



Desenvolvimento de uma CNN para classificar personagens de Big Bang Theory

Deivid Braian Smarzaro



Introdução

Labels:

leonard: 23 samples

howard: 68 samples

sheldon: 176 samples

raj: 70 samples

total 337 samples

kaggle



Pré-processamento dos dados

```
img_size = 50

#Enumera os labels
class_dict = {}
#celebrity_file_names_dict contém apenas caminhos das imagens de cada personagem
for i,celebrity_name in enumerate(celebrity_file_names_dict.keys()):
    print(celebrity_name,end=' ')
    class_dict[celebrity_name] = i
    print(len(celebrity_file_names_dict[celebrity_name]),'samples')

#Coloca todos os dados e seus labels em duas listas paralelas
x,y = [],[]
for k in celebrity_file_names_dict:
    for i,v in enumerate(celebrity_file_names_dict[k]):
        x.append(v)
        y.append(k)
print('total',len(x))

#Uniformiza o tamanho das imagens e as deixa P&B
for i,endereco in enumerate(x):

    gray = ImageOps.grayscale(Image.open(endereco))
    x[i] = (np.asarray(gray.resize((img_size,img_size))))

#Troca str para o int correspondente
for i,c in enumerate(y):
    y[i] = class_dict[c]
```



Pré-processamento dos dados



#Separa o treinamento do restante

```
x_train, x_rem, y_train, y_rem = train_test_split(x,y, train_size=0.7, random_state = 42)
```

#Separa o restante em validação e teste

```
x_valid, x_test, y_valid, y_test = train_test_split(x_rem,y_rem, train_size=0.5, random_state = 42)
```



One-Hot encoding



```
num_classes = 4
```


```
y_train = keras.utils.to_categorical(y_train, num_classes)  
y_test = keras.utils.to_categorical(y_test, num_classes)  
y_valid = keras.utils.to_categorical(y_valid, num_classes)
```




```
array([0., 0., 0., 1.], dtype=float32)
```



Normalização



```
x_valid = np.array(x_valid)
x_train = np.array(x_train)
x_test = np.array(x_test)
if x_train.max() == 255:
    x_valid = x_valid/255
    x_train = x_train/255
    x_test = x_test/255
```



```
array([[0.48235294, 0.48627451, 0.40392157],
       [0.12156863, 0.12156863, 0.0627451 ],
       [0.0745098 , 0.07058824, 0.01176471],
       ...,
       ...])
```



Reshaping



#Mantém a quantidade e faz reshape para ter também a quantidade de canais na ultima dimensão, sendo gray, há apenas 1

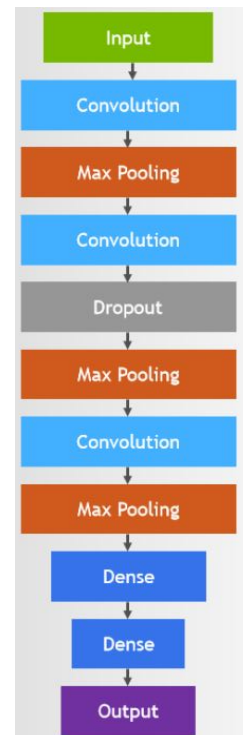
```
x_train = x_train.reshape(-1, img_size, img_size, 1)
x_valid = x_valid.reshape(-1, img_size, img_size, 1)
x_test = x_test.reshape(-1, img_size, img_size, 1)
x_train.shape, x_test.shape, x_valid.shape
```

```
>>>((235, 28, 28, 1), (51, 28, 28, 1), (51, 28, 28, 1))
```

Desenvolvimento do Modelo



```
def create_model(*args):  
    name = str(args)  
    model = Sequential()  
    model.add(Conv2D(args[0], (3, 3), strides=1, padding="same", activation="relu", input_shape=(img_size, img_size,  
3)))  
    model.add(BatchNormalization())  
    model.add(MaxPool2D((2, 2), strides=2, padding="same"))  
    model.add(Conv2D(args[1], (3, 3), strides=1, padding="same", activation="relu"))  
    model.add(Dropout(args[2]))  
    model.add(BatchNormalization())  
    model.add(MaxPool2D((2, 2), strides=2, padding="same"))  
    model.add(Conv2D(args[3], (3, 3), strides=1, padding="same", activation="relu"))  
    model.add(BatchNormalization())  
    model.add(MaxPool2D((2, 2), strides=2, padding="same"))  
  
    model.add(Flatten())  
    model.add(Dense(units=args[4], activation="relu"))  
    model.add(Dropout(args[5]))  
    model.add(Dense(units=args[4], activation="relu"))  
    model.add(Dropout(args[5]))  
    model.add(Dense(units=num_classes, activation="softmax"))  
    model.compile(loss="categorical_crossentropy", optimizer = 'adam', metrics=["accuracy"])  
    return model, name
```



Summary do Modelo

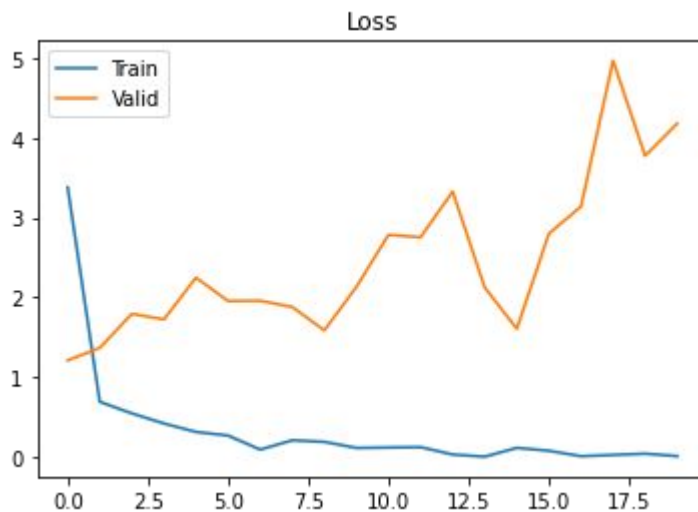
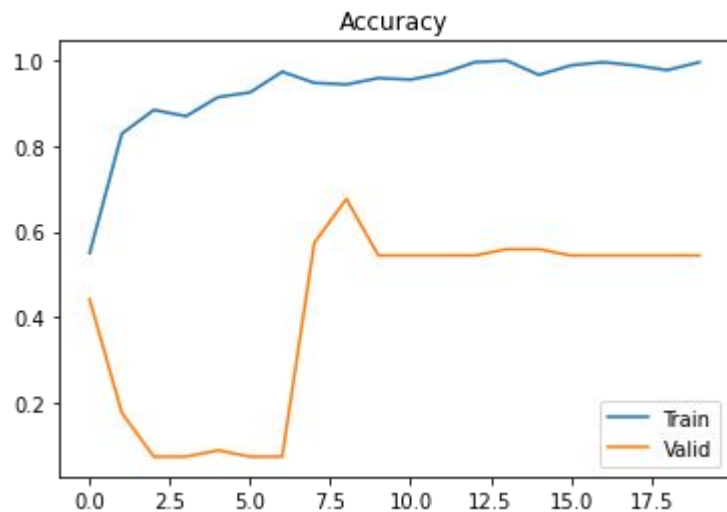


Model: "sequential_18"

Layer (type)	Output Shape	Param #
conv2d_54 (Conv2D)	(None, 28, 28, 225)	2250
batch_normalization_54 (Batch Normalization)	(None, 28, 28, 225)	900
max_pooling2d_54 (MaxPooling2D)	(None, 14, 14, 225)	0
conv2d_55 (Conv2D)	(None, 14, 14, 75)	151950
dropout_48 (Dropout)	(None, 14, 14, 75)	0
batch_normalization_55 (Batch Normalization)	(None, 14, 14, 75)	300
max_pooling2d_55 (MaxPooling2D)	(None, 7, 7, 75)	0
conv2d_56 (Conv2D)	(None, 7, 7, 25)	16900
batch_normalization_56 (Batch Normalization)	(None, 7, 7, 25)	100
max_pooling2d_56 (MaxPooling2D)	(None, 4, 4, 25)	0
flatten_18 (Flatten)	(None, 400)	0
dense_48 (Dense)	(None, 512)	205312
dropout_49 (Dropout)	(None, 512)	0
dense_49 (Dense)	(None, 4)	2052
Total params: 379,764		
Trainable params: 379,114		
Non-trainable params: 650		

Resultados

1ª Tentativa com os mesmos parâmetros do curso (Imagem 80x80) - Parâmetros (75, 50, 0.2, 25, 512, 0.2)





Data augmentation



```
datagen = ImageDataGenerator(  
    #samplewise_center=True, # set each sample mean to 0  
    rotation_range=10, # randomly rotate images in the range (degrees, 0 to 180)  
    zoom_range=0.1, # Randomly zoom image  
    width_shift_range=0.1, # randomly shift images horizontally (fraction of total width)  
    height_shift_range=0.1, # randomly shift images vertically (fraction of total height)  
    horizontal_flip=True, # randomly flip images horizontally  
    vertical_flip=False, # Don't randomly flip images vertically  
)  
  
batch_size = 32  
img_iter = datagen.flow(x_train, y_train, batch_size=batch_size)
```

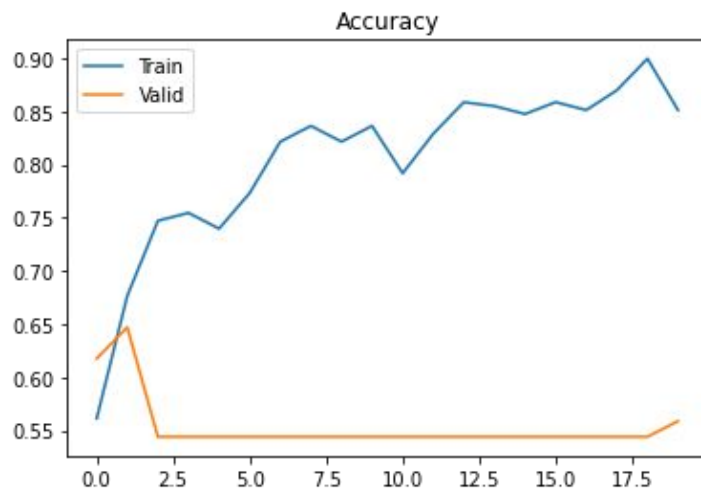


Data augmentation visualization



Resultados

2ª Tentativa com os mesmos parâmetros do curso e usando o gerador (Imagem 80x80)





Early Stop

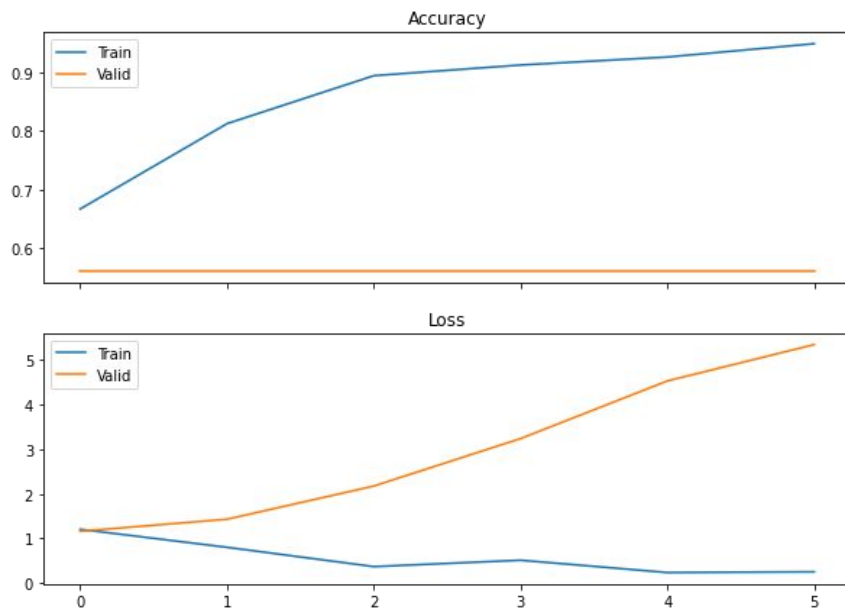


```
model, name = create_model(225, 75, 0.2, 25, 256, .2)
callback = keras.callbacks.EarlyStopping(monitor='val_accuracy', min_delta=0.001, patience=5, restore_best_weights =
True)
history = model.fit(img_iter,
                    epochs=20,
                    verbose=1,
                    steps_per_epoch=len(x_train)/batch_size, # Run same number of steps we would if we were not using a
generator.
                    validation_data=(x_valid, y_valid),
                    callbacks= [callback])

show_results(history, name)
```

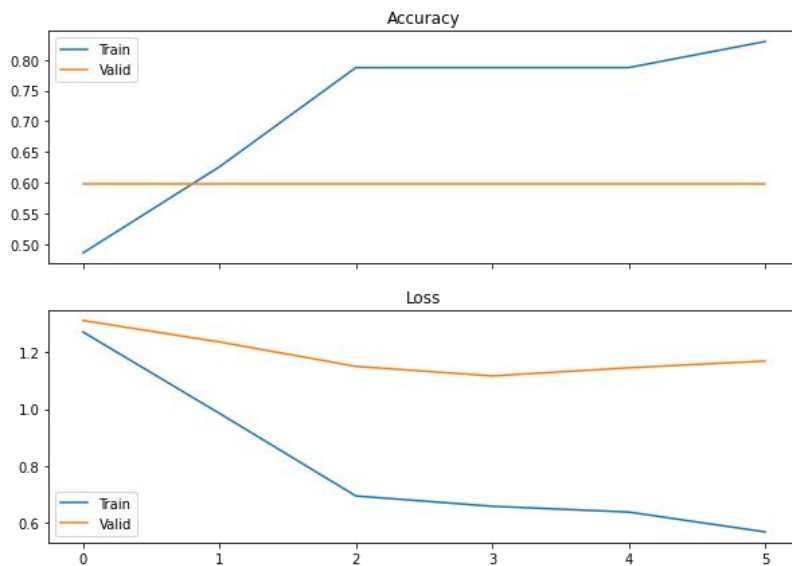
Resultados

3ª Tentativa (Imagem 100x100) - Parâmetros (225, 100, 0.2, 25, 512, 0.2); batch_size = 32

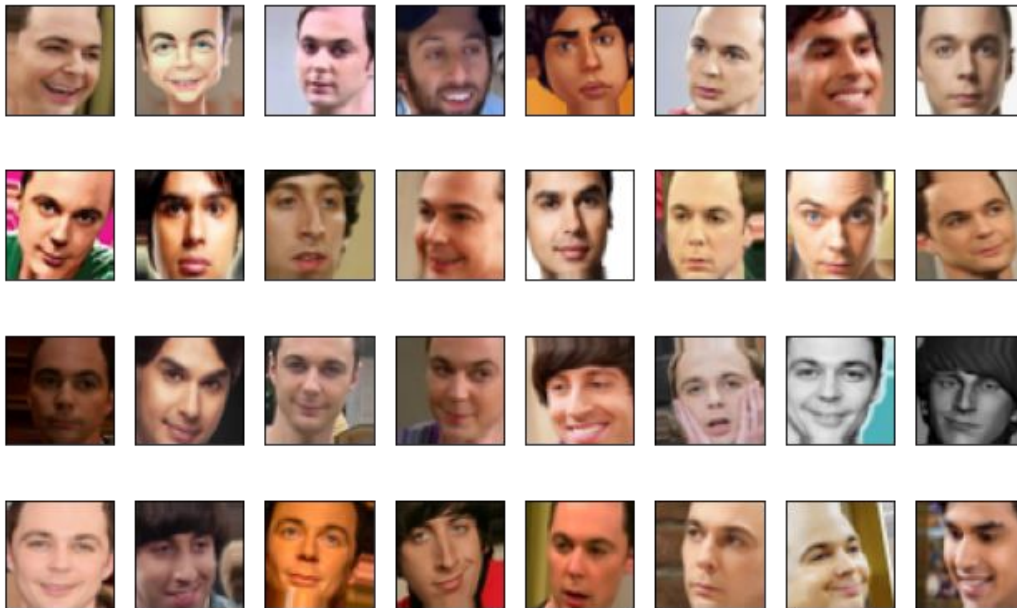


Resultados

4ª Tentativa (Imagem 28x28) - Parâmetros (80, 20, 0.4, 5, 256, 0.2); batch_size = 16

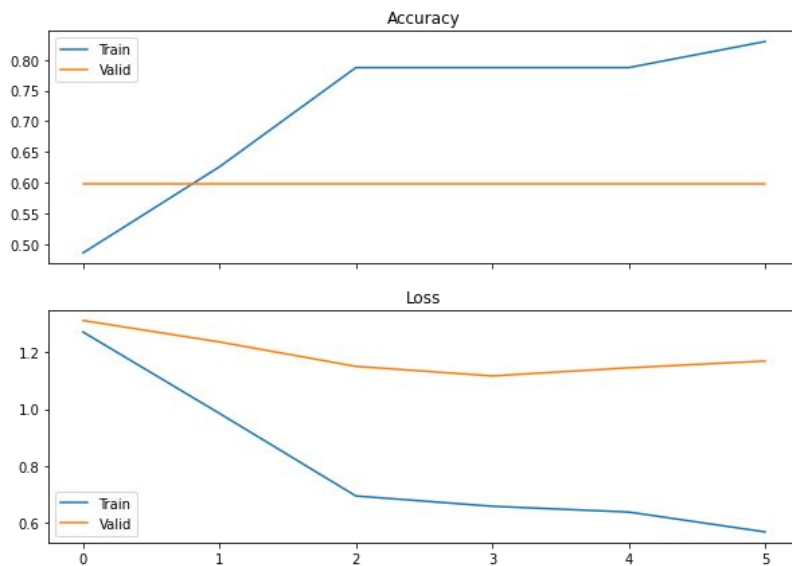


Tentativa em RGB



Resultados

5ª Tentativa (Imagem 50x50 RGB) - Parâmetros (225, 75, 0.2, 25, 512, 0.2); batch_size = 32





Salvando o Modelo



```
model.save('Last_Model')
```

```
!zip -r /content/Last_Model.zip /content/Last_Model  
files.download('Last_Model.zip')
```



```
model = keras.models.load_model('Last_Model')
```

Aplicação da Validação

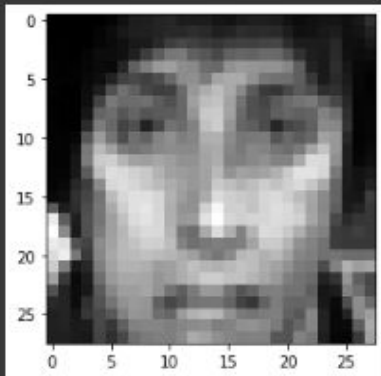
```
● ● ●  
  
#Recupera o model salvo  
model = keras.models.load_model('Last_Model')  
  
correct = 0  
for xv,yv in zip(x_valid,y_valid):  
    #Realiza o predict para cada imagem no set "valid"  
    pred=model.predict(xv.reshape(1,img_size,img_size,1))  
    print(pred)  
    #Plota a imagem  
    plt.imshow(xv.reshape(img_size,img_size),cmap='gray')  
  
    #Compara as respostas obtidas com o gabarito  
    print('resposta:',list(class_dict.keys())[np.argmax(pred)],'. Correto:',list(class_dict.keys())  
[np.argmax(yv)],'\n')  
    if list(class_dict.keys())[np.argmax(pred)]==list(class_dict.keys())[np.argmax(yv)]:  
        correct+=1  
  
    #Exibe  
    plt.show()  
    print('\n')  
#Compilado de % de acerto ao final  
print(correct/len(x_valid)*100,'% de acerto')
```

Validação

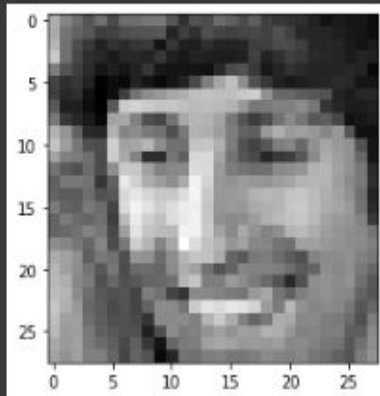
(Imagem 28x28) - Parâmetros (225, 75, 0.2, 25, 512, 0.2); batch_size = 32

54.9 % de acerto

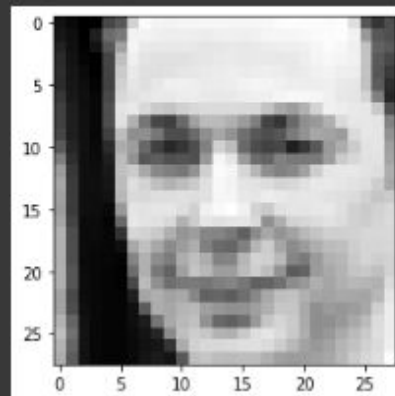
[[0.21881491 0.23600985 0.27155173 0.27362362]]
resposta: raj . Correto: howard



[[0.2245844 0.2340137 0.28635132 0.25505057]]
resposta: sheldon . Correto: howard



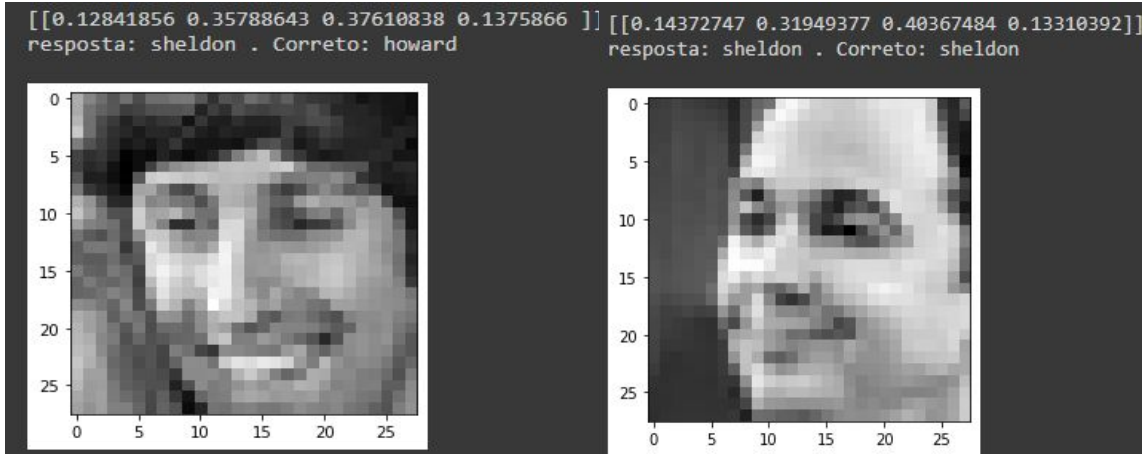
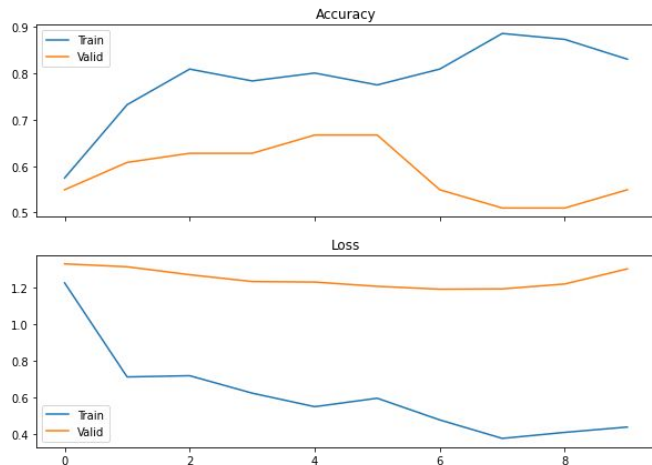
[[0.2214553 0.22505222 0.30241647 0.25107604]]
resposta: sheldon . Correto: sheldon



Resultados e Validação

6ª Tentativa (Imagem 28x28) - Parâmetros (75, 50, 0.2, 25, 512, 0.2); batch_size = 32

66.6% de acerto





Outras tentativas sem progresso em *val_accuracy*

Alterar taxas de dropout entre 0.1 e 0.5

Adicionar mais um conjunto de Conv2D, Maxpooling e BatchNormalization

Duplicar a quantidade de camadas Conv2D

Adicionar uma camada Dense

Diversas combinações de tamanhos das camadas Conv2D

Reduzir quantidade de imagens de sheldon para 60



Conclusões

- Desenvolvimento de conhecimento do funcionamento e layers de uma CNN
- Uso e boas práticas em Keras
- Metodologia para realizar pré-processamento dos dados
- Visualização geral dos dados
- Detecção de overfitting
- Métodos para evitar overfitting, como Data Augmentation e Early Stop
- Possíveis causas para o overfit: Base desbalanceada; layers insuficientes ou desconfigurados

Obrigado
