UNIVERSITÀ DEGLI STUDI DI MILANO-BICOCCA
MSC IN DATA SCIENCE
A.Y. 2023-2024

CLASSIFYING IMAGES USING A CNN

Course in Foundation of Deep Learning

By Andrea Cardinali

UNIVERSITÀ DEGLI STUDI DI MILANO-BICOCCA
MSC IN DATA SCIENCE
A.Y. 2023-2024

Team presentation

Andrea Cardinali
Student ID: 911556
1st year student of MSc in Data Science

Aim of this project

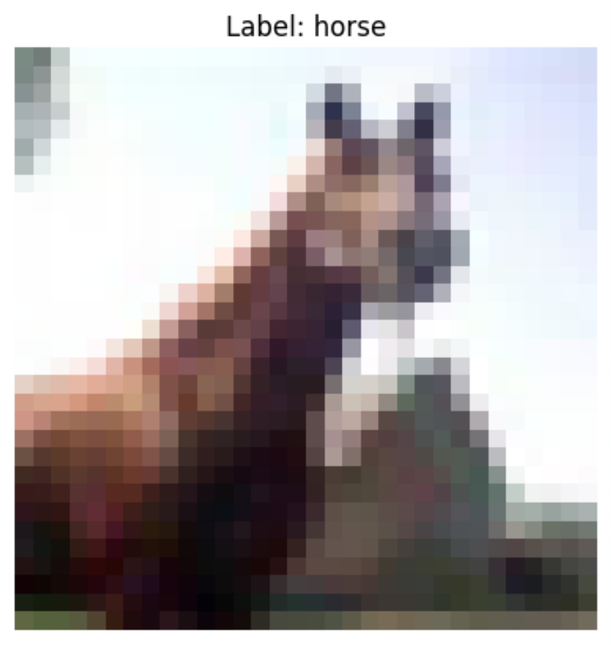
Given a set of 10.000 images, the aim is to predict the category of objects to which each photo belongs.

The photo are divided into ten categories:

- airplane
- automobile
- bird
- cat
- deer
- dog
- frog
- horse
- ship
- truck

CIFAR-10 Dataset

The CIFAR-10 dataset contains 50.000 images in the training set and 10.000 images in the test set.



All the images are coloured and have a dimension of 32x32 pixel, with 3 RGB channels.

Each image comes also with its label that classifies the object.

Normalisation

Both the training and test set are transformed to have float 32 value and then divided by 255, the highest pixel value. The images now take values between 0 and 1.

The two variables containing the labels of the categories are converted in a one-hot encoded matrix. For example, the label of the first category would be: [1, 0, 0, 0, 0, 0, 0, 0, 0, 0].

The reason behind this choice is that NNs perform better when the input values are small.

Quick dataset analysis

Both sets are analysed to check if the categories have all the same number of images.

The classes are balanced, i.e. there are the same number of photos for each category, both in the train and test sets.

```
Categories in training set:
Category 1 has 5000 images
Category 2 has 5000 images
Category 3 has 5000 images
Category 4 has 5000 images
Category 5 has 5000 images
Category 6 has 5000 images
Category 7 has 5000 images
Category 8 has 5000 images
Category 9 has 5000 images
Category 10 has 5000 images

Categories in test set:
Category 1 has 1000 images
Category 2 has 1000 images
Category 3 has 1000 images
Category 4 has 1000 images
Category 5 has 1000 images
Category 6 has 1000 images
Category 7 has 1000 images
Category 8 has 1000 images
Category 9 has 1000 images
Category 10 has 1000 images
```

CNN

The best way to correctly predict the images is to create and train a convolutional neural network.

First it was created a rather simple sequential model. Then, with different tries and error, the model complexity was increased until reasonable good results were obtained.

CNN

The layer schema used in this NN is the following:

- 2D Convolutional layers
- Batch normalisation layers
- ReLU activation function
- Max pooling 2D layers
- Dropout layer

CNN

A first NN was build with three convolutional layers of 32, 64 and 128 filters, a batch normalisation layer and a dropout layer with a parameter of 0.2.

During the fitting process, a batch of 32 samples is taken in each epoch.

CNN

The model did not produce satisfying results: accuracy was extremely low (≈ 0.05) and loss high.

Accuracy increased when the sample batches were removed and the learning rate lowered.

CNN

To further increase the accuracy, an l2 kernel regulariser was added in the second last dense layer to prevent overfitting and it was chosen the Adam optimiser over RMSprop.

The new architecture seems to reach better results.

CNN

The ultimate model has different convolutional layers going from 32 filters and doubling it up to 256 filters. Each Conv2D, batch normalisation and activation function layers are applied twice.
Then the MaxPool2D and the dropout layers are applied.

The dropout layer is particularly useful as it tries to prevent the overfitting of the model.

CNN

```
kernel = (3, 3)
model = Sequential()

model.add(Conv2D(filters = 32, kernel_size = kernel, input_shape = (32, 32, 3), padding = 'same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(filters = 32, kernel_size = kernel, padding = 'same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPool2D(pool_size = (2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters = 64, kernel_size = kernel, padding = 'same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(filters = 64, kernel_size = kernel, padding = 'same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPool2D(pool_size = (2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(filters = 128, kernel_size = kernel, padding = 'same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(filters = 128, kernel_size = kernel, padding = 'same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPool2D(pool_size = (2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(filters = 256, kernel_size = kernel, padding = 'same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPool2D(pool_size = (2, 2)))
model.add(Conv2D(filters = 256, kernel_size = kernel, padding = 'same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.2))
```

CNN

```
model.add(GlobalAveragePooling2D())
model.add(Dense(128, activation = 'relu', kernel_regularizer = l2(0.001)))
model.add(Dropout(0.3))
model.add(Dense(10, activation = 'softmax'))
```

Finally, a dense layer of 10 neurones with a softmax activation function is able to produce the output of the model.
This is going to compute a probability for each category.

Data Augmentation

A data augmentation phase is also performed to have more data available and get better performance.

The ImageDataGenerator is able to create new images by rotating, shifting (both horizontally and vertically) and horizontally flipping the training images.

```
new_data = ImageDataGenerator(
    rotation_range = 10,
    width_shift_range = 0.15,
    height_shift_range = 0.15,
    horizontal_flip = True)
```

Model fit

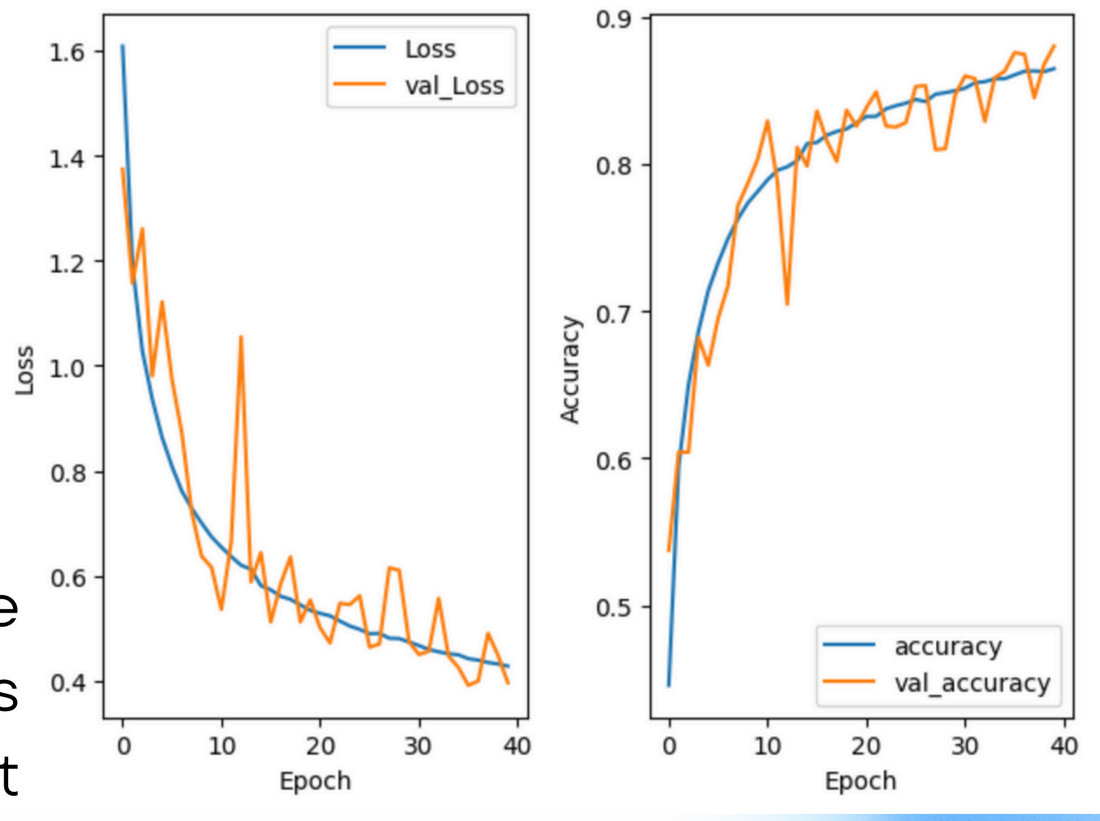
The model is trained for all the 40 epochs without calling the early stopping function.
It was used both the RMSprop and Adam optimiser with a learning rate of 0.001, however the ladder performed better.

```
Epoch 31/40
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 44s 28ms/step - accuracy: 0.8532 - loss: 0.4604 - val_accuracy: 0.8598 - val_loss: 0.4499
Epoch 32/40
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 45s 29ms/step - accuracy: 0.8554 - loss: 0.4599 - val_accuracy: 0.8579 - val_loss: 0.4559
Epoch 33/40
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 45s 29ms/step - accuracy: 0.8569 - loss: 0.4537 - val_accuracy: 0.8289 - val_loss: 0.5570
Epoch 34/40
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 79s 27ms/step - accuracy: 0.8569 - loss: 0.4497 - val_accuracy: 0.8586 - val_loss: 0.4468
Epoch 35/40
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 83s 28ms/step - accuracy: 0.8590 - loss: 0.4465 - val_accuracy: 0.8629 - val_loss: 0.4251
Epoch 36/40
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 43s 27ms/step - accuracy: 0.8598 - loss: 0.4430 - val_accuracy: 0.8756 - val_loss: 0.3911
Epoch 37/40
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 44s 28ms/step - accuracy: 0.8654 - loss: 0.4328 - val_accuracy: 0.8745 - val_loss: 0.3994
Epoch 38/40
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 43s 27ms/step - accuracy: 0.8627 - loss: 0.4314 - val_accuracy: 0.8449 - val_loss: 0.4900
```

Result

The result of the model accuracy of 87.56% is quite good. The loss value however is a little bit high.

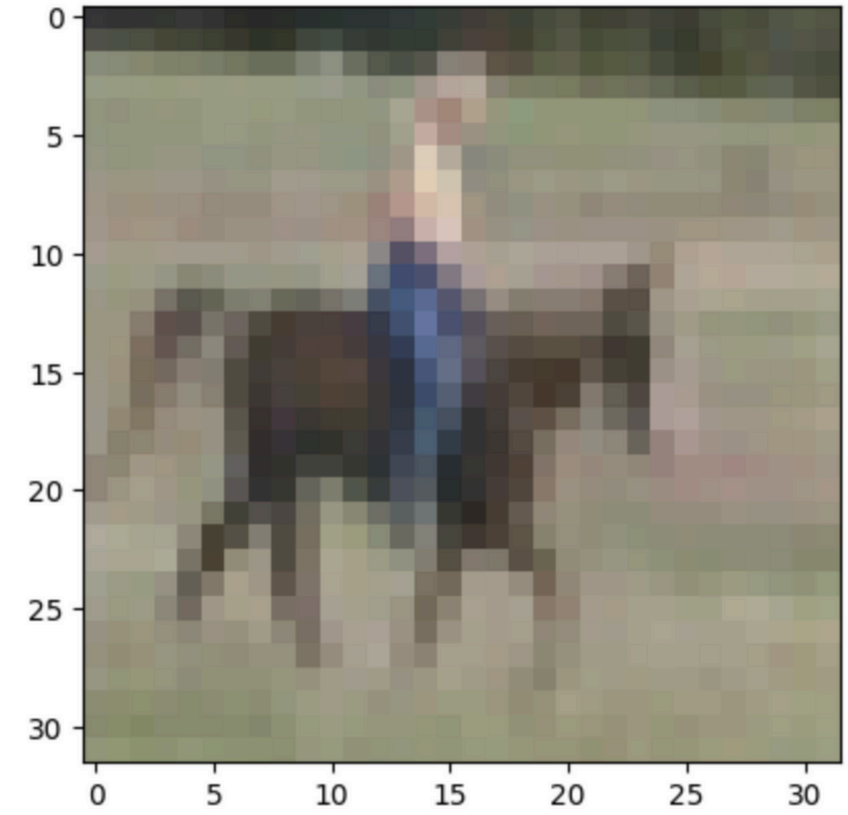
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8806 - loss: 0.3859
Accuracy on test set : 87.56%
Loss on test set : 39.11%
```



The val_Loss and val_accuracy oscillate quite a lot when compared to the Loss and accuracy. This probably means that the model is struggling to get better results.

Testing

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
Image 6504 is a(n) horse and the model predicts that it's a(n) horse
```



A random test is performed to see how the model works.

Here there is a random picture of a horse taken from the test set. The model correctly predicts that it's an image of a horse.

Possible improvements

The result obtained gives us a hint that the model can be improved further to lower the loss value (or increase the accuracy).

This could be done by:

- Adding more Conv2D layers or using more complex architectures.
- Using pre-trained models.
- Or even try using a validation dataset.

UNIVERSITÀ DEGLI STUDI DI MILANO-BICOCCA
MSC IN DATA SCIENCE
A.Y. 2023-2024

MANY THANKS FOR YOUR TIME