Importando as bibliotecas necessárias

```
#TensorFlow + Keras
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import *

#Auxiliares
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import cv2
```

Importando a base de dados

Foi utilizada um dataset de peças de xadrez, que contém aproximadamente 550 imagens, esta base é divida em subpastas com o devido nome das classes(Bispo,Cavalo,Peao,Rainha,Rei,Torre) respectivamente.

Disponível em: https://drive.google.com/drive/folders/1rgrVLOh7yE7-38uo7InyDS0HnL942T1Y?usp=sharing

```
#Importando o dataset e direcionando para o treinamento
train = keras.preprocessing.image_dataset_from_directory(
    '/content/drive/MyDrive/Chessman-image-dataset/Xadrez',
#Parâmetros padrões para definir as configurações dos dados e direcionar em um conjunto de
    validation_split=.2,
    subset='training',
    seed=42,
    image_size=(48, 48),
    batch_size=32,
    label_mode='categorical'
)
```

```
Found 552 files belonging to 6 classes.
Using 442 files for training.
```

```
#Importando o dataset e direcionando para a validação
validation = keras.preprocessing.image_dataset_from_directory(
    '/content/drive/MyDrive/Chessman-image-dataset/Xadrez',
    #Parâmetros padrões para definir as configurações dos dados
    #(como o tamanho ou batch_size que é o tamanho do conjunto(número de amostras)) e
    #direcionar em um conjunto de treinamento
    validation_split=.2,
    subset='validation',
    seed=42,
    image_size=(48, 48),
    batch_size=32
```

```
    batch_size=32,
    label_mode='categorical'
)
```

```
Found 552 files belonging to 6 classes.
Using 110 files for validation.
```

```
#Dataset disponível em: https://drive.google.com/drive/folders/1rgrVLOh7yE7-38uo7InyDS0HnL
#Está no drive, pois o tensor flow é conectado neste meio, é só montar a pasta do drive
#Ele reconhecerá todas as pastas, basta selecionar e usar '/content/drive/Mydrive/nome_dat
```

Criando o modelo

Montando as camadas

```
#Montando as camadas
image_size = (48, 48)
#Definindo o tamanho do lote(dado) de treinamento, os tensores terão a mesma dimensão(3)
input_shape = (image_size[0], image_size[1], 3)
#Definindo o modelo para camadas simples(sequenciais)
model = keras.Sequential([
    #Cria um kernel de convolução que é convolvido com a entrada da camada para produzir u
    #"Resumindo" as dimensões, definindo com espaço zero e informando as funções de ativaç
    Conv2D(64, (3,3), input_shape=image_size + (3,), padding='same', activation='relu'),
    Conv2D(128, (3,3), padding='same', activation='relu'),
    #Reduz a resolução
    MaxPooling2D(pool_size=(2,2)),
    #Achata a entrada(reformata os dados)
    Flatten(),
    #Conecta a camada e passa a função de ativação
    Dense(256, activation='relu'),
    #Define a fração das unidades de entrada(flutua entre 0-1)
    Dropout(0.5),
    #Dimensionando a saída(espaço)
    Dense(6, activation='softmax')
])
```

```
#Classes das imagens
train.class_names
```

```
['Bispo', 'Cavalo', 'Peao', 'Rainha', 'Rei', 'Torre']
```

Compilando o modelo

```
#Compilando o modelo
model.compile(
    #Definindo como o modelo se atualiza mediante a função loss
    optimizer='adam',
    #Medição da precisão do modelo
    loss='categorical_crossentropy',
```

```
#Monitoraçao dos dados de treinamento, a acc usa uma traçao das imagens que foram prev
metrics=['accuracy'])
```

Treinando o modelo

```
#Alimentando com os dados, para que a IA possa aprender e associar cada imagem com sua res
model.fit(train,epochs=50,validation_data=validation)
```

```
14/14 [==============================] - 17s 1s/step - loss: 0.2425 - accuracy: 0.
Epoch 23/50
14/14 [==============================] - 17s 1s/step - loss: 0.2162 - accuracy: 0.
Epoch 24/50
14/14 [==============================] - 17s 1s/step - loss: 0.1896 - accuracy: 0.
Epoch 25/50
14/14 [==============================] - 17s 1s/step - loss: 0.1816 - accuracy: 0.
Epoch 26/50
14/14 [==============================] - 17s 1s/step - loss: 0.1621 - accuracy: 0.
Epoch 27/50
14/14 [==============================] - 17s 1s/step - loss: 0.1588 - accuracy: 0.
Epoch 28/50
14/14 [==============================] - 17s 1s/step - loss: 0.1636 - accuracy: 0.
Epoch 29/50
14/14 [==============================] - 17s 1s/step - loss: 0.2652 - accuracy: 0.
Epoch 30/50
14/14 [==============================] - 17s 1s/step - loss: 0.2180 - accuracy: 0.
Epoch 31/50
14/14 [==============================] - 17s 1s/step - loss: 0.2381 - accuracy: 0.
Epoch 32/50
14/14 [==============================] - 17s 1s/step - loss: 0.2449 - accuracy: 0.
Epoch 33/50
14/14 [==============================] - 17s 1s/step - loss: 0.3024 - accuracy: 0.
Epoch 34/50
14/14 [==============================] - 17s 1s/step - loss: 0.3054 - accuracy: 0.
Epoch 35/50
14/14 [==============================] - 17s 1s/step - loss: 0.3141 - accuracy: 0.
Epoch 36/50
14/14 [==============================] - 17s 1s/step - loss: 0.1800 - accuracy: 0.
Epoch 37/50
14/14 [==============================] - 17s 1s/step - loss: 0.1545 - accuracy: 0.
Epoch 38/50
14/14 [==============================] - 17s 1s/step - loss: 0.2023 - accuracy: 0.
Epoch 39/50
14/14 [==============================] - 17s 1s/step - loss: 0.1185 - accuracy: 0.
Epoch 40/50
14/14 [==============================] - 17s 1s/step - loss: 0.1373 - accuracy: 0.
Epoch 41/50
14/14 [==============================] - 17s 1s/step - loss: 0.2052 - accuracy: 0.
Epoch 42/50
14/14 [==============================] - 17s 1s/step - loss: 0.1244 - accuracy: 0.
Epoch 43/50
14/14 [==============================] - 17s 1s/step - loss: 0.1638 - accuracy: 0.
Epoch 44/50
14/14 [==============================] - 17s 1s/step - loss: 0.1532 - accuracy: 0.
Epoch 45/50
14/14 [==============================] - 17s 1s/step - loss: 0.0611 - accuracy: 0.
Epoch 46/50
14/14 [==============================] - 17s 1s/step - loss: 0.0811 - accuracy: 0.
Epoch 47/50
14/14 [==============================] - 17s 1s/step - loss: 0.1666 - accuracy: 0.
```

```
Epoch 48/50
14/14 [==============================] - 17s 1s/step - loss: 0.1376 - accuracy: 0.
Epoch 49/50
14/14 [==============================] - 17s 1s/step - loss: 0.2302 - accuracy: 0.
Epoch 50/50
14/14 [==============================] - 17s 1s/step - loss: 0.1216 - accuracy: 0.
<keras.callbacks.History at 0x7f819e9c6d90>
```

Avaliando o modelo

```
#Testando a accurácia, que retorna a perfomance do teste
test_loss, test_acc = model.evaluate(train, verbose=2)
print('\nTest accuracy:', test_acc)
```

```
14/14 - 6s - loss: 0.0424 - accuracy: 0.9887

Test accuracy: 0.9886877536773682
```

Fazendo uma predição de uma imagem

```
#Fazendo uma predição de uma imagem qualquer
predictions = model.predict(train)
predictions[10]
```

```
array([1.0000000e+00, 2.7140792e-27, 9.6554466e-20, 3.8031630e-14,
       1.3500624e-17, 1.5313590e-14], dtype=float32)
```

```
#Pegando o array do resultado e convertendo em inteiro(número respectivo da classe indenti
np.argmax(predictions[10])
```

```
0
```

```
#Apontando o resultado da predição com o, resultou em '0', as classes vão do 0 ao 6
train.class_names[0]
```

```
'Bispo'
```

Resultados

Foi testado uma imagem de cada classe

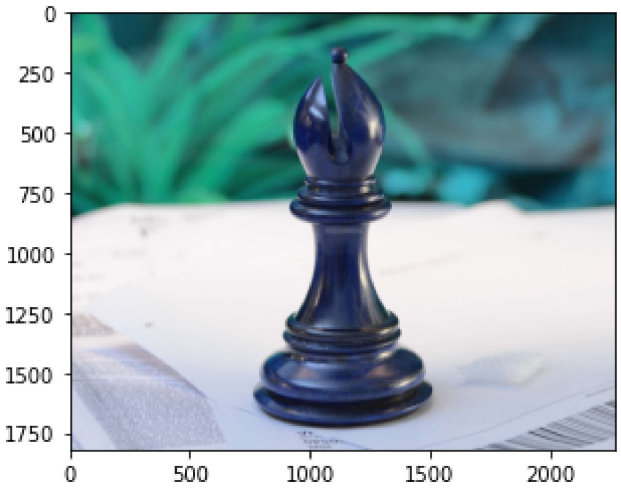
['Bispo', 'Cavalo', 'Peao', 'Rainha', 'Rei', 'Torre']=[0 , 1 , 2 , 3 , 4 , 5]

Bispo

```
#Pegando uma imagem da base de dados e exibindo
objeto=·cv2.imread('/content/drive/MyDrive/Chessman-image-dataset/Xadrez/Bispo/00000012.jp
```

```
plt.imshow(objeto)
```

```
<matplotlib.image.AxesImage at 0x7f819da9f690>
```




```
#Exibindo as dimensões da imagem
print(objeto.shape)
```

```
(1819, 2274, 3)
```

```
#Reduzindo a imagem para 48x48 antes de realizar o teste
objeto=cv2.resize(objeto,·(48,48))
```

```
#Conferindo as dimensões e atribuindo para outra variável, por padronização
img=objeto
print(img.shape)
```

```
(48, 48, 3)
```

```
#O keras realiza predições em massa, muitos itens de uma vez, mesmo que uma imagem é neces
img·=·(np.expand_dims(img,0))
```

```
#Fazendo a predição da imagem
predictions_single·=·model.predict(img)

print(predictions_single)
```

```
[[9.9667740e-01 3.9651655e-09 8.2459702e-04 9.5761068e-05 2.4010406e-03
  1.1792142e-06]]
```

```
#Utilizando a função para converter o array num número inteiro, este que será o identifica
#E ao invés de imprimir o número, será impresso a classe que a IA apontou
#Pegando o número e colocando como posição da lista de classes
#Se retornar 0 é o bispo, se 1 Cavalo...
print(train.class_names[np.argmax(predictions_single[0])])
```

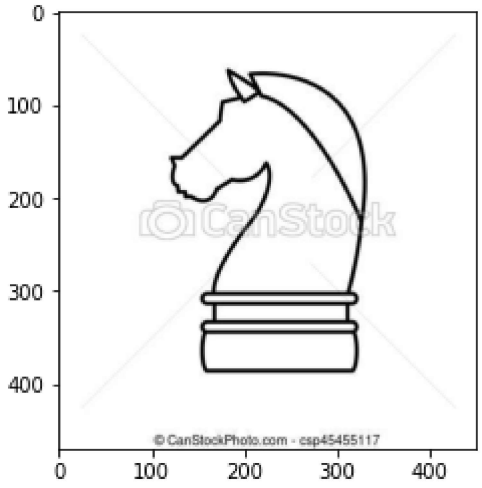
```
Bispo
```

Cavalo

```
#Teste com o cavalo
objeto= cv2.imread('/content/drive/MyDrive/Chessman-image-dataset/Xadrez/Cavalo/00000097.j
plt.imshow(objeto)

objeto=cv2.resize(objeto, (48,48))
img=objeto

img = (np.expand_dims(img,0))
```




```
predictions_single = model.predict(img)
print(predictions_single)

print(train.class_names[np.argmax(predictions_single[0])])
```

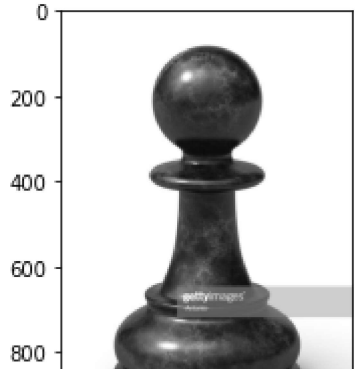
```
[[3.7402267e-28 1.0000000e+00 2.8161447e-32 1.3336948e-21 1.1327457e-24
  6.5952989e-36]]
Cavalo
```

Peão

```
#Teste com o peão
objeto= cv2.imread('/content/drive/MyDrive/Chessman-image-dataset/Xadrez/Peao/00000004.jpg
plt.imshow(objeto)

objeto=cv2.resize(objeto, (48,48))
img=objeto

img = (np.expand_dims(img,0))
```




```
predictions_single = model.predict(img)
print(predictions_single)

print(train.class_names[np.argmax(predictions_single[0])])
```

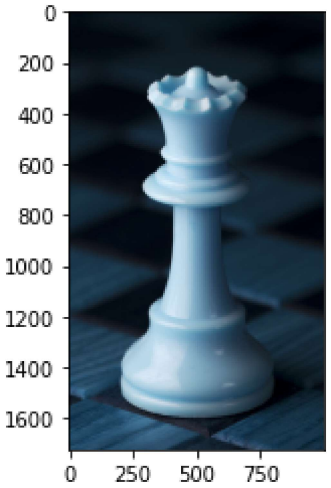
```
[[1.2649301e-28 1.0934346e-25 1.0000000e+00 2.0395660e-17 6.1136903e-32
  3.7730671e-23]]
Peao
```

Rainha

```
#Teste com a rainha
objeto=·cv2.imread('/content/drive/MyDrive/Chessman-image-dataset/Xadrez/Rainha/00000003.j
plt.imshow(objeto)

objeto=cv2.resize(objeto,·(48,48))
img=objeto

img·=·(np.expand_dims(img,0))
```




```
predictions_single·=·model.predict(img)
print(predictions_single)

print(train.class_names[np.argmax(predictions_single[0])])
```

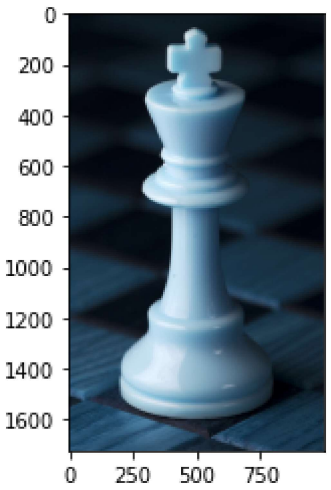
```
[[1.6428129e-12 1.0504461e-12 6.3674907e-12 1.0000000e+00 1.5825424e-10
  5.6114835e-08]]
Rainha
```

Rei

```
#Teste com o rei
objeto=·cv2.imread('/content/drive/MyDrive/Chessman-image-dataset/Xadrez/Rei/00000007.jpg'
plt.imshow(objeto)

objeto=cv2.resize(objeto,·(48,48))
img=objeto

img·=·(np.expand_dims(img,0))
```




```
predictions_single·=·model.predict(img)
print(predictions_single)

print(train.class_names[np.argmax(predictions_single[0])])
```

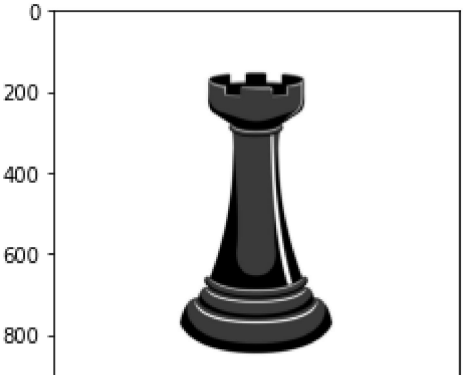
```
[[5.8465295e-11 3.0694760e-09 5.8213483e-11 1.0095705e-12 1.0000000e+00
  3.2345091e-16]]
Rei
```

Torre

```
#Por último, teste com a torre
objeto= cv2.imread('/content/drive/MyDrive/Chessman-image-dataset/Xadrez/Torre/00000006.jp
plt.imshow(objeto)

objeto=cv2.resize(objeto, (48,48))
img=objeto

img = (np.expand_dims(img,0))
```




```
predictions_single = model.predict(img)
print(predictions_single)

print(train.class_names[np.argmax(predictions_single[0])])
```

```
[[4.4254560e-18 6.7787437e-10 6.1557930e-06 1.3062663e-12 3.3015502e-20
  9.9999380e-01]]
Torre
```

```
#By George Trindade
```

✓ 0s conclusão: 18:34