Relatório 22 - Reconhecimento de Emoções com TensorFlow 2.0 e Python

Guilherme Loan Schneider

Descrição da atividade

2: Reconhecimento de emoções em imagens

O primeiro tópico abordado é o de reconhecimento de emoções em imagens, que, a partir de um modelo já treinado com o dataset de imagens de rostos FER2013, é feito um teste de detecção de faces/emoções em um conjunto de imagens. Esse primeiro contato é realizado de forma bem manual.

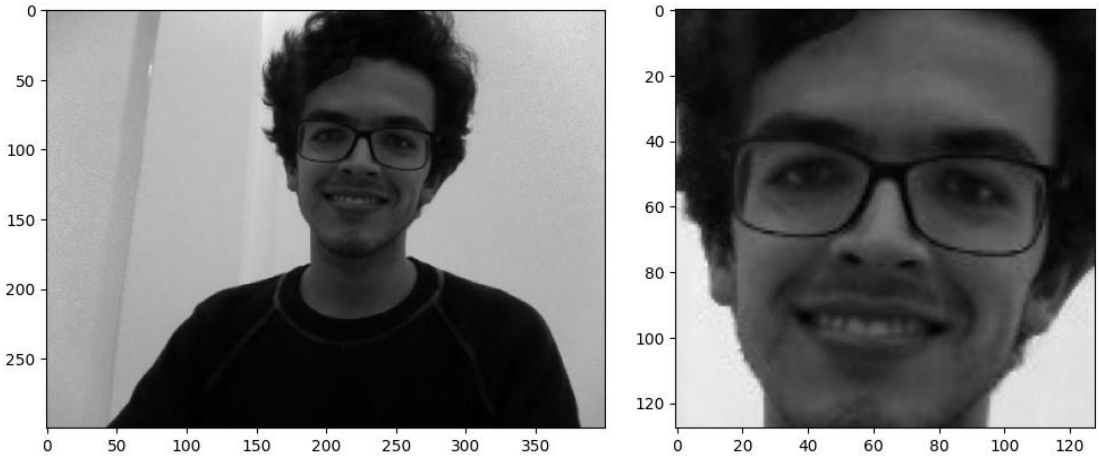
O reconhecimento de emoções realizado automaticamente pelo computador é um tópico importante para coletar dados de usuários que utilizam determinado serviço. A aplicação desse reconhecimento parte desde o reconhecimento de expressões de alunos em aulas remotas, como Google Meet, Zoom, etc, até o monitoramento do motorista em tempo real em veículos de condução semiautônoma/autônoma. O principal foco é melhorar a interação entre humanos e computadores.

Os modelos utilizados consistem em dois:

- Rede Neural Convolucional: Essa rede neural será implementada mais à frente do curso, mas ela é utilizada aqui também. Consiste em uma rede de quase seis milhões de parâmetros, com 5 blocos principais de convolução.
- Modelo de detecção de rostos: Esse modelo criado por Rainer Lienhart é um modelo pronto e treinado para reconhecer rostos. Ele é baseado no algoritmo Haar Cascade, que utiliza características como bordas, linhas e contrastes para detectar objetos em imagens, no nosso caso, rostos.

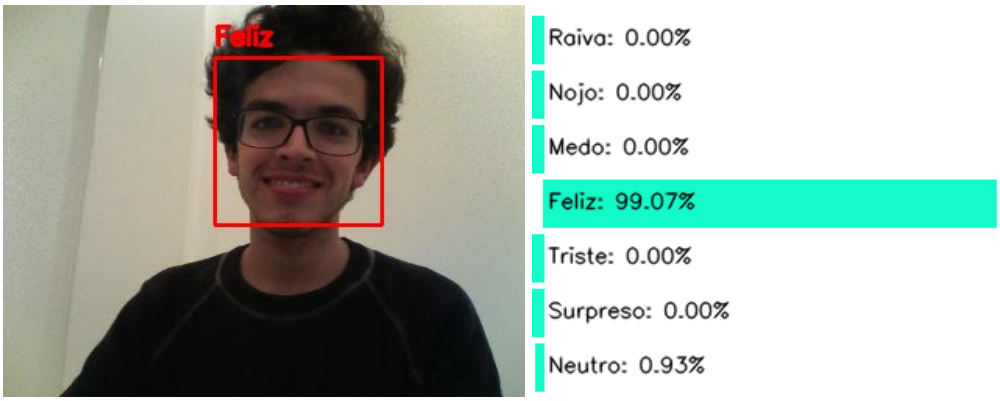
Vale ressaltar que as manipulações feitas nas imagens que estamos utilizando são realizadas pixel a pixel, tornando o trabalho pouco escalável, dado que os recortes aplicados são feitos manualmente e especificamente para a imagem em utilização.

Aplicando o modelo de detecção de rostos, obtivemos o seguinte resultado:



É possível perceber que o algoritmo desempenhou bem ao reconhecer o rosto nessa disposição. No entanto, foi notado que quando os rostos estão mais inclinados (de lado), o reconhecimento é prejudicado. Na imagem abaixo, podemos verificar essa peculiaridade.

Aplicando o modelo de reconhecimento de emoções, o resultado do algoritmo foi a label “Feliz”, com 99.07% de probabilidade.



3: Reconhecimento de Emoções com Redes Neurais Convolucionais

As redes neurais convolucionais são, de maneira simplificada, uma rede neural com algumas etapas antes de chegar no processamento tradicional de rede (neurônios de entrada, ocultos e de saída). Esse tipo é principalmente utilizado para reduzir a complexidade de imagens, tendo aplicações no DLSS (Deep Learning Super Sampling) de placas de vídeo da NVIDIA.

A primeira etapa da rede convolucional é reduzir a complexidade de uma imagem. É possível visualizar que a imagem ilustrada abaixo está no espectro RGB, possuindo 3 canais de cor, vermelho, verde e azul. Normalmente é aplicada uma redução para a escala de cinza que consegue preservar as características da imagem e reduzir a complexidade da rede como um todo, tornando a imagem com apenas um canal de cor.

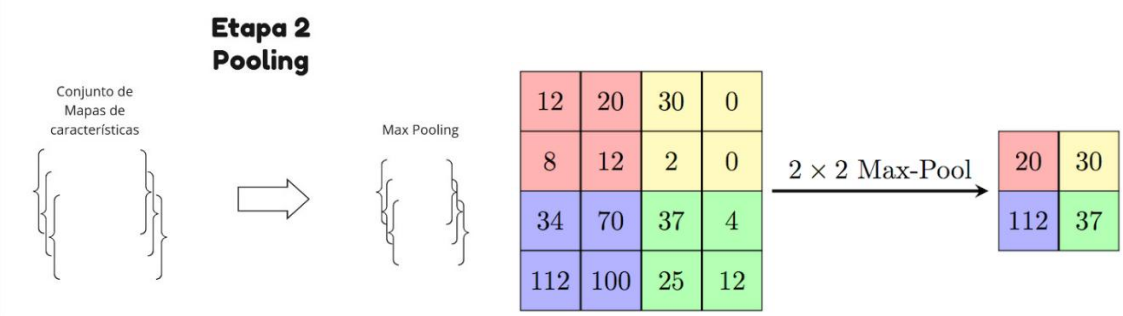
Em seguida o algoritmo define os melhores valores para o detector de características, que é uma matriz, de acordo com a imagem passada. O tamanho da matriz também e definido pelo algoritmo, variando de tamanho conforme a altura e largura da imagem passada.

É interessante destacar também que existem vários tipos de matrizes para o detector de características, como uma matriz para deixar a imagem mais nítida, para adicionar Blur, remover o Blur, identificar bordas, dentre outras (https://en.wikipedia.org/wiki/Kernel_(image_processing)).

Por fim, após aplicar o detector de características na imagem, tem-se como resultado um conjunto de mapas de características, que, dependendo a finalidade, possuem tamanho menor que a imagem original, e tentam preservar as principais diferenças de uma imagem para outra.

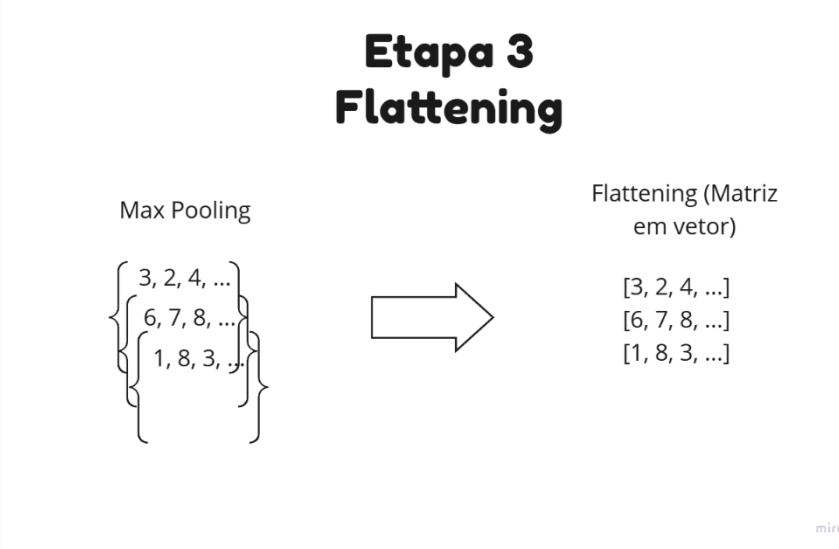
Pooling

Finalizada a etapa 1, os valores obtidos nos mapas de características são refinados mais uma vez, passando pela técnica Max Pooling (existe também o Average Pooling), que consiste em, dado uma seleção de valores μ x μ em um mapa de características, acessar o maior valor e armazená-lo em uma matriz de Pooling.

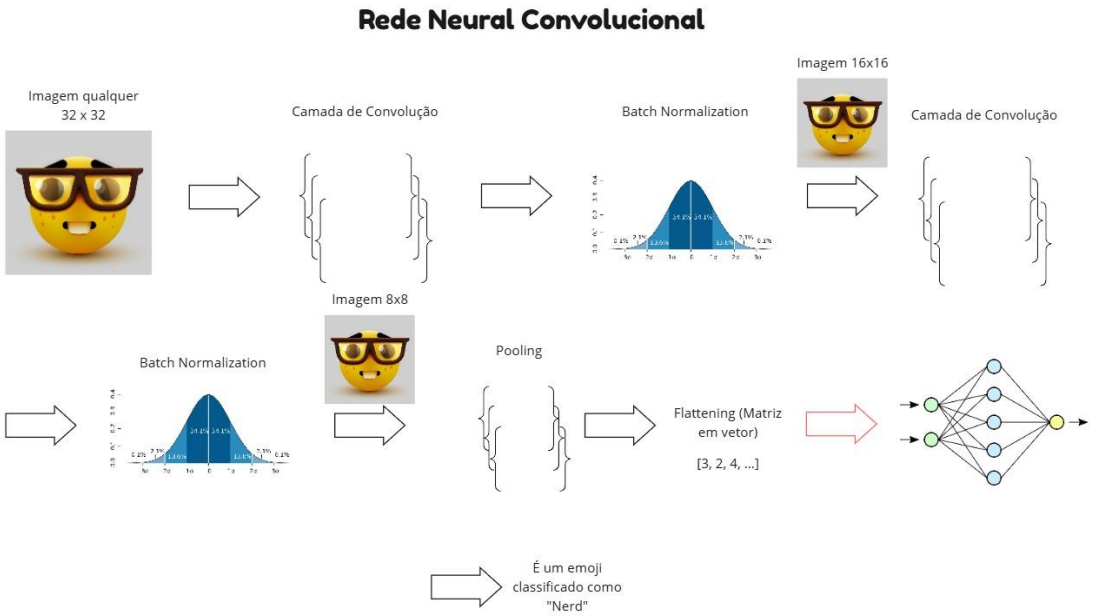


Flattening

A última etapa da rede convolucional transforma as matrizes obtidas no processo anterior em vetores. Os valores desses vetores serão utilizados na camada de entrada da rede neural.

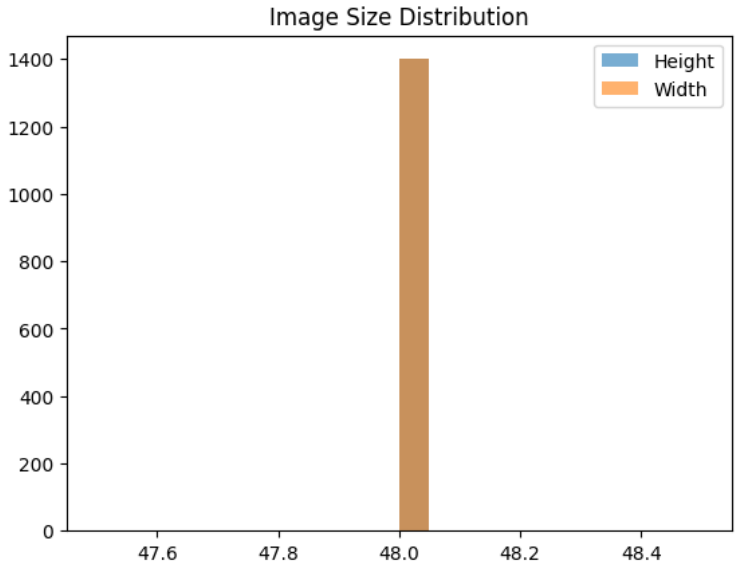


Por fim, após todas essas etapas, o vetor resultante é passado para a camada de entrada de uma rede neural (seta pintada em vermelho na figura abaixo). A partir daqui a rede neural precisa seguir as especificidades da imagem passada, como o tamanho da imagem de entrada, camada de saída de acordo com o total de saídas possíveis, dentre outros.



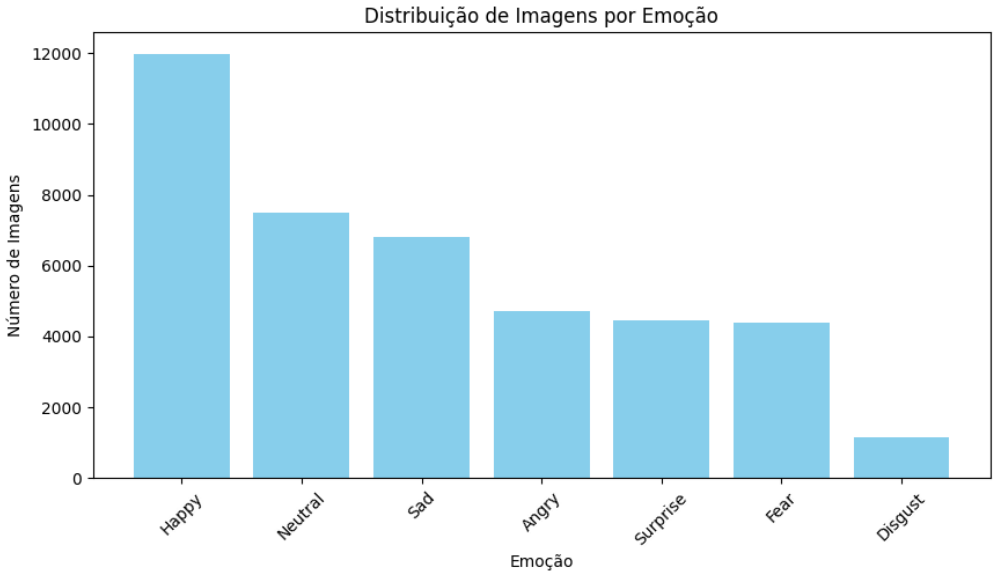
Essa seção teve a sua parte prática também, que lidou com o mesmo dataset FER2013, mas dessa vez a rede neural convolucional foi montada por nós, onde utilizamos apenas o haar cascade pronto.

O primeiro passo feito é analisar as imagens que temos, ou seja, verificar o tamanho (altura x largura) e a distribuição dessas imagens por classe. Na figura abaixo, podemos verificar que a base de dados já possui tamanho normalizado, onde todas as imagens estão no tipo 48x48 (as colunas de Height e Width estão sobrepostas).



Em seguida analisamos a quantidade de imagens presentes em cada classe (Happy, Neutral, Sad, Angry, Surprise, Fear e Disgust). Podemos verificar um grande

desbalanceamento de clases, onde a maior possui quase 12 mil imagens, e a menor possui menos de duas mil. Esse problema pode ser solucionado a partir de um data augmentation nas imagens, no entanto, essa técnica não será aplicada nessa execução.



O autor das aulas fez uma abordagem que manipulou arrays com todos os pixels de cada imagem (os pixels estavam em um arquivo csv que possuía uma coluna de nome da imagem, e outra com os valores de pixels), algo que no arquivo prático eu não fiz, por não ser algo comum de existir e ser manipulado.

```
pixels = data['pixels'].tolist()
pixels
```

Python

```
['70 80 82 72 58 58 60 63 54 58 60 48 89 115 121 119 115 110 98 91 84 84 90 99 110 126 143 153 158 171 169 172 169 165 129 110 113 107 95 79 66 62 56 57 61 52 43 4
 '151 150 147 155 148 133 111 140 170 174 182 154 153 164 173 178 185 185 189 187 186 193 194 185 183 186 180 173 166 161 147 133 172 151 114 161 161 146 131 104 9
 '231 212 156 164 174 138 161 173 182 200 106 38 39 74 138 161 164 179 190 201 210 216 220 224 222 218 216 213 217 220 220 218 217 212 174 160 162 160 139 135 137
 '24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 19 43 52 13 26 40 59 65 12 20 63 99 98 98 111 75 62 41 73 118 140 192 186 187 188 190 190 187 182 176 173 172 173 25
 '4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84 115 127 137 142 151 156 155 149 153 152 157 160 162 159 145 121 83 58 48 38 21 17 7 5 25 27 24 25 1 0 0 0 0 0 0 0
 '55 55 55 55 55 54 60 68 54 85 151 163 170 179 181 185 188 188 191 196 189 194 198 197 195 194 190 193 195 184 175 172 161 159 158 159 147 136 137 136 146 120 86
 '20 17 19 21 25 38 42 42 46 54 56 62 63 66 82 108 118 130 139 134 132 126 113 97 126 148 157 161 155 154 154 164 189 204 194 168 180 188 214 214 214 216 208 220 2
 '77 78 79 79 78 75 60 55 47 48 58 73 77 79 57 50 37 44 56 70 80 82 87 91 86 80 73 66 54 57 68 69 68 68 49 46 75 71 69 70 70 72 72 71 72 74 77 76 83 84 82 81 81 69
 '85 84 90 121 101 102 133 153 153 169 177 189 195 199 205 207 209 216 221 225 221 220 218 222 223 217 220 217 211 196 188 173 170 133 117 131 121 88 73 73 50 27 3
 '255 254 255 254 254 179 122 107 95 124 149 150 169 178 179 179 181 181 184 190 191 191 193 190 190 195 194 192 193 196 193 192 188 182 173 162 152 144 129 116 11
 '30 24 21 23 25 25 49 67 84 103 120 125 130 139 140 139 148 171 178 175 176 174 180 180 178 178 182 185 183 186 186 178 180 172 175 171 155 152 141 136 132 137 13
 '39 75 78 58 58 45 49 48 103 156 81 45 41 38 49 56 60 49 32 31 28 52 83 81 78 75 62 31 18 19 19 20 17 20 16 15 12 10 11 10 23 36 65 59 9 3 5 7 93 69 86 90 84 75 5
 '219 213 206 202 209 217 216 215 219 218 223 230 227 227 233 235 234 236 237 238 234 226 219 212 208 201 190 183 176 161 74 15 24 22 22 22 21 19 19 20 20 23 7 89
 '148 144 130 129 119 122 129 131 139 153 140 128 139 144 146 143 132 133 134 130 140 142 150 152 150 134 128 149 142 138 156 155 140 136 143 143 139 144 160 170 1
 '4 2 13 41 56 62 67 87 95 62 65 70 80 107 127 149 153 150 165 168 177 187 176 167 152 128 130 149 149 146 130 139 139 143 134 105 78 56 36 50 69 82 64 35 10 11 13
 '107 107 109 109 109 109 110 101 123 140 144 144 149 153 160 161 161 167 168 169 172 172 173 175 176 171 170 166 165 162 162 157 150 149 145 140 136 132 128 111 9
 '14 14 18 28 27 22 21 30 42 61 77 86 88 95 100 99 101 99 98 99 99 96 101 102 96 95 94 88 78 72 65 55 40 25 20 20 42 64 74 129 133 125 144 151 153 154 154 155 16 1
 '255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 2
 '134 124 167 180 197 194 203 210 204 203 209 204 206 211 211 216 219 224 228 230 230 226 222 220 217 217 210 207 213 210 199 191 190 188 177 172 148 142 90 46 81
 '219 192 179 148 208 254 192 98 121 103 145 185 83 58 114 227 225 220 203 202 168 154 157 164 182 211 164 94 122 155 176 238 240 242 192 87 43 39 60 85 34 57 82 1
 '1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 7 12 23 45 38 35 14 43 27 31 24 18 20 29 18 6 2 4 2 0 1 1 1 1 0 5 13 16 16 16 15 14 1 1 1 1 1 1 1 1 1 1 1 1 0 0 3 7 9 27 44 43 42
 '174 51 37 37 38 41 22 25 22 24 35 51 70 83 98 113 119 127 136 149 149 141 125 107 77 50 30 21 9 38 96 79 72 87 60 23 25 43 29 24 33 51 36 33 26 20 136 255 107 46
 '123 125 124 142 209 226 234 236 231 232 235 223 211 196 184 181 182 186 185 193 208 211 208 201 196 192 191 192 194 201 207 216 225 225 223 220 219 220 172 124 1
 '8 9 14 21 26 32 37 46 52 62 72 70 71 73 76 83 98 92 80 90 110 148 158 149 166 172 166 166 164 179 196 197 189 185 194 189 174 173 177 179 186 197 192 191 185 176
 '252 250 246 229 182 140 98 72 53 44 67 95 95 89 89 90 90 93 94 89 88 88 83 82 81 73 74 69 69 64 58 49 46 49 35 32 27 25 24 21 13 9 11 15 15 15 21 24 254 246 221
...
 '130 131 132 132 133 134 134 134 136 138 136 143 114 28 57 86 103 107 107 110 108 102 95 81 74 75 68 64 60 54 45 42 35 32 33 32 29 27 27 26 15 8 4 5 6 4 2 3 128 1
 '30 25 19 5 5 4 3 1 1 0 2 26 31 15 10 4 3 3 5 6 8 14 13 14 11 10 22 24 18 35 42 46 43 27 25 18 14 21 25 15 15 11 1 1 1 0 0 0 34 27 23 14 5 6 5 2 0 8 45 44 19 14 6
 '177 203 157 122 156 178 194 202 199 191 193 171 178 177 171 178 192 182 166 202 216 212 203 165 164 177 161 159 156 144 153 169 171 162 177 221 225 224 229 229 2
 '22 24 23 25 23 21 21 23 21 23 22 26 99 115 121 152 190 208 216 217 215 216 210 200 195 179 160 137 121 112 87 69 75 88 102 111 110 111 102 97 93 85 74 69 64 56 4
 ...]
```

Em seguida uma normalização foi aplicada nas imagens.

```
def normalizar(x):
    x = x.astype('float32')
    x = x / 255.0
    return x
```

Agora partimos para a etapa de divisão do dataset em conjuntos de treino, teste e validação. A função train_test_split pertence a biblioteca do sklearn, não fizemos a implementação dessa função. Analisamos também o tamanho de cada dataset montado.

Ao fim salvamos as divisões em um arquivo .npy para caso precisemos carregar esse modelo em outro lugar.

```
# Extrai caminhos e rótulos do dataset
X = [item[0] for item in emotion_dataset]
y = [item[1] for item in emotion_dataset]

# Divide em treino, teste e validação
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42, stratify=y)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.1, random_state=41, stratify=y_train)
```

```
print('Número de imagens no conjunto de treinamento:', len(X_train))
print('Número de imagens no conjunto de teste:', len(X_test))
print('Número de imagens no conjunto de validação:', len(X_val))
```

```
Número de imagens no conjunto de treinamento: 33193
Número de imagens no conjunto de teste: 4098
Número de imagens no conjunto de validação: 3689
```

```
np.save('mod_xtest', X_test)
np.save('mod_ytest', y_test)
```

Temos agora a arquitetura do modelo (CNN), essa arquitetura foi baseada em uma implementação já feita <https://medium.com/@birdortyedi_23820/deep-learning-lab-episode-3-fer2013-c38f2e052280>.

Temos quatro blocos principais de convolução, com batch normalization em todas menos a primeira camada. Além disso, utiliza-se regularização L2 (ajuda a evitar o overfitting). Nessa implementação, o autor utilizou o MaxPooling, com stride 2x2. Temos também a aplicação do dropout, com valor alto, ao meu ver, de 0.5. Essa arquitetura totaliza quase seis milhões de parâmetros, 5.905.863.

Na compilação do modelo, utilizamos a loss sendo o categorical cross entropy, que é utilizado para problemas multiclasse, com otimizador Adam configurado com learning rate de 1e-2, beta_1 representa o momentum, beta_2 representa a média móvel dos gradientes e por fim o épsilon, que nada mais é que um valor para evitar divisão por 0.

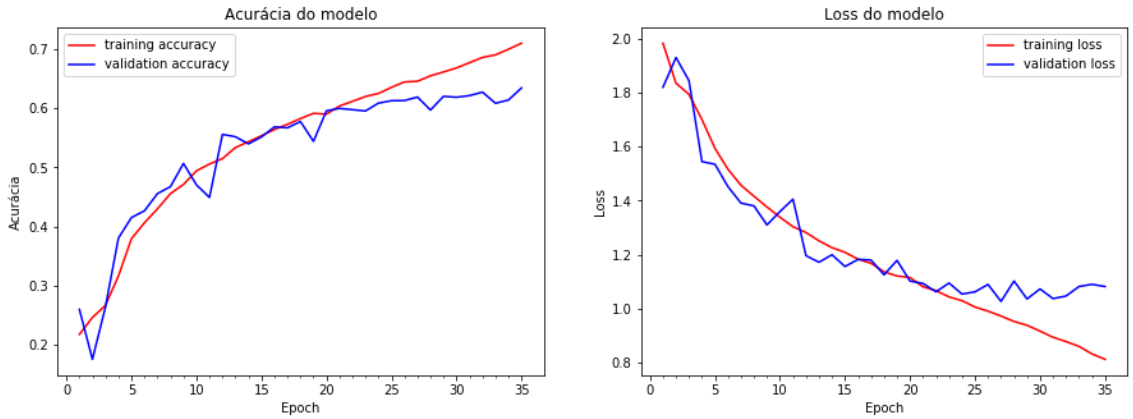
Utilizamos a métrica de acurácia, que nesse caso pode não ser muito interessante por possuir classes desbalanceadas. Além disso, utilizamos três funções adicionais, a ReduceLROnPlateau, que reduz o learning rate quando o modelo para de melhorar, o EarlyStopping determina quando o modelo irá parar após ficar X épocas sem melhorar (nesse caso representado pelo argumento patience). O ModelCheckpoint irá salvar o modelo durante as épocas, mas apenas se houver melhora nas métricas.

```
model.compile(loss = 'categorical_crossentropy',
              optimizer = Adam(learning_rate = 0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-7),
              metrics = ['accuracy'])

arquivo_modelo = 'modelo_01_expressoes.h5'
arquivo_modelo_json = 'modelo_01_expressoes.json'

lr_reducer = ReduceLROnPlateau(monitor='val_loss', factor = 0.9, patience=3, verbose = 1)
early_stopper = EarlyStopping(monitor='val_loss', min_delta=0, patience = 8, verbose = 1, mode = 'auto')
checkpointer = ModelCheckpoint(arquivo_modelo, monitor='val_loss', verbose = 1, save_best_only=True)
```

Em seguida o treinamento é feito, com 100 épocas. A figura abaixo demonstra o desempenho do modelo durante as épocas. É interessante ressaltar o número de épocas executadas, que totalizam 35 (early stopping).

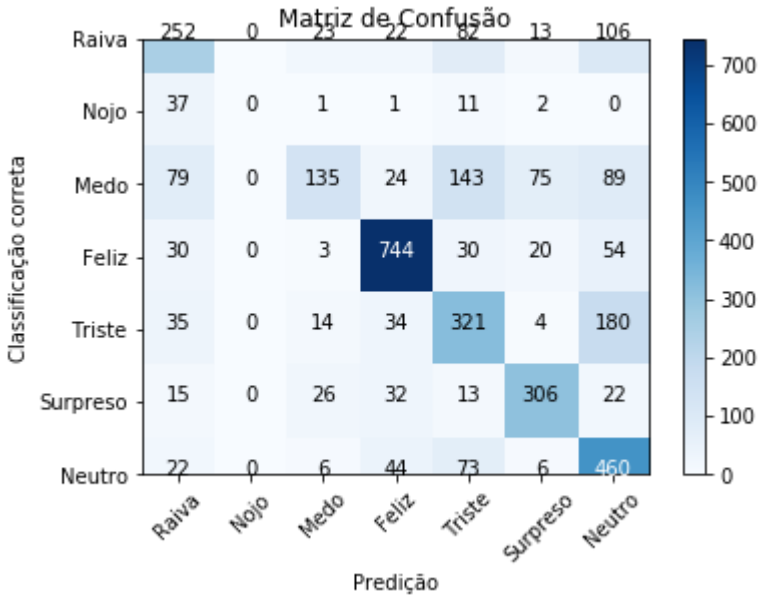


Analisando o desempenho do algoritmo, podemos verificar que ele atingiu acurácia de 61.79%, com erro de 1.1469.

```
print('Acurácia: ' + str(scores[1]))
print('Erro: ' + str(scores[0]))
```

```
Acurácia: 0.61799943
Erro: 1.1469147179611017
```

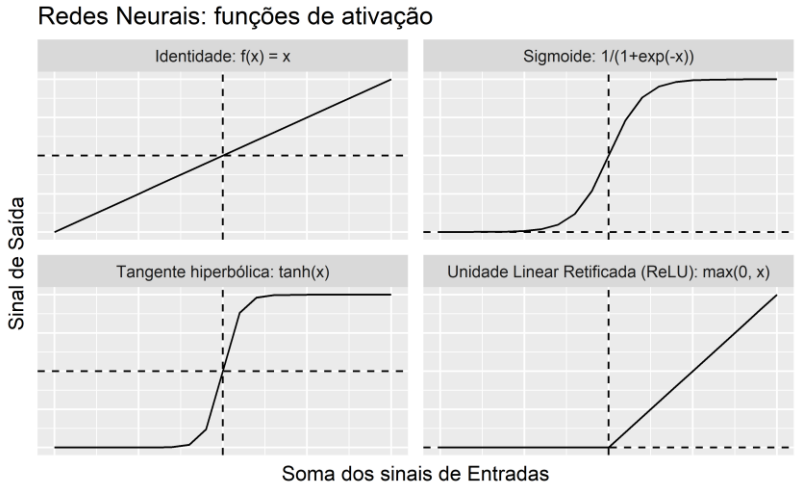
A matriz de confusão é dada na imagem abaixo. Verifica-se que temos resultados satisfatórios em algumas classes, mas em classes menores como a “Nojo”, o desempenho foi péssimo. Algumas dessas variações podem ser a dificuldade de interpretar as expressões, visto que ele confundiu bastante com “Raiva” e “Triste”. Além disso, é interessante considerar a troca para uma outra métrica, como F1 Score.



4: Anexo 1 - Redes Neurais Artificiais

A seção começa com o autor comentando sobre as redes perceptron de uma camada, que são utilizados para classificação de operações lógicas, como o bit AND, onde só há o caso de o bit ser 1 no momento em que ambos os bits são verdadeiros, permitindo traçar uma única linha reta em um gráfico 2D para separar o ponto (1,1) dos outros três.

Em seguida aborda o assunto das redes neurais multicamada, que são utilizadas para problemas de classificação binária e multiclasses. Além disso, comenta também sobre as funções de ativação que podem ser utilizadas, elas estão representadas na figura abaixo.



Existem vários conjuntos de técnicas para tentar reduzir o valor do erro, abaixo estão algumas que foram utilizadas no desenvolvimento do curso:

Ajuste dos pesos: Utiliza o backpropagation para realizar os ajustes nos pesos da rede neural.

Cálculo do Erro: A rede realiza uma previsão, compara com o valor real e calcula o erro usando uma função de custo (ex: erro quadrático médio ou entropia cruzada).

Descida do Gradiente: Utiliza o gradiente da função de custo em relação aos pesos para determinar a direção e a intensidade da atualização necessária para minimizar o erro.

Descida do Gradiente Estocástico (SGD - Stochastic Gradient Descent): Em vez de calcular o gradiente com todos os dados de treinamento (descida do gradiente batch), o SGD atualiza os pesos com base em um pequeno subconjunto (batch) aleatório, tornando o treinamento mais rápido e eficiente.

Cálculo do Parâmetro Delta: O delta representa a correção necessária para os pesos com base no erro propagado pela retropropagação (backpropagation). Ele é calculado a partir da derivada da função de ativação multiplicada pelo gradiente do erro.

Referencias

Reconhecimento de Emoções com TensorFlow 2.0 e Python (Seção 1 a 4)

Keras 3 API documentation