CIFAR-10 dataset

For this practical work, we will use the very famous CIFAR-10 dataset. It consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The training and testing set can be found directly from keras. All images were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

1 Loading and observing data

1- The first thing you have to do is to load the datasets and check the number of samples as well as the size of images

2- Then make some plots to observe some images see the different classes. You should obtain some- thing like that :

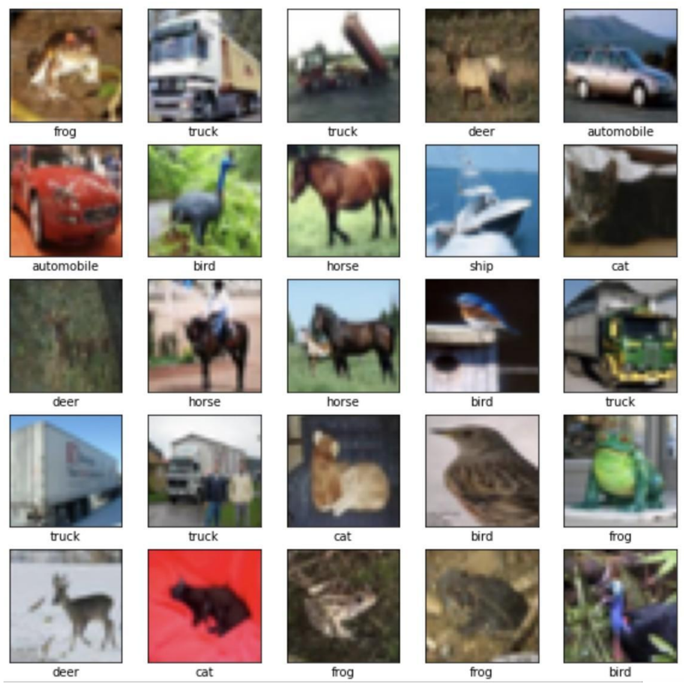
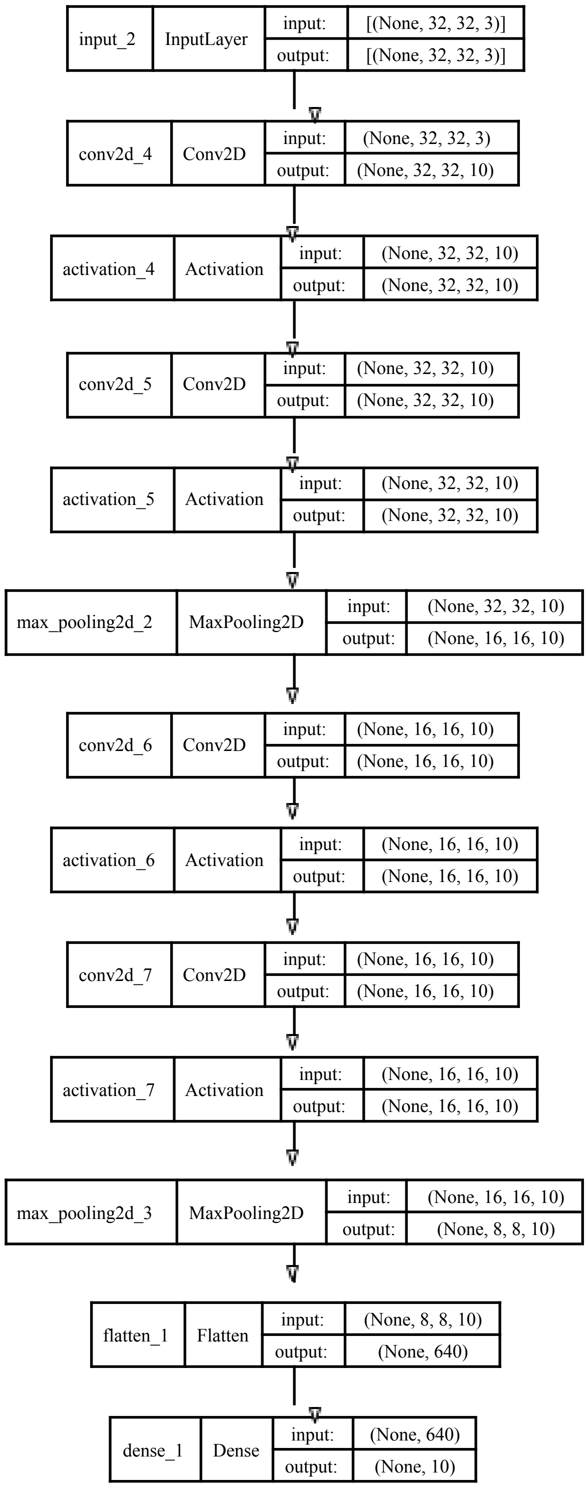


FIGURE 1 – Examples of some images from CIFAR-10 dataset

3- Apply traditional preprocessing on the datasets

2 Convolutional Neural Network

In this practical work, we will consider a tiny version of the famous VGG architecture. We will first build a network with two blocks of convolutions (instead of 5 in the VGG-16) [1] :



input_2 | InputLayer | input: [(None, 32, 32, 3)] | output: [(None, 32, 32, 3)]
conv2d_4 | Conv2D | input: (None, 32, 32, 3) | output: (None, 32, 32, 10)
activation_4 | Activation | input: (None, 32, 32, 10) | output: (None, 32, 32, 10)
conv2d_5 | Conv2D | input: (None, 32, 32, 10) | output: (None, 32, 32, 10)
activation_5 | Activation | input: (None, 32, 32, 10) | output: (None, 32, 32, 10)
max_pooling2d_2 | MaxPooling2D | input: (None, 32, 32, 10) | output: (None, 16, 16, 10)
conv2d_6 | Conv2D | input: (None, 16, 16, 10) | output: (None, 16, 16, 10)
activation_6 | Activation | input: (None, 16, 16, 10) | output: (None, 16, 16, 10)
conv2d_7 | Conv2D | input: (None, 16, 16, 10) | output: (None, 16, 16, 10)
activation_7 | Activation | input: (None, 16, 16, 10) | output: (None, 16, 16, 10)
max_pooling2d_3 | MaxPooling2D | input: (None, 16, 16, 10) | output: (None, 8, 8, 10)
flatten_1 | Flatten | input: (None, 8, 8, 10) | output: (None, 640)
dense_1 | Dense | input: (None, 640) | output: (None, 10)

FIGURE 2 – Tiny VGG architecture

4- Build this network using Keras. In particular all kernels have a size of 3×3 and pooling have a size of 2×2.

5- Prepare and compile your model with the Adam optimizer

1. Simonyan, K., and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv pre- print arXiv :1409.1556.

3 Training and evaluation

6- Train your model on the training set for 100 epochs, with a batch size of 128 and with 30% of data used for validation

7- Evaluate your model on the test set

8- Plot the train and test curves (accuracy and loss)

9- Note that the model performance depends on the random initialization of weights. In order to evaluate its performance more robustly, a good practice is to run the whole process several times (e.g 5 or 10 times) and finally report the average accuracy among runs **with standard deviation**. A mean value only (without standard deviation) is not meaningful. This will show of the model is robust to random initialization. Run the experiment to analyze the model's robustness

4 Visualization

An interesting characteristic of CNN applied on images is that we can easily visualize intermediate features to interpret how an image has been processed.

10- Try to visualize the intermediate features of the tiny VGG architecture except for the flatten layers and the last layers as we can not see them as images. By selecting one of the 10 filters, you should obtain something like that :

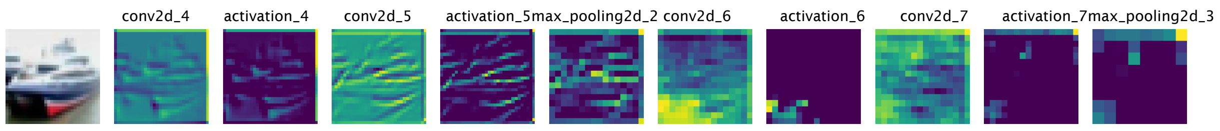


FIGURE 3 – Intermediate features corresponding to the 10*th* filter for an input image

11- Instead of visualizing intermediate features, we can also observe the filters (kernels). Try to visua- lize all the weights of the first layer for the three channels. You should obtain something like that :

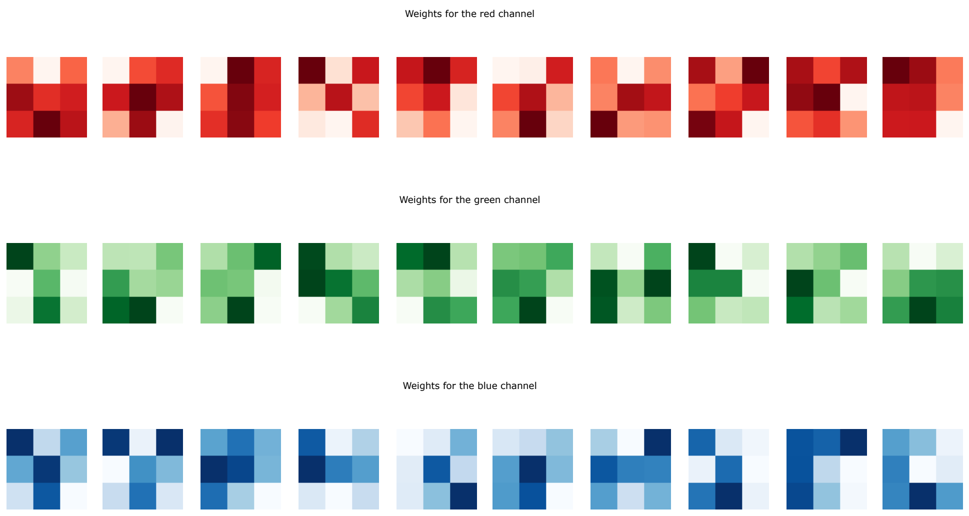


FIGURE 4 – All filters of the first layer for the three channels

12- With this architecture, as kernel sizes are 3 × 3, it is not easy to interpret them. Just for this ques- tion, load the ResNet50 model (in a new model variable to erase your previous model) pre-trained on the Imagenet dataset and observe the 64 filters of the first layer. The pre-trained model can be found in from `keras.applications.resnet50`. Instead of considered each channel separately, consider each filter as an RGB image. You should obtain something like the following :

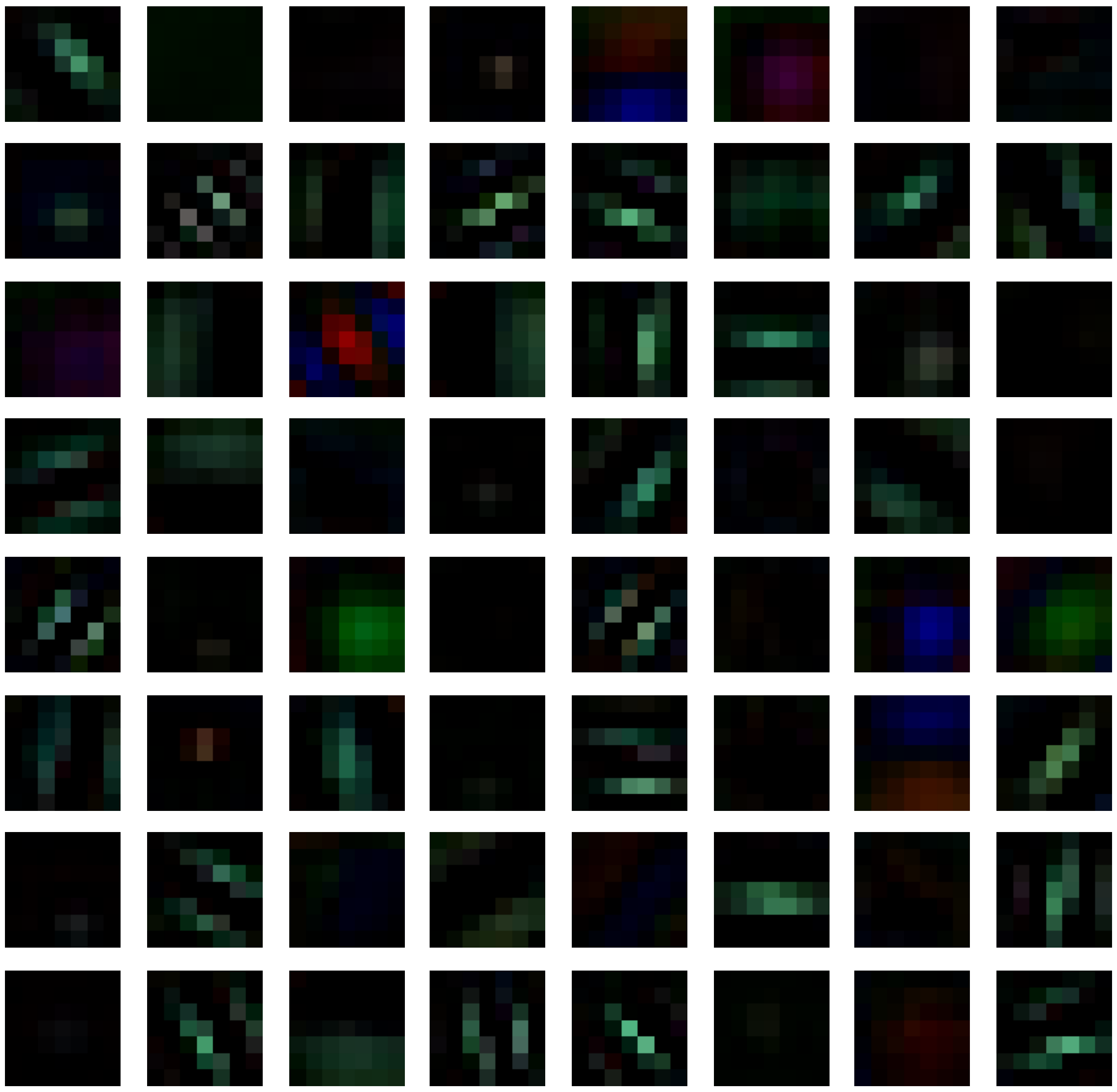


FIGURE 5 – 64 filters of the first convolutional layer of ResNet50 pre-trained on ImageNet

5 Improving the model

13- In your **original tiny VGG model**, add BatchNormalization layers after each convolutions (before activation) and run the experiments again to evaluate and analyze the new model's performance and ro- bustness.

14- Add an additional convolution block to the **original tiny VGG model** and run the experiments again in order to see if the results are improved and more robust.

Bonus- Transform your model into a tiny resnet-like architecture :

- Before the blocks of 3×3 convolutions, add a first convolutional layer with a kernel size of 7×7, followed by a BatchNormalization layer, an Activation layer and a MaxPooling layer (pool_size of 2)
- In each block, remove the MaxPooling Layers
- For each block, create residual connections from the input of a block to the output of the same block. For that, you can use the `Add` layer from Keras. In Resnet, the final activation of a block is done after the residual connection.
- Finally, before the flatten layer, add a global average pooling with $pool\_size = 4$.

Run the experiments to observe if it improves the performance. Actually, residual connections are useful for very deep networks. In our tiny version, the improvement may be very slight (or null)