UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Departamento de Ciências de Computação

Reconhecimento Facial aplicado a trava de segurança

Gabriel Alves Kuabara - 11275043
Gabriel Freitas Ximenes de Vasconcelos - 11819084
Gabriel Victor Cardoso Fernandes - 11878296

Relatório da disciplina **SCC0910 - Tópicos Avançados em Ciências de Computação I** do Departamento de Ciências de Computação do Instituto de Ciências Matemáticas e de Computação – ICMC-USP.

USP – São Carlos
23/06/2023

Introdução

Nosso trabalho visa desenvolver um sistema de fechadura automática por reconhecimento facial a partir dos conhecimentos obtidos na disciplina SCC0910 - Tópicos Avançados em Ciências de Computação I.

O projeto consiste no seguinte fluxo: haverá uma câmera na porta de uma casa como se fosse um olho mágico que irá gravar uma foto da pessoa com uma determinada frequência. Então, ao captar essa imagem, enviará para o servidor conectado na mesma rede que irá realizar o reconhecimento da pessoa dizendo quem seria o sujeito ou se é desconhecido. Após o reconhecimento, irá enviar uma flag para a fechadura destravando-a ou não.

Para construção do reconhecimento facial, utilizaremos da rede VGGFace modelada a partir de uma VGG16 por meio de Transfer Learning.

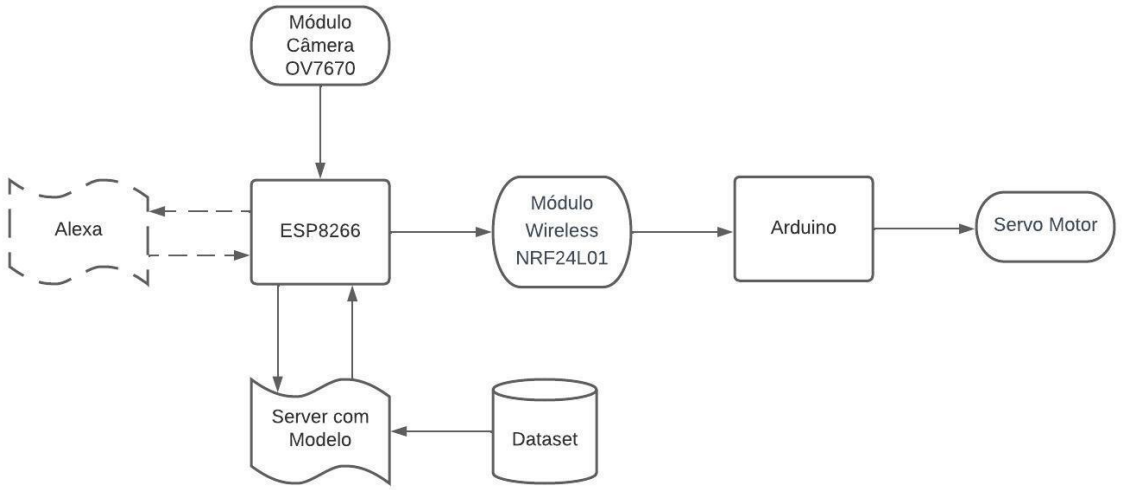
Para o sistema eletrônico, utilizaremos um Arduino Uno, um ESP32 (microprocessador com wi-fi integrado), uma câmera e uma série de componentes para compô-lo.

As próximas seções buscam elucidar o que foi feito até então e quais serão os próximos passos.

O GitHub para o trabalho é https://github.com/kibonusp/fechadura-automatica.

Funcionamento do Sistema

O sistema foi modelado considerando uma comunicação em tempo real entre o ESP32 e o servidor com o modelo. A comunicação deveria ser rápida para que assim que a câmera, acoplada ao ESP32, obtivesse a imagem de uma pessoa, o reconhecimento fosse feito de maneira rápida e então liberada sua entrada. Dessa forma, um fator importante para a escolha do modelo foi o tempo de latência, por isso escolhemos um modelo rápido e validado para reconhecimento facial, como o VGG16.



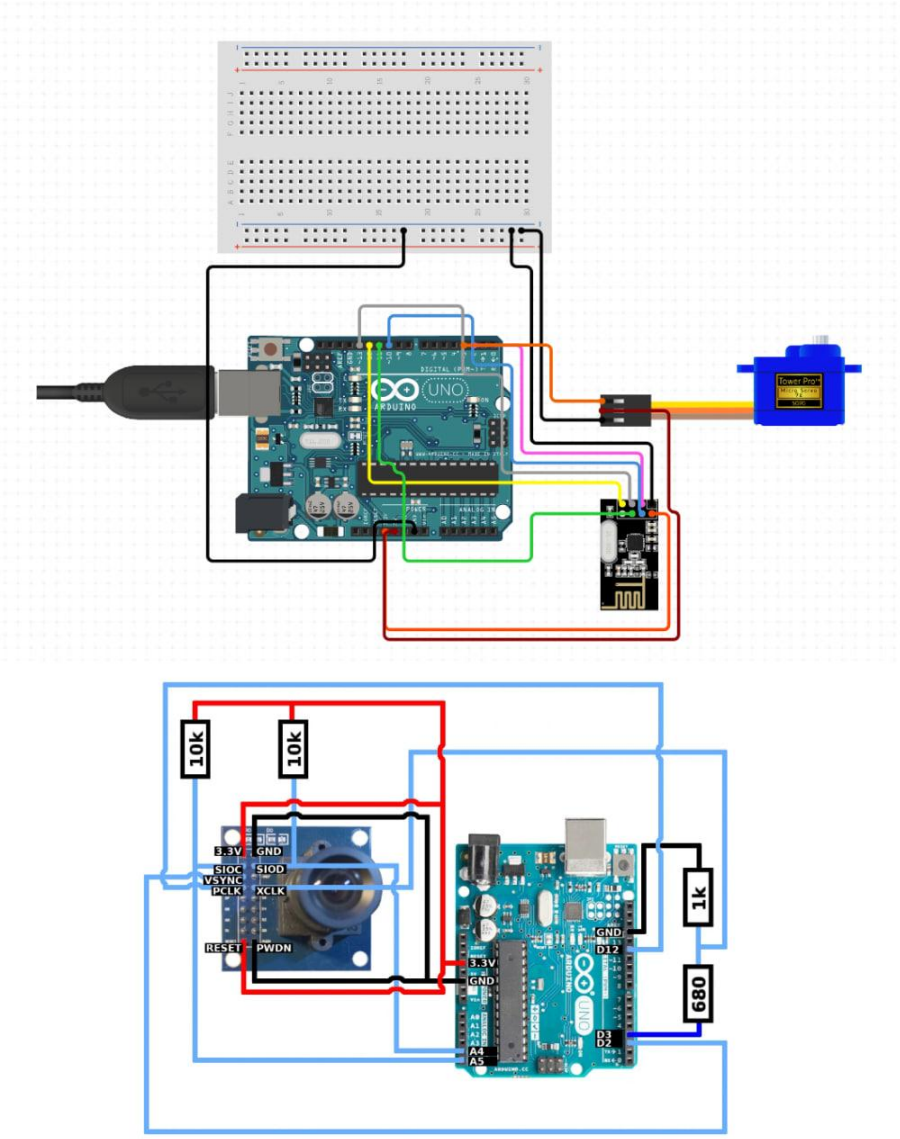
O módulo de câmera estará acoplado na ESP32 e constantemente enviando imagens para o microprocessador. Ao receber uma imagem, o ESP32 irá enviar uma requisição para o servidor com nosso modelo de predição que retornará se a imagem contém uma das pessoas cadastradas pelo modelo. Caso seja uma pessoa cadastrada, a ESP envia para o Arduino abrir a porta, ou seja, fazer uma rotação de abertura no Servo Motor. Após 10 segundos, o Arduino fechará a porta. Após isso, a ESP32 pede uma nova imagem para a câmera.

A Alexa está tracejada pois é um módulo adicional que pode ser adicionado ao sistema. Com ela, mensagens como “Porta aberta” e “Porta fechada” podem ser adicionadas, além de comandos de interação para possíveis versões futuras do projeto. Entre esses comandos, pode-se adicionar um comando de cadastro de usuário no sistema, por exemplo.

Circuito Eletrônico

O circuito eletrônico seria orquestrado da maneira abaixo. Porém ao invés de ser um arduino conectado a câmera, seria uma ESP32 que tem wi-fi integrado para comunicação com o servidor na rede local.

Abaixo na imagem temos na parte superior o circuito do Arduino Uno ligado ao servo motor e na parte inferior temos o ESP32 ligado a câmera.



Processamento da ESP32

A aplicação na ESP32, que está acoplada fisicamente à câmera, utiliza dos drivers disponibilizados na biblioteca esp32-camera [1], com a configuração da imagem no formato PIXFORMAT_GRAYSCALE e resolução FRAMESIZE_QVGA . Esses valores foram escolhidos pois a ESP32 comum não possui memória RAM suficiente para processar imagens coloridas no formato JPEG, e o QVGA (320x240) é a primeira resolução que atende ao tamanho especificado pelo modelo de 224x224. Após obter a imagem, ela é transmitida por protocolo HTTP para o endpoint do servidor.

Processamento do Servidor

O nosso servidor consiste basicamente de um script usando Python e principalmente a biblioteca Flask [2] que recebe as imagens do microprocessador através de um método POST. Posteriormente, utiliza a imagem recebida no reconhecimento facial que é feito por um método de uma classe construída por nós. Ao receber a resposta do reconhecimento dizendo se é uma pessoa conhecida ou não, esse envia para a ESP.

Processamento do Arduino

O Arduino está conectado a um servo motor que gira em 180º e a um buzzer. O Arduino funcionará como um servidor web que receberá requisição da ESP32 quando esta receber a resposta de que o rosto detectado está registrado. Ao receber essa requisição, girará o servo e apitará o buzzer. Após 10 segundos, o servo irá voltar para a posição inicial.

Detecção Facial

Para essa parte, utilizamos um algoritmo de detecção facial conhecido como Haar Cascade Classifier [3]. Um classificador Haar Cascade é usado para detectar o objeto para o qual foi treinado. Dando um resumo do seu funcionamento:

Primeiro, um conjunto de imagens positivas (imagens de rostos) e um conjunto de imagens negativas (imagens com rostos) são usados para treinar o classificador. Em seguida, extrai-se as características das imagens. Para detectar rostos em uma imagem, você procura a presença de vários recursos normalmente encontrados em rostos humanos, como a sobrancelha, em que a região acima da sobrancelha é mais clara do que a região abaixo dela. Quando uma imagem contém uma combinação de todos esses recursos, considera-se que ela contém um rosto. Exemplo de características:

Felizmente, o OpenCV [4] pode realizar a detecção de faces imediatamente, usando o Haar Cascade pré-treinado, juntamente com outros Haar Cascade para reconhecer outros objetos.

Dataset

Para o dataset, usamos um achado nesse repositório do Kaggle, contendo aproximadamente 7200 fotos diversas. Ademais, tiramos uma quantidade de fotos por volta de 20 de cada membro do grupo para adicionar ao treinamento.

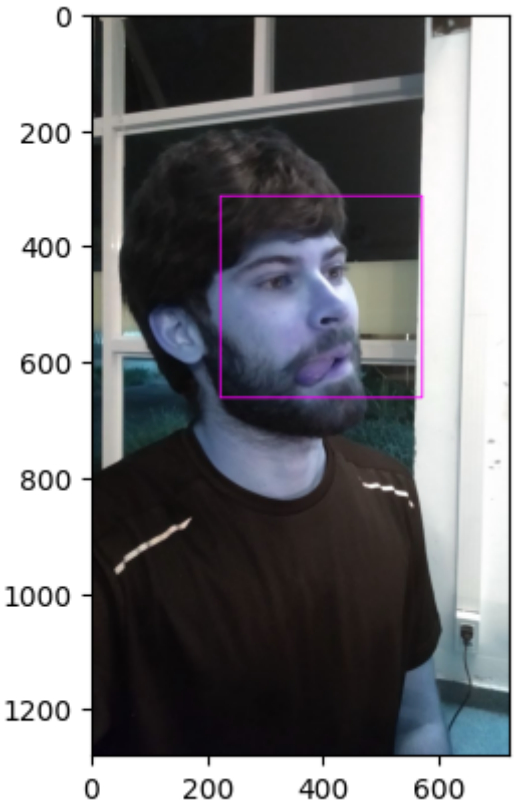
Dado essa quantidade de imagens, elas foram incluídas de maneira com que houvesse uma grande variância de características faciais, iluminação, cores, acessórios faciais, entre outros, para contemplar um bom treinamento.

Por fim, para aumentar a quantidade de imagens no dataset usamos augmentation [5] com o batch_size default de 32. Dessa forma, aumentamos em 32 vezes a quantidade de imagens disponíveis para treinamento.

Pré-processamento

Com esse conhecimento, podemos adentrar o funcionamento do pré-processamento de dados. Após termos coletado o dataset, tivemos que usar o algoritmo de detecção facial para realizar um corte no rosto de cada imagem. As imagens que não houveram rostos detectados foram excluídas do dataset. Já nas imagens no qual foram detectadas coisas que não eram rostos, elas permaneceram no dataset. Fizemos isso, pois em funcionamento real, pode ser que a câmera obtenha imagens de pessoas passando pela sua frente ou de outros objetos, o que deverá ser reconhecido como desconhecido. Dessa forma, esses "rostos falsos" servem também para contemplar mais um caso da realidade. Apesar disso, as imagens dos participantes do grupo que não foram detectados os rostos corretamente foram retiradas a mão para evitar o treinamento indevido.

Por fim, as imagens que tiveram o rosto detectado corretamente foram cortadas para o rosto e substituíram as originais, sobrando um total de aproximadamente 6400 fotos do dataset original, ou seja, sem sofrer augmentation. Agora temos todos os itens para montar o modelo e treiná-lo. Exemplo de rosto detectado:



Modelagem

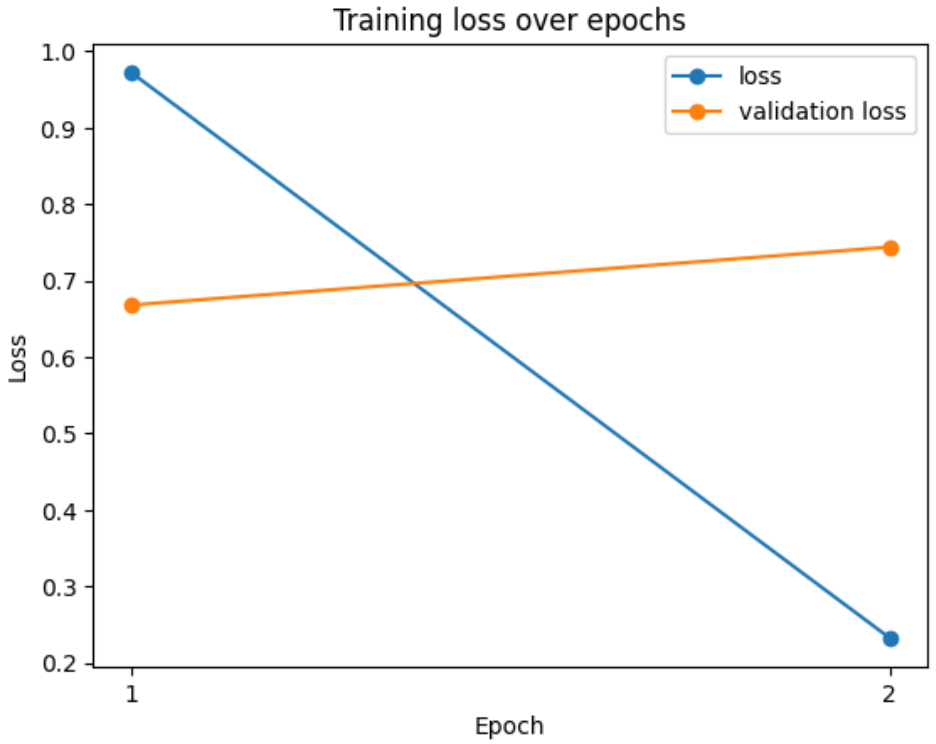
Como já dito anteriormente, usamos a rede VGGFace modelada a partir de uma VGG16 por meio de Transfer Learning como base para a nossa rede. A partir dessa rede adicionamos algumas camadas:

1. Uma camada de Average Pooling 2D
2. Duas camadas densas de dimensão 1024 com função de ativação ReLU
3. Uma camada densa de dimensão 512 com função de ativação ReLU

Estado feito o modelo, o treinamento do modelo foi composto pelo método K-fold cross-validation. Alguns dos parâmetros utilizados:

- A função de loss usada foi a categorical cross entropy
- O otimizador usado foi o Adam
- A métrica foi a acurácia
- O batch size foi 1
- A quantidade de epochs foi 20
- A quantidade de folds foi k=5

Ademais, utilizamos do algoritmo Early Stopping [6] que monitora o desempenho do modelo para cada epoch em um conjunto de validação durante o treinamento, encerrando-o dependendo do desempenho da validação. Utilizamos esse algoritmo para fazer com que o treinamento pare assim que o desempenho do modelo parar de melhorar e também para evitar o overfitting. Apresentando o gráfico comparando a loss e a accuracy do treinamento:

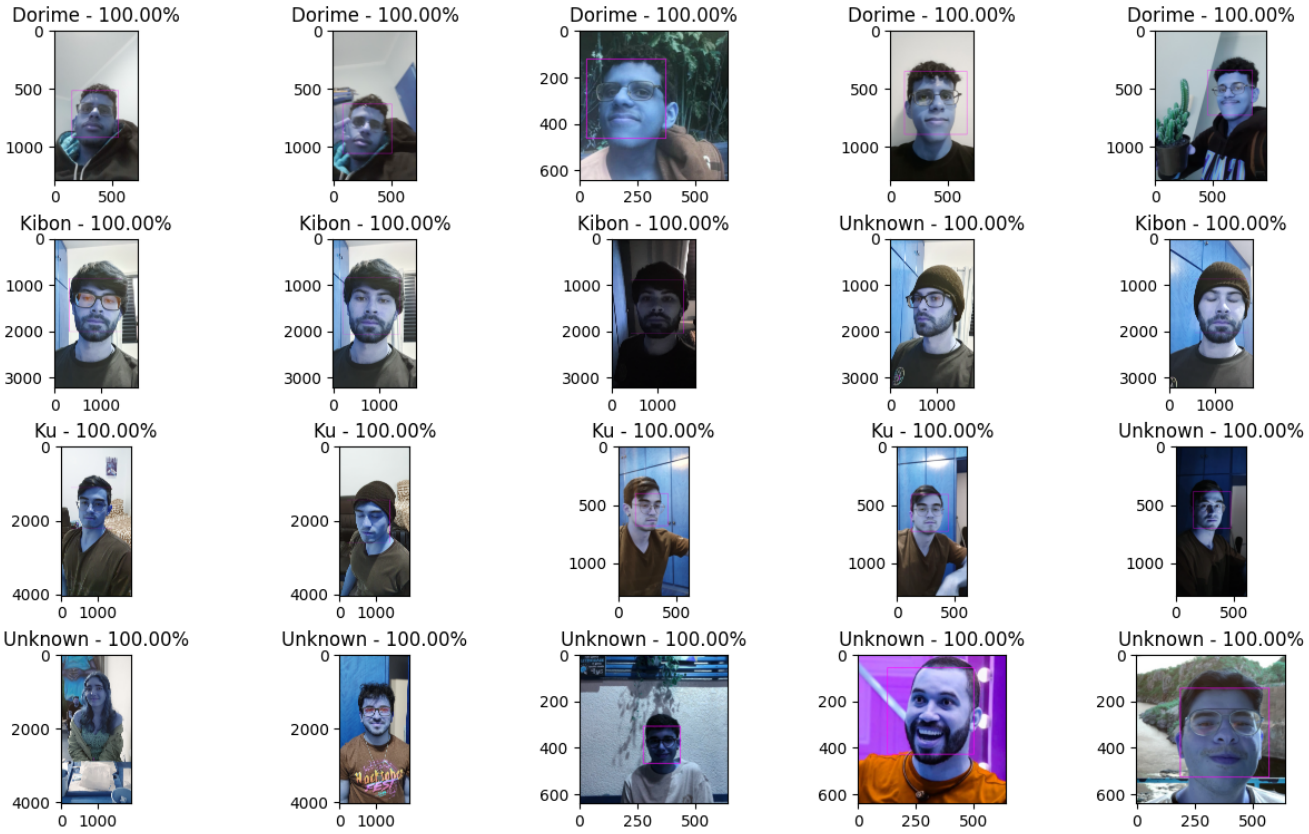


Outro ponto importante de ressaltar é que utilizamos um dataset de rostos desconhecidos muito maior do que o dataset de rostos conhecidos para que o modelo esteja condicionado a detectar rostos desconhecidos com maior precisão. Fizemos isso pois pensando numa aplicação real do nosso projeto é perigoso que tenhamos falsos positivos. Por exemplo, quando um rosto desconhecido é dado como conhecido. Esse caso tornaria o sistema de segurança fraco.

Por fim, obtemos ao final uma acurácia média entre todos os k-folds de 97.58926153182983 (+- 0.4697043813101986) e uma loss de 85.06835403442383.

Testes

Uma vez treinado o modelo, fizemos os testes com cerca de 20 imagens. Nesse conjunto há imagens de integrantes do grupo e de pessoas desconhecidas, como podemos ver na imagem abaixo:



Interpretação dos Resultados

Ao final do projeto, acreditamos que o modelo criado apresenta uma acurácia alta sem overfitting, pois utilizamos o Early Stopping para evitar o mesmo e as predições do modelo estavam majoritariamente corretas para uma quantidade boa de testes. Portanto, acreditamos que nosso objetivo com o modelo foi concluído.

Dificuldades e Considerações Finais

Sobre diversas partes do projeto, gostaríamos de ressaltar alguns pontos de dificuldades e observações acerca do trabalho realizado.

O maior dos nossos problemas foi de fato a utilização do hardware, como o microprocessador, a câmera, o Arduino e o servo motor, causando um atraso no trabalho o que não permitiu que concluímos todas as funcionalidades do projeto em questão. Esses

estavam fora do nosso escopo de aprendizado e costume, mas tínhamos nos proposto a esse desafio para aprender e criar uma aplicação real que pudesse ser usada. Criar um código que funcionasse com as especificações de hardware dos componentes foi bem complexo e fez com que gastássemos mais tempo do que esperado lidando com essas partes do projeto. Além disso, havia o problema da resolução das fotos tiradas pela câmera e também a questão do envio e recebimento de dados pelo servidor, o que poderia ser complicado dependendo do componente. Por exemplo, o Arduino não tem um módulo wi-fi embutido, levando nosso grupo a tentar outras maneiras de enviar o sinal para o mesmo. Tentamos usar um módulo de radiofrequência para enviar o sinal, porém a ESP32 não tinha esse módulo para enviar a resposta para o Arduino. Seria necessário a compra ou substituição de um dos componentes para concluir o projeto como um todo.

Tivemos outros problemas relacionados ao modelo também. Anteriormente quando havíamos treinado o modelo, atingimos o 100% de acurácia em poucas epochs fazendo com que possivelmente o modelo atingisse o overfitting. Devido a isso, acrescentamos ao modelo o uso do Early Stopping.

Referências

1. https://github.com/espressif/esp32-camera
2. https://flask.palletsprojects.com/en/2.3.x/
3. https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf
4. https://github.com/opencv/opencv/tree/master/data/haarcascades
5. https://www.tensorflow.org/tutorials/images/data_augmentation