# Eduardo Oliveira Coelho

**Repositório GitHub: https://github.com/eduardoocoelho/ai-lists**

1. **Definindo a Arquitetura da Rede Neural**
    a. Inicialização de uma instância da classe **"Sequential"** para criar uma pilha linear de camadas.
    b. Adição da primeira camada de convolução **(Conv2D)** com ativação ReLU e especificação da forma de entrada.
    c. Adição de uma camada de pooling **(MaxPooling2D)**.
    d. Adição da segunda camada de convolução com ativação ReLU e outra camada de pooling.
    e. Flattening da saída para transformar os mapas de características em um vetor.
    f. Adição de duas camadas totalmente conectadas **(`Dense`)** com ativação ReLU e sigmoid (para a camada de saída).

```
# Inicializando a Rede Neural Convolucional
classifier = Sequential()

# Passo 1 - Primeira Camada de Convolução
classifier.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))

# Passo 2 - Pooling
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Adicionando a Segunda Camada de Convolução
classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Passo 3 - Flattening
classifier.add(Flatten())

# Passo 4 - Full connection
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = 1, activation = 'sigmoid'))

# Compilando a rede
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

2. **Data Augmentation e Configuração dos Conjuntos de Treinamento e Validação:**
    a. Configuração de geradores de dados **(ImageDataGenerator)** para realizar data augmentation.

b. Criação de instâncias de **flow_from_directory** para os conjuntos de treinamento e validação, especificando o tamanho da imagem, o tamanho do lote e a classe de modo binário.

Criando os objetos train_datagen e validation_datagen com as regras de pré-processamento das imagens

```
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

validation_datagen = ImageDataGenerator(rescale = 1./255)
```

Pré-processamento das imagens de treino e validação

```
training_set = train_datagen.flow_from_directory('training_set',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'binary')

validation_set = validation_datagen.flow_from_directory('test_set',
                                                        target_size = (64, 64),
                                                        batch_size = 32,
                                                        class_mode = 'binary')
```

3. **Criação de um Gerador Personalizado:**
   a. Criação de uma função **custom_generator** que utiliza um loop infinito **(while True)** para repetir o conjunto de treinamento indefinidamente.
   b. A função **next(generator)** é usada para obter o próximo lote do gerador original.

```
def custom_generator(generator):
    while True:
        data = next(generator)
        yield data

repeated_train = custom_generator(training_set)
```

4. **Treinamento do Modelo:**
   a. Compilação do modelo usando o otimizador **Adam**, a função de perda binária e a métrica de acurácia.
   b. Uso do método **fit** para treinar o modelo. O gerador personalizado é utilizado como entrada, e o número de etapas por época e épocas são especificados.

Executando o treinamento (esse processo pode levar bastante tempo, dependendo do seu computador)

```
classifier.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
                   loss='binary_crossentropy', metrics=['accuracy'])

classifier.fit(repeated_train,
                         steps_per_epoch = 100,
                         epochs = 5,
                         validation_data = validation_set,
                         validation_steps = 25)
```

5. **Avaliação do Modelo e Predições:**
    a. Carregamento de imagens de teste, pré-processamento e predição usando o modelo treinado.
    b. Comparação da predição com o rótulo verdadeiro para determinar se é "Homer" ou "Bart".

```
path = 'test_set/bart/bart17.bmp'
img = PilImage.open(path)
img.save(path.replace('.bmp', '.png'))

test_image = image.load_img(path.replace('.bmp', '.png'), target_size = (64, 64))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = classifier.predict(test_image)
training_set.class_indices

if result[0][0] == 1:
    prediction = 'Homer'
else:
    prediction = 'Bart'

prediction
```

6. **Avaliação do Desempenho do Modelo no Conjunto de Validação:**
    a. Uso do método **evaluate** para calcular a perda e a acurácia do modelo no conjunto de validação.

Avaliar o modelo

```
score = classifier.evaluate(validation_set, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```