UNIVERSIDADE FEDERAL DO ESTADO DO RIO GRANDE DO NORTE

Disciplina / Área de conhecimento:

ECT2702 - TÓPICOS AVANÇADOS EM INFORMÁTICA I - T01 (2019.2)

Teacher Sensor

Luís Fernando Tavares

Natal, 2019

Sumário

1 Introdução

O objetivo a ser atingido com esse programa é: usa uma base de dados com o rosto de diversas pessoas aleatórias, comparar com o rosto do seu professor para verificar se ele está se aproximando do seu computador e mudar para uma imagem pré-selecionada em tela cheia, o ponto desse sistema é para quando estiver na sala de aula e estiver fazendo algo que não tenha haver com a aula e seu professor se aproximar de você ou seu computador, mudar a tela para uma imagem que tenha haver com a aula, seja um documento, slide, código e etc.

A base de dados possui dois tipos de imagem, as primeiras são diversas imagem de seu professor e a segunda são de pessoas aleatórias, as do professor podem ser obtidas através de um vídeo e as das pessoas pela internet, as imagens devem conter principalmente o rosto da pessoa.

2 Metodologia

O modelo de Machine Learning utilizado para a construção do código é a Rede Neural Convolucional (RNN ou CNN - Convolutional Neural Network), ela é um algoritmo de aprendizagem profunda (Deep Learning) que pode captar uma imagem de entrada, atribuir importância (pesos e vieses aprendíveis) a vários aspectos / objetos da imagem e ser capaz de diferenciar um do outro.

O pré-processamento necessário em uma CNN é muito menor em comparação com outros algoritmos de classificação. Enquanto nos métodos primitivos os filtros são projetados manualmente, com treinamento suficiente, as CNN têm a capacidade de aprender esses filtros/características.

Em outras palavras, a rede pode ser treinada para entender melhor a sofisticação da imagem.

Ela possui diversas partes para sua construção, mas pode ser demonstrada pelos seguintes passos:

1.Entrada de Imagens:

Camada de entrada na CNN deve conter dados de imagem. Os dados da imagem são representados por uma matriz tridimensional. É preciso remodelá-lo em uma única coluna. Suponha que você tenha uma imagem da dimensão 28 x 28 = 784, é necessário convertê-la em 784 x 1 antes de alimentar a entrada.

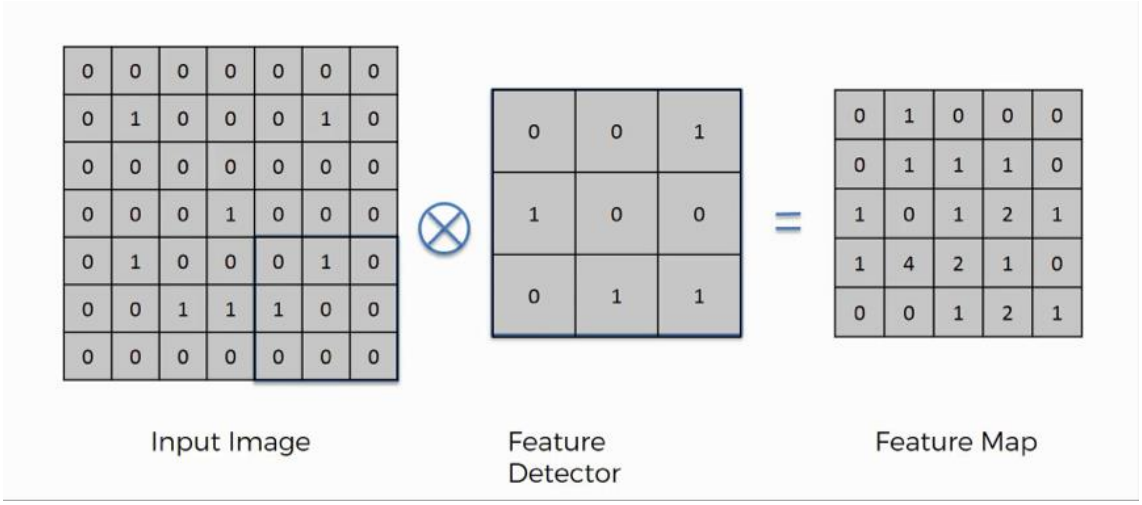
A imagem da CNN são representadas de forma que cada pixel é formado por um byte, as cores variam em uma escala de 0 à 255 (preto à branco), imagens em preto e branco são de duas dimensões e coloridas são de três dimensões ou três canais de cores (RGB).

O papel da CNN é reduzir as imagens para um formato mais fácil de processar, sem perder recursos essenciais para obter uma boa previsão.

2.Camada de Convuloção:

A camada Convolacional às vezes é chamada de camada extratora de recurso, porque os recursos da imagem são extraídos nessa camada. Antes de tudo, uma parte da imagem é conectada à camada para executar a operação de convolução, é calculado o produto escalar entre o campo receptivo (é uma região local da imagem de entrada que tem o mesmo tamanho do filtro) e o filtro.

O resultado da operação é um número inteiro único do volume de saída. Em seguida, o filtro é deslizado sobre o próximo campo receptivo da mesma imagem de entrada por um Stride (a quantidade de pixels que o filtro é deslocado) e fazemos a mesma operação novamente. Repetiremos o mesmo processo repetidamente até passarmos por toda a imagem. A saída será a entrada para a próxima camada.

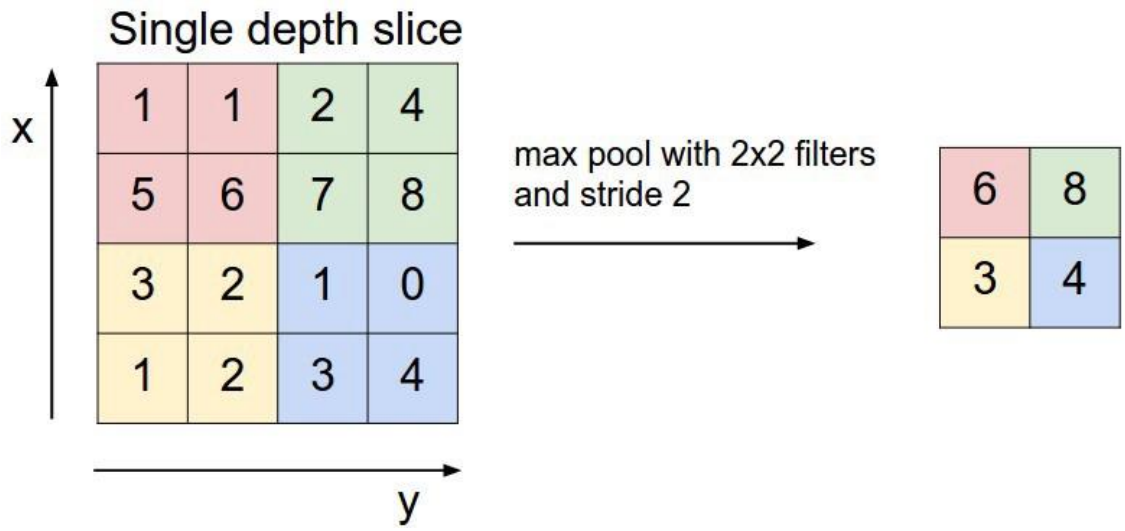


A camada também contém a ativação ReLU para fazer com que todo o valor negativo seja zero.

Para facilitar o processamento das imagens, elas passam por um processo que transforma as cores das imagens para preto e branco, deixando o processamento mais simples.

3.Camada de Pooling:

A camada de pooling é usada para reduzir o volume espacial da imagem de entrada após a convolução. É usado entre duas camadas de convolução. Se aplicarmos o FC* (Fully Connected Layer ou Camada Totalmente Conectada) após a camada Convolacional sem aplicar o pool ou o pool máximo, será computacionalmente caro e não o queremos. Portanto, o pool máximo é a única maneira de reduzir o volume espacial da imagem de entrada.



*Camada Totalmente Conectada envolve pesos, preconceitos e neurônios. Ele conecta neurônios em uma camada a neurônios em outra camada. É usado para classificar imagens entre diferentes categorias por treinamento.

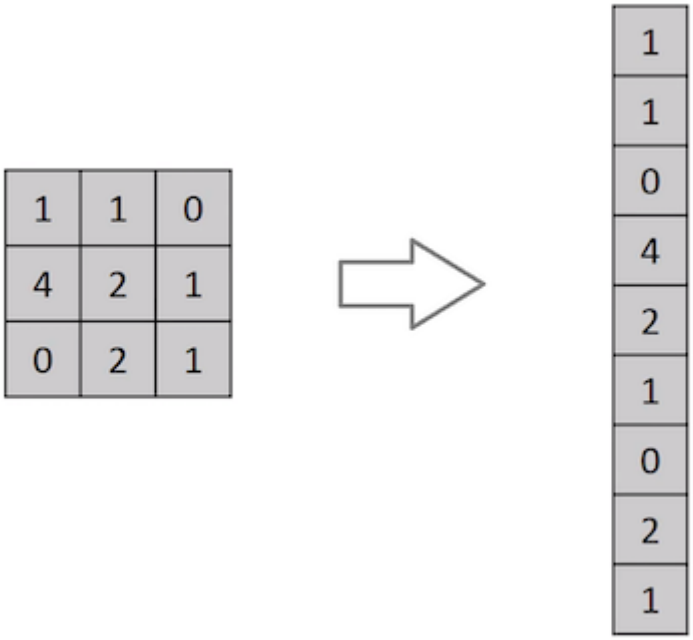
Logo acima, é aplicado o pool máximo em uma fatia de profundidade única com Stride de 2. A entrada da matriz de dimensão 4 x 4 é reduzida para 2 x 2.

Não há parâmetro na camada, mas ele possui dois hiperparâmetros - Filter (F) e Stride (S).

4.Flattening (Achatamento):

Agora que a imagem de entrada está convertida em um formato adequado para nosso Perceptron de vários níveis, achataremos a imagem em um vetor de coluna.

Esse processo é chamado de Flattening e é usado para transformar a matriz obtida através do processo de Pooling em uma coluna, e cada linha da coluna se torna uma entrada para uma rede neural.



5.Rede Neural:

Adicionar uma Camada Totalmente Conectada (FC) é uma maneira (geralmente) barata de aprender combinações não lineares dos recursos de alto nível, representados pela saída da camada convolucional.

A saída achatada é alimentada a uma rede neural feed-forward e a retropropagação é aplicada a todas as iterações do treinamento.

Ao longo de uma série de epochs, o modelo é capaz de distinguir entre características dominantes e certas características de baixo nível nas imagens e classificá-las usando a técnica de Classificação Softmax.

3 Código

O código deste programa foi dividido em quatro partes, marcadas como: Teacher Input, Teacher Train, Camera Reader e Image Show.

A seguir, mostraremos como cada um dos códigos funcionam:

1.Teacher Input:

1º Redimensionamento:

```
IMAGE_SIZE = 64


def resize_with_pad(image, height=IMAGE_SIZE, width=IMAGE_SIZE):

    def get_padding_size(image):
        h, w, _ = image.shape
        longest_edge = max(h, w)
        top, bottom, left, right = (0, 0, 0, 0)
        if h < longest_edge:
            dh = longest_edge - h
            top = dh // 2
            bottom = dh - top
        elif w < longest_edge:
            dw = longest_edge - w
            left = dw // 2
            right = dw - left
        else:
            pass
        return top, bottom, left, right

    top, bottom, left, right = get_padding_size(image)
    BLACK = [0, 0, 0]
    constant = cv2.copyMakeBorder(image, top , bottom, left, right, cv2.BORDER_CONSTANT, value=BLACK)

    resized_image = cv2.resize(constant, (height, width))

    return resized_image
```

Define o tamanho das imagens para um tamanho pré-selecionado, no caso 64X64, e as armazena em resized_image e retorna ele.

2º Atravessar Diretório:

```
images = []
labels = []
def traverse_dir(path):
    for file_or_dir in os.listdir(path):
        abs_path = os.path.abspath(os.path.join(path, file_or_dir))
        print(abs_path)
        if os.path.isdir(abs_path):  # dir
            traverse_dir(abs_path)
        else:                        # file
            if file_or_dir.endswith('.jpg'):
                image = read_image(abs_path)
                images.append(image)
                labels.append(path)

    return images, labels
```

Atravessa o diretório path e retorna uma lista contendo os nomes das entradas no diretório fornecido pelo path, normaliza e as adiciona em abs_path, retorna verdadeiro se todas as imagens estão no diretório path e atravessa o diretório abs_path, senão as armazena se ela termina com .jpg, passa pela função read_image e adiciona na lista images e o resto na labels e retorna às duas listas.

3º Ler Imagem:

```
def read_image(file_path):
    image = cv2.imread(file_path)
    image = resize_with_pad(image, IMAGE_SIZE, IMAGE_SIZE)

    return image
```

Le a imagem e chama a função resize_with_pad para modificar o tamanho da imagem e retorna a imagem

4º Extrair Dados:

```
def extract_data(path):
    images, labels = traverse_dir(path)
    images = np.array(images)
    labels = np.array([0 if label.endswith('Teacher') else 1 for label in labels])

    return images, labels
```

Extrai as imagens das lista images e labels, passa elas pela função traverse_dir, cria dois vetores, para images e labels, no labels retorna valor 0 se as imagens que estão na pasta 'Teacher', senão o valor é 1 em seguida armazena no vetor labels, e para images se forem da lista images e retorna os dois vetores.

2.Teacher Train:

1º Importação:

```
from Teacher_Input import extract_data, resize_with_pad, IMAGE_SIZE
```

Importa do código Teacher_Input as funções extract_data e resize_with_pad, e a variavel IMAGE_SIZE.

2º Conjunto de dados:

```
class Dataset(object):

    def __init__(self):
        self.X_train = None
        self.X_valid = None
        self.X_test = None
        self.Y_train = None
        self.Y_valid = None
        self.Y_test = None
```

Armazena toda a base de dados para o treinamento.

2.1º Ler:

```
def read(self, img_rows=IMAGE_SIZE, img_cols=IMAGE_SIZE, img_channels=3, nb_classes=2):
    images, labels = extract_data('./data/')
    labels = np.reshape(labels, [-1])
    # numpy.reshape
    X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.3, random_state=random.randint(0, 100))
    X_valid, X_test, y_valid, y_test = train_test_split(images, labels, test_size=0.5, random_state=random.randint(0, 100))
```

Lê as imagens que estão na pasta data, ordená-las e separá-las em dados de teste e treinamento.

2.2º Dados:

```
if K.common.image_dim_ordering() == 'th':
    X_train = X_train.reshape(X_train.shape[0], 3, img_rows, img_cols)
    X_valid = X_valid.reshape(X_valid.shape[0], 3, img_rows, img_cols)
    X_test = X_test.reshape(X_test.shape[0], 3, img_rows, img_cols)
    input_shape = (3, img_rows, img_cols)
else:
    X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols, 3)
    X_valid = X_valid.reshape(X_valid.shape[0], img_rows, img_cols, 3)
    X_test = X_test.reshape(X_test.shape[0], img_rows, img_cols, 3)
    input_shape = (img_rows, img_cols, 3)

print('X_train shape:', X_train.shape)
print(X_train.shape[0], 'train samples')
print(X_valid.shape[0], 'valid samples')
print(X_test.shape[0], 'test samples')
```

Redimensiona os dados, os embaralha e divide entre conjuntos de treinos e testes.

2.3º Classe Binária:

```
Y_train = np_utils.to_categorical(y_train, nb_classes)
Y_valid = np_utils.to_categorical(y_valid, nb_classes)
Y_test = np_utils.to_categorical(y_test, nb_classes)

X_train = X_train.astype('float32')
X_valid = X_valid.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_valid /= 255
X_test /= 255

self.X_train = X_train
self.X_valid = X_valid
self.X_test = X_test
self.Y_train = Y_train
self.Y_valid = Y_valid
self.Y_test = Y_test
```

Converter vetores de classe em matrizes de classe binária.

3º Modelo:

Criação de uma rede neural.

3.1º Arquivo:

```
FILE_PATH = './store/model.h5'
```

O Arquivo onde o modelo criado será salvo.

3.2º Construção do Modelo:

```
def __init__(self):
    self.model = None

def build_model(self, dataset, nb_classes=2):
    self.model = Sequential()

    self.model.add(Convolution2D(32, 3, 3, border_mode='same', input_shape=dataset.X_train.shape[1:]))
    self.model.add(Activation('relu'))
    self.model.add(Convolution2D(32, 3, 3))
    self.model.add(Activation('relu'))
    self.model.add(MaxPooling2D(pool_size=(2, 2)))
    self.model.add(Dropout(0.25))

    self.model.add(Convolution2D(64, 3, 3, border_mode='same'))
    self.model.add(Activation('relu'))
    self.model.add(Convolution2D(64, 3, 3))
    self.model.add(Activation('relu'))
    self.model.add(MaxPooling2D(pool_size=(2, 2)))
    self.model.add(Dropout(0.25))

    self.model.add(Flatten())
    self.model.add(Dense(512))
    self.model.add(Activation('relu'))
    self.model.add(Dropout(0.5))
    self.model.add(Dense(nb_classes))
    self.model.add(Activation('softmax'))

    self.model.summary()
```

Construir uma rede neural utilizando a CNN e criando as camadas de neurônios.

3.3º Treinamento:

```
def train(self, dataset, batch_size=32, nb_epoch=40, data_augmentation=True):
```

Treinamento da base de dados (das imagens).

3.3.1º SGD + Momentum:

```
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
self.model.compile(loss='categorical_crossentropy',
                   optimizer=sgd,
                   metrics=['accuracy'])
if not data_augmentation:
    print('Not using data augmentation.')
    self.model.fit(dataset.X_train, dataset.Y_train,
                   batch_size=batch_size,
                   nb_epoch=nb_epoch,
                   validation_data=(dataset.X_valid, dataset.Y_valid),
                   shuffle=True)
else:
    print('Using real-time data augmentation.')
```

Realizar o treinamento do modelo usando o SGD + Momentum

A Stochastic Gradient Descent (SGD ou Descida Estocástica do Gradiente) é a variação preferida da descida do gradiente que estima o gradiente a partir de uma pequena amostra de entrada de treinamento escolhida aleatoriamente em cada iteração chamada minibatches,os quais são gerados baralhando os dados de treinamento e selecionando aleatoriamente certo número de amostras de treinamento.

A técnica de Momentum é uma abordagem que fornece uma regra de atualização motivada da perspectiva física da otimização.

A vantagem do Momentum é que ele faz uma alteração muito pequena no SGD, mas fornece um grande impulso à velocidade do aprendizado.

3.3.2º Pré-Processamento:

```
datagen = ImageDataGenerator(
    featurewise_center=False,              #1
    samplewise_center=False,               #2
    featurewise_std_normalization=False,   #3
    samplewise_std_normalization=False,    #4
    zca_whitening=False,                   #5
    rotation_range=20,                     #6
    width_shift_range=0.2,                 #7
    height_shift_range=0.2,                #8
    horizontal_flip=True,                  #9
    vertical_flip=False)                   #10
```

Cria dados de imagem aprimorados para o modelo em tempo real, cada código realizando as mudanças a seguir:

1.Defina a média de entrada como 0 no conjunto de dados

2.Defina cada média da amostra como 0

3.Dividir entradas pelo padrão do conjunto

4.Divida cada entrada pelo seu padrão

5.Aplicar o clareamento ZCA

6.Gire aleatoriamente as imagens no intervalo

7.Alternar aleatoriamente imagens

8.Alternar aleatoriamente imagens verticalmente

9.Virar aleatoriamente imagens(horizontal)

10.Virar aleatoriamente imagens(vertical)

3.3.3º Ajuste de Dados:

```
datagen.fit(dataset.X_train)
```

Quantidades de cálculos necessários para normalização em termos de recursos,ou seja, calcula todas as estatísticas necessárias para realmente realizar as transformações nos dados da imagem.

3.3.4º Ajustar Modelo:

```
self.model.fit_generator(datagen.flow(dataset.X_train, dataset.Y_train,
                                       batch_size=batch_size),
                         samples_per_epoch=dataset.X_train.shape[0],
                         nb_epoch=nb_epoch,
                         validation_data=(dataset.X_valid, dataset.Y_valid))
```

Ajusta o modelo nos lotes gerados pela datagen.flow(), ou seja, podemos configurar o tamanho do lote e preparar o gerador de dados e obter lotes de imagens.

3.4º Salvar e Carregar:

```
def save(self, file_path=FILE_PATH):
    print('Model Saved.')
    self.model.save(file_path)


def load(self, file_path=FILE_PATH):
    print('Model Loaded.')
    self.model = load_model(file_path)
```

Salvar o modelo no arquivo e carregá-lo.

3.5º Prever:

```
def predict(self, image):
    if K.common.image_dim_ordering() == 'th' and image.shape != (1, 3, IMAGE_SIZE, IMAGE_SIZE):
        image = resize_with_pad(image)
        image = image.reshape((1, 3, IMAGE_SIZE, IMAGE_SIZE))
    elif K.common.image_dim_ordering() == 'tf' and image.shape != (1, IMAGE_SIZE, IMAGE_SIZE, 3):
        image = resize_with_pad(image)
        image = image.reshape((1, IMAGE_SIZE, IMAGE_SIZE, 3))
    image = image.astype('float32')
    image /= 255
    result = self.model.predict_proba(image)
    print(result)
    result = self.model.predict_classes(image)

    return result[0]
```

Prever a imagem do treinamento utilizando funções extract_data e resize_with_pad para comparar com a imagem retirada da Webcam.

3.6º Avaliar:

```
def evaluate(self, dataset):
    score = self.model.evaluate(dataset.X_test, dataset.Y_test, verbose=0)
    print("%s: %.2f%%" % (self.model.metrics_names[1], score[1] * 100))
```

Avaliar o treinamento, a taxa de acerto do treinamento

3.7º Funcionamento:

```
if __name__ == '__main__':
    dataset = Dataset()
    dataset.read()

    model = Model()
    model.build_model(dataset)
    model.train(dataset, nb_epoch=10)
    model.save()

    model = Model()
    model.load()
    model.evaluate(dataset)
```

Ativar o código para treinar as imagens.

3.Camera Reader:

1º Importação:

```
from Teacher_Train import Model
from Image_Show import show_image
```

Importa do código Teacher_Train a classe modelo, e a função show_image da Image_Show

2º Captura de tela:

```
if __name__ == '__main__':
    cap = cv2.VideoCapture(0)
    cascade_path = "C:\\Users\\User\\Downloads\\opencv\\sources\\data\\haarcascades\\haarcascade_frontalface_default.xml"
    model = Model()
    model.load()
```

Liga a webcam, captura a imagem transmitida por ela e carrega o modelo criado pelo código Teacher_Train.

3º Reconhecimento do Rosto:

```
  while True:
      _, frame = cap.read()


      frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)


      cascade = cv2.CascadeClassifier(cascade_path)


      facerect = cascade.detectMultiScale(frame_gray, scaleFactor=1.2, minNeighbors=3, minSize=(10, 10))
```

Enquanto ocorre a captura de tela, a conversão em escala de cinza, um filtro, é salvo em frame_gray, os recursos do classificador em cascata são salvos em cascade, com isso é executado o reconhecimento de objeto (reconhecimento de rosto) e é salvo na facerect.

4º Rosto Detectado:

```
    if len(facerect) > 0:
        print('face detected')
        color = (255, 255, 255)
        for rect in facerect:


            x, y = rect[0:2]
            width, height = rect[2:4]
            image = frame[y - 10: y + height, x: x + width]

            result = model.predict(image)
            if result == 0:
                print('Teacher is approaching')
                show_image()
            else:
                print('Not teacher')


    k = cv2.waitKey(100)

    if k == 27:
        break


cap.release()
cv2.destroyAllWindows()
```

Se o valor de facerect for maior que zero, cria-se um retângulo ao redor da face detectada e a imagem é inserida na função predict do modelo, se o resultado for 0, o professor está se aproximando e ativamons a função show_image, mostrando a imagem pré-selecionada e ele para ou se a tecla Esc for pressionada, senão o código continua rodando.

4.Image Show:

1º Mostrar Imagem:

```
def show_image(image_path='s_pycharm.jpg'):
    app =  QtWidgets.QApplication(sys.argv)
    pixmap = QtGui.QPixmap(image_path)
    screen = QtWidgets.QLabel()
    screen.setPixmap(pixmap)
    screen.showFullScreen()
    sys.exit(app.exec_())
```

Lê a tela, redimensiona a imagem pré-selecionada e depois a mostra na tela em tela cheia.

4 Experimentos

Os tipos de testes executados foram para, de acordo com as bases de dados fornecidos, ter um gral de certeza de é possível identificar o rosto do seu professor, capturado pela webcam, utilizando uma base de dados das imagens dele para comparação, no treinamento foi utilizado uma rede neural com 18 camadas com 512 neurônios, sendo utilizadas nas camadas de ativação a função ReLU, no Max Pooling uma matriz 2x2 e na de saída a função SoftMax, com uma saída de 2 neurônios, e para descobrir a estimativa do gradiente, o batch_size = 32 e os epochs = 40, e em seguida foi inserida a imagem capturada foi remodelada para uma matriz 64X64, passada por um "filtro" para deixar preto e branco e comparada a previsão do modelo com os dados de treinamento para serem averiguada se a imagem era do professor e abrindo a imagem pré-selecionada em tela cheia.

Durante a execução do código houve dificuldades para fazê-lo funcionar, como o código foi baseado em um já existente há 3 anos (2016), várias das bibliotecas estavam desatualizadas e certas funções já não existiam mais e foi necessário atualizar e mudar essas bibliotecas e funções.

Parâmetro avaliado foram uma base de dados das imagem do professor, com 716 imagens, e imagem de outras pessoas, com 166 imagens, a primeira para comparar com a imagem capturada pela webcam e saber se era o professor, e a segunda para comparar com a mesma imagem capturada e saber se era outra pessoa diferente do professor.

Os resultados obtidos foram, de acordo com os testes e treinamento, uma taxa de 100% de acurácia na identificação do rosto do professor, o que foi confirmado ser veridico quando ele se aproximou da webcam e em 4 a 5 segundos o programa o reconheceu e colocou a imagem pré-selecionada em tela cheia, assim este programa pode ser usado para identificar o rosto do professor e colocar uma imagem em tela cheia quando ele se aproxima e esconder algo que estivesse fazendo que não tinha algo haver com o trabalho.

Foi feita uma validação dos dados utilizando outras 2 bases de dados, substituindo as anteriores, a do professor com 33 imagens e de outras pessoas com 25 imagens, e foram passadas pelo modelo já treinado com as bases de dados anteriores e a taxa de acurácia foi de 79.31%, uma taxa consideravel visto que as bases de validação são muito menores que as anteriores.

Link do cogido no Github:

https://github.com/LtavaresII/TeacherSensor