1. Classificação de objetos com Redes Convolucionais

1.1 Importando as bibliotecas necessárias

```
import os
import tensorflow as tf
import numpy as np
from sklearn.model_selection import StratifiedShuffleSplit
import matplotlib.pyplot as plt
```

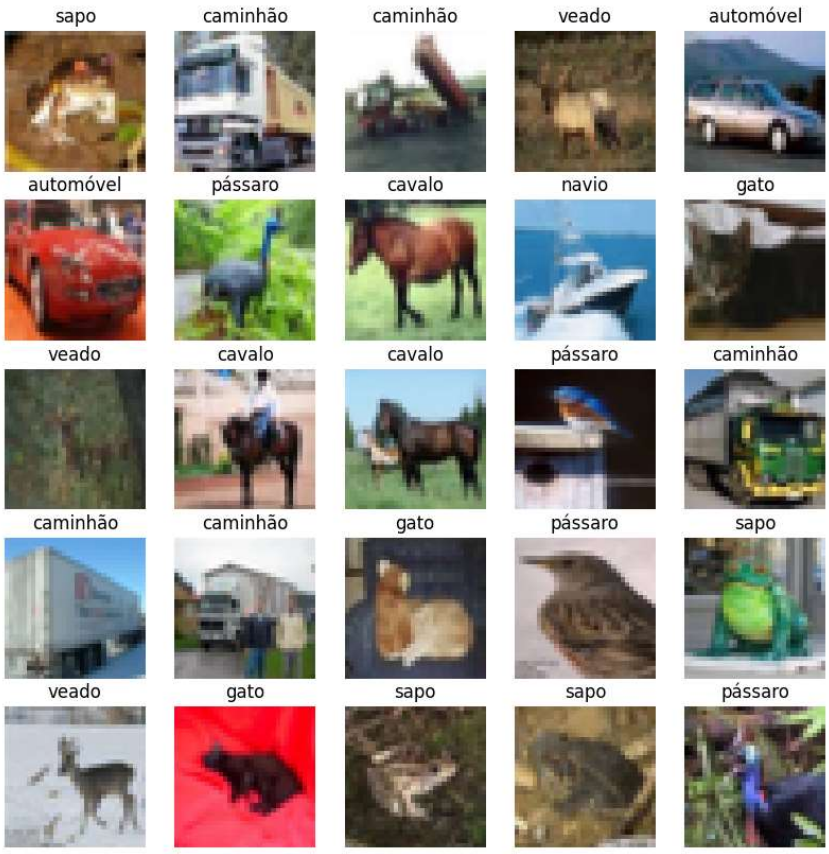
1.2 Carregando o conjunto de dados

```
(dados_treino, classes_treino), (dados_teste, classes_teste) = tf.keras.datasets.cifar10.load_data()
```

1.2.1 Exibindo os dados

```
class_names = ['avião', 'automóvel', 'pássaro', 'gato', 'veado',
               'cachorro', 'sapo', 'cavalo', 'navio', 'caminhão']

plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.imshow(dados_treino[i])
    plt.title(class_names[classes_treino[i][0]])
    plt.axis('off')
#plt.tight_layout()
#plt.show()
```



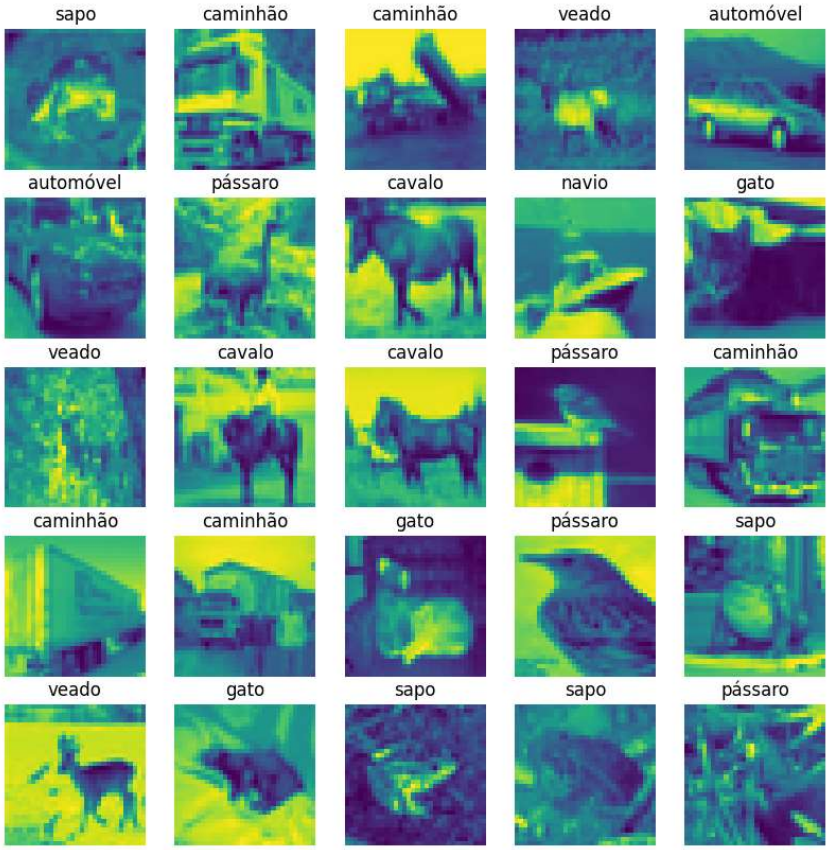
1.3 Normalizando os dados

```
# normalize inputs from 0-255 to 0.0-1.0
#Normalize pixel values
dados_treino = dados_treino.astype('float32')
dados_teste = dados_teste.astype('float32')
dados_treino = dados_treino / 255.0
dados_teste = dados_teste / 255.0
```

```
dados_treino=np.dot(dados_treino[...,:3], [0.299, 0.587, 0.114])
dados_teste=np.dot(dados_teste[...,:3], [0.299, 0.587, 0.114])

# add empty color dimension
dados_treino = np.expand_dims(dados_treino, -1)
dados_teste = np.expand_dims(dados_teste, -1)
```

```
# Resultado
plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.imshow(dados_treino[i])
    plt.title(class_names[classes_treino[i][0]])
    plt.axis('off')
```



1.4 Dividindo os dados em 80% para treino e 20% para teste

```
#Split the dataset into train and valid
s = StratifiedShuffleSplit(n_splits=5, random_state=0, test_size=0.2)
train_index, valid_index = next(s.split(dados_treino, classes_treino))

dados_validacao, classes_validacao = dados_treino[valid_index], classes_treino[valid_index]
dados_treino, classes_treino = dados_treino[train_index], classes_treino[train_index]
```

```
print(dados_treino.shape, dados_validacao.shape, dados_teste.shape)
```

```
(40000, 32, 32, 1) (10000, 32, 32, 1) (10000, 32, 32, 1)
```

1.5 Modelo CNN

```
# modelo com 3 camadas convolucionais
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.BatchNormalization(input_shape=dados_treino.shape[1:]))
model.add(tf.keras.layers.Conv2D(64, (5, 5), padding='same', activation='elu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=(2,2)))
model.add(tf.keras.layers.Dropout(0.25))

#model.add(tf.keras.layers.BatchNormalization(input_shape=dados_treino.shape[1:]))
model.add(tf.keras.layers.Conv2D(128, (5, 5), padding='same', activation='elu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(tf.keras.layers.Dropout(0.25))

#model.add(tf.keras.layers.BatchNormalization(input_shape=dados_treino.shape[1:]))
model.add(tf.keras.layers.Conv2D(256, (5, 5), padding='same', activation='elu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=(2,2)))
model.add(tf.keras.layers.Dropout(0.25))

model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(256))
model.add(tf.keras.layers.Activation('elu'))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(10))
model.add(tf.keras.layers.Activation('softmax'))
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 batch_normalization_1 (Batc  (None, 32, 32, 1)        4
 hNormalization)

 conv2d_6 (Conv2D)           (None, 32, 32, 64)        1664

 max_pooling2d_6 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 dropout_8 (Dropout)         (None, 16, 16, 64)        0

 conv2d_7 (Conv2D)           (None, 16, 16, 128)       204928

 max_pooling2d_7 (MaxPooling  (None, 8, 8, 128)        0
 2D)

 dropout_9 (Dropout)         (None, 8, 8, 128)         0

 conv2d_8 (Conv2D)           (None, 8, 8, 256)         819456

 max_pooling2d_8 (MaxPooling  (None, 4, 4, 256)        0
 2D)

 dropout_10 (Dropout)        (None, 4, 4, 256)         0

 flatten_2 (Flatten)         (None, 4096)              0

 dense_4 (Dense)             (None, 256)               1048832

 activation_4 (Activation)   (None, 256)               0

 dropout_11 (Dropout)        (None, 256)               0

 dense_5 (Dense)             (None, 10)                2570

 activation_5 (Activation)   (None, 10)                0

=================================================================
Total params: 2,077,454
Trainable params: 2,077,452
Non-trainable params: 2
_________________________________________________________________
```

1.6 Compilação do modelo

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

1.7 Treinamento

```
history = model.fit(dados_treino, classes_treino, epochs=10, batch_size=128, validation_data=(dados_validacao, classes_validacao), shu
```

```
Epoch 1/10
313/313 [==============================] - 698s 2s/step - loss: 1.5636 - accuracy: 0.4501 - val_loss: 1.3265 - val_accuracy: 0.5442
Epoch 2/10
313/313 [==============================] - 686s 2s/step - loss: 1.2170 - accuracy: 0.5764 - val_loss: 1.0607 - val_accuracy: 0.6329
Epoch 3/10
313/313 [==============================] - 684s 2s/step - loss: 1.0593 - accuracy: 0.6330 - val_loss: 0.9815 - val_accuracy: 0.6630
Epoch 4/10
313/313 [==============================] - 688s 2s/step - loss: 0.9543 - accuracy: 0.6694 - val_loss: 0.9494 - val_accuracy: 0.6724
Epoch 5/10
313/313 [==============================] - 685s 2s/step - loss: 0.8730 - accuracy: 0.6967 - val_loss: 1.0013 - val_accuracy: 0.6639
Epoch 6/10
313/313 [==============================] - 685s 2s/step - loss: 0.8020 - accuracy: 0.7221 - val_loss: 0.8908 - val_accuracy: 0.6994
Epoch 7/10
313/313 [==============================] - 689s 2s/step - loss: 0.7338 - accuracy: 0.7446 - val_loss: 0.8872 - val_accuracy: 0.7094
Epoch 8/10
313/313 [==============================] - 682s 2s/step - loss: 0.6675 - accuracy: 0.7653 - val_loss: 0.9165 - val_accuracy: 0.7065
Epoch 9/10
313/313 [==============================] - 691s 2s/step - loss: 0.6279 - accuracy: 0.7796 - val_loss: 0.8870 - val_accuracy: 0.7209
Epoch 10/10
313/313 [==============================] - 682s 2s/step - loss: 0.5781 - accuracy: 0.7983 - val_loss: 0.8743 - val_accuracy: 0.7190
```

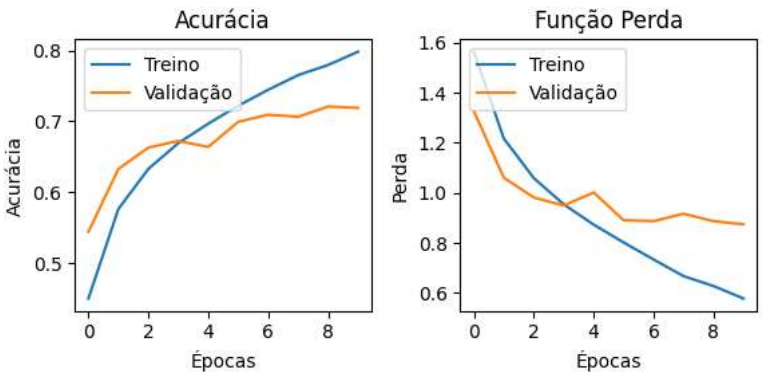
1.8 Curvas

```
# Plot training & validation accuracy values
plt.figure(figsize=(6, 3))

# Subplot for accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Acurácia')
plt.ylabel('Acurácia')
plt.xlabel('Épocas')
plt.legend(['Treino', 'Validação'], loc='upper left')

# Subplot for loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Função Perda')
plt.ylabel('Perda')
plt.xlabel('Épocas')
plt.legend(['Treino', 'Validação'], loc='upper left')

# Adjust layout to prevent overlap
plt.tight_layout()
# Show the subplots
plt.show()
```



```
# Acurácia nos dados de teste

scores = model.evaluate(dados_teste, classes_teste)
print("Acurácia: %.2f%%" % (scores[1]*100))
```

```
313/313 [==============================] - 49s 157ms/step - loss: 0.8854 - accuracy: 0.7157
Acurácia: 71.57%
```

1.9 Teste do modelo

```
fig = plt.figure(figsize=(10,10))
for i in range(25):
    ax = fig.add_subplot(5,5, i + 1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(dados_teste[i]))
    predict = np.argmax(model.predict(dados_teste[i].reshape(1, 32, 32, 1), verbose=0)[0], axis=-1)
    ax.set_title("{}".format(class_names[predict]))
    #ax.set_xlabel("{}".format(classes[y_test[i]]))
    ax.set_ylabel("{}".format(predict == classes_teste[i]))
```