incluíam, principalmente, facilidade de uso (interface *user-friendly*) e disponibilidade de informações sobre o sistema, foram atendidos.

Para a identificação da falta de comida no pote foi necessário realizar alguns ajustes na arquitetura do modelo utilizado devido à escolha da placa de desenvolvimento. A Raspberry Pi apresenta certa limitação no poder de processamento, mas essa tarefa poderia ser facilmente delegada a um serviço de nuvem se o protótipo vier a ser integrado a cenários de Internet das Coisas.

O uso de um sistema com câmera e processamento de imagens por meio de redes neurais, apesar de parecer dispensável em alguns cenários, é relevante para casos em que deve haver um rígido controle na dieta do animal. Além disso, o estado atual deste sistema permite que novas funcionalidades sejam desenvolvidas e beneficiem-se do suporte ao tratamento de imagens já existente.

6.1 Dificuldades encontradas

Algumas dificuldades foram encontradas no desenvolvimento do projeto. Pelo fato da escolha da placa de desenvolvimento Raspberry Pi, que possui poder de processamento limitado, o processo de criação do modelo de *deep learning* teve que ser revisto diversas vezes para que o uso de memória RAM não prejudicasse a operação de outras *threads* do dispositivo.

Foram encontradas dificuldades também em decidir como funcionaria o servimento da porção e como montar a parte física responsável por essa tarefa, já que a pá tem que rodar apenas em um certo ângulo por servimento, e tem que permanecer na posição até que seja acionada novamente, no mesmo sentido, eventualmente completando uma volta. A primeira sugestão foi utilizar um motor de passo, porém com a evolução da idéia foi constatado que a parte física teria que ser muito mais complexa, pelo fato do motor de passo não dar uma volta

completa. No fim, foi decidido utilizar o motor DC, que não garante uma precisão tão boa em relação ao ângulo, porém resolve o problema descrito de maneira mais apropriada, sem comprometer a aplicação.

Em relação ao *strain gauge*, durante os testes foi percebido que cada vez que o sistema era utilizado depois de um longo período, era necessário recalibrá-lo, pois os valores medidos ficavam ligeiramente mais altos que os valores reais.

6.2 Trabalhos futuros

Como trabalhos futuros, tem-se:

- No Design do aplicativo, apesar de ter atendido os requisitos do projeto, seria necessário ter uma melhor apresentação ao usuário, para que o produto tenha um melhor engajamento no mercado;
- Criar um sistema de cadastro e login para o aplicativo;
- Buscando atingir o maior número de consumidores possível, seria de extrema importância desenvolver o aplicativo para *smartphones e tablets* que utilizam o sistema operacional iOS, da Apple;
- Uma opção para superar o problema de hardware seria utilizar algum servidor externo e manter todo processamento da rede neural em nuvem como, por exemplo, o Amazon AWS;
- Utilizar uma câmera móvel para identificar outras informações sobre o animal (satisfação, presença no cômodo, busca por comida);
- Criar uma base de dados para cada usuário e gerar um relatório para prever algum tipo de comportamento que possa denotar problemas de saúde.

Referências

ABINPET. No brasil, gastos com animais de estimaÇÃo variam de 24% a menos de 1% da renda familiar. 2016. Disponível em: <http://abinpet.org.br/no-brasil-gastos-com-animais-de-estimacao-variam-de-24-a-menos-de-1-da-renda-familiar/>. Acesso em: 2 de março de 2019. Citado na página 13.

AGGARWAL, C. C. **Neural Networks and Deep Learning: A Textbook**. [S.l.], 2018. 497 p. Citado na página 19.

AVIA. **24-Bit Analog-to-Digital Converter (ADC) for Weigh Scales**. 2019. 10 p. Disponível em: <https://www.mouser.com/ds/2/813/hx711_english-1022875.pdf>. Acesso em: 06 jun. 2019. Citado na página 25.

CHOLLET, F. et al. **Keras**. [S.l.]: GitHub, 2015. <https://github.com/fchollet/keras>. Citado na página 21.

CNDL; SPC. Mercado de consumo pet. p. 22, 2017. Citado na página 57.

DENG, L.; YU, D. Deep learning:methods and applications. 2014. Citado na página 26.

DEVELOPERS, G. **Android Developers**. [S.l.], 2019. Disponível em: <https://developer.android.com>. Acesso em: 2 de fevereiro de 2019. Citado na página 22.

ELPROCUS. **Basics Of Embedded System and Applications**. 2017. Disponível em: <https://www.elprocus.com/basics-of-embedded-system-and-applications/>. Acesso em: 19 de dezembro de 2018. Citado na página 16.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>. Citado na página 19.

HAMSTRA, S. Smart consumption for smart products: Understanding mqtt. 2017. Disponível em: <https://spindance.com/smart-consumption-smart-products-understanding-mqtt/>. Acesso em: 17 de dezembro de 2018. Citado na página 17.

IBGE. **Pesquisa Nacional de Saúde 2013**. [S.l.], 2015. 105 p. Citado na página 13.

LAMPKIN, V. et al. **Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry**. [S.l.], 2012. 250 p. Citado na página 17.

LECUN, Y.; BENGIO, Y.; HINTON, G. **Deep learning**. [S.l.]: Nature, 2015. 10 p. Citado na página 19.

LECUN, Y. et al. **Gradient-Based Learning Applied to Document Recognition**. [S.l.]: IEEE, 1998. 46 p. Citado na página 20.

MANANDHAR, S. **MQTT BASED COMMUNICATION IN IOT**. Novembro de 2017. Dissertação (Mestrado) — Tampere University of Technology, 2017. Citado na página 17.

MQTT.ORG AND SCALAGENT. **Benchmark of MQTT servers**. [S.l.], 2015. 15 p. Acesso em: 25 de maio de 2019. Citado 2 vezes nas páginas 49 e 50.

NOERGAARD, T. **Embedded System Arquiteture: A comprehensive guide for engineers and programmers.** [S.l.], 2005. 640 p. Citado na página 16.

PAHO. 2019. Disponível em: <http://www.steves-internet-guide.com/publishing-messages-mqtt-client/>. Acesso em: 12 jun. 2019. Citado na página 51.

PAZINI, R.; PEREIRA, R. Dispenser automático de alimentos para animais domésticos. 2016. Citado na página 23.

RASPBERRY PI. **Raspberry PI 3 Model B+**. 2019. Disponível em: <https://www.raspberrypi.org/>. Acesso em: 19 de dezembro de 2018. Citado 2 vezes nas páginas 24 e 26.

RELU. 2018. Disponível em: <https://medium.com/@kanchansarkar/relu-not-a-differentiable-function-why-used-in-gradient-based-optimization-7fef3a4cecec>. Acesso em: 06 jun. 2019. Citado na página 20.

RUSSEL, S. J.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. [S.l.]: Prentice Hall, 2009. 932 p. Citado na página 18.

SPARKFUN. **Load cell**. 2019. Disponível em: <https://www.sparkfun.com/products/13329>. Acesso em: 06 jun. 2019. Citado na página 25.

TENSORFLOW. **Tensorflow**. [S.l.], 2019. Disponível em: <https://www.tensorflow.org>. Acesso em: 7 de fevereiro de 2019. Citado na página 21.

THE HIVEMQ TEAM. **MQTT Essentials**. 2015. Citado na página 18.

YUAN, M. Conhecendo o mqtt. 2017. Disponível em: <https://www.ibm.com/developerworks/br/library/iot-mqtt-why-good-for-iot/index.html>. Acesso em: 17 de dezembro de 2018. Citado 2 vezes nas páginas 17 e 18.

Apêndices

APÊNDICE A – Código fonte

O código fonte desenvolvido para este projeto está disponível no link https://github.com/vinisklin/PetFeeder.