In [1]:

```
from keras.utils import image_dataset_from_directory
import tensorflow as tf
train_dir_1 = 'trainning/train1'
train_dir_2 = 'trainning/train2'
train_dir_3 = 'trainning/train3'
validation_dir = 'train4' # Validation
train_dir_5 = 'trainning/train5'
test_dir = 'test'

trainning = [train_dir_1, train_dir_2,train_dir_3,train_dir_5]

train_dir = train_dir_2
IMG_SIZE = 32 # 32x32

# image_dataset_from_directory with labels="inferred" for
# getting the images in the subdirectories and translating the subdirectory as a class
# of type categorical
#train_dataset = image_dataset_from_directory(train_dir,image_size=(IMG_SIZE, IMG_SIZE),b
atch_size=32, labels="inferred", label_mode="categorical")
test_dataset = image_dataset_from_directory(test_dir,image_size=(IMG_SIZE, IMG_SIZE), lab
els="inferred",label_mode="categorical")
validation_dataset = image_dataset_from_directory(validation_dir,image_size=(IMG_SIZE, IM
G_SIZE), labels="inferred",label_mode="categorical")

train_dataset = tf.data.Dataset

for i in trainning:
    if i == trainning[0]:
        train_dataset = image_dataset_from_directory(i, image_size=(IMG_SIZE, IMG_SIZE),
labels="inferred", label_mode="categorical")
        continue
    train_dataset = train_dataset.concatenate( image_dataset_from_directory(i, image_size
=(IMG_SIZE, IMG_SIZE), labels="inferred", label_mode="categorical"))
```

```
Found 10000 files belonging to 10 classes.
Found 10000 files belonging to 10 classes.
Found 10000 files belonging to 10 classes.
Found 10000 files belonging to 10 classes.
Found 10000 files belonging to 10 classes.
Found 10000 files belonging to 10 classes.
```

In [2]:

```
#imports
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
import keras
from keras import layers
import numpy as np
IMG_SIZE = 32

inputs = keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))

x = layers.Rescaling(1./255)(inputs)

x = layers.Conv2D(filters=32, kernel_size=3, activation="relu", padding='same', kernel_r
egularizer=keras.regularizers.l2(0.001))(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.25)(x)

x = layers.Conv2D(filters=64, kernel_size=3, activation="relu", padding='same', kernel_r
egularizer=keras.regularizers.l2(0.001))(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.25)(x)

x = layers.Conv2D(filters=128, kernel_size=3, activation="relu", padding='same', kernel_
```

```
regularizer=keras.regularizers.l2(0.001))(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.25)(x)


x = layers.Flatten()(x)
x = layers.Dense(256, activation="relu", kernel_regularizer=keras.regularizers.l2(0.001)
)(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(128, activation="relu", kernel_regularizer=keras.regularizers.l2(0.001)
)(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.5)(x)

# Output layer
outputs = layers.Dense(10, activation="softmax")(x)

model = keras.Model(inputs=inputs, outputs=outputs)

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Callbacks
checkpoint = ModelCheckpoint(
    'models_S/S_without_DA_First.h5', monitor='val_loss', verbose=0,
    save_best_only=True, mode='min'
)

early_stopping = EarlyStopping(
    monitor='val_loss', patience=5, verbose=1, mode='min',
    restore_best_weights=True
)

rl = ReduceLROnPlateau(
    monitor="val_loss",
    factor=0.1,
    patience=4,
    verbose=0,
    mode="min",
)


# batch size 32
# Test loss: 0.6775595545768738
# Test accuracy: 0.8391000032424927

# batch size 64
# Test loss: 0.693570613861084
# Test accuracy: 0.8421000242233276

# batchsize 128
# Test loss: 0.7256516814231873
# Test accuracy: 0.8317000269889832

# sem BatchNormalization
#Test loss: 0.7569888234138489
# Test accuracy: 0.8009999990463257

# sem Droupout
#Test loss: 1.2996251583099365
#Test accuracy: 0.7465000152587891



#64 BATCH_SIZE - pacience 10

# Epoch 102: early stopping
# 313/313 [==============================] - 1s 3ms/step - loss: 0.7004 - accuracy: 0.835
0
# Test loss: 0.7004380226135254
```

```
# Test accuracy: 0.8349999785423279

#128
# Test loss: 0.7256516814231873
# Test accuracy: 0.8317000269889832
```

In [4]:

```
history = model.fit(
    train_dataset,
    epochs=200,
    batch_size=128,
    validation_data=validation_dataset,
    callbacks=[checkpoint, early_stopping, rl]
  )


# Avaliar o modelo (com o teste)
test_loss, test_accuracy = model.evaluate(test_dataset)
print(f'Test loss: {test_loss}')
print(f'Test accuracy: {test_accuracy}')
```

```
Epoch 1/200
1252/1252 [==============================] - 122s 98ms/step - loss: 2.4612 - accuracy: 0.
3994 - val_loss: 3.0313 - val_accuracy: 0.3178 - lr: 0.0010
Epoch 2/200
1252/1252 [==============================] - 35s 28ms/step - loss: 1.8749 - accuracy: 0.5
286 - val_loss: 1.7325 - val_accuracy: 0.5480 - lr: 0.0010
Epoch 3/200
1252/1252 [==============================] - 35s 28ms/step - loss: 1.6502 - accuracy: 0.5
878 - val_loss: 1.7617 - val_accuracy: 0.5382 - lr: 0.0010
Epoch 4/200
1252/1252 [==============================] - 33s 26ms/step - loss: 1.5788 - accuracy: 0.6
197 - val_loss: 1.6868 - val_accuracy: 0.5654 - lr: 0.0010
Epoch 5/200
1252/1252 [==============================] - 34s 27ms/step - loss: 1.5629 - accuracy: 0.6
341 - val_loss: 1.4883 - val_accuracy: 0.6629 - lr: 0.0010
Epoch 6/200
1252/1252 [==============================] - 33s 26ms/step - loss: 1.5438 - accuracy: 0.6
480 - val_loss: 1.4054 - val_accuracy: 0.6954 - lr: 0.0010
Epoch 7/200
1252/1252 [==============================] - 37s 29ms/step - loss: 1.5290 - accuracy: 0.6
582 - val_loss: 1.4709 - val_accuracy: 0.6735 - lr: 0.0010
Epoch 8/200
1252/1252 [==============================] - 37s 30ms/step - loss: 1.5218 - accuracy: 0.6
642 - val_loss: 2.0779 - val_accuracy: 0.4837 - lr: 0.0010
Epoch 9/200
1252/1252 [==============================] - 37s 30ms/step - loss: 1.5052 - accuracy: 0.6
737 - val_loss: 1.3864 - val_accuracy: 0.7057 - lr: 0.0010
Epoch 10/200
1252/1252 [==============================] - 35s 28ms/step - loss: 1.4894 - accuracy: 0.6
787 - val_loss: 1.4003 - val_accuracy: 0.7007 - lr: 0.0010
Epoch 11/200
1252/1252 [==============================] - 31s 24ms/step - loss: 1.4732 - accuracy: 0.6
821 - val_loss: 1.6594 - val_accuracy: 0.6134 - lr: 0.0010
Epoch 12/200
1252/1252 [==============================] - 31s 25ms/step - loss: 1.4794 - accuracy: 0.6
819 - val_loss: 1.4633 - val_accuracy: 0.6770 - lr: 0.0010
Epoch 13/200
1252/1252 [==============================] - 30s 24ms/step - loss: 1.4632 - accuracy: 0.6
842 - val_loss: 1.5156 - val_accuracy: 0.6592 - lr: 0.0010
Epoch 14/200
1252/1252 [==============================] - 32s 25ms/step - loss: 1.3000 - accuracy: 0.7
300 - val_loss: 1.1259 - val_accuracy: 0.7745 - lr: 1.0000e-04
Epoch 15/200
1252/1252 [==============================] - 29s 23ms/step - loss: 1.1918 - accuracy: 0.7
477 - val_loss: 1.0401 - val_accuracy: 0.7861 - lr: 1.0000e-04
Epoch 16/200
1252/1252 [==============================] - 29s 23ms/step - loss: 1.1153 - accuracy: 0.7
599 - val_loss: 1.0010 - val_accuracy: 0.7886 - lr: 1.0000e-04
Epoch 17/200
1252/1252 [==============================] - 35s 28ms/step - loss: 1.0620 - accuracy: 0.7
```

```
660 - val_loss: 0.9698 - val_accuracy: 0.7904 - lr: 1.0000e-04
Epoch 18/200
1252/1252 [==============================] - 37s 29ms/step - loss: 1.0134 - accuracy: 0.7
770 - val_loss: 0.9261 - val_accuracy: 0.7955 - lr: 1.0000e-04
Epoch 19/200
1252/1252 [==============================] - 31s 25ms/step - loss: 0.9798 - accuracy: 0.7
778 - val_loss: 0.8985 - val_accuracy: 0.7994 - lr: 1.0000e-04
Epoch 20/200
1252/1252 [==============================] - 31s 25ms/step - loss: 0.9507 - accuracy: 0.7
840 - val_loss: 0.8773 - val_accuracy: 0.8014 - lr: 1.0000e-04
Epoch 21/200
1252/1252 [==============================] - 34s 27ms/step - loss: 0.9253 - accuracy: 0.7
851 - val_loss: 0.8573 - val_accuracy: 0.8041 - lr: 1.0000e-04
Epoch 22/200
1252/1252 [==============================] - 36s 29ms/step - loss: 0.9051 - accuracy: 0.7
903 - val_loss: 0.8501 - val_accuracy: 0.8023 - lr: 1.0000e-04
Epoch 23/200
1252/1252 [==============================] - 32s 25ms/step - loss: 0.8801 - accuracy: 0.7
947 - val_loss: 0.8245 - val_accuracy: 0.8090 - lr: 1.0000e-04
Epoch 24/200
1252/1252 [==============================] - 33s 27ms/step - loss: 0.8613 - accuracy: 0.8
004 - val_loss: 0.8263 - val_accuracy: 0.8056 - lr: 1.0000e-04
Epoch 25/200
1252/1252 [==============================] - 38s 30ms/step - loss: 0.8389 - accuracy: 0.8
010 - val_loss: 0.8209 - val_accuracy: 0.8083 - lr: 1.0000e-04
Epoch 26/200
1252/1252 [==============================] - 43s 34ms/step - loss: 0.8281 - accuracy: 0.8
054 - val_loss: 0.8155 - val_accuracy: 0.8068 - lr: 1.0000e-04
Epoch 27/200
1252/1252 [==============================] - 35s 28ms/step - loss: 0.8169 - accuracy: 0.8
075 - val_loss: 0.7864 - val_accuracy: 0.8156 - lr: 1.0000e-04
Epoch 28/200
1252/1252 [==============================] - 51s 41ms/step - loss: 0.8062 - accuracy: 0.8
072 - val_loss: 0.8055 - val_accuracy: 0.8053 - lr: 1.0000e-04
Epoch 29/200
1252/1252 [==============================] - 47s 37ms/step - loss: 0.7923 - accuracy: 0.8
099 - val_loss: 0.7782 - val_accuracy: 0.8168 - lr: 1.0000e-04
Epoch 30/200
1252/1252 [==============================] - 57s 45ms/step - loss: 0.7815 - accuracy: 0.8
135 - val_loss: 0.7685 - val_accuracy: 0.8178 - lr: 1.0000e-04
Epoch 31/200
1252/1252 [==============================] - 34s 27ms/step - loss: 0.7763 - accuracy: 0.8
136 - val_loss: 0.7654 - val_accuracy: 0.8208 - lr: 1.0000e-04
Epoch 32/200
1252/1252 [==============================] - 33s 26ms/step - loss: 0.7669 - accuracy: 0.8
145 - val_loss: 0.7630 - val_accuracy: 0.8203 - lr: 1.0000e-04
Epoch 33/200
1252/1252 [==============================] - 31s 25ms/step - loss: 0.7573 - accuracy: 0.8
186 - val_loss: 0.7838 - val_accuracy: 0.8110 - lr: 1.0000e-04
Epoch 34/200
1252/1252 [==============================] - 32s 26ms/step - loss: 0.7524 - accuracy: 0.8
181 - val_loss: 0.7768 - val_accuracy: 0.8156 - lr: 1.0000e-04
Epoch 35/200
1252/1252 [==============================] - 32s 25ms/step - loss: 0.7457 - accuracy: 0.8
216 - val_loss: 0.7907 - val_accuracy: 0.8090 - lr: 1.0000e-04
Epoch 36/200
1252/1252 [==============================] - 32s 25ms/step - loss: 0.7332 - accuracy: 0.8
250 - val_loss: 0.7484 - val_accuracy: 0.8176 - lr: 1.0000e-04
Epoch 37/200
1252/1252 [==============================] - 31s 25ms/step - loss: 0.7290 - accuracy: 0.8
238 - val_loss: 0.7578 - val_accuracy: 0.8178 - lr: 1.0000e-04
Epoch 38/200
1252/1252 [==============================] - 32s 25ms/step - loss: 0.7223 - accuracy: 0.8
254 - val_loss: 0.7600 - val_accuracy: 0.8126 - lr: 1.0000e-04
Epoch 39/200
1252/1252 [==============================] - 32s 25ms/step - loss: 0.7228 - accuracy: 0.8
248 - val_loss: 0.7463 - val_accuracy: 0.8214 - lr: 1.0000e-04
Epoch 40/200
1252/1252 [==============================] - 31s 25ms/step - loss: 0.7174 - accuracy: 0.8
270 - val_loss: 0.7490 - val_accuracy: 0.8201 - lr: 1.0000e-04
Epoch 41/200
1252/1252 [==============================] - 31s 24ms/step - loss: 0.7071 - accuracy: 0.8
```

```
299 - val_loss: 0.7392 - val_accuracy: 0.8241 - lr: 1.0000e-04
Epoch 42/200
1252/1252 [==============================] - 30s 24ms/step - loss: 0.7018 - accuracy: 0.8
306 - val_loss: 0.7566 - val_accuracy: 0.8180 - lr: 1.0000e-04
Epoch 43/200
1252/1252 [==============================] - 31s 25ms/step - loss: 0.6984 - accuracy: 0.8
320 - val_loss: 0.7507 - val_accuracy: 0.8212 - lr: 1.0000e-04
Epoch 44/200
1252/1252 [==============================] - 34s 27ms/step - loss: 0.6996 - accuracy: 0.8
311 - val_loss: 0.7530 - val_accuracy: 0.8207 - lr: 1.0000e-04
Epoch 45/200
1252/1252 [==============================] - 34s 27ms/step - loss: 0.6914 - accuracy: 0.8
332 - val_loss: 0.7345 - val_accuracy: 0.8229 - lr: 1.0000e-04
Epoch 46/200
1252/1252 [==============================] - 34s 27ms/step - loss: 0.6899 - accuracy: 0.8
355 - val_loss: 0.7343 - val_accuracy: 0.8243 - lr: 1.0000e-04
Epoch 47/200
1252/1252 [==============================] - 37s 29ms/step - loss: 0.6887 - accuracy: 0.8
350 - val_loss: 0.7346 - val_accuracy: 0.8239 - lr: 1.0000e-04
Epoch 48/200
1252/1252 [==============================] - 32s 25ms/step - loss: 0.6864 - accuracy: 0.8
358 - val_loss: 0.7577 - val_accuracy: 0.8200 - lr: 1.0000e-04
Epoch 49/200
1252/1252 [==============================] - 34s 27ms/step - loss: 0.6761 - accuracy: 0.8
397 - val_loss: 0.7444 - val_accuracy: 0.8222 - lr: 1.0000e-04
Epoch 50/200
1252/1252 [==============================] - 33s 27ms/step - loss: 0.6797 - accuracy: 0.8
371 - val_loss: 0.7548 - val_accuracy: 0.8166 - lr: 1.0000e-04
Epoch 51/200
1252/1252 [==============================] - 31s 24ms/step - loss: 0.6434 - accuracy: 0.8
496 - val_loss: 0.7239 - val_accuracy: 0.8275 - lr: 1.0000e-05
Epoch 52/200
1252/1252 [==============================] - 34s 27ms/step - loss: 0.6410 - accuracy: 0.8
498 - val_loss: 0.7178 - val_accuracy: 0.8293 - lr: 1.0000e-05
Epoch 53/200
1252/1252 [==============================] - 33s 27ms/step - loss: 0.6409 - accuracy: 0.8
486 - val_loss: 0.7131 - val_accuracy: 0.8313 - lr: 1.0000e-05
Epoch 54/200
1252/1252 [==============================] - 34s 27ms/step - loss: 0.6309 - accuracy: 0.8
503 - val_loss: 0.7156 - val_accuracy: 0.8294 - lr: 1.0000e-05
Epoch 55/200
1252/1252 [==============================] - 34s 27ms/step - loss: 0.6344 - accuracy: 0.8
514 - val_loss: 0.7137 - val_accuracy: 0.8294 - lr: 1.0000e-05
Epoch 56/200
1252/1252 [==============================] - 36s 29ms/step - loss: 0.6203 - accuracy: 0.8
529 - val_loss: 0.7124 - val_accuracy: 0.8301 - lr: 1.0000e-05
Epoch 57/200
1252/1252 [==============================] - 34s 27ms/step - loss: 0.6195 - accuracy: 0.8
539 - val_loss: 0.7077 - val_accuracy: 0.8316 - lr: 1.0000e-05
Epoch 58/200
1252/1252 [==============================] - 31s 25ms/step - loss: 0.6064 - accuracy: 0.8
592 - val_loss: 0.7095 - val_accuracy: 0.8315 - lr: 1.0000e-05
Epoch 59/200
1252/1252 [==============================] - 32s 26ms/step - loss: 0.6137 - accuracy: 0.8
571 - val_loss: 0.7062 - val_accuracy: 0.8330 - lr: 1.0000e-05
Epoch 60/200
1252/1252 [==============================] - 32s 25ms/step - loss: 0.6135 - accuracy: 0.8
561 - val_loss: 0.7031 - val_accuracy: 0.8318 - lr: 1.0000e-05
Epoch 61/200
1252/1252 [==============================] - 30s 24ms/step - loss: 0.6070 - accuracy: 0.8
604 - val_loss: 0.7084 - val_accuracy: 0.8308 - lr: 1.0000e-05
Epoch 62/200
1252/1252 [==============================] - 30s 24ms/step - loss: 0.5985 - accuracy: 0.8
612 - val_loss: 0.7026 - val_accuracy: 0.8336 - lr: 1.0000e-05
Epoch 63/200
1252/1252 [==============================] - 29s 23ms/step - loss: 0.6001 - accuracy: 0.8
593 - val_loss: 0.7045 - val_accuracy: 0.8313 - lr: 1.0000e-05
Epoch 64/200
1252/1252 [==============================] - 32s 26ms/step - loss: 0.5939 - accuracy: 0.8
640 - val_loss: 0.6997 - val_accuracy: 0.8339 - lr: 1.0000e-05
Epoch 65/200
1252/1252 [==============================] - 33s 26ms/step - loss: 0.5949 - accuracy: 0.8
```

```
615 - val_loss: 0.7037 - val_accuracy: 0.8312 - lr: 1.0000e-05
Epoch 66/200
1252/1252 [==============================] - 30s 24ms/step - loss: 0.6000 - accuracy: 0.8
601 - val_loss: 0.6988 - val_accuracy: 0.8343 - lr: 1.0000e-05
Epoch 67/200
1252/1252 [==============================] - 29s 23ms/step - loss: 0.5950 - accuracy: 0.8
608 - val_loss: 0.6999 - val_accuracy: 0.8329 - lr: 1.0000e-05
Epoch 68/200
1252/1252 [==============================] - 32s 26ms/step - loss: 0.5920 - accuracy: 0.8
629 - val_loss: 0.6994 - val_accuracy: 0.8332 - lr: 1.0000e-05
Epoch 69/200
1252/1252 [==============================] - 32s 25ms/step - loss: 0.5882 - accuracy: 0.8
622 - val_loss: 0.6990 - val_accuracy: 0.8312 - lr: 1.0000e-05
Epoch 70/200
1252/1252 [==============================] - 32s 26ms/step - loss: 0.5826 - accuracy: 0.8
662 - val_loss: 0.6960 - val_accuracy: 0.8334 - lr: 1.0000e-05
Epoch 71/200
1252/1252 [==============================] - 32s 26ms/step - loss: 0.5820 - accuracy: 0.8
632 - val_loss: 0.6970 - val_accuracy: 0.8329 - lr: 1.0000e-05
Epoch 72/200
1252/1252 [==============================] - 30s 24ms/step - loss: 0.5861 - accuracy: 0.8
643 - val_loss: 0.6963 - val_accuracy: 0.8344 - lr: 1.0000e-05
Epoch 73/200
1252/1252 [==============================] - 35s 28ms/step - loss: 0.5806 - accuracy: 0.8
637 - val_loss: 0.6948 - val_accuracy: 0.8347 - lr: 1.0000e-05
Epoch 74/200
1252/1252 [==============================] - 42s 34ms/step - loss: 0.5793 - accuracy: 0.8
662 - val_loss: 0.6929 - val_accuracy: 0.8348 - lr: 1.0000e-05
Epoch 75/200
1252/1252 [==============================] - 41s 33ms/step - loss: 0.5787 - accuracy: 0.8
654 - val_loss: 0.6930 - val_accuracy: 0.8326 - lr: 1.0000e-05
Epoch 76/200
1252/1252 [==============================] - 45s 36ms/step - loss: 0.5741 - accuracy: 0.8
670 - val_loss: 0.6886 - val_accuracy: 0.8362 - lr: 1.0000e-05
Epoch 77/200
1252/1252 [==============================] - 42s 33ms/step - loss: 0.5696 - accuracy: 0.8
662 - val_loss: 0.6928 - val_accuracy: 0.8339 - lr: 1.0000e-05
Epoch 78/200
1252/1252 [==============================] - 40s 32ms/step - loss: 0.5745 - accuracy: 0.8
673 - val_loss: 0.6903 - val_accuracy: 0.8343 - lr: 1.0000e-05
Epoch 79/200
1252/1252 [==============================] - 42s 33ms/step - loss: 0.5632 - accuracy: 0.8
703 - val_loss: 0.6892 - val_accuracy: 0.8349 - lr: 1.0000e-05
Epoch 80/200
1252/1252 [==============================] - 44s 35ms/step - loss: 0.5609 - accuracy: 0.8
715 - val_loss: 0.6952 - val_accuracy: 0.8342 - lr: 1.0000e-05
Epoch 81/200
1252/1252 [==============================] - ETA: 0s - loss: 0.5635 - accuracy: 0.8692Res
toring model weights from the end of the best epoch: 76.
1252/1252 [==============================] - 40s 32ms/step - loss: 0.5635 - accuracy: 0.8
692 - val_loss: 0.6932 - val_accuracy: 0.8346 - lr: 1.0000e-06
Epoch 81: early stopping
313/313 [==============================] - 6s 19ms/step - loss: 0.6838 - accuracy: 0.8331
Test loss: 0.6837673187255859
Test accuracy: 0.8331000208854675
```

O modelo contém 3 camadas convolucionais para reconhecer padrões presentes nas imagens, e para analisar os dados espaciais. Adicionamos uma camada convolucional (layers.Conv2D) com x filtros, cada um com tamanho (janela de convolução) 3x3, utilizando a função de ativação ReLU. O padding='same' garante que a saída tenha o mesmo tamanho da entrada (uma vez que stride é 1). O kernel_regularizer=keras.regularizers.l2(0.001) aplica a regularização L2 para evitar overfitting. A regularização L2 ajuda a reduzir a importância de variáveis correlacionadas, evitando que elas dominem o modelo e prejudiquem a generalização.

Com o objetivo de normalizar a ativação da camada anterior, acelerar o treino, e melhorar a estabilidade do modelo adiciona-mos o layer layers.BatchNormalization. Este aplica uma transformação que mantém a saída média próxima de 0 e o desvio padrão da saída próximo de 1.

Usamos layers.MaxPooling2D para reduzir a dimensão espacial da entrada pela metade ao utilizar uma janela de 2x2.

de 2x2.

Para evitar overfitting aplicamos uma camada de dropout (**layers.Dropout**) com uma taxa de 0.25, esta desativa aleatoriamente 25% dos neurónios da camada durante o treino.

Este processo é repetido 3 vezes mas com um numero de filtros maior em cada layer (a dobrar), estas camadas convolucionais cada vez extraem características mais especificas. Depois de extrair as características através das camadas convolucionais, os dados são achatados (flatten) e passados para as camadas densas para classificação. Este modelo tem 2 camadas densas. Primeiro tranformamos a entrada 2D num vetor 1D usando **layers.Flatten**, uma vez que as camadas densas recebem um vetor de uma dimensão. Depois foi adicionada uma camada densa (totalmente conectada) com 256 neurónios e a função de ativação ReLU (**layers.Dense**). Esta inclui regularização L2. Normaliza-se a saída da camada densa (**layers.BatchNormalization**). E aplica-se dropout com uma taxa de 0.5 para evitar overfitting (**layers.Dropout**).

Depois na camada de saída temos uma uma camada densa (**layers.Dense**) com 10 neurónios, um para cada classe, utilizando a função de ativação softmax para obter as probabilidades de classificação.

Após o modelo estar construido definimos o modelo com as entradas e saídas especificadas (**keras.Model**). E compilamos o modelo (**model.compile**)utilizando a perda categorical_crossentropy, o otimizador adam, e a métrica de accuracy. Usamos o categorical_crossentropy uma vez que este é adequado para problemas multiclasse, ele mede a diferença entre a distribuição de probabilidade verdadeira das classes e a distribuição de probabilidade prevista, incentivando previsões de probabilidades que são mais próximas das distribuições reais das classes. Usamos o otimizador adam pois este proporciona uma taxa de aprendizagem adaptativa e é eficiente, funcionando bem com diferentes tipos de dados e problemas. A accuracy fornece uma métrica simples e intuitiva para avaliar o desempenho do modelo, é útil para verificar rapidamente se o modelo está a aprender corretamente. Para acabar usamos callbacks, primeiro salvamos o melhor modelo com base na métrica val_loss (**ModelCheckpoint**). Usamos o **EarlyStopping** para parar o treino em caso que a val_loss não melhor em 5 épocas consecutivas e restaura os melhores pesos do modelo. Durante o treino ajustamos a taxa de aprendizado (**LearningRateScheduler**). Esta mantém a taxa de aprendizado constante nas primeiras 10 épocas e a reduz exponencialmente depois disso.

Ao longo do processo de construção deste modelo, testámos várias camadas convolucionais com diferentes parâmetros e quantidades. Além disso, experimentámos diversas estratégias para combater o sobreajuste (overfitting). Este modelo final foi o que apresentou os melhores resultados.

Testou-se com diferentes valores de batch size e chegou-se à conclusão que não parece haver uma relação direta entre a accuracy e este valor, a nível de tempo com um batch size maior obviamente cada época demora mais tempo, mas depois vão ser precisas menos épocas para convergir. Consoante a seguinte tabela 128 para o batch size foi o que permitiu maior accuracy, mas uma vez que estou a usar o colab fica dificil fazer todos os testes que gostaria para tirar mais conclusões. Mas uma vez que por volta de uns 10 treinos a maior accuracy que tivemos foi com um batch size de 128, ficou esse. E no modeloS extra usámos o batch size de 128, uma vez que este é mais rápido e obtivemos melhores resultados.

Como se pode ver sem BatchNormalization a accuracy é bem mais baixa: Test loss: 0.75 Test accuracy: 0.80

E sem Droupout a accuracy nem chega aos 0.75 : Test loss: 1.29 Test accuracy: 0.74

Tendo o modelo final : Test loss: 0.693570613861084 Test accuracy: 0.8421000242233276

Após "finalizar" com uma boa accuracy o modelo S decidiu-se adaptar o modelo com data augmentation.

\

Batch size	Max. Test accuracy	At Epoch
32	70.41	13
64	71.79	14
128	73.02	11

256	71.76	13
512	71.98	15

layers.Conv2D:https://www.ibm.com/topics/convolutional-neural-networks
layers.BatchNormalization:https://medium.com/@ilyurek/demystifying-batch-normalization-a-practical-guide-with-python-24c58956d3e

layers.MaxPooling2D:https://keras.io/api/layers/pooling_layers/max_pooling2d/

layers.Dropout:https://databasecamp.de/en/ml/dropout-layer-en

L2 Regularization: https://medium.com/@fernando.dijkinga/explaining-l1-and-l2-regularization-in-machine-learning-2356ee91c8e3

MODELO S COM DATA AUGMENTATION

Com o objetivo de aumentar a quantidade e a diversidade dos dados de treino, o que ajuda a melhorar a generalização e a robustez do modelo. Adicionámos data augmentation.

In [6]:

```
from keras.callbacks import ModelCheckpoint, EarlyStopping,ReduceLROnPlateau
import keras
from keras import layers

data_augmentation = keras.Sequential(
        [layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),])

    # rotation_range=20,
    # width_shift_range=0.1,
    # height_shift_range=0.1,
    # zoom_range=0.2,
    # horizontal_flip=True,
    # fill_mode='nearest',
    # # shear_range=0.2,
    # # zoom_range=0.2,
    # # horizontal_flip=True,
    # # fill_mode='nearest',
    # # channel_shift_range=0.2,

inputs = keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)

x = layers.Conv2D(filters=32, kernel_size=3, activation="relu", padding='same', kernel_r
egularizer=keras.regularizers.l2(0.001))(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.25)(x)

x = layers.Conv2D(filters=64, kernel_size=3, activation="relu", padding='same', kernel_r
egularizer=keras.regularizers.l2(0.001))(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.25)(x)

x = layers.Conv2D(filters=128, kernel_size=3, activation="relu", padding='same', kernel_
regularizer=keras.regularizers.l2(0.001))(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(0.25)(x)

x = layers.Flatten()(x)
```

```
x = layers.Dense(256, activation="relu", kernel_regularizer=keras.regularizers.l2(0.001)
)(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(128, activation="relu", kernel_regularizer=keras.regularizers.l2(0.001)
)(x)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.5)(x)

# Output layer
outputs = layers.Dense(10, activation="softmax")(x)

model2 = keras.Model(inputs=inputs, outputs=outputs)

# compilar o modelo
model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Callbacks
checkpoint = ModelCheckpoint(
    'models_S/S_with_DA_First.h5', monitor='val_loss', verbose=0,
    save_best_only=True, mode='min'
)

early_stopping = EarlyStopping(
    monitor='val_loss', patience=4, verbose=1, mode='min',
    restore_best_weights=True
)

learning_rate = ReduceLROnPlateau(
    monitor="val_loss",
    factor=0.1,
    patience=4,
    verbose=0,
    mode="min",
)
```

In [ ]:

```
history2 = model.fit(
    train_dataset,
    epochs=200,
    batch_size=128,
    validation_data=validation_dataset,
    callbacks=[checkpoint, early_stopping, rl]
  )


# Avaliar o modelo (com o teste)
test_loss, test_accuracy = model2.evaluate(test_dataset)
print(f'Test loss: {test_loss}')
print(f'Test accuracy: {test_accuracy}')
```

```
Epoch 1/200
1248/1250 [============================>.] - ETA: 0s - loss: 1.7520 - accuracy: 0.5494
Epoch 1: val_loss improved from 1.59985 to 1.57618, saving model to best_model3.h5
1250/1250 [==============================] - 33s 26ms/step - loss: 1.7520 - accuracy: 0.5
493 - val_loss: 1.5762 - val_accuracy: 0.6225 - lr: 0.0010
Epoch 2/200
1248/1250 [============================>.] - ETA: 0s - loss: 1.7445 - accuracy: 0.5595
Epoch 2: val_loss improved from 1.57618 to 1.47002, saving model to best_model3.h5
1250/1250 [==============================] - 33s 26ms/step - loss: 1.7445 - accuracy: 0.5
594 - val_loss: 1.4700 - val_accuracy: 0.6492 - lr: 0.0010
Epoch 3/200
1249/1250 [============================>.] - ETA: 0s - loss: 1.7224 - accuracy: 0.5674
Epoch 3: val_loss did not improve from 1.47002
1250/1250 [==============================] - 33s 26ms/step - loss: 1.7222 - accuracy: 0.5
675 - val_loss: 1.5922 - val_accuracy: 0.5942 - lr: 0.0010
Epoch 4/200
1249/1250 [============================>.] - ETA: 0s - loss: 1.7099 - accuracy: 0.5759
Epoch 4: val_loss improved from 1.47002 to 1.37626, saving model to best_model3.h5
1250/1250 [==============================] - 33s 26ms/step - loss: 1.7096 - accuracy: 0.5
760 - val_loss: 1.3763 - val_accuracy: 0.6870 - lr: 0.0010
```

```
Epoch 5/200
1249/1250 [============================>.] - ETA: 0s - loss: 1.6998 - accuracy: 0.5763
Epoch 5: val_loss did not improve from 1.37626
1250/1250 [==============================] - 33s 27ms/step - loss: 1.6997 - accuracy: 0.5
763 - val_loss: 1.9492 - val_accuracy: 0.5015 - lr: 0.0010
Epoch 6/200
1249/1250 [============================>.] - ETA: 0s - loss: 1.6868 - accuracy: 0.5809
Epoch 6: val_loss did not improve from 1.37626
1250/1250 [==============================] - 33s 26ms/step - loss: 1.6869 - accuracy: 0.5
808 - val_loss: 1.7475 - val_accuracy: 0.5766 - lr: 0.0010
Epoch 7/200
1249/1250 [============================>.] - ETA: 0s - loss: 1.6736 - accuracy: 0.5862
Epoch 7: val_loss did not improve from 1.37626
1250/1250 [==============================] - 33s 26ms/step - loss: 1.6735 - accuracy: 0.5
863 - val_loss: 1.5403 - val_accuracy: 0.6164 - lr: 0.0010
Epoch 8/200
1248/1250 [============================>.] - ETA: 0s - loss: 1.6666 - accuracy: 0.5860
Epoch 8: val_loss did not improve from 1.37626
1250/1250 [==============================] - 33s 26ms/step - loss: 1.6668 - accuracy: 0.5
859 - val_loss: 1.7032 - val_accuracy: 0.5741 - lr: 0.0010
Epoch 9/200
1250/1250 [==============================] - ETA: 0s - loss: 1.6605 - accuracy: 0.5897
Epoch 9: val_loss did not improve from 1.37626
1250/1250 [==============================] - 33s 26ms/step - loss: 1.6605 - accuracy: 0.5
897 - val_loss: 1.6398 - val_accuracy: 0.5918 - lr: 0.0010
Epoch 10/200
1250/1250 [==============================] - ETA: 0s - loss: 1.6628 - accuracy: 0.5903
Epoch 10: val_loss did not improve from 1.37626
1250/1250 [==============================] - 33s 26ms/step - loss: 1.6628 - accuracy: 0.5
903 - val_loss: 1.3978 - val_accuracy: 0.6754 - lr: 0.0010
Epoch 11/200
1249/1250 [============================>.] - ETA: 0s - loss: 1.6256 - accuracy: 0.5947
Epoch 11: val_loss did not improve from 1.37626
1250/1250 [==============================] - 33s 26ms/step - loss: 1.6256 - accuracy: 0.5
948 - val_loss: 1.5941 - val_accuracy: 0.5975 - lr: 9.0484e-04
Epoch 12/200
1249/1250 [============================>.] - ETA: 0s - loss: 1.5843 - accuracy: 0.6100
Epoch 12: val_loss improved from 1.37626 to 1.27970, saving model to best_model3.h5
1250/1250 [==============================] - 33s 26ms/step - loss: 1.5845 - accuracy: 0.6
099 - val_loss: 1.2797 - val_accuracy: 0.6996 - lr: 8.1873e-04
Epoch 13/200
1250/1250 [==============================] - ETA: 0s - loss: 1.5284 - accuracy: 0.6180
Epoch 13: val_loss did not improve from 1.27970
1250/1250 [==============================] - 33s 26ms/step - loss: 1.5284 - accuracy: 0.6
180 - val_loss: 1.3611 - val_accuracy: 0.6656 - lr: 7.4082e-04
Epoch 14/200
1249/1250 [============================>.] - ETA: 0s - loss: 1.4909 - accuracy: 0.6260
Epoch 14: val_loss did not improve from 1.27970
1250/1250 [==============================] - 33s 26ms/step - loss: 1.4908 - accuracy: 0.6
261 - val_loss: 1.3628 - val_accuracy: 0.6625 - lr: 6.7032e-04
Epoch 15/200
1250/1250 [==============================] - ETA: 0s - loss: 1.4504 - accuracy: 0.6352
Epoch 15: val_loss improved from 1.27970 to 1.20225, saving model to best_model3.h5
1250/1250 [==============================] - 33s 26ms/step - loss: 1.4504 - accuracy: 0.6
352 - val_loss: 1.2022 - val_accuracy: 0.7139 - lr: 6.0653e-04
Epoch 16/200
1250/1250 [==============================] - ETA: 0s - loss: 1.4235 - accuracy: 0.6380
Epoch 16: val_loss did not improve from 1.20225
1250/1250 [==============================] - 33s 26ms/step - loss: 1.4235 - accuracy: 0.6
380 - val_loss: 1.2951 - val_accuracy: 0.6806 - lr: 5.4881e-04
Epoch 17/200
1250/1250 [==============================] - ETA: 0s - loss: 1.3893 - accuracy: 0.6428
Epoch 17: val_loss did not improve from 1.20225
1250/1250 [==============================] - 33s 26ms/step - loss: 1.3893 - accuracy: 0.6
428 - val_loss: 1.4012 - val_accuracy: 0.6443 - lr: 4.9659e-04
Epoch 18/200
1248/1250 [============================>.] - ETA: 0s - loss: 1.3568 - accuracy: 0.6483
Epoch 18: val_loss improved from 1.20225 to 1.05067, saving model to best_model3.h5
1250/1250 [==============================] - 33s 26ms/step - loss: 1.3571 - accuracy: 0.6
482 - val_loss: 1.0507 - val_accuracy: 0.7489 - lr: 4.4933e-04
Epoch 19/200
1250/1250 [==============================] - ETA: 0s - loss: 1.3272 - accuracy: 0.6546
```

```
Epoch 19: val_loss did not improve from 1.05067
1250/1250 [==============================] - 33s 26ms/step - loss: 1.3272 - accuracy: 0.6
546 - val_loss: 1.0600 - val_accuracy: 0.7383 - lr: 4.0657e-04
Epoch 20/200
1249/1250 [============================>.] - ETA: 0s - loss: 1.2939 - accuracy: 0.6647
Epoch 20: val_loss did not improve from 1.05067
1250/1250 [==============================] - 33s 27ms/step - loss: 1.2940 - accuracy: 0.6
647 - val_loss: 1.0654 - val_accuracy: 0.7355 - lr: 3.6788e-04
Epoch 21/200
1249/1250 [============================>.] - ETA: 0s - loss: 1.2688 - accuracy: 0.6639
Epoch 21: val_loss did not improve from 1.05067
1250/1250 [==============================] - 33s 26ms/step - loss: 1.2687 - accuracy: 0.6
640 - val_loss: 1.0940 - val_accuracy: 0.7242 - lr: 3.3287e-04
Epoch 22/200
1249/1250 [============================>.] - ETA: 0s - loss: 1.2488 - accuracy: 0.6673
Epoch 22: val_loss improved from 1.05067 to 0.99854, saving model to best_model3.h5
1250/1250 [==============================] - 33s 26ms/step - loss: 1.2488 - accuracy: 0.6
674 - val_loss: 0.9985 - val_accuracy: 0.7503 - lr: 3.0119e-04
Epoch 23/200
1249/1250 [============================>.] - ETA: 0s - loss: 1.2227 - accuracy: 0.6740
Epoch 23: val_loss did not improve from 0.99854
1250/1250 [==============================] - 33s 26ms/step - loss: 1.2226 - accuracy: 0.6
741 - val_loss: 1.0481 - val_accuracy: 0.7249 - lr: 2.7253e-04
Epoch 24/200
1249/1250 [============================>.] - ETA: 0s - loss: 1.1959 - accuracy: 0.6801
Epoch 24: val_loss improved from 0.99854 to 0.96532, saving model to best_model3.h5
1250/1250 [==============================] - 33s 26ms/step - loss: 1.1957 - accuracy: 0.6
801 - val_loss: 0.9653 - val_accuracy: 0.7549 - lr: 2.4660e-04
Epoch 25/200
1250/1250 [==============================] - ETA: 0s - loss: 1.1763 - accuracy: 0.6842
Epoch 25: val_loss did not improve from 0.96532
1250/1250 [==============================] - 33s 27ms/step - loss: 1.1763 - accuracy: 0.6
842 - val_loss: 0.9741 - val_accuracy: 0.7479 - lr: 2.2313e-04
Epoch 26/200
1250/1250 [==============================] - ETA: 0s - loss: 1.1451 - accuracy: 0.6912
Epoch 26: val_loss did not improve from 0.96532
1250/1250 [==============================] - 33s 26ms/step - loss: 1.1451 - accuracy: 0.6
912 - val_loss: 0.9684 - val_accuracy: 0.7436 - lr: 2.0190e-04
Epoch 27/200
1249/1250 [============================>.] - ETA: 0s - loss: 1.1282 - accuracy: 0.6939
Epoch 27: val_loss improved from 0.96532 to 0.90483, saving model to best_model3.h5
1250/1250 [==============================] - 33s 26ms/step - loss: 1.1281 - accuracy: 0.6
939 - val_loss: 0.9048 - val_accuracy: 0.7680 - lr: 1.8268e-04
Epoch 28/200
1250/1250 [==============================] - ETA: 0s - loss: 1.1142 - accuracy: 0.6944
Epoch 28: val_loss did not improve from 0.90483
1250/1250 [==============================] - 33s 26ms/step - loss: 1.1142 - accuracy: 0.6
944 - val_loss: 0.9417 - val_accuracy: 0.7496 - lr: 1.6530e-04
Epoch 29/200
1249/1250 [============================>.] - ETA: 0s - loss: 1.1057 - accuracy: 0.6948
Epoch 29: val_loss improved from 0.90483 to 0.86827, saving model to best_model3.h5
1250/1250 [==============================] - 33s 26ms/step - loss: 1.1056 - accuracy: 0.6
948 - val_loss: 0.8683 - val_accuracy: 0.7761 - lr: 1.4957e-04
Epoch 30/200
1248/1250 [============================>.] - ETA: 0s - loss: 1.0818 - accuracy: 0.7020
Epoch 30: val_loss improved from 0.86827 to 0.84073, saving model to best_model3.h5
1250/1250 [==============================] - 33s 26ms/step - loss: 1.0819 - accuracy: 0.7
020 - val_loss: 0.8407 - val_accuracy: 0.7792 - lr: 1.3534e-04
Epoch 31/200
1250/1250 [==============================] - ETA: 0s - loss: 1.0712 - accuracy: 0.7041
Epoch 31: val_loss improved from 0.84073 to 0.83603, saving model to best_model3.h5
1250/1250 [==============================] - 33s 26ms/step - loss: 1.0712 - accuracy: 0.7
041 - val_loss: 0.8360 - val_accuracy: 0.7800 - lr: 1.2246e-04
Epoch 32/200
1248/1250 [============================>.] - ETA: 0s - loss: 1.0576 - accuracy: 0.7070
Epoch 32: val_loss did not improve from 0.83603
1250/1250 [==============================] - 33s 26ms/step - loss: 1.0576 - accuracy: 0.7
070 - val_loss: 0.8801 - val_accuracy: 0.7643 - lr: 1.1080e-04
Epoch 33/200
1249/1250 [============================>.] - ETA: 0s - loss: 1.0536 - accuracy: 0.7065
Epoch 33: val_loss improved from 0.83603 to 0.82786, saving model to best_model3.h5
1250/1250 [==============================] - 33s 26ms/step - loss: 1.0537 - accuracy: 0.7
```

```
064 - val_loss: 0.8279 - val_accuracy: 0.7802 - lr: 1.0026e-04
Epoch 34/200
1249/1250 [============================>.] - ETA: 0s - loss: 1.0346 - accuracy: 0.7128
Epoch 34: val_loss did not improve from 0.82786
1250/1250 [==============================] - 33s 26ms/step - loss: 1.0346 - accuracy: 0.7
126 - val_loss: 0.8291 - val_accuracy: 0.7777 - lr: 9.0718e-05
Epoch 35/200
1248/1250 [============================>.] - ETA: 0s - loss: 1.0193 - accuracy: 0.7134
Epoch 35: val_loss improved from 0.82786 to 0.79306, saving model to best_model3.h5
1250/1250 [==============================] - 33s 26ms/step - loss: 1.0194 - accuracy: 0.7
135 - val_loss: 0.7931 - val_accuracy: 0.7882 - lr: 8.2085e-05
Epoch 36/200
1250/1250 [==============================] - ETA: 0s - loss: 1.0133 - accuracy: 0.7148
Epoch 36: val_loss did not improve from 0.79306
1250/1250 [==============================] - 33s 26ms/step - loss: 1.0133 - accuracy: 0.7
148 - val_loss: 0.8269 - val_accuracy: 0.7743 - lr: 7.4274e-05
Epoch 37/200
1249/1250 [============================>.] - ETA: 0s - loss: 1.0103 - accuracy: 0.7164
Epoch 37: val_loss did not improve from 0.79306
1250/1250 [==============================] - 33s 26ms/step - loss: 1.0104 - accuracy: 0.7
163 - val_loss: 0.8361 - val_accuracy: 0.7749 - lr: 6.7206e-05
Epoch 38/200
1250/1250 [==============================] - ETA: 0s - loss: 0.9969 - accuracy: 0.7196
Epoch 38: val_loss did not improve from 0.79306
1250/1250 [==============================] - 32s 26ms/step - loss: 0.9969 - accuracy: 0.7
196 - val_loss: 0.8652 - val_accuracy: 0.7615 - lr: 6.0810e-05
Epoch 39/200
1248/1250 [============================>.] - ETA: 0s - loss: 0.9879 - accuracy: 0.7191
Epoch 39: val_loss improved from 0.79306 to 0.79157, saving model to best_model3.h5
1250/1250 [==============================] - 33s 26ms/step - loss: 0.9880 - accuracy: 0.7
192 - val_loss: 0.7916 - val_accuracy: 0.7858 - lr: 5.5023e-05
Epoch 40/200
1248/1250 [============================>.] - ETA: 0s - loss: 0.9808 - accuracy: 0.7241
Epoch 40: val_loss improved from 0.79157 to 0.75789, saving model to best_model3.h5
1250/1250 [==============================] - 33s 26ms/step - loss: 0.9809 - accuracy: 0.7
241 - val_loss: 0.7579 - val_accuracy: 0.7979 - lr: 4.9787e-05
Epoch 41/200
1250/1250 [==============================] - ETA: 0s - loss: 0.9795 - accuracy: 0.7208
Epoch 41: val_loss did not improve from 0.75789
1250/1250 [==============================] - 33s 26ms/step - loss: 0.9795 - accuracy: 0.7
208 - val_loss: 0.7605 - val_accuracy: 0.7964 - lr: 4.5049e-05
Epoch 42/200
1249/1250 [============================>.] - ETA: 0s - loss: 0.9719 - accuracy: 0.7215
Epoch 42: val_loss did not improve from 0.75789
1250/1250 [==============================] - 33s 26ms/step - loss: 0.9719 - accuracy: 0.7
214 - val_loss: 0.7787 - val_accuracy: 0.7873 - lr: 4.0762e-05
Epoch 43/200
1248/1250 [============================>.] - ETA: 0s - loss: 0.9617 - accuracy: 0.7260
Epoch 43: val_loss improved from 0.75789 to 0.73982, saving model to best_model3.h5
1250/1250 [==============================] - 33s 26ms/step - loss: 0.9621 - accuracy: 0.7
259 - val_loss: 0.7398 - val_accuracy: 0.8026 - lr: 3.6883e-05
Epoch 44/200
1248/1250 [============================>.] - ETA: 0s - loss: 0.9606 - accuracy: 0.7256
Epoch 44: val_loss did not improve from 0.73982
1250/1250 [==============================] - 33s 26ms/step - loss: 0.9602 - accuracy: 0.7
257 - val_loss: 0.7530 - val_accuracy: 0.7943 - lr: 3.3373e-05
Epoch 45/200
1250/1250 [==============================] - ETA: 0s - loss: 0.9595 - accuracy: 0.7268
Epoch 45: val_loss did not improve from 0.73982
1250/1250 [==============================] - 33s 26ms/step - loss: 0.9595 - accuracy: 0.7
268 - val_loss: 0.7564 - val_accuracy: 0.7937 - lr: 3.0197e-05
Epoch 46/200
1250/1250 [==============================] - ETA: 0s - loss: 0.9531 - accuracy: 0.7278
Epoch 46: val_loss did not improve from 0.73982
1250/1250 [==============================] - 33s 26ms/step - loss: 0.9531 - accuracy: 0.7
278 - val_loss: 0.7872 - val_accuracy: 0.7838 - lr: 2.7324e-05
Epoch 47/200
1249/1250 [============================>.] - ETA: 0s - loss: 0.9497 - accuracy: 0.7286
Epoch 47: val_loss did not improve from 0.73982
1250/1250 [==============================] - 33s 26ms/step - loss: 0.9494 - accuracy: 0.7
287 - val_loss: 0.7637 - val_accuracy: 0.7906 - lr: 2.4724e-05
Epoch 48/200
```

```
1248/1250 [============================>.] - ETA: 0s - loss: 0.9501 - accuracy: 0.7301
Epoch 48: val_loss did not improve from 0.73982
1250/1250 [==============================] - 33s 26ms/step - loss: 0.9501 - accuracy: 0.7
301 - val_loss: 0.7448 - val_accuracy: 0.7959 - lr: 2.2371e-05
Epoch 49/200
1249/1250 [============================>.] - ETA: 0s - loss: 0.9440 - accuracy: 0.7301
Epoch 49: val_loss did not improve from 0.73982
1250/1250 [==============================] - 33s 26ms/step - loss: 0.9441 - accuracy: 0.7
300 - val_loss: 0.7599 - val_accuracy: 0.7903 - lr: 2.0242e-05
Epoch 50/200
1250/1250 [==============================] - ETA: 0s - loss: 0.9369 - accuracy: 0.7323
Epoch 50: val_loss did not improve from 0.73982
1250/1250 [==============================] - 33s 27ms/step - loss: 0.9369 - accuracy: 0.7
323 - val_loss: 0.7488 - val_accuracy: 0.7942 - lr: 1.8316e-05
Epoch 51/200
1250/1250 [==============================] - ETA: 0s - loss: 0.9274 - accuracy: 0.7350
Epoch 51: val_loss did not improve from 0.73982
1250/1250 [==============================] - 33s 26ms/step - loss: 0.9274 - accuracy: 0.7
350 - val_loss: 0.7478 - val_accuracy: 0.7938 - lr: 1.6573e-05
Epoch 52/200
1249/1250 [============================>.] - ETA: 0s - loss: 0.9359 - accuracy: 0.7311
Epoch 52: val_loss did not improve from 0.73982
1250/1250 [==============================] - 33s 26ms/step - loss: 0.9360 - accuracy: 0.7
311 - val_loss: 0.7556 - val_accuracy: 0.7919 - lr: 1.4996e-05
Epoch 53/200
 347/1250 [=======>......................] - ETA: 21s - loss: 0.9255 - accuracy: 0.7356
```

Começa-mos com estas transformações, mas chegamos à conclusão que era demasiada agressiva uma vez que piorou em muito a precisão do modelo... Algo que não estava à espera.

data_augmentation = Sequential([layers.RandomFlip("horizontal_and_vertical"), layers.RandomRotation(0.2), layers.RandomZoom(0.3), layers.RandomTranslation(0.1, 0.1), layers.RandomContrast(0.2), layers.RandomBrightness(0.2),])

outras: dataAug = ImageDataGenerator(rotation_range=40,
width_shift_range=0.2,
height_shift_range=0.2,
shear_range=0.2,
zoom_range=0.2,
horizontal_flip=True,
fill_mode='nearest',
brightness_range=[0.8, 1.2],
channel_shift_range=0.2,
)

Epoch 47/100 625/625 [==============================] - ETA: 0s - loss: 1.1264 - accuracy: 0.6619

Então decidimos diminuir a intensidade das alterações.

data_augmentation = keras.Sequential([layers.RandomFlip("horizontal"), layers.RandomRotation(0.1), layers.RandomZoom(0.2),])

Test loss: 0.8087025117874146 Test accuracy: 0.7913000273704529

ANALISE DE RESULTADOS **sobre S sem data augmentation**

In [2]:

```python
from keras.models import load_model
model = load_model('/content/best_modelS_WITHOUT_64.h5')
```

In [10]:

```python
import pandas as pd
import numpy as np

# usar para instalar...: pip install scikit-learn
```

```python
from sklearn import metrics
from sklearn.model_selection import train_test_split

import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense

from sklearn.preprocessing import LabelEncoder

# to install IPython, use: pip install ipython
from IPython.display import display

from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import accuracy_score

import matplotlib.pyplot as plt

from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
```

Para avaliar o desempenho do modelo usamos a matriz de confusão, esta mostra o número de verdadeiros positivos, falsos positivos, verdadeiros negativos e falsos negativos para cada classe. E também apresentamos os valores para a accuracy, esta indica o desempenho geral do modelo que consiste na proporção de previsões corretas em relação ao total de previsões feitas. Dentre todas as classificações, quantas o modelo classificou corretamente. A precisão mede a exatidão das previsões positivas do modelo. A proporção de verdadeiros positivos (TP) em relação ao total de previsões positivas (TP + FP). Recall/Revocação/Sensibilidade: A proporção de verdadeiros positivos em relação ao total de exemplos reais positivos (TP + FN). Mede a capacidade do modelo de encontrar todos os exemplos positivos. E o F1-Scor é a média harmônica entre precisão e recall. É uma métrica balanceada que considera tanto falsos positivos quanto falsos negativos.

In [11]:

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report


# Definir nomes das classes
class_names = {
    0: "Airplane",
    1: "Automobile",
    2: "Bird",
    3: "Cat",
    4: "Deer",
    5: "Dog",
    6: "Frog",
    7: "Horse",
    8: "Ship",
    9: "Truck"
}

# Prever para todo o conjunto de teste
y_pred = []
y_true = []

for x, y in test_dataset:
    predictions = model.predict(x)
    y_pred.extend(np.argmax(predictions, axis=1))
    y_true.extend(np.argmax(y, axis=1))

y_pred = np.array(y_pred)
y_true = np.array(y_true)

# Matriz de confusão
conf_matrix = confusion_matrix(y_true, y_pred)
```

```python
# Plotar a matriz de confusão
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names.values(), yticklabels=class_names.values())
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

# Relatório de classificação
class_report = classification_report(y_true, y_pred, target_names=class_names.values())
print("Classification Report:\n", class_report)
```

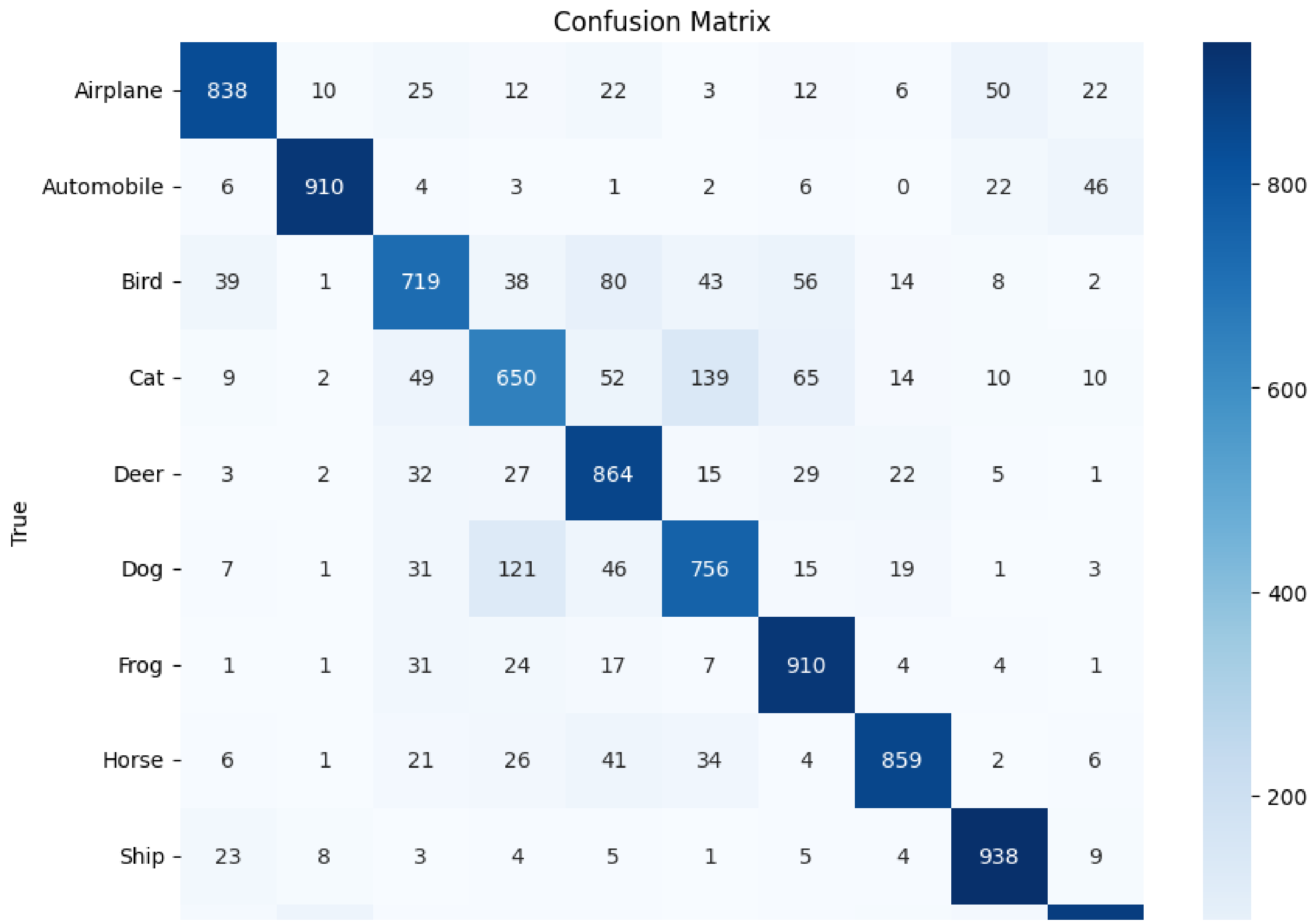
```
1/1 [==============================] - 1s 880ms/step
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 99ms/step
1/1 [==============================] - 0s 64ms/step
1/1 [==============================] - 0s 84ms/step
1/1 [==============================] - 0s 83ms/step
1/1 [==============================] - 0s 68ms/step
1/1 [==============================] - 0s 50ms/step
1/1 [==============================] - 0s 90ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 67ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 65ms/step
1/1 [==============================] - 0s 57ms/step
1/1 [==============================] - 0s 66ms/step
1/1 [==============================] - 0s 68ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 51ms/step
1/1 [==============================] - 0s 66ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 44ms/step
1/1 [==============================] - 0s 66ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 62ms/step
1/1 [==============================] - 0s 78ms/step
1/1 [==============================] - 0s 72ms/step
1/1 [==============================] - 0s 67ms/step
1/1 [==============================] - 0s 78ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 55ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 69ms/step
1/1 [==============================] - 0s 68ms/step
1/1 [==============================] - 0s 81ms/step
1/1 [==============================] - 0s 45ms/step
1/1 [==============================] - 0s 82ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 87ms/step
1/1 [==============================] - 0s 79ms/step
1/1 [==============================] - 0s 69ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 45ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 58ms/step
1/1 [==============================] - 0s 52ms/step
1/1 [==============================] - 0s 53ms/step
1/1 [==============================] - 0s 57ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 59ms/step
```

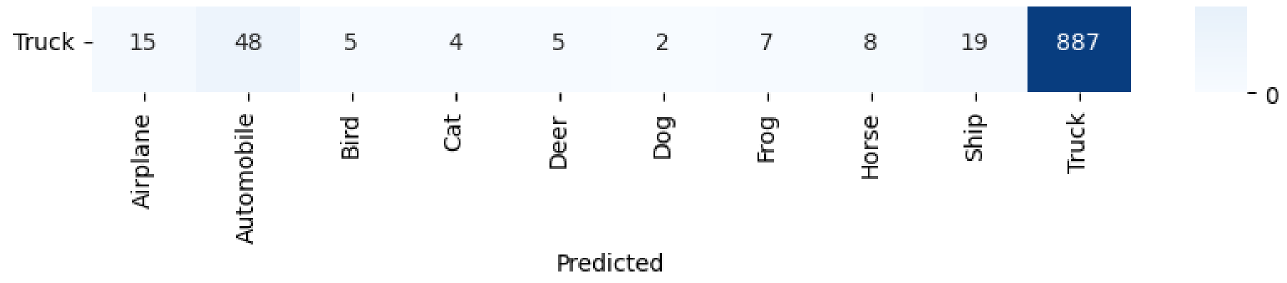
```
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 69ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 51ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 46ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 52ms/step
1/1 [==============================] - 0s 72ms/step
1/1 [==============================] - 0s 52ms/step
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 46ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 44ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 55ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 66ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 52ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 58ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 53ms/step
1/1 [==============================] - 0s 46ms/step
1/1 [==============================] - 0s 48ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 57ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 27ms/step
```

```
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 46ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 46ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 68ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 48ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 138ms/step
1/1 [==============================] - 0s 54ms/step
1/1 [==============================] - 0s 48ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 45ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 46ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 48ms/step
1/1 [==============================] - 0s 41ms/step
```

```
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 57ms/step
1/1 [==============================] - 0s 46ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 45ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 41ms/step
```

```
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 44ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 48ms/step
1/1 [==============================] - 0s 48ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 44ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 48ms/step
1/1 [==============================] - 0s 46ms/step
1/1 [==============================] - 0s 53ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 45ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 45ms/step
1/1 [==============================] - 0s 44ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 232ms/step
```



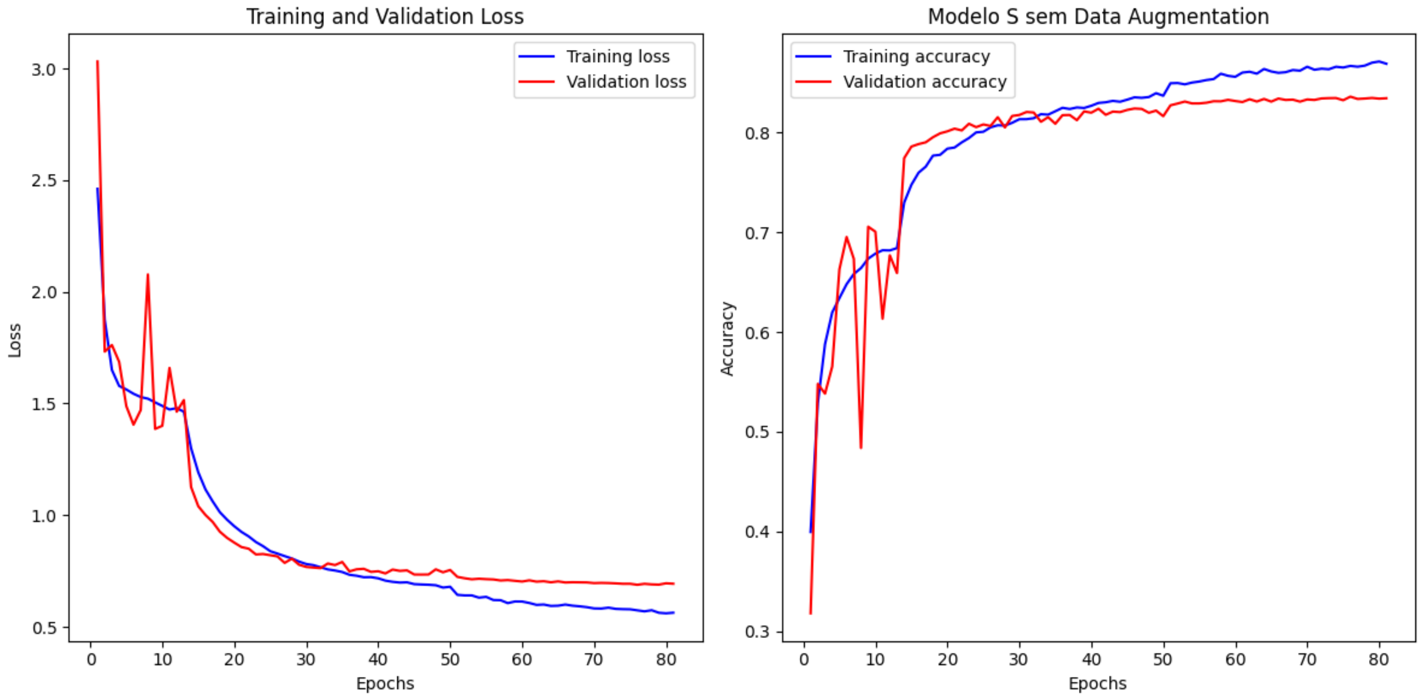



```
Classification Report:
              precision    recall  f1-score   support

    Airplane       0.88      0.84      0.86      1000
  Automobile       0.92      0.91      0.92      1000
        Bird       0.78      0.72      0.75      1000
         Cat       0.72      0.65      0.68      1000
        Deer       0.76      0.86      0.81      1000
         Dog       0.75      0.76      0.76      1000
        Frog       0.82      0.91      0.86      1000
       Horse       0.90      0.86      0.88      1000
        Ship       0.89      0.94      0.91      1000
       Truck       0.90      0.89      0.89      1000

    accuracy                           0.83     10000
   macro avg       0.83      0.83      0.83     10000
weighted avg       0.83      0.83      0.83     10000
```

Agora com o objetivo de mobnitorizar o processo de treino e verificar se o modelo está a aprender corretamente ao longo do tempo. Usamos as curvas de perda de treino (loss) e de validação (val_loss) ao longo das épocas. E a curva de accuracy de treino (acc) e de validação (val_acc) ao longo das épocas. Estas permitem por exemplo ao comparar as curvas de treino e validação, é possível identificar problemas de overfitting (onde a perda de validação aumenta enquanto a perda de treino diminui) ou underfitting (onde ambas as perdas são altas). Com base nas curvas, ajustes podem ser feitos aos hiperparâmetros do modelo, como a taxa de aprendizagem, o número de épocas, e outros.

In [12]:

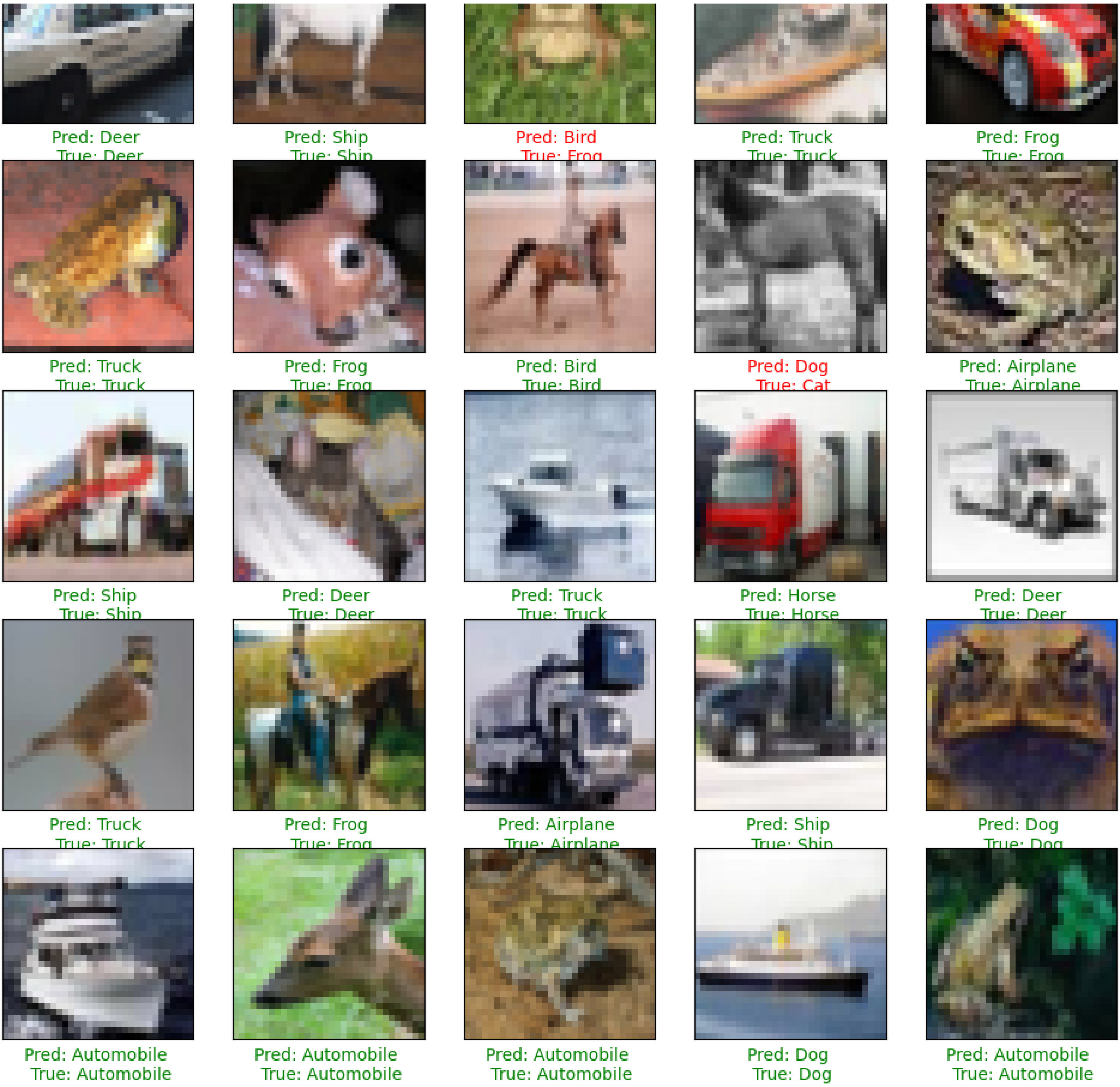
```python
def graph(history, title):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs = range(1, len(acc) + 1)

    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.plot(epochs, loss, 'b', label='Training loss')
    plt.plot(epochs, val_loss, 'r', label='Validation loss')
    plt.title('Training and Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(epochs, acc, 'b', label='Training accuracy')
    plt.plot(epochs, val_acc, 'r', label='Validation accuracy')
    plt.title(title)
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.tight_layout()
    plt.show()

graph(history, "Modelo S sem Data Augmentation")
```



Para visualizar exemplos de imagens do conjunto de dados, juntamente com as previsões feitas pelo modelo. Com o objetivo de permitir uma análise visual rápida e intuitiva do desempenho do modelo.

In [13]:

```
import matplotlib.pyplot as plt
import numpy as np

# Assuming you have already trained your model and obtained predictions and true labels
# Example lists for demonstration


# Extract images and labels from test_dataset
test_images = []
test_labels = []

for images, labels in test_dataset:
    test_images.append(images.numpy())  # Assuming you convert images to numpy arrays
    test_labels.append(labels.numpy())

test_images = np.concatenate(test_images)
test_labels = np.concatenate(test_labels)

# Normalize the images to [0, 1]
test_images = test_images.astype(np.float32) / 255.0

# Display images with predictions and true labels
plt.figure(figsize=(12, 12))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(test_images[i], cmap=plt.cm.binary)  # Display the ith image
    predicted_label = class_names[y_pred[i]]
    true_label = class_names[y_true[i]]
    if predicted_label == true_label:
        color = 'green'
    else:
        color = 'red'
    plt.xlabel(f'Pred: {predicted_label} \n True: {true_label}', color=color)

plt.show()
```



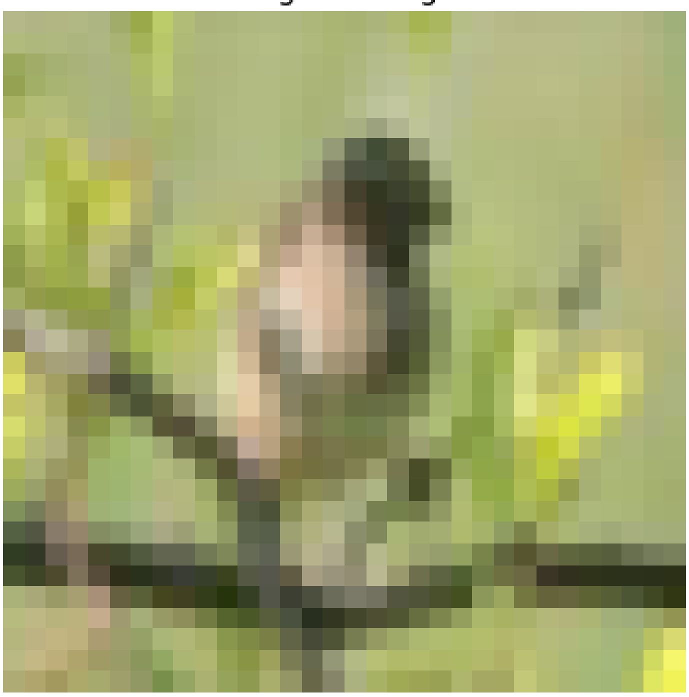
Com o objetivo de entender como as camadas convolucionais e de pooling do modelo de CNN processam e extraem as características das imagens de entrada ao longo da rede neural usámos a visualização de feature maps. É útil para " o modelo, verificar se as características esperadas estão a ser aprendidas e entender como as informações são transformadas e representadas nas diferentes camadas da rede.

In [16]:

```
import matplotlib.pyplot as plt
from keras.models import Model
from tensorflow.keras.utils import load_img, img_to_array
import numpy as np
import random

# Assuming you have defined your test_dataset and model somewhere

# Get the first image from test_dataset
first_batch = next(iter(test_dataset))  # Get the first batch from the dataset
first_image = first_batch[0][0]  # Extract the first image from the batch

# Display the original image
plt.imshow(first_image.numpy().astype("uint8"))
plt.title('Original Image')
plt.axis('off')
plt.show()
```

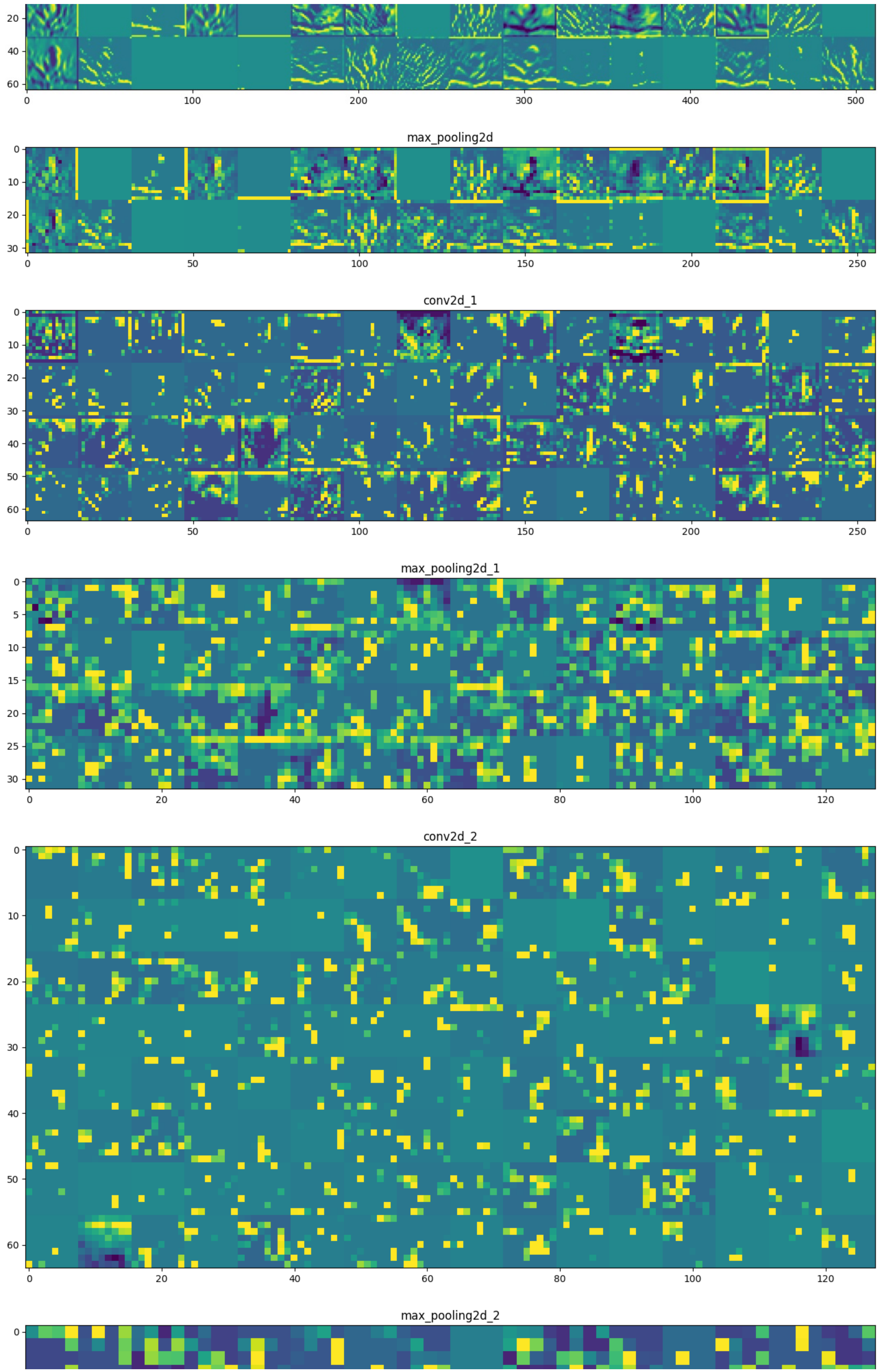
```python
# Preprocess the image for visualization
img = first_image.numpy()
img = np.expand_dims(img, axis=0)  # Now img has shape (1, 32, 32, 3)

def visualize_filters(model, img):
    layer_outputs = [layer.output for layer in model.layers if 'conv' in layer.name or 'pool' in layer.name]

    activation_model = Model(inputs=model.input, outputs=layer_outputs)

    activations = activation_model.predict(img)

    layer_names = [layer.name for layer in model.layers if 'conv' in layer.name or 'pool' in layer.name]

    for layer_name, layer_activation in zip(layer_names, activations):
        n_features = layer_activation.shape[-1]
        size = layer_activation.shape[1]

        n_cols = n_features // 16  # Number of columns in the grid
        display_grid = np.zeros((size * n_cols, size * 16))

        for col in range(n_cols):
            for row in range(16):
                channel_image = layer_activation[0, :, :, col * 16 + row]
                channel_image -= channel_image.mean()
                channel_image /= (channel_image.std() + 1e-5)
                channel_image *= 64
                channel_image += 128
                channel_image = np.clip(channel_image, 0, 255).astype('uint8')
                display_grid[col * size : (col + 1) * size, row * size : (row + 1) * size] = channel_image

        scale = 1. / size
        plt.figure(figsize=(scale * display_grid.shape[1], scale * display_grid.shape[0]))
        plt.title(layer_name)
        plt.grid(False)
        plt.imshow(display_grid, aspect='auto', cmap='viridis')

# Visualize filters for the model and the first image
visualize_filters(model, img)
```

Original Image

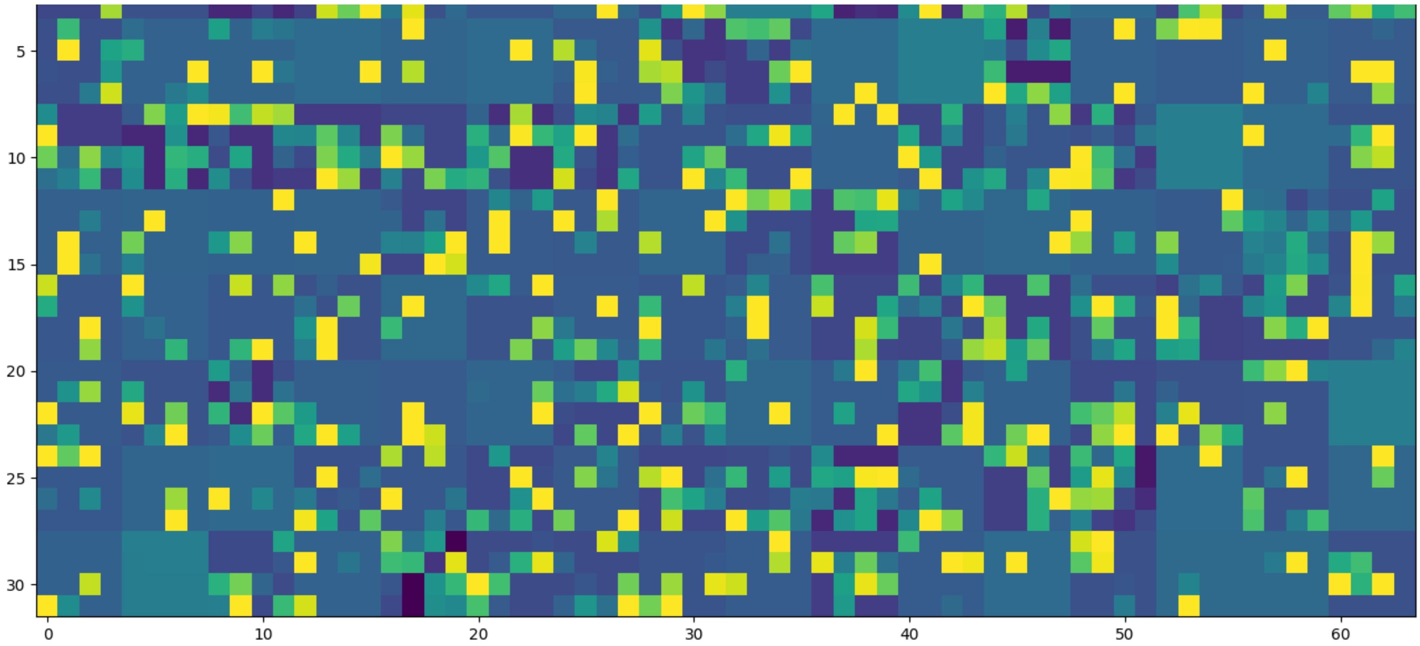


```
1/1 [==============================] - 0s 195ms/step
```

conv2d



max_pooling2d
conv2d_1
max_pooling2d_1
conv2d_2
max_pooling2d_2

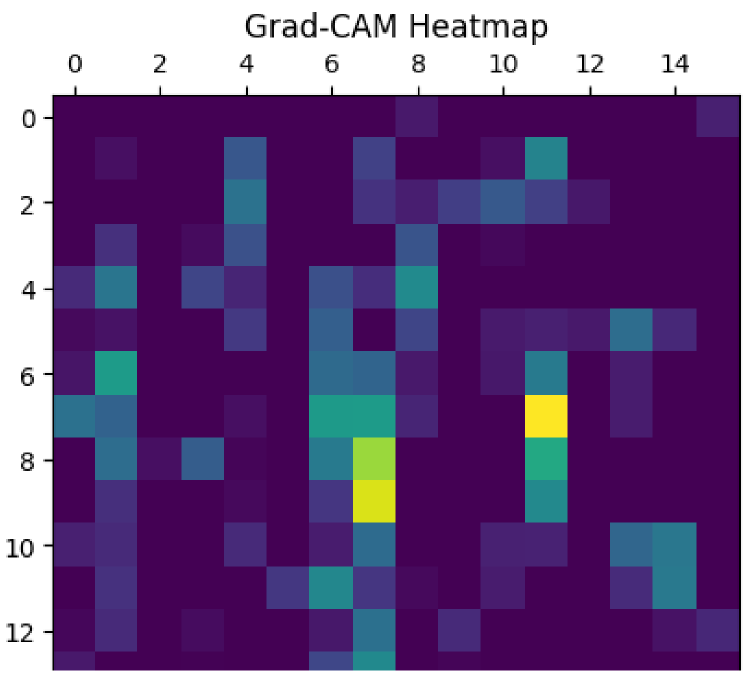


Para complementar usamos heat maps (mapas de calor) para visualizar as regiões de uma imagem que são mais importantes para a decisão de classificação feita pelo modelo. Este mapa mostra visualmente onde o modelo está a "focar" mais para fazer uma decisão sobre a classe da imagem. Isto é útil para interpretar e entender quais características da imagem que estão a ser consideradas mais relevantes pelo modelo durante o processo de classificação.

In [21]:

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import random
import cv2

# Assuming test_dataset and model are already defined

random_index = random.randint(0, 500)

first_batch = next(iter(test_dataset))  # Get the first batch from the dataset
img = first_batch[0][0]  # Extract the first image from the batch

img_array = img.numpy()
img_array = np.expand_dims(img_array, axis=0)  # Now img_array has shape (1, 32, 32, 3)

plt.imshow(img_array[0].astype("uint8"))  # Display the original image
plt.title('Original Image')
plt.axis('off')
plt.show()

def make_gradcam_heatmap(img_array, model, last_conv_layer_name, pred_index=None):
    grad_model = tf.keras.models.Model(
        [model.inputs], [model.get_layer(last_conv_layer_name).output, model.output]
    )

    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(preds[0])
        class_channel = preds[:, pred_index]

    grads = tape.gradient(class_channel, last_conv_layer_output)
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
    last_conv_layer_output = last_conv_layer_output[0]
    heatmap = last_conv_layer_output @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)

    heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
    return heatmap.numpy()
```

```python
last_conv_layer_name = 'conv2d_1'  # Replace with your actual last conv layer name
heatmap = make_gradcam_heatmap(img_array, model, last_conv_layer_name)

plt.matshow(heatmap)
plt.title('Grad-CAM Heatmap')
plt.show()

def superimpose_heatmap(img, heatmap, alpha=0.4):
    heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))
    heatmap = np.uint8(255 * heatmap)
    heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)
    superimposed_img = heatmap * alpha + img
    return np.uint8(superimposed_img)

# Convert img_array to (32, 32, 3) for superimpose_heatmap function
img_for_superimpose = img_array[0].astype("uint8")

superimposed_img = superimpose_heatmap(img_for_superimpose, heatmap)

plt.imshow(superimposed_img)
plt.title('Image with Grad-CAM Heatmap')
plt.axis('off')
plt.show()
```



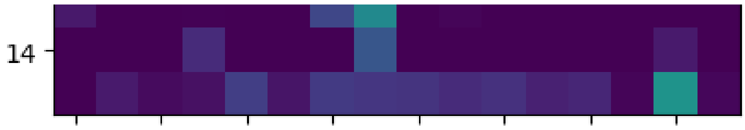


Image with Grad-CAM Heatmap

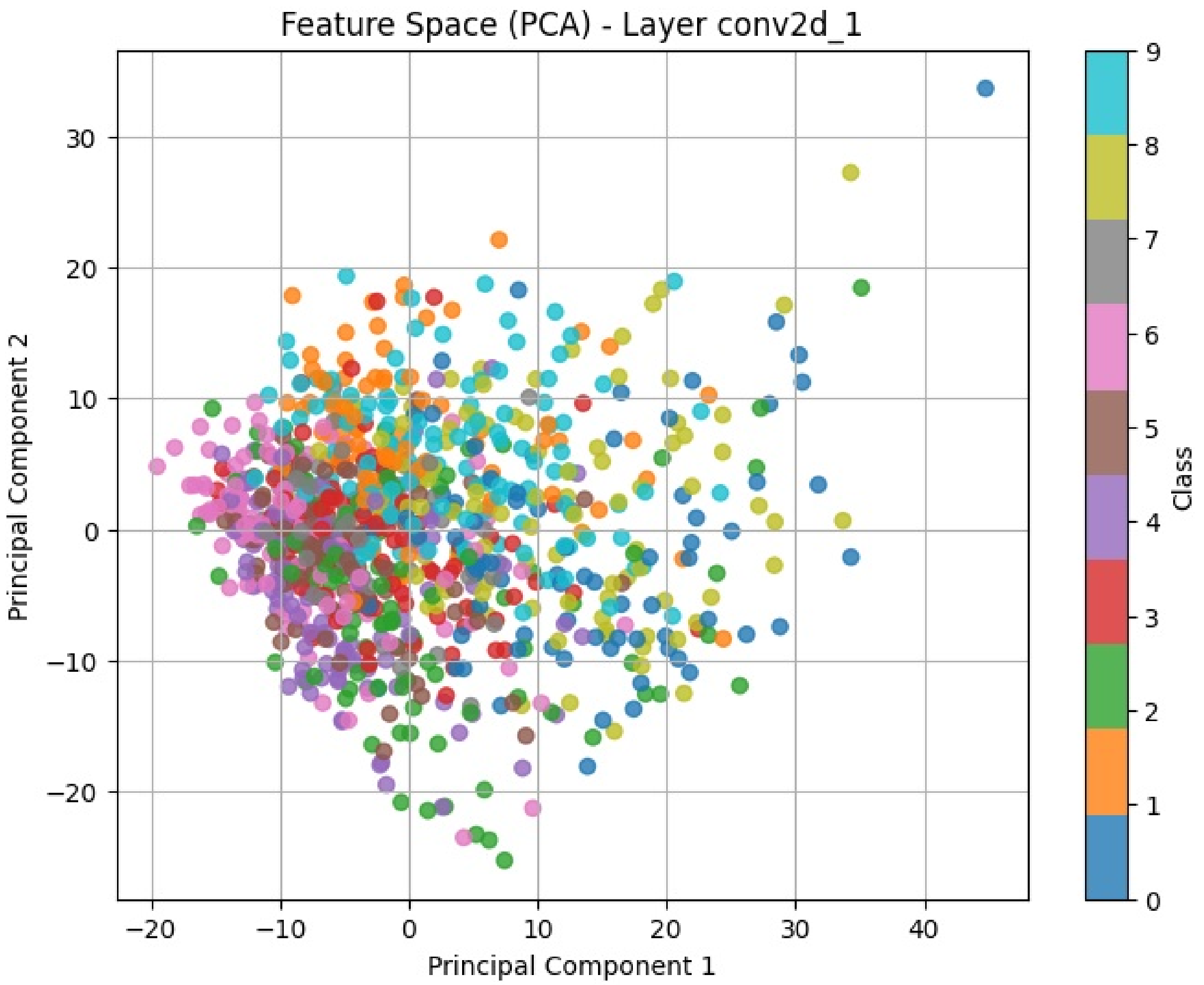
O Feature Space ajuda a verificar como as características são distribuídas no espaço, ao revelar padrões ou agrupamentos que podem indicar a capacidade do modelo de discriminar entre as diferentes classes. Ao comparar os gráficos entre as camadas, podemos ver como estas (por exemplo, camadas convolucionais, em comparação com as camadas densas) aprendem e representam as características das imagens.

In [19]:

```
# Function to extract images and labels from a tf.data.Dataset
from sklearn.decomposition import PCA

def get_images_and_labels(dataset, num_samples):
    dataset = dataset.unbatch().take(num_samples)
    images = []
    labels = []
    for img, label in dataset:
        images.append(img.numpy())
        labels.append(label.numpy())
    return np.array(images), np.array(labels)

# Extract images and labels from the test dataset
x_test, y_test = get_images_and_labels(test_dataset, num_samples=1000)

# Visualization function
def visualize_feature_space(layer_name, x_data, y_data):
    feature_extractor = tf.keras.models.Model(inputs=model.input, outputs=model.get_laye
r(layer_name).output)

    num_samples = len(x_data)
    sample_indices = np.random.choice(len(x_data), num_samples, replace=False)
    sample_images = x_data[sample_indices]

    features = feature_extractor.predict(sample_images)

    pca = PCA(n_components=2)
    features_reduced = pca.fit_transform(features.reshape(num_samples, -1))

    plt.figure(figsize=(8, 6))
    plt.scatter(features_reduced[:, 0], features_reduced[:, 1], c=np.argmax(y_data[sampl
e_indices], axis=1), cmap='tab10', marker='o', alpha=0.8)
```

```
    plt.colorbar(label='Class')
    plt.title(f'Feature Space (PCA) - Layer {layer_name}')
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.grid(True)
    plt.show()

# Assuming you have x_test and y_test as your test dataset
layer_name = 'conv2d_1'
visualize_feature_space(layer_name, x_test, y_test)
```

```
32/32 [==============================] - 0s 4ms/step
```



Para mostrar a arquitetura do modelo

In [17]:

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0

 rescaling (Rescaling)       (None, 32, 32, 3)         0

 conv2d (Conv2D)             (None, 32, 32, 32)        896

 batch_normalization (BatchN  (None, 32, 32, 32)       128
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 dropout (Dropout)           (None, 16, 16, 32)        0

 conv2d_1 (Conv2D)           (None, 16, 16, 64)        18496
```

```
 batch_normalization_1 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 8, 8, 64)          0

 conv2d_2 (Conv2D)           (None, 8, 8, 128)         73856

 batch_normalization_2 (Batc  (None, 8, 8, 128)        512
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 dropout_2 (Dropout)         (None, 4, 4, 128)         0

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 256)               524544

 batch_normalization_3 (Batc  (None, 256)              1024
 hNormalization)

 dropout_3 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 128)               32896

 batch_normalization_4 (Batc  (None, 128)              512
 hNormalization)

 dropout_4 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 10)                1290

=================================================================
Total params: 654,410
Trainable params: 653,194
Non-trainable params: 1,216
_________________________________________________________________
```