**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.

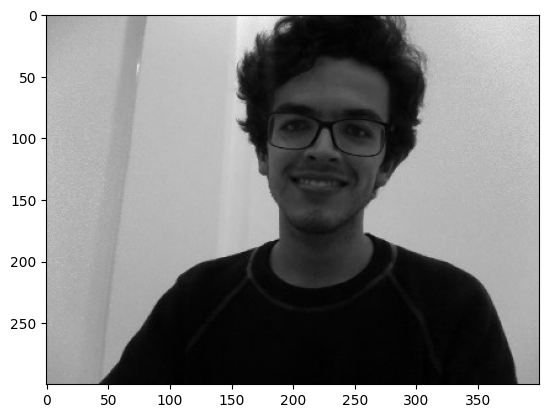
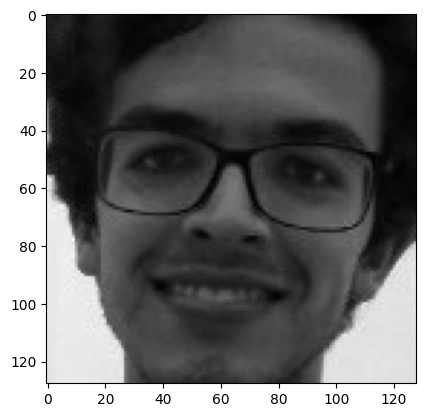
O reconhecimento de emoções realizado automaticamente pelo computador é um tópico importante para coletar dados de usuários que utilizam determinado serviço. A aplicação desse reconhecimento parte desde o reconhecimento de expressões de alunos em aulas remotas, como Google Meet, Zoom, etc, até o monitoramento 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.

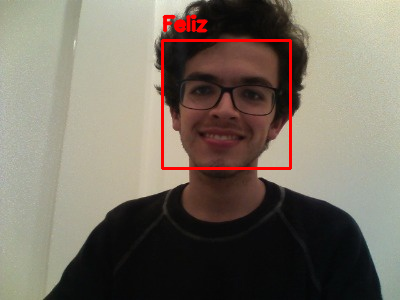
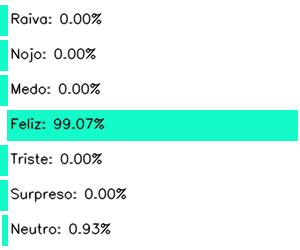
Aplicando o modelo de detecção de rostos, obtivemos o seguinte resultado:

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.



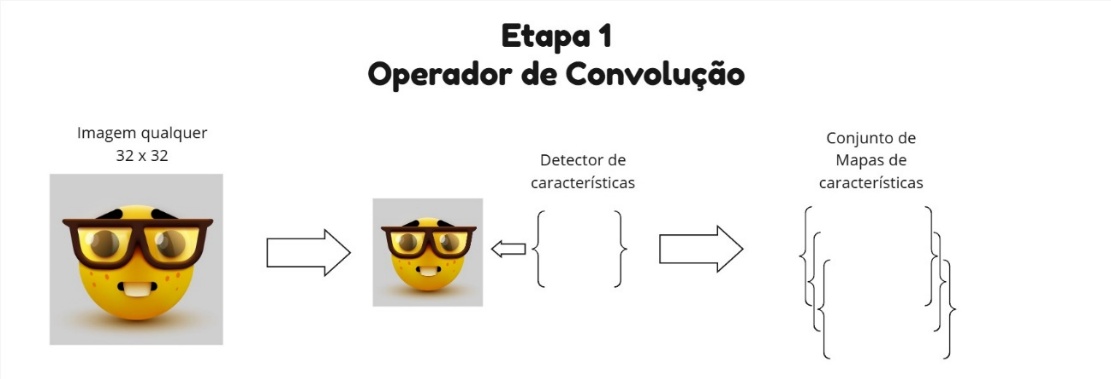
Aplicando o modelo de reconhecimento de emoções, o resultado do algoritmo foi a label “Feliz”, com 99.07% de probabilidade.

**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.



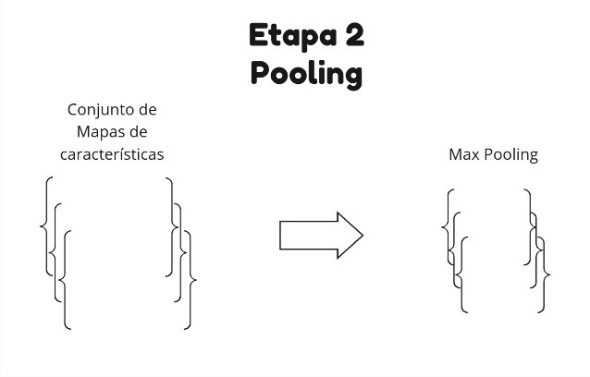
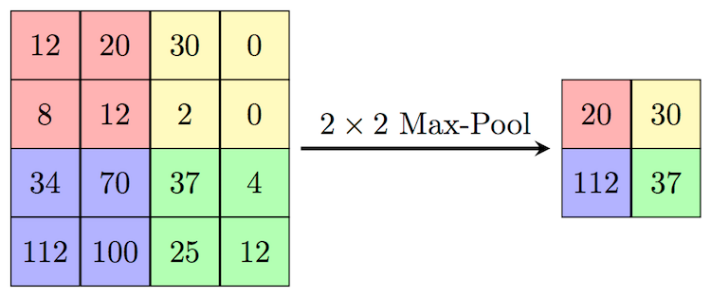
Em seguida o algoritmo define os melhores valores para o detector de características, que é uma matriz, de acordo com a imagem passada. O tamanho da matriz também e definido pelo algoritmo, variando de tamanho conforme a altura e largura da imagem passada.

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.

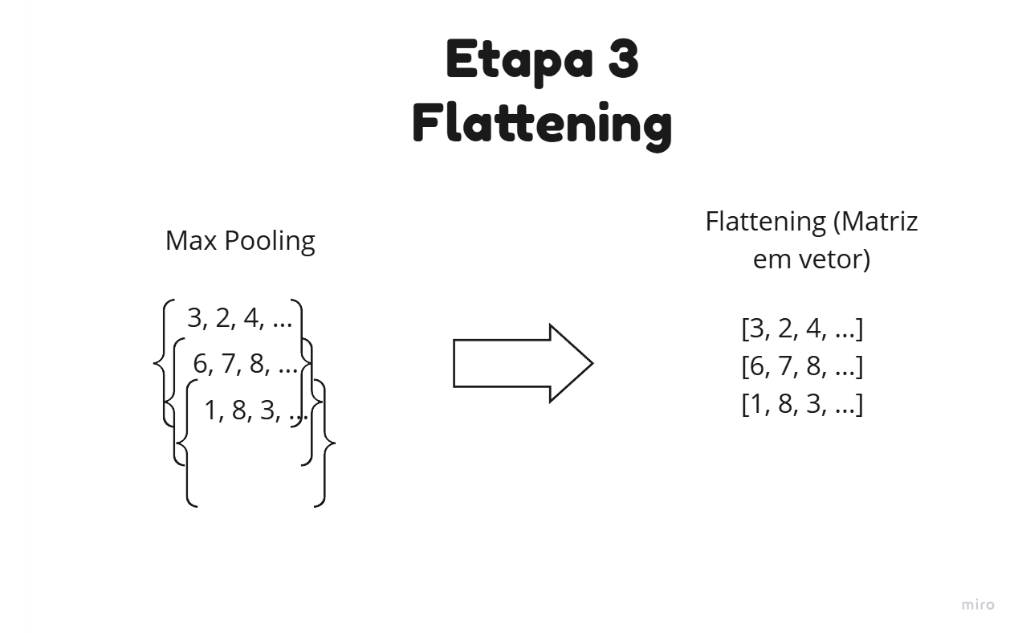
Pooling

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.

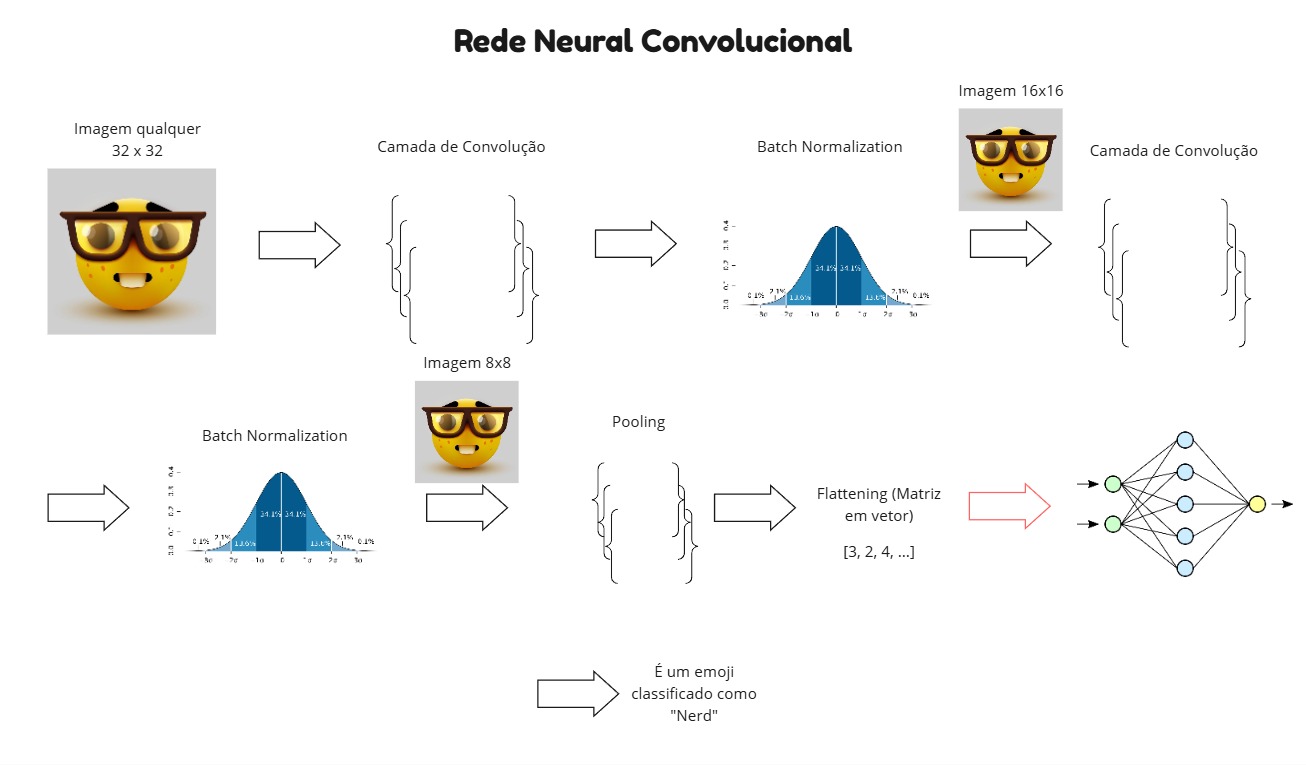
 

Flattening

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.

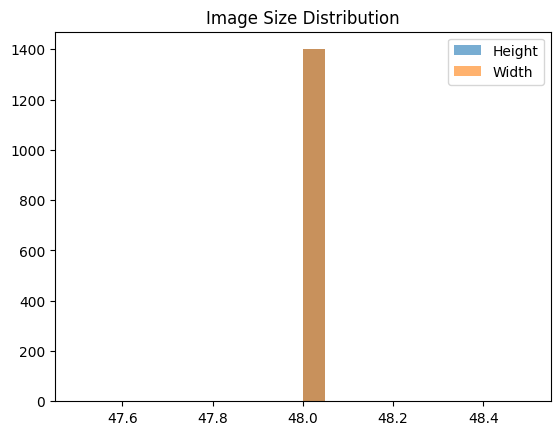


Por fim, após todas essas etapas, o vetor resultante é passado para a camada de entrada de uma rede neural (seta pintada em vermelho na figura abaixo). A partir daqui a rede neural precisa seguir as especificidades da imagem passada, como o tamanho da imagem de entrada, camada de saída de acordo com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

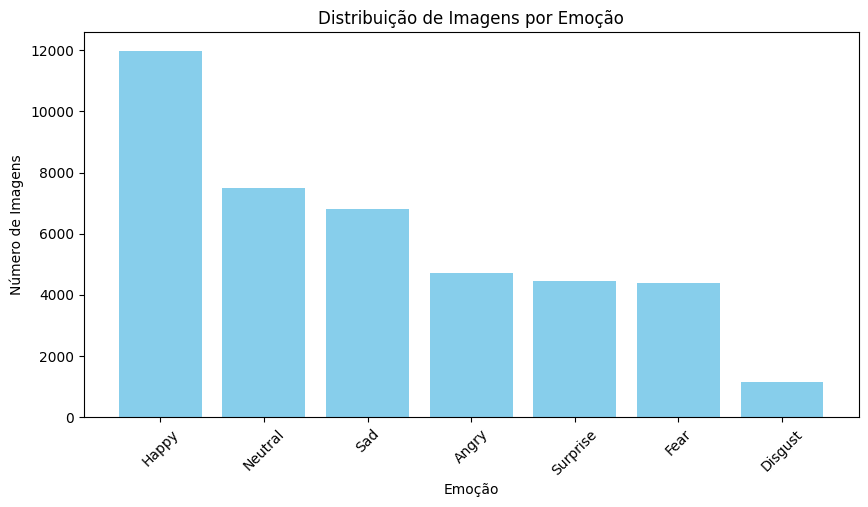


Essa seção teve a sua parte prática também, que lidou com o mesmo dataset FER2013, mas dessa vez a rede neural convolucional foi montada por nós, onde utilizamos apenas o haar cascade pronto.

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).



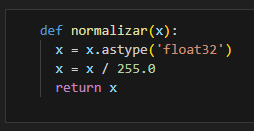
Em seguida analisamos a quantidade de imagens presentes em cada classe (Happy, Neutral, Sad, Angry, Surprise, Fear e Disgust). Podemos verificar um grande desbalanceamento de clases, onde a maior possui quase 12 mil imagens, e a menor possui menos de duas mil. Esse problema pode ser solucionado a partir 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.

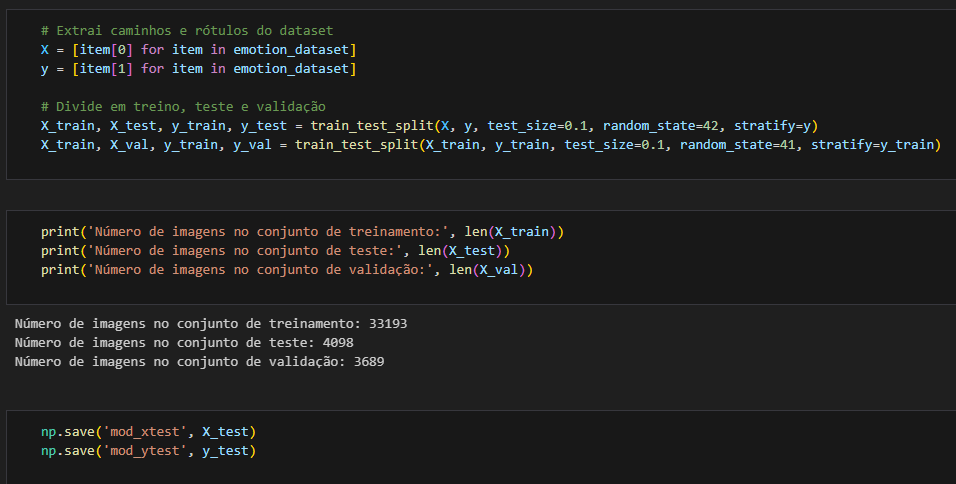


Em seguida uma normalização foi aplicada nas imagens.



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.

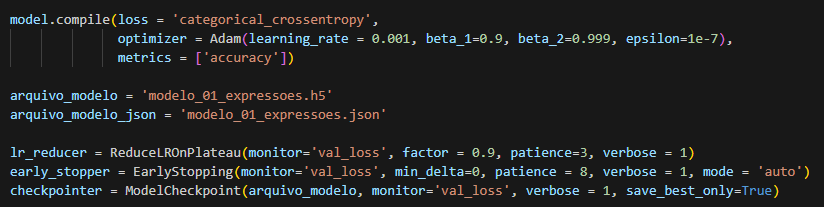


Temos agora a arquitetura do modelo (CNN), essa arquitetura foi baseada em uma implementação já feita <https://medium.com/@birdortyedi\_23820/deep-learning-lab-episode-3-fer2013-c38f2e052280>.

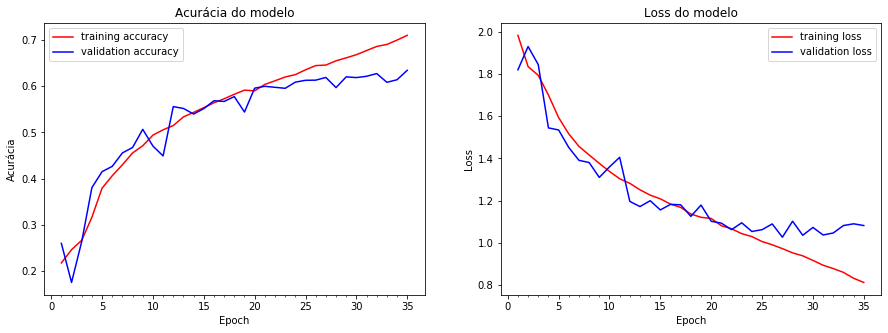
Temos quatro blocos principais de convolução, com batch normalization em todas menos a primeira camada. Além disso, utiliza-se regularização L2 (ajuda a evitar o overfitting). Nessa implementação, o autor utilizou o MaxPooling, com stride 2x2. Temos também a aplicação do dropout, com valor alto, ao meu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

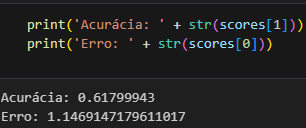
Utilizamos a métrica de acurácia, que nesse caso pode não ser muito interessante por possuir classes desbalanceadas. Além disso, utilizamos três funções adicionais, a ReduceLROnPlateau, que reduz o learning rate quando o modelo para de melhorar, o EarlyStopping determina quando o modelo irá parar após 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.



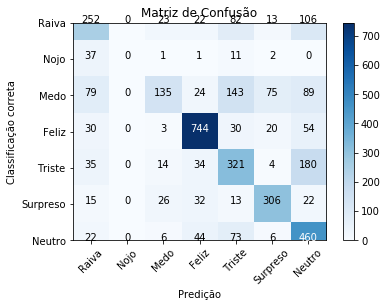
Em seguida o treinamento é feito, com 100 épocas. A figura abaixo demonstra o desempenho do modelo durante as épocas. É interessante ressaltar o número de épocas executadas, que totalizam 35 (early stopping).



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



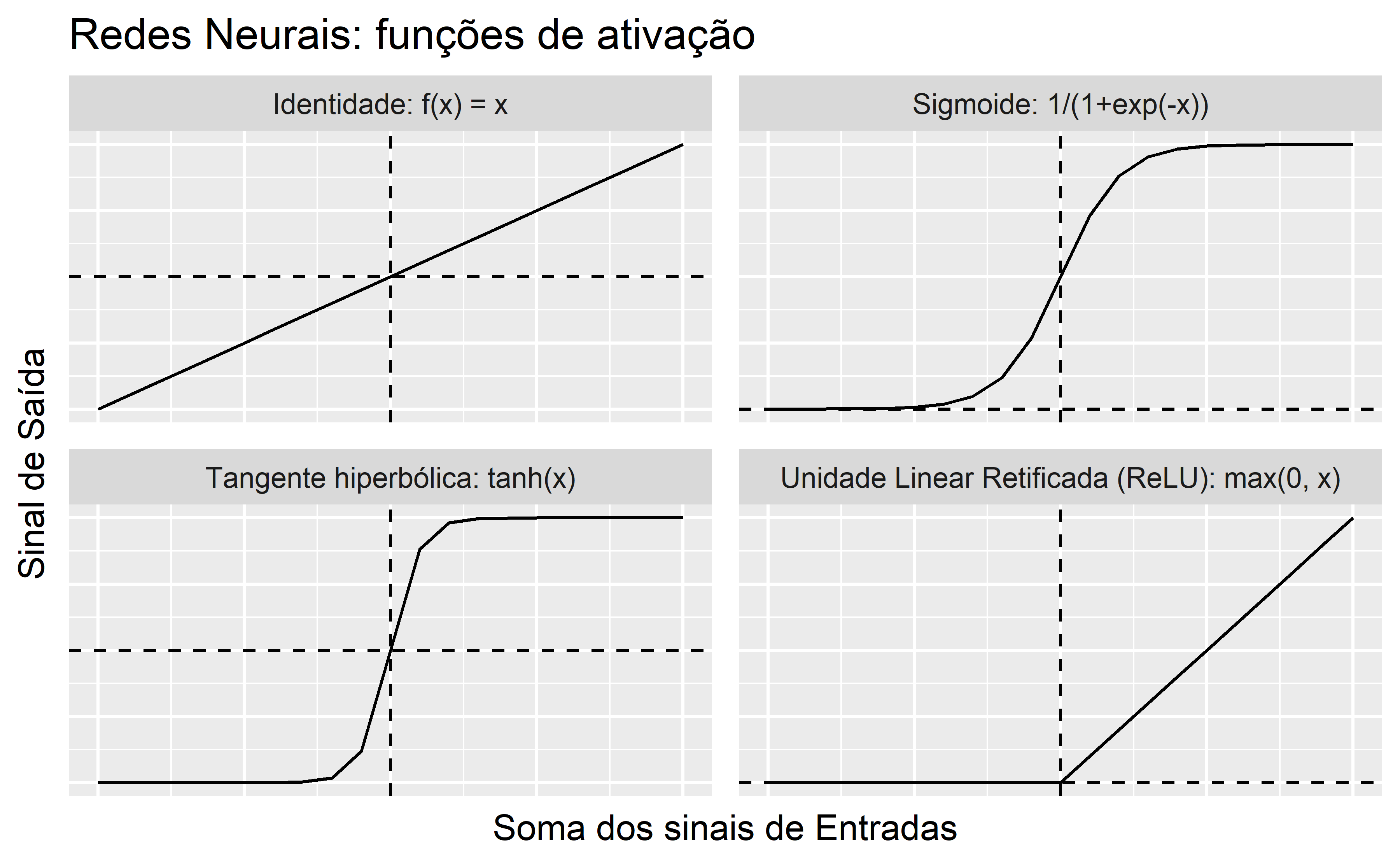
A matriz de confusão é dada na imagem abaixo. Verifica-se que temos resultados satisfatórios em algumas classes, mas em classes menores como a “Nojo”, o desempenho foi péssimo. Algumas dessas variações podem ser a dificuldade de interpretar as expressões, visto que ele confundiu bastante com “Raiva” 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)](https://docs.google.com/document/d/1rAvJ0FiWZG5fgecj4VhHqDJcgQWh_O5CrmZCw812GGU/edit?tab=t.0#heading=h.qai1l4sen9sb)

[Keras 3 API documentation](https://keras.io/api/)