UNIVERSIDADE FEDERAL DO PARANÁ

Alessandro Lick Cordeiro
Giovanni Geraldo Szychta
Luca Raphael Martins

RECONHECIMENTO FACIAL

CURITIBA
2019

Alessandro Lick Cordeiro
Giovanni Geraldo Szychta
Luca Raphael Martins

RECONHECIMENTO FACIAL

Relatório apresentado à disciplina de Fundamentos de Programação de Computadores do Curso de Graduação em Agronomia da Universidade Federal do Paraná sob orientação do Orientador: Prof. Jackson Antônio do Prado Lima.

Curitiba, junho de 2018.

Sumário

INTRODUÇÃO 4
DESENVOLVIMENTO 5
ESTRUTURA DO PROJETO 7
CÓDIGO 8
EXECUÇÃO DO PROGRAMA 10
CONCLUSÃO 11
REFERÊNCIAS 12

INTRODUÇÃO

Esse relatório apresenta informações relativas ao trabalho da disciplina Fundamentos de Programação de Computadores. O trabalho foi realizado na linguagem Python, pois além de ser uma linguagem fácil, ele tem um suporte muito bom para projetos relacionado a aprendizagem de máquinas e modelos utilizando imagens, o modelo para realização deste projeto foi retirado da Universidade de Berkeley, e se chama Caffe model.

Com a facilidade da tecnologia e como ela está sendo implementada ao redor do mundo, existe um apelo para o reconhecimento de padrões, principalmente programas que possam utilizar o reconhecimento de imagens, para monitoramentos, analise de segurança e dispositivos moveis, a ideia portanto é trabalhar com padrões de reconhecimento, inicialmente de face para então expandir para projetos que tenham demandas com apelo agroeconômico

O objetivo é mostrar por meio de reconhecimento de padrões qual indivíduo está aparecendo pela câmera ou webcam, o próprio programa é capaz de reconhecer e calcular um grau de confiabilidade, através do treinamento com um conjunto de imagens já disponibilizadas pelo programador.

DESENVOLVIMENTO

A manipulação das imagens é realizada utilizando a biblioteca OpenCV. Em conjunto, utilizaremos o aprendizado em duas etapas:

- Primeiramente identificamos a presença e a localização dos rostos
- Em seguida, extraímos os vetores (embeddings) que quantificam cada face em uma imagem

O modelo responsável por quantificar cada face de uma imagem é o projeto OpenFace, uma implementação Python e Torch de reconhecimento facial com aprendizado profundo.

Primeiro, inserimos uma imagem ou quadro de vídeo em nosso canal de reconhecimento de rosto. Dada a imagem de entrada, aplicamos a detecção de rosto para detectar a localização de uma face na imagem. Opcionalmente, podemos calcular pontos de referência faciais, permitindo-nos pré-processar e alinhar a face. O alinhamento do rosto, como o nome sugere, é o processo de (1) identificar a estrutura geométrica das faces e (2) tentar obter um alinhamento canônico da face com base na translação, rotação e escala.

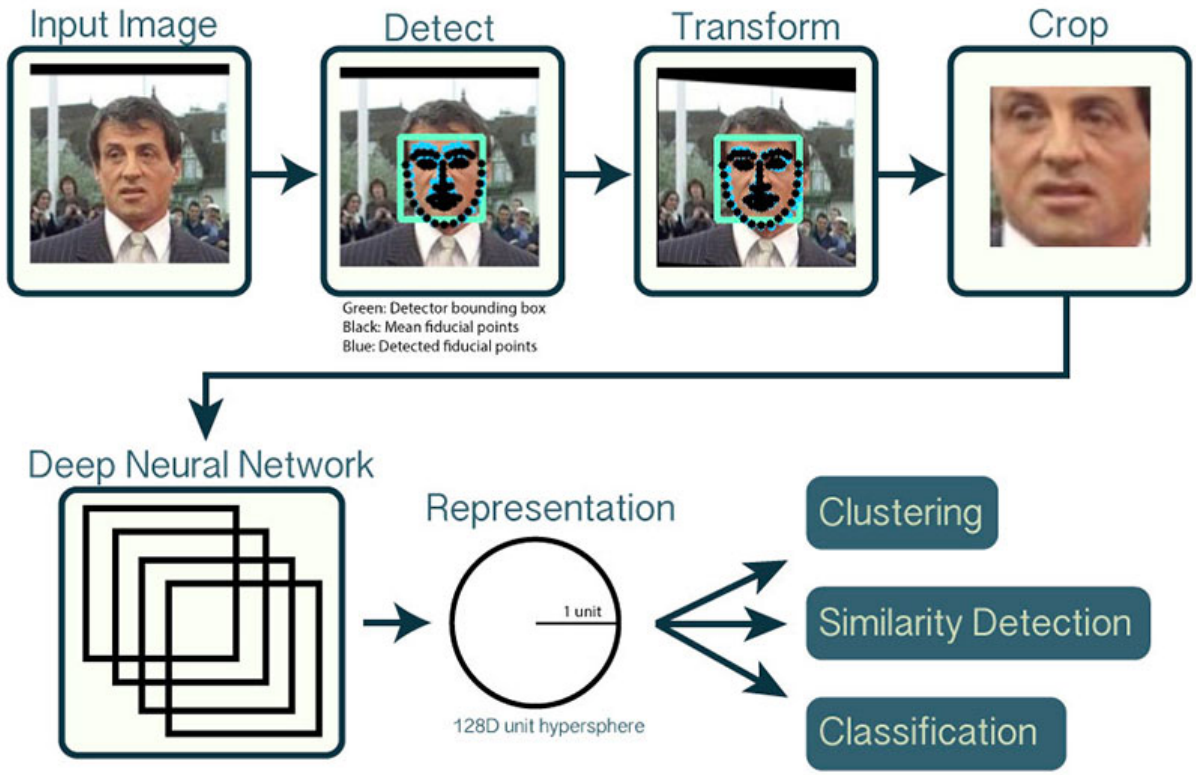


Figura 1: Funcionamento do OpenCV

Embora opcional, o alinhamento facial demonstrou aumentar a precisão do reconhecimento de faces.

Depois de aplicar o alinhamento e corte da face, passamos a face de entrada por meio de uma rede neural.

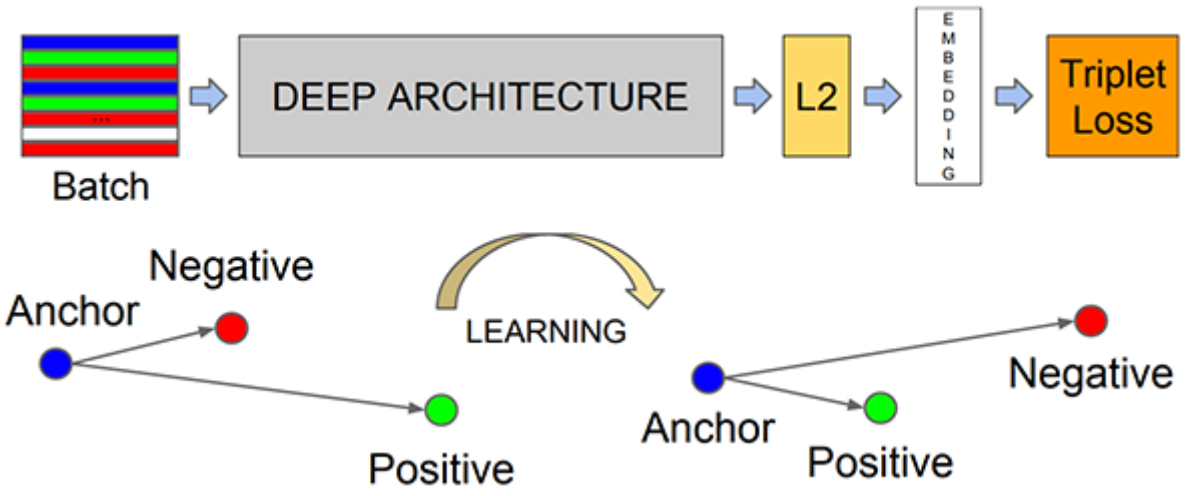


Figura 2: Como o modelo de reconhecimento facial de aprendizagem profunda calcula a incorporação de faces.

O modelo de aprendizagem profunda FaceNet calcula uma incorporação de 128-d que quantifica a face em si.

O processo de treinamento segue os seguintes passos:

- Os dados de entrada para a rede
- A função de perda de tripla

Para treinar um modelo de reconhecimento de rosto com aprendizado profundo, cada lote de dados de entrada inclui três imagens:

- A âncora
- A imagem positiva
- A imagem negativa
- A âncora é a nossa face atual e tem identidade A.

A segunda imagem é a nossa imagem positiva - esta imagem também contém uma face da pessoa A. A imagem negativa, por outro lado, não tem a mesma identidade e pode pertencer à pessoa B, C ou mesmo Y.

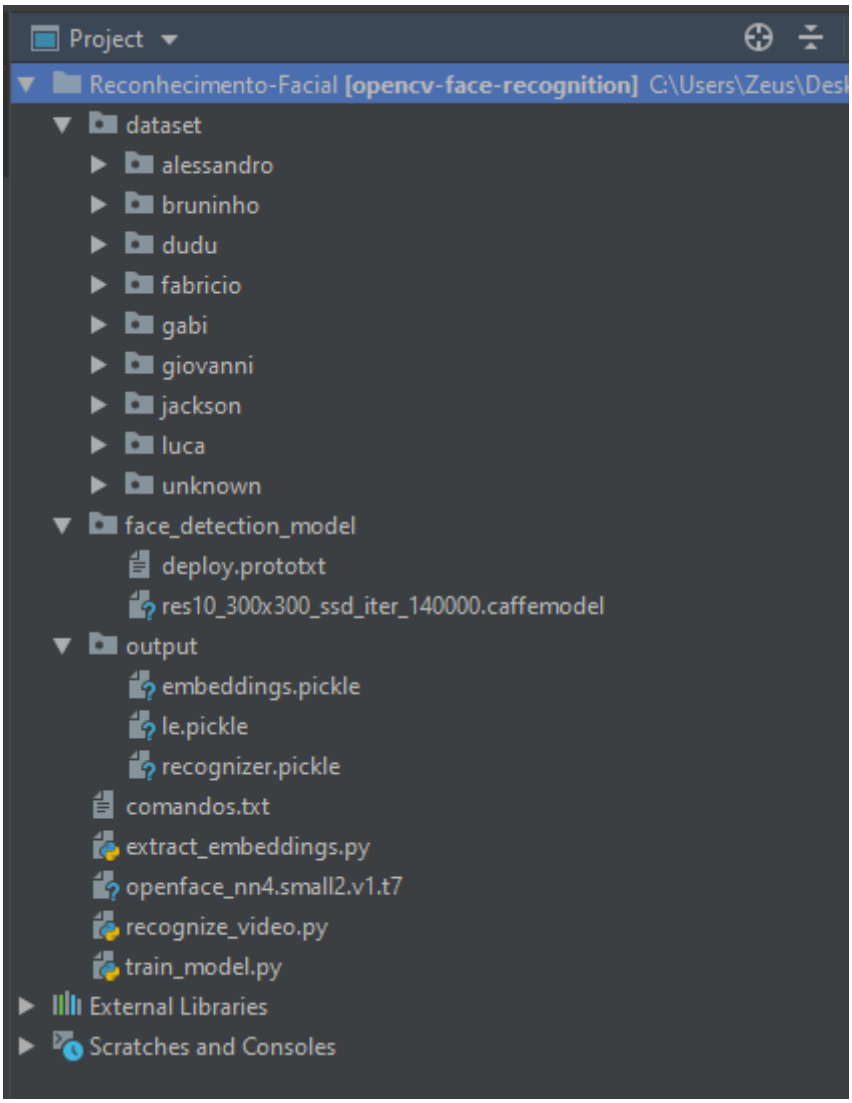
O ponto é que a âncora e a imagem positiva pertencem à mesma pessoa / face, enquanto a imagem negativa não contém a mesma face.

A rede neural calcula a aproximação para cada face e, em seguida, ajusta os pesos da rede (por meio da função de perda de tripla) de forma que:

- Os embeddings da âncora e imagem positiva estão mais próximos
- Enquanto, ao mesmo tempo, deslocando os embeddings para o pai da imagem negativa
  para uma posição mais distante e com um peso menor

Dessa maneira, a rede é capaz de aprender a quantificar faces e retornar integrações altamente robustas e diferenciadas, adequadas para reconhecimento de faces.

ESTRUTURA DO PROJETO



Dataset – Imagens utilizadas para o treinamento do modelo. Cada pasta deve ser nomeada de acordo com o nome da pessoa. Cada imagem deve ser numerada iniciando com a sequencia 00000 e ir sendo incrementada

Face_detection_model – Pasta que contem o modelo que será treinado

Output – saída dos vetores de embeddings que serão utilizados/atualizados através da detecção dos padrões

Openface_nn4.small2.v1.t7 – Modelo utilizado para a detecção das faces

CÓDIGO

```
# Loop feito pra analisar os frames
while True:
    # Coletando os frames
    frame = vs.read()

    # Deixando o frame das imagens com 600px pra nao distorcer demais
    # e assim mantendo o aspecto da imagem mais natural
    frame = imutils.resize(frame, width=600)
    (h, w) = frame.shape[:2] #mantem a proporção do tamanho do frame

    # Contruindo os padrões a partir das imagens
    imageBlob = cv2.dnn.blobFromImage(
        cv2.resize(frame, (300, 300)), 1.0, (300, 300),
        (104.0, 177.0, 123.0), swapRB=False, crop=False)

    # Aplica os padrões de aprendizado da biblioteca OpenCV para localizar as faces
    # Localiza as faces baseadas na entrada da camera
    detector.setInput(imageBlob)
    detections = detector.forward()

    # Laço em que ocorrem as detecções
    for i in range(0, detections.shape[2]):
        # Estimativa dos graus de confiança e de previsões
        confidence = detections[0, 0, i, 2]

        # Filtrando as detecções
        if confidence > args["confidence"]:
            # Coordenadas para as caixas ao redor das faces
            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")

            # Extraindo as faces
            face = frame[startY:endY, startX:endX]
            (fH, fW) = face.shape[:2]

            # Verifica se o tamanho das faces está no padrao
            if fW < 20 or fH < 20:
                continue

            # Controi um blob para a estimativa da face com base no modelo
            # Aplica a estimativa do blob ao modelo treinado
            # quantificação da face
            faceBlob = cv2.dnn.blobFromImage(face, 1.0 / 255,
                (96, 96), (0, 0, 0), swapRB=True, crop=False)
            embedder.setInput(faceBlob)
```

```
# Laço em que ocorrem as detecções
for i in range(0, detections.shape[2]):
    # Estimativa dos graus de confiança e de previsões
    confidence = detections[0, 0, i, 2]

    # Filtrando as detecções
    if confidence > args["confidence"]:
        # Coordenadas para as caixas ao redor das faces
        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
        (startX, startY, endX, endY) = box.astype("int")

        # Extraindo as faces
        face = frame[startY:endY, startX:endX]
        (fH, fW) = face.shape[:2]

        # Verifica se o tamanho das faces está no padrao
        if fW < 20 or fH < 20:
            continue

        # Controi um blob para a estimativa da face com base no modelo
        # Aplica a estimativa do blob ao modelo treinado
        # quantificação da face
        faceBlob = cv2.dnn.blobFromImage(face, 1.0 / 255,
            (96, 96), (0, 0, 0), swapRB=True, crop=False)
        embedder.setInput(faceBlob)
        vec = embedder.forward()

        #Classificação e reconhecimento das faces
        preds = recognizer.predict_proba(vec)[0]
        j = np.argmax(preds)
        proba = preds[j]
        name = le.classes_[j]

        # Contrução da caixa em volta das faces
        # Mostra a probabilidade associada a cada face
        text = "{}: {:.2f}%".format(name, proba * 100)
        y = startY - 10 if startY - 10 > 10 else startY + 10
        cv2.rectangle(frame, (startX, startY), (endX, endY),
            (0, 0, 255), 2)
        cv2.putText(frame, text, (startX, y),
            cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)
```

```
    # Atualização do contador de frames
    fps.update()

    # Exibindo os frames
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF

    # Condição de saída do loop
    if key == ord("q"):
        break

# Parada do timer e exibição das informações de FPS
fps.stop()
print("[INFO] elasped time: {:.2f}".format(fps.elapsed()))
print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))

# Destruição das telas
cv2.destroyAllWindows()
vs.stop()
```

EXECUÇÃO DO PROGRAMA

1. Reunir as imagens para o treinamento e colocar nas pastas corretas
2. Executar o comando na pasta do projeto ou via terminal do pycharm:
   python extract_embeddings.py -i dataset -e output/embeddings.pickle -d face_detection_model -m openface_nn4.small2.v1.t7
3. Em seguida executar o treinamento do modelo. Para isso, executar o seguinte comando:
   python train_model.py -e output/embeddings.pickle -r output/recognizer.pickle -l output/le.pickle
4. Para executar o reconhecimento facial, executar o seguinte comando:
   python recognize_video.py -d face_detection_model -m openface_nn4.small2.v1.t7 -r output/recognizer.pickle -l output/le.pickle

CONCLUSÃO

A realização desse trabalho é importante para a iniciação de projetos voltados a programação, através do contato com a linguagem Python, os integrantes da equipe tiveram a ideia de buscar métodos que pudessem estar relacionados com reconhecimento de padrões, e ao desenvolver este projeto os alunos puderam compreender que não é tão difícil o desenvolvimento de programas que utilizam recursos disponíveis no nosso dia a dia como computadores, tablets e aparelhos móveis e que estes facilitam muito a vida de seus usuários.

Através do “aprendizado de máquinas”, outros projetos podem ser desenvolvidos, o que indica que o reconhecimento facial é só o início para o reconhecimento de padrões, a intenção portanto é que no futuro possamos dar continuidade e aplicarmos na vida profissional como engenheiros agrônomos, um exemplo de planos para o futuro são os programas envolvendo o reconhecimento de plantas (identificação taxonômica) outros exemplo é o reconhecimento de padrões de doenças agrícolas, feitas por drones.

REFERÊNCIAS

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html

https://circuitdigest.com/tutorial/real-life-object-detection-using-opencv-python-detecting-objects-in-live-video

https://www.datacamp.com/community/tutorials/face-detection-python-opencv

https://caffe.berkeleyvision.org/

https://www.learnopencv.com/blob-detection-using-opencv-python-c/

https://cmusatyalab.github.io/openface/models-and-accuracies/