"Training a CNN using Keras "

Deep Learning - Lab 04

MEKKI Meriem
MESSIOUD Mohammed Amir

ESI SBA - April 2024

Sommaire

1. Introduction

In the realm of computer vision, Convolutional Neural Networks (CNNs) reign supreme, offering unparalleled capabilities in image analysis and classification. In this Lab we used the Keras framewrok to train a CNN on the famous CIFAR-10 dataset, a cornerstone in image classification research, comprising 60,000 32x32 color images across 10 distinct classes.

2. Data Loading

We utilized the Keras framework's convenient function load_data() to seamlessly load the CIFAR-10 dataset into our program. This single line of code

```
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
])
```

efficiently fetches the training and validation data.

3. Preprocessing

For pre-processing we used the Rescaling layer to normalize the data (transform the pixel values to a range between 0 and 1).

4. Define the model architecure: CNN

This CNN architecture begins with a preprocessing step, rescaling the input images to a range between 0 and 1. It then employs alternating convolutional layers, with the first two layers extracting low-level features and the subsequent layers refining these features. We utilized Max-pooling layers to reduce spatial dimensions and extract dominant features. Following additional convolutional layers for higher-level feature extraction, the flatten layer reshapes the output for the final fully connected layer. This dense layer, with softmax activation, produces class probabilities, enabling accurate classification into one of the predefined classes.

Here is the exact architecture:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 32, 32, 3)         0

 conv2d (Conv2D)             (None, 30, 30, 16)        448

 conv2d_1 (Conv2D)           (None, 28, 28, 16)        2320

 max_pooling2d (MaxPooling2  (None, 14, 14, 16)        0
 D)

 conv2d_2 (Conv2D)           (None, 12, 12, 32)        4640

 conv2d_3 (Conv2D)           (None, 10, 10, 32)        9248

 max_pooling2d_1 (MaxPoolin  (None, 5, 5, 32)          0
 g2D)

 flatten (Flatten)           (None, 800)               0

 dense (Dense)               (None, 10)                8010

=================================================================
Total params: 24666 (96.35 KB)
Trainable params: 24666 (96.35 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 1: Model architecture

5. Fit the neural network for the training data

To train the neural network on the CIFAR-10 dataset, we employed the Adam optimizer with its default settings, We set a batch size of 64. Additionally, we utilized accuracy as a metric, categorical cross-entropy as loss function, and trained the model for 5 epochs.

Here are the first results we obtained:

at Epoch 5/5 loss: 1.0118 - accuracy: 0.6464 - val_loss: 1.0134 - val_accuracy: 0.6444

Running time : 80.61359524726868 seconds

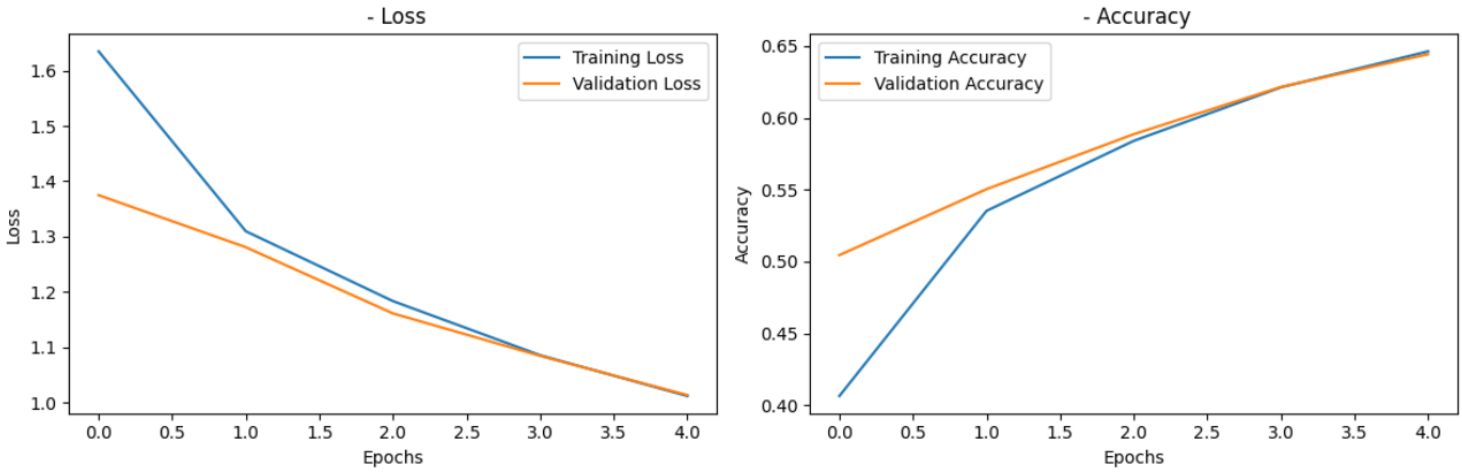

Figure 2: Loss and Accuracy for the 1st Model

6. Improving the previous architecure

For the purpose for improving the performance of this architecure we tried some approaches such as adding dropout layers and data augmentation, then we added depth to the model to acheive 70% accuracy

Here is the final architecture we utilized:

Layer (type)	**Output Shape**	**Param #**	**Trainable**
rescaling_3 (Rescaling)	(None, 32, 32, 3)	0	
conv2d_12 (Conv2D)	(None, 32, 32, 32)	896	
batch_normalization (BatchNormalization)	(None, 32, 32, 32)	128	
conv2d_13 (Conv2D)	(None, 32, 32, 32)	9248	
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 32)	128	
max_pooling2d_6 (MaxPooling2D)	(None, 16, 16, 32)	0	
dropout_4 (Dropout)	(None, 16, 16, 32)	0	
conv2d_14 (Conv2D)	(None, 16, 16, 64)	18496	
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 64)	256	
conv2d_15 (Conv2D)	(None, 16, 16, 64)	36928	
batch_normalization_3 (BatchNormalization)	(None, 16, 16, 64)	256	
max_pooling2d_7 (MaxPooling2D)	(None, 8, 8, 64)	0	
dropout_5 (Dropout)	(None, 8, 8, 64)	0	
conv2d_16 (Conv2D)	(None, 8, 8, 128)	73856	
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 128)	512	
conv2d_17 (Conv2D)	(None, 8, 8, 128)	147584	
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 128)	512	
max_pooling2d_8 (MaxPooling2D)	(None, 4, 4, 128)	0	
dropout_6 (Dropout)	(None, 4, 4, 128)	0	
flatten_3 (Flatten)	(None, 2048)	0	
dense_3 (Dense)	(None, 128)	262272	
batch_normalization_6 (BatchNormalization)	(None, 128)	512	
dropout_7 (Dropout)	(None, 128)	0	
dense_4 (Dense)	(None, 10)	1290	
Total params: 552874 (2.11 MB)			
Trainable params: 551722 (2.10 MB)			
Non-trainable params: 1152 (4.50 KB)			

Table 1: Final Model Summary

Here are the results we obtained:

Epoch 5/5 loss: 0.8121 - accuracy: 0.7174 - val_loss: 0.7499 - val_accuracy: 0.7417

Running time : 470.0587821006775 seconds

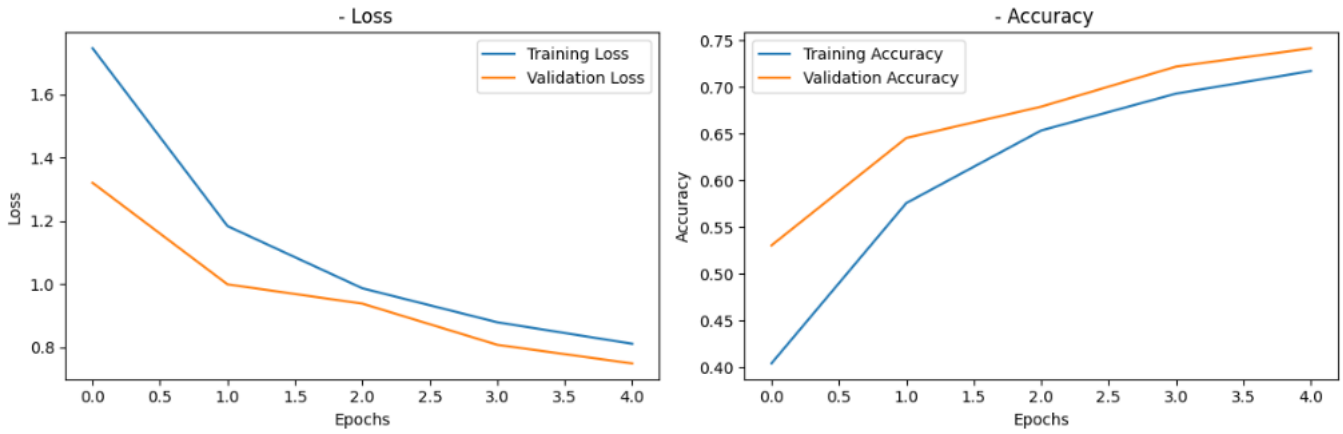

Figure 3: Accuracy and Loss for the Final Model