CS-577 Deep Learning Homework 4

Guillermo Lopez-Areal

ILLINOIS INSTITUTE
OF TECHNOLOGY

Contenido

Programming questions

Binary classification

In this part of the homework, we are asked to download a set of images from Kaggle. Those images are images of cats and dogs from different angles, in which there are lots of animals breeds, and images where the resolution and dimensions are completely different from each other.

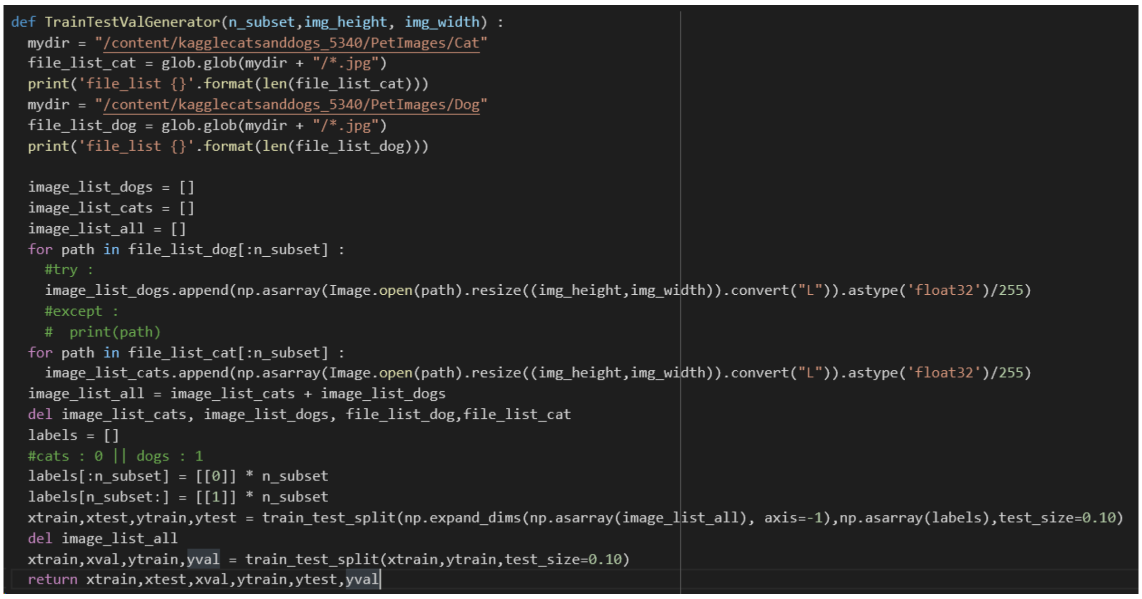
To do this first exercise we define a python notebook, called CNN.ipynb, in which we'll define the neural network and we'll train the model so as the model can recognize which photo corresponds to cats and which to dogs.

Convolutional Neural Network (CNN.ipynb)

Firstly, and as we download the dataset as a zip file, we'll have to unzip it in our folder, with this command.

```
shutil.unpack_archive("/content/kagglecatsanddogs_5340.zip","kaggle
catsanddogs_5340")
```

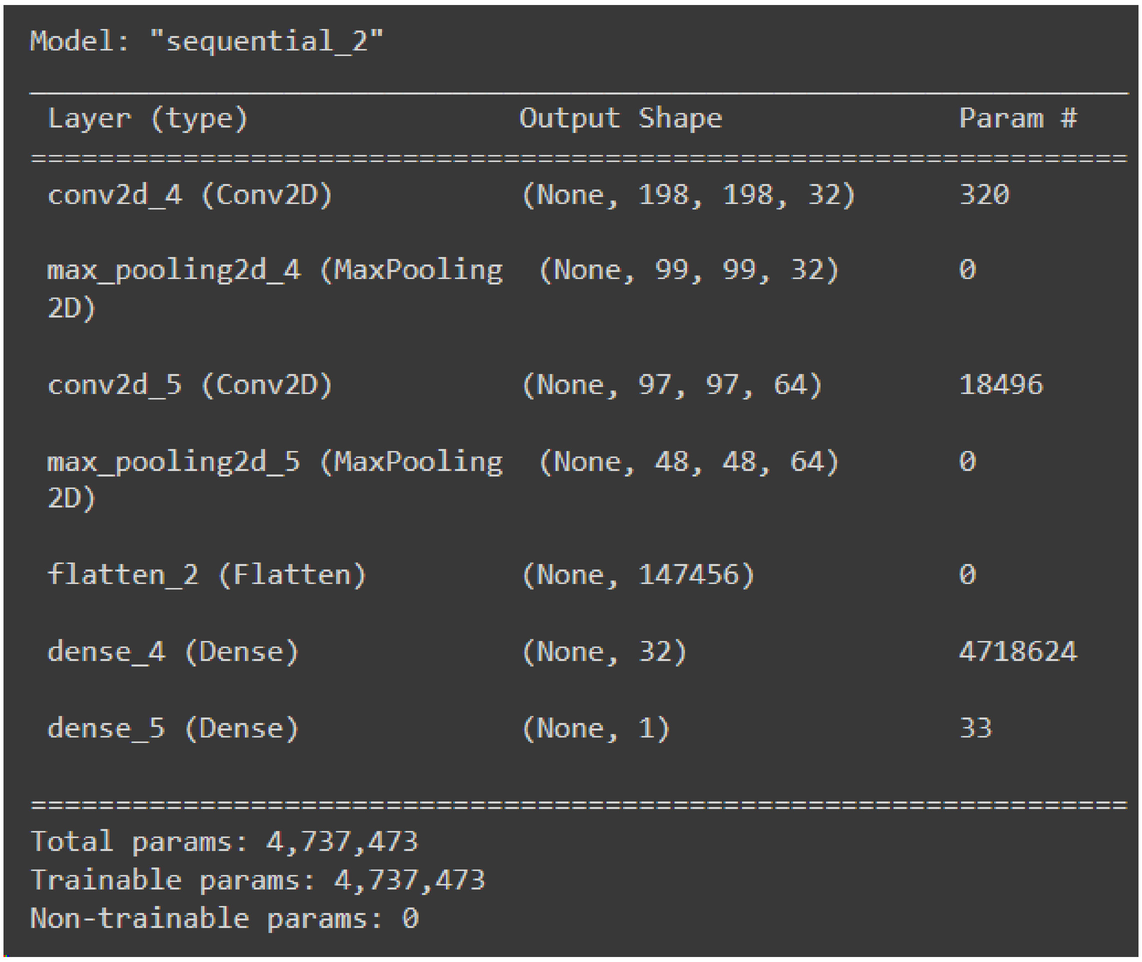
Then we'll define a function in which we'll prepare the data to fit our model. As in the dataset there are multiple images with different formats, we'll pick exclusively the '.png' files.



```python
def TrainTestValGenerator(n_subset,img_height, img_width) :
  mydir = "/content/kagglecatsanddogs_5340/PetImages/Cat"
  file_list_cat = glob.glob(mydir + "/*.jpg")
  print('file_list {}'.format(len(file_list_cat)))
  mydir = "/content/kagglecatsanddogs_5340/PetImages/Dog"
  file_list_dog = glob.glob(mydir + "/*.jpg")
  print('file_list {}'.format(len(file_list_dog)))

  image_list_dogs = []
  image_list_cats = []
  image_list_all = []
  for path in file_list_dog[:n_subset] :
    #try :
    image_list_dogs.append(np.asarray(Image.open(path).resize((img_height,img_width)).convert("L")).astype('float32')/255)
    #except :
    #  print(path)
  for path in file_list_cat[:n_subset] :
    image_list_cats.append(np.asarray(Image.open(path).resize((img_height,img_width)).convert("L")).astype('float32')/255)
  image_list_all = image_list_cats + image_list_dogs
  del image_list_cats, image_list_dogs, file_list_dog,file_list_cat
  labels = []
  #cats : 0 || dogs : 1
  labels[:n_subset] = [[0]] * n_subset
  labels[n_subset:] = [[1]] * n_subset
  xtrain,xtest,ytrain,ytest = train_test_split(np.expand_dims(np.asarray(image_list_all), axis=-1),np.asarray(labels),test_size=0.10)
  del image_list_all
  xtrain,xval,ytrain,yval = train_test_split(xtrain,ytrain,test_size=0.10)
  return xtrain,xtest,xval,ytrain,ytest,yval
```

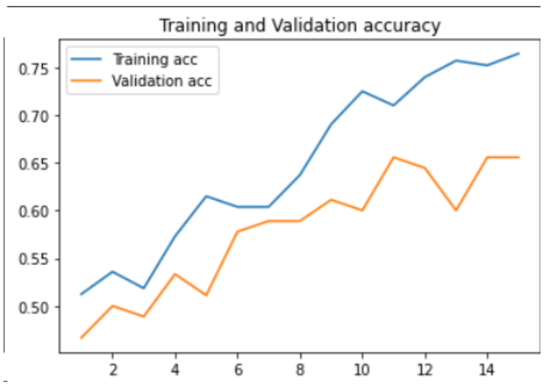
Then we'll create 3 lists. In the first one(image_list_dogs), we'll append the dog pictures, in the second one (image_list_cats), we'll append every cat picture in the dataset. In the remaining list we'll append both lists. There are multiple delete commands (del X) throughout the function, which are used for memory purposes. Once we have our list of dogs and cats images, we have to create labels, so as our network can reference certain images as dogs or cats. To do that, and as the images in our list are not randomized yet, we apply an integer to reference both animals, 0 to dogs and 1 to cats. And finally, we call the function train_test_split which will not only split our dataset into training testing and validation sets, but also will randomize the features of our dataset. Once the function is called we'll exclusively, operate with the variables, xtrain, xtest, xval, ytrain, ytest, yval. We'll leave three variables as input objects to our function, which are the number of subset images (2000), and since the images are shaped differently, the resizing is also a parameter as input to our function.

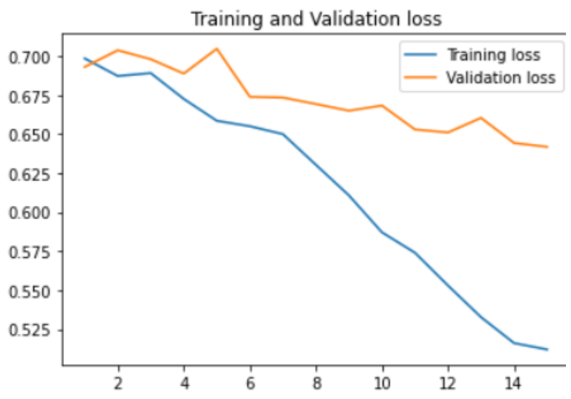
The summary of the model's architecture is shown in the figure below :



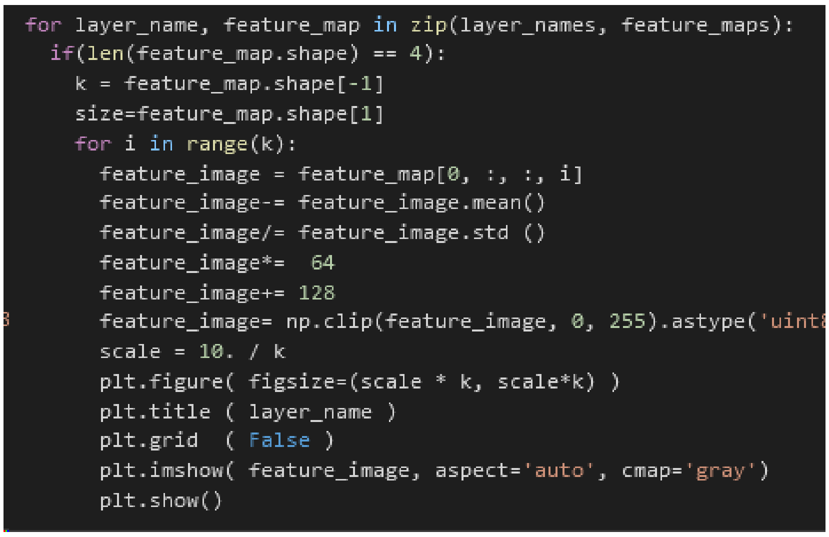
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_4 (Conv2D)           (None, 198, 198, 32)      320       
                                                                 
 max_pooling2d_4 (MaxPooling  (None, 99, 99, 32)       0         
 2D)                                                             
                                                                 
 conv2d_5 (Conv2D)           (None, 97, 97, 64)        18496     
                                                                 
 max_pooling2d_5 (MaxPooling  (None, 48, 48, 64)       0         
 2D)                                                             
                                                                 
 flatten_2 (Flatten)         (None, 147456)            0         
                                                                 
 dense_4 (Dense)             (None, 32)                4718624   
                                                                 
 dense_5 (Dense)             (None, 1)                 33        
                                                                 
=================================================================
Total params: 4,737,473
Trainable params: 4,737,473
Non-trainable params: 0
```

The results are plotted below :



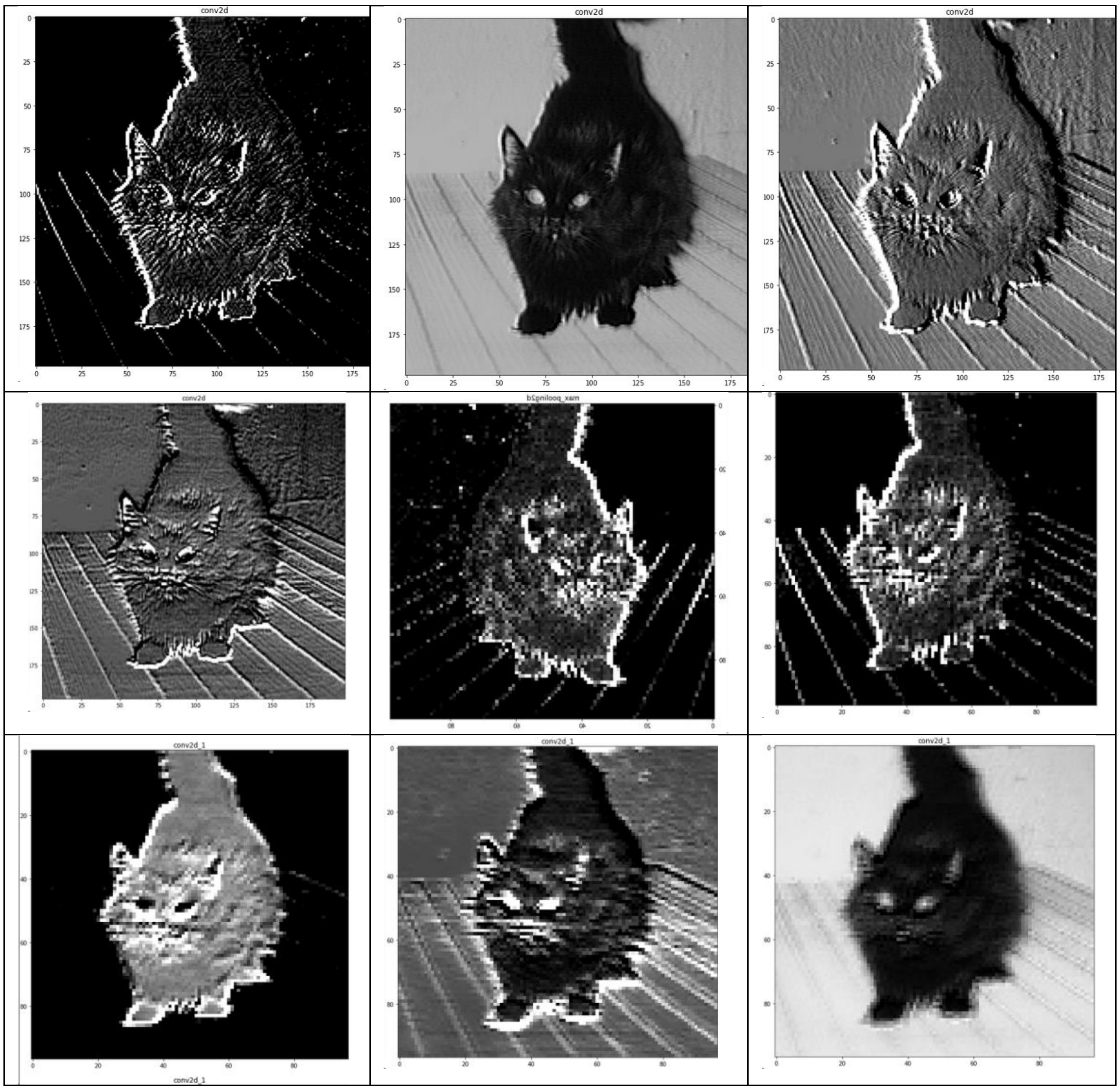


Then, we have to observe the activation of some convolution layers, which has been performed with the following piece of code :



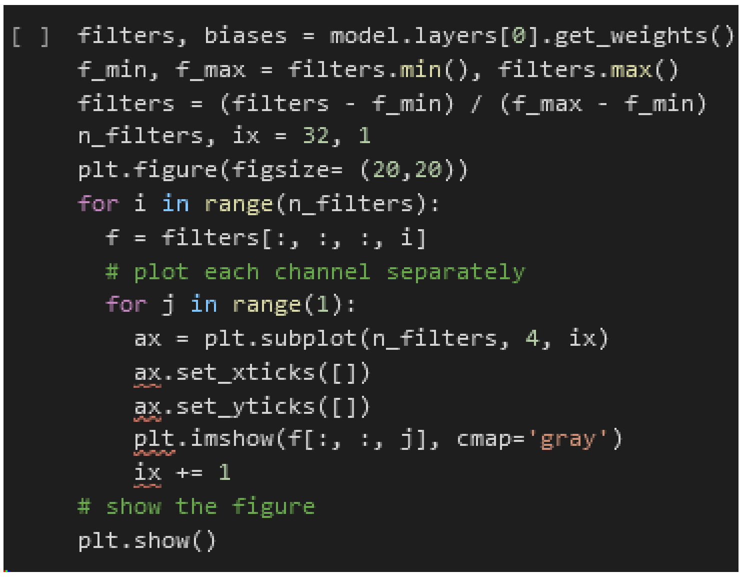
```
for layer_name, feature_map in zip(layer_names, feature_maps):
  if(len(feature_map.shape) == 4):
    k = feature_map.shape[-1]
    size=feature_map.shape[1]
    for i in range(k):
      feature_image = feature_map[0, :, :, i]
      feature_image-= feature_image.mean()
      feature_image/= feature_image.std ()
      feature_image*=  64
      feature_image+= 128
      feature_image= np.clip(feature_image, 0, 255).astype('uint
      scale = 10. / k
      plt.figure( figsize=(scale * k, scale*k) )
      plt.title ( layer_name )
      plt.grid  ( False )
      plt.imshow( feature_image, aspect='auto', cmap='gray')
      plt.show()
```

The results below has been done for the first cat image, however in the neural network it is done for every image.



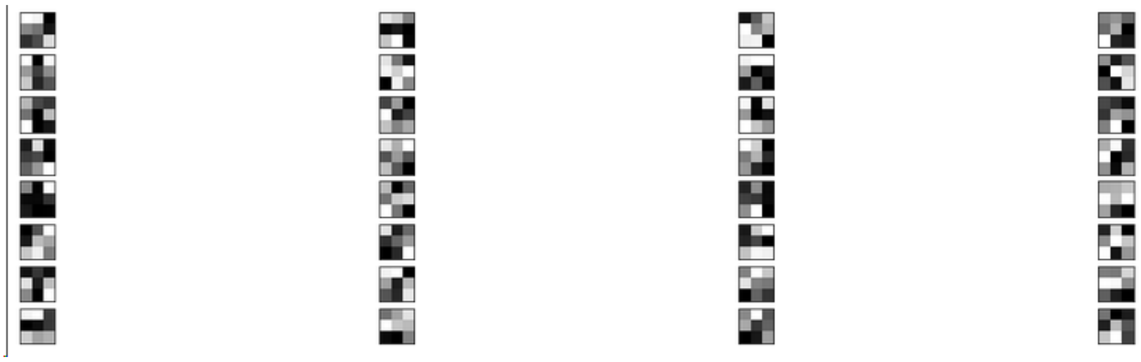
We have displayed only 9, however there much more to be displayed.

Then, we have to visualize the filters learned in training. This next part is done with the following code.



```
[ ]  filters, biases = model.layers[0].get_weights()
     f_min, f_max = filters.min(), filters.max()
     filters = (filters - f_min) / (f_max - f_min)
     n_filters, ix = 32, 1
     plt.figure(figsize= (20,20))
     for i in range(n_filters):
       f = filters[:, :, :, i]
       # plot each channel separately
       for j in range(1):
         ax = plt.subplot(n_filters, 4, ix)
         ax.set_xticks([])
         ax.set_yticks([])
         plt.imshow(f[:, :, j], cmap='gray')
         ix += 1
     # show the figure
     plt.show()
```

And here are the results :



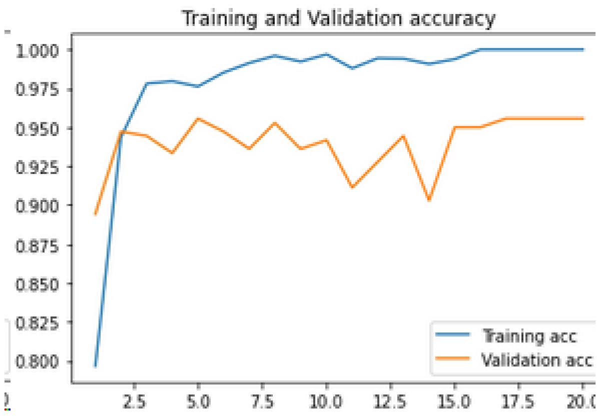
As the filter is applied multiple times to the input array, the result is a two-dimensional array of output values that represent a filtering of the input. As such, the two-dimensional output array from this operation is called a “feature map“

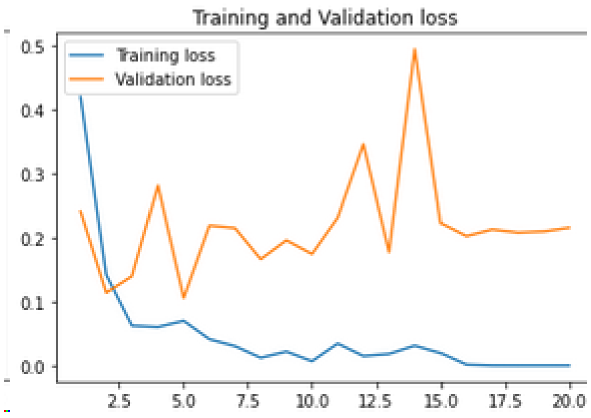
VGC16

Now we replace our regular CNN, with pre-trained convolutional base of VGC16. In this part of the homework, we will consider no data augmentation (every trainable feature has been done with a 200x200 sizing) until the very end, in which we will consider 500x500 images. Firstly we will train and plot the results with an unfrozen convolutional base, and afterwards with a frozen convolutional base.

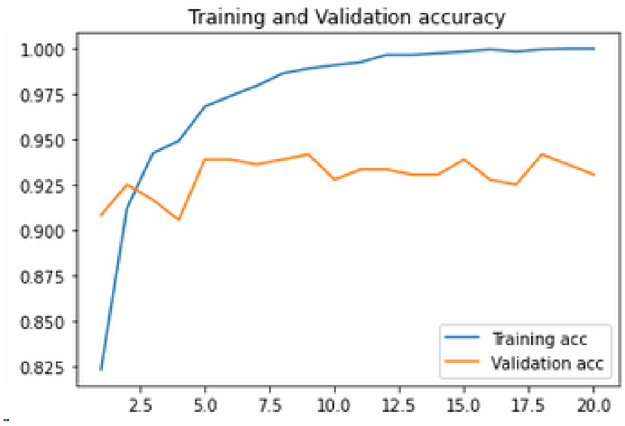
Freeze/Unfreeze

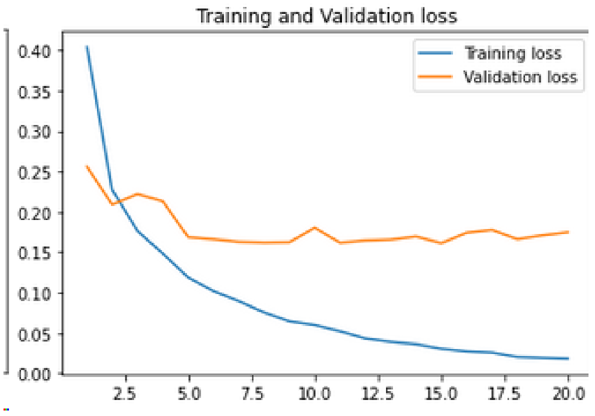
- With freeze deactivated:





- With freeze activated :

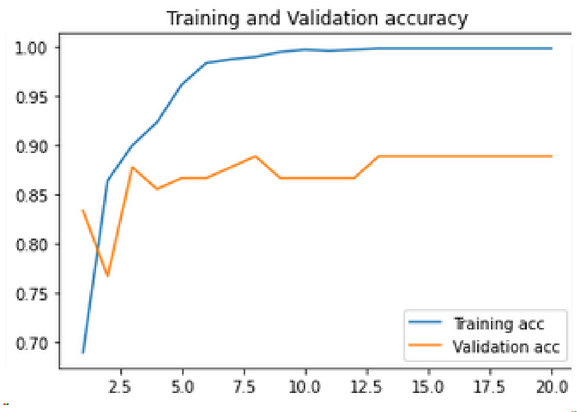


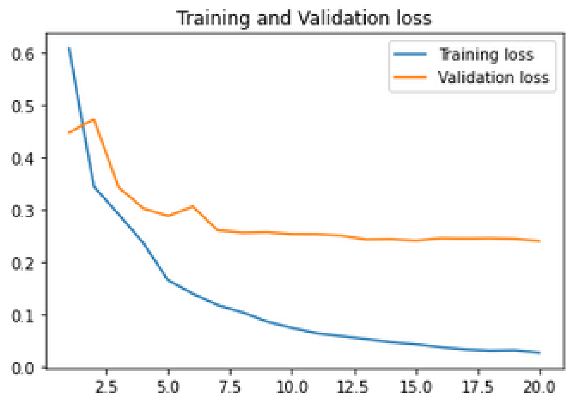


Data augmentation :

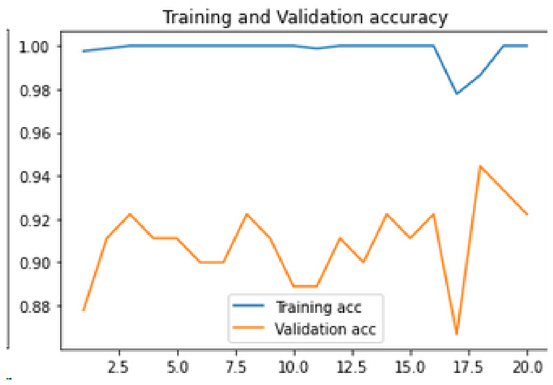
Finally, we will try with data augmentation. For that, we will change the parameters of resizing of our input images, converting them from 200x200 (original value), which is the values from which every calculation has been made from, to 500x500, and compare the results. However, since we can't complete 500x500 size with 2000 images, because of the excessive RAM required, we will do it only with 500 images. We obtain the following results :

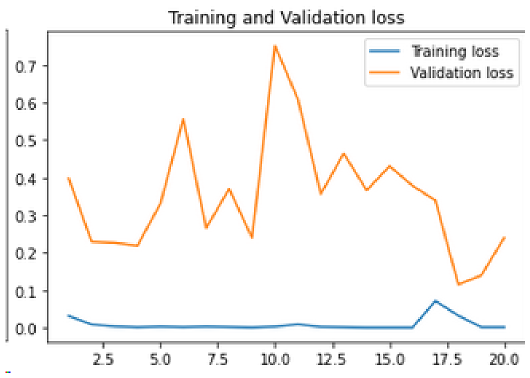
Freeze activated





Freeze deactivated



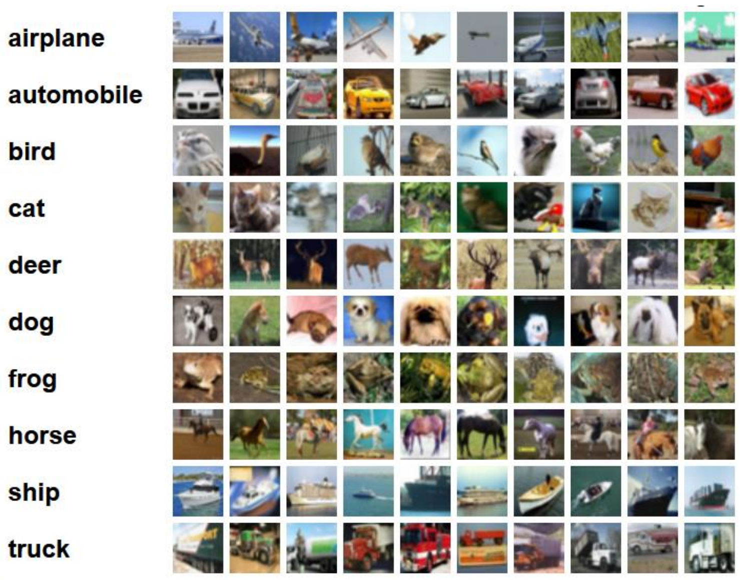


Multiclass classification

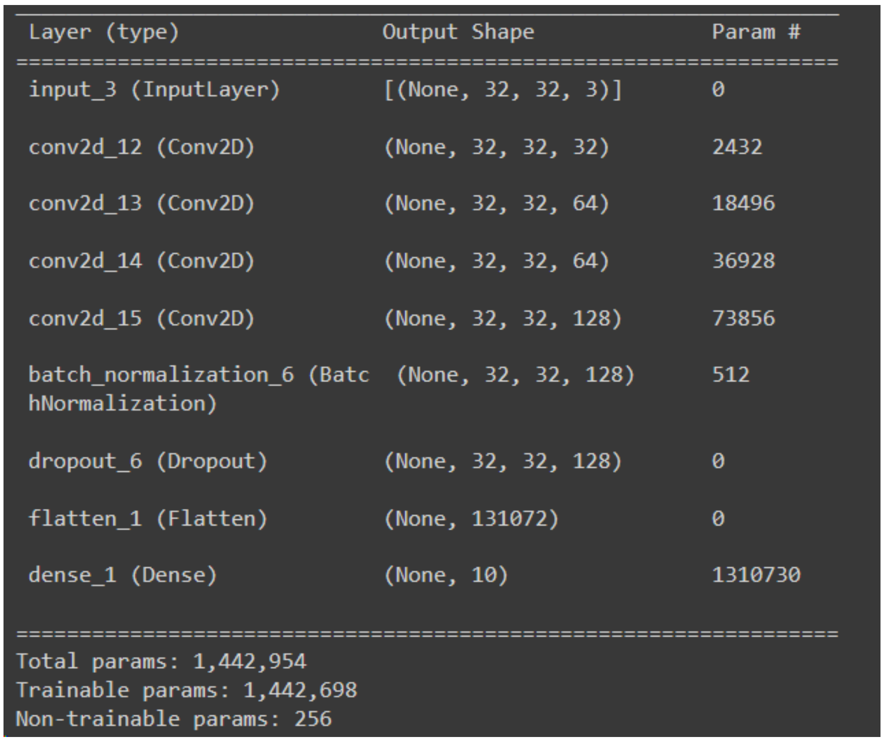
CIFAR10

This second exercise has been done in another python notebook called : ‘CIFAR10_HW4.ipynb’. The dataset is divided into five training batches and one test batch, each with 10000 images. In this dataset, we don’t have images of cats and dogs, we have 10 different type of images, and for that reason, it is a multiclassification exercise.

The classes are completely mutually exclusive. Below there is a sample of some images of the dataset and their corresponding class.

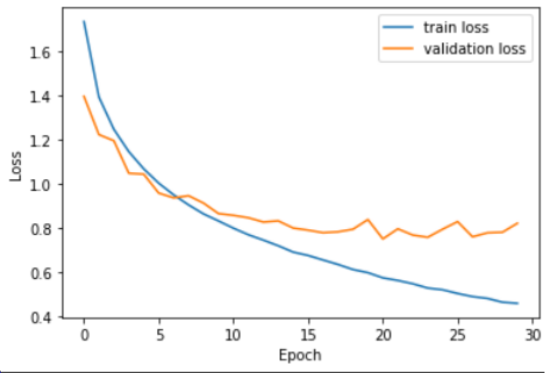


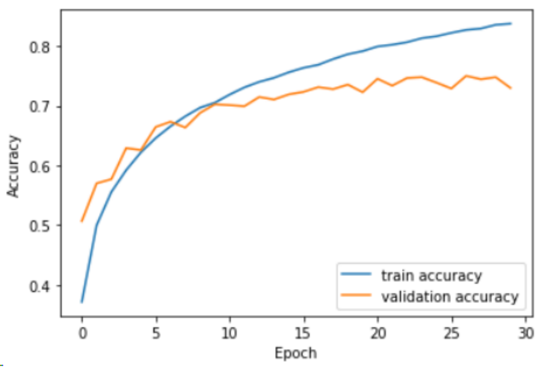
The architecture which we will be using at first, without the use of residual blocks or inception blocks es described below :



```
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 32, 32, 3)]       0

 conv2d_12 (Conv2D)          (None, 32, 32, 32)        2432

 conv2d_13 (Conv2D)          (None, 32, 32, 64)        18496

 conv2d_14 (Conv2D)          (None, 32, 32, 64)        36928

 conv2d_15 (Conv2D)          (None, 32, 32, 128)       73856

 batch_normalization_6 (Batc  (None, 32, 32, 128)      512
 hNormalization)

 dropout_6 (Dropout)         (None, 32, 32, 128)       0

 flatten_1 (Flatten)         (None, 131072)            0

 dense_1 (Dense)             (None, 10)                1310730

=================================================================
Total params: 1,442,954
Trainable params: 1,442,698
Non-trainable params: 256
```

The results obtained are displayed below:

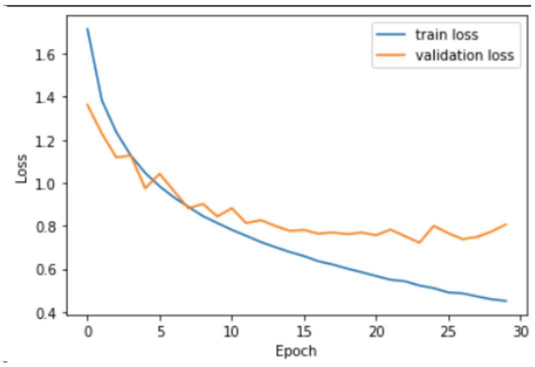


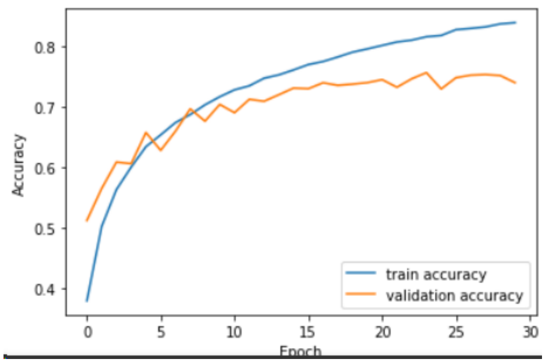


```
Test accuracy:  0.736400008201599
```

Addition of inception blocks

The addition of inception blocks allow for more efficient computation and deeper networks through a dimensionality reduction with stacked 1×1 convolutions. That is that inception modules are incorporated into convolutional neural networks (CNNs) as a way of reducing computational expense. Here is the design of our inception blocks, in which we concatenate multiple blocks in order to

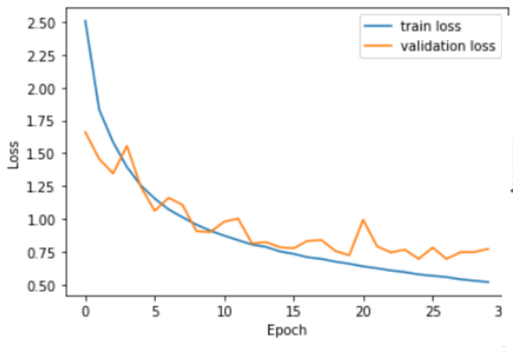


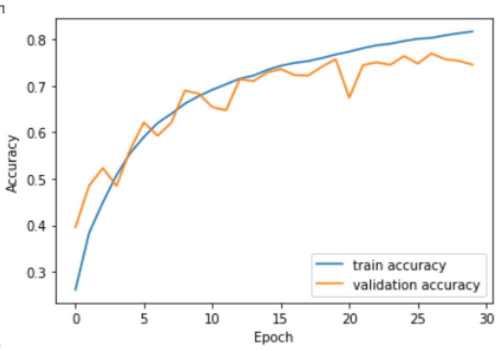


```
Test accuracy:  0.7364000082015991
```

Addition of residual blocks

Theoretically, residual blocks, are able to preserve the weights and continuously optimize and get better accuracy. In this section we’ll eliminate the inception blocks used in the previous exercise, and we’ll incorporate residual blocks and we’ll compare the results.





```
Test accuracy:  0.7480000257492065
```

As we can see from the test accuracy in the 3 cases, although the 3 accuracies are pretty close to each other, there is slight improvement if we apply residual blocks (maximum accuracy among the 3 tests), and even if we don’t apply residual blocks, and only apply inception blocks, the test accuracy is greater compared to if we don’t apply nor inception blocks or residual blocks.