UNIVERSITÀ DI PISA

COMPUTATIONAL INTELLIGENCE
AND DEEP LEARNING PROJECT

ACADEMIC YEAR 2021-22

CNN for Medical Images Analysis: Diabetic Retinopathy Detection

Casini Mirko

Lagna Andrea

Martorana Michelangelo

Summary

1. Introduction

The objective of the project is to build, develop and compare different neural networks suitable to distinguish the different levels of diabetic retinopathy using Deep Learning techniques.

Diabetic retinopathy, also known as diabetic eye disease (DED), is a medical condition in which damage occurs to the retina due to diabetes mellitus. Diabetic retinopathy affects up to 80 percent of those who have had diabetes for 20 years or more. At least 90% of new cases could be reduced with proper treatment and monitoring of the eyes. The longer a person has diabetes, the higher his or her chances of developing diabetic retinopathy. [1]

Diabetic retinopathy is the main cause of blindness in the working-age population of the developed world. It is estimated that it affects more than 93 million people. Diabetic retinopathy (DR) is an eye disease associated with long-standing diabetes. Progression to impaired vision can be slowed down or avoided if DR is detected in time; however, this can be difficult as the disease often shows few symptoms until it is too late to provide effective treatment. The need for a comprehensive and automated DR screening method has long been recognised, and previous efforts have made good progress using image classification, model recognition and machine learning.

The dataset consists of a large number of high-resolution retina images (~4752x3168) taken in a variety of conditions. For each subject are provided photos of both the left and right retina. Images are labelled with a subject ID in addition to a label that specifies whether the photo is left or right retina (e.g. 1_left.jpeg is the patient's left eye with ID 1).

A medical specialist then evaluated the presence of diabetic retinopathy in each image on a scale of 0 to 4, reported below:

0. NO DR
1. Mild
2. Moderate
3. Severe
4. DR Proliferating

The images in the dataset come from different models and types of cameras that can affect the visual appearance of left and right. Some images are shown for how you could see the retina anatomically (macula on the left, optic nerve on the right for the right eye). Others are shown as they would be seen through a microscope condensing lens (inverted) as seen in a typical ocular examination. To have the dataset balanced and maintain the consistency of data in each photo, some of them have been reversed in order to always display the same way photos of right or left retinas. Like any real data set, a minimal amount of noise is present in both images and labels. Images, for example, may be out of focus, under-exposed or overexposed.

Due to the large size of the dataset (~89GB) a preprocessing phase is required to be able to manage the material through Google Colab. During this phase we decided to reduce the resolution of the images, as found in the literature, to 786x524 so as to reduce the size of the dataset by 99% bringing it up to ~1GB.

The DR has two major types: the Non-Proliferative Diabetic Retinopathy (NPDR) and Proliferative Diabetic Retinopathy (PDR). The DR in the early stages is called NPDR which is further divided into Mild, Moderate, and Severe stages.

- **Mild** stage has one micro-aneurysm, a small circular red dot at the end of blood vessels.
- In the **Moderate** stage the micro-aneurysm rapture into deeper layers and form a flame-shaped haemorrhage in the retina.
- The **Severe** stage contains more than 20 intraretinal haemorrhages in each of the four quadrants, having definite venous bleeding with prominent intraretinal microvascular abnormalities.
- **PDR** is the advanced stage of DR which leads to neovascularization, a natural formation of new blood vessels in the form of functional microvascular networks that grow on the inside surface of the retina.

The Figure 1.1 Diabetic Retinopathy Level Examples visually presents the different stages of DR. It is clear from the given figure the Normal and Mild stage looks visually similar. Hence, it is difficult to detect the Mild stage. [2]

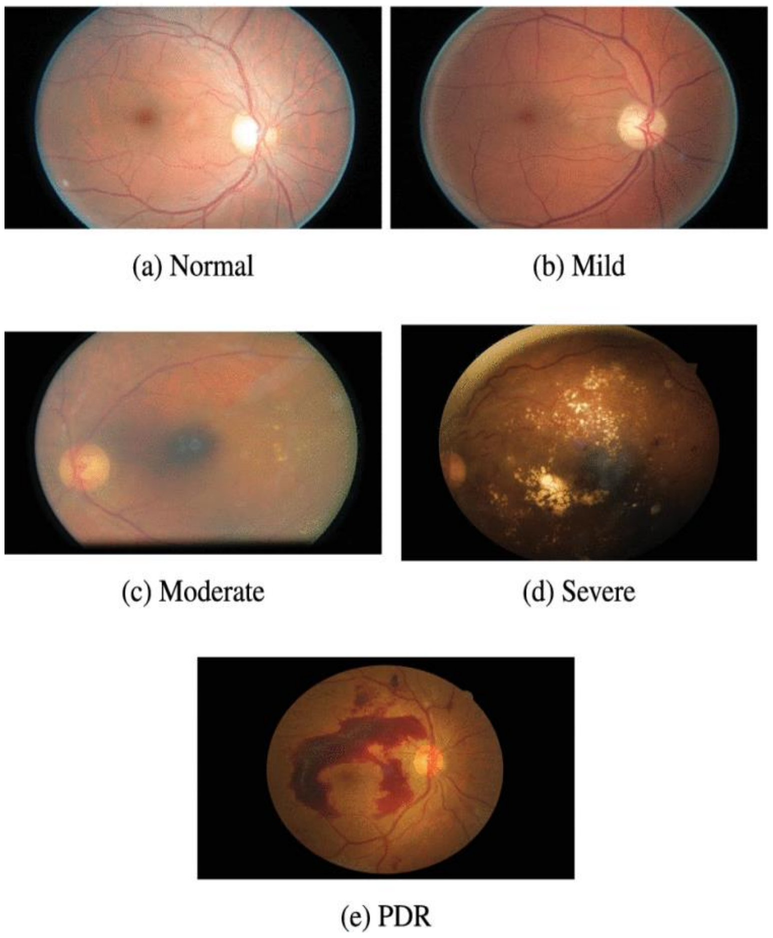


Figure 1.1 Diabetic Retinopathy Level Examples

The classes distribution of images is:

Training Set

Level	**Name**	**# of instances**
Level 0	NO DR	25808
Level 1	Mild	2443
Level 2	Moderate	5292
Level 3	Severe	873
Level 4	DR Proliferating	708

Table 1 Number of instances of the training set for each class

Test Set

Level	**Name**	**# of instances**
Level 0	NO DR	39533
Level 1	Mild	3762
Level 2	Moderate	7861
Level 3	Severe	1214
Level 4	DR Proliferating	1206

Table 2 Number of instances of the test set for each class

1.1 Review of state-of-the-art works

There are several state-of-the-art works in the literature that try to solve the problem of the classification of diabetic retinopathy. In this document, the most relevant, according to the task of classifying into the 5 classes, are exploited.

- "Deep convolutional neural networks for diabetic retinopathy detection by image classification" published in October 2018 by Shaohua Wan, Yan Liang, Yin Zhang [3]: In this paper four different models of CNNs are evaluated and compared. In the pre-process phase normalization schemas, data augmentation and noisy reduction are applied in order to balance the dataset and make it usable in the successive phases. For the classification phase four different models (AlexNet, VggNet, GoogleNet, ResNet) are applied coupled with transfer learning and hyper-parameter tuning. Finally, the metrics used for performance evaluation are accuracy, sensitivity, specificity and AUC and they highlight that the best model is VggNet-s.

- "Diabetic Retinopathy Detection and Classification using Pre-trained Convolutional Neural Networks" published on June 2020 by Sanskruti Patel [4]: In this paper two different models of CNNs are evaluated and compared. In the pre-process phase images are resized and organized in five folder (one for each class). Then in the successive phase, normalization is carried out and after a Gaussian blur is applied to the images in order to reduce the noise. CNN model requires enough training data to train a model, for that data augmentation is necessary, in particular in this paper real time data augmentation is performed each epoch.

For what concern experiments in the paper is highlighted the fact that transfer learning and fine tuning are used on the acquire dataset. After for the feature extraction 2 different approaches are performed, the first one is using pre-trained CNN by replacing the last fully connected layer that gives results in terms of accuracy around 90% and in terms of loss 0.3 for VGG16 and 0.45 for MobileNetV1. The second approach utilize fine-tuning on the pre-trained CNN model by replacing set of convolutional layers and gives results in term of accuracy around 89% and in terms of loss 0.3 both for VGG16 and MobileNetV1. Concluding the authors of the papers highlight the fact that fine tuning the pre trained CNN model by replacing higher level layers is giving more accuracy compared to only replace the last fully connected layer. Moreover, MobileNetV1 provided slightly better accuracy compared to VGG16.

- "Automatic Detection and Classification of Diabetic Retinopathy stages using CNN" published in 2018 by Ratul Ghosh, Kuntal Ghosh and Sanjit Maitra [5]: In this paper a single model of CNNs is evaluated. The first phase is the preprocessing and is performed by cropping and resizing the images due their high dimensions. Then augmented images were created to increase the class size exploiting brightness adjustment, image scaling and image rotation, this also makes the model immune to different orientations and lightning conditions. The successive steps are normalization and denoising that is implemented by Non-Local Means Denoising (NLMD). For what concern the weight initialization the Xavier initialization is used and the SoftMax layer is responsible of the final prediction, with a related loss function. The regularization of the model is obtained using dropout and the parameters are updated using Nesterov momentum.

2. Preprocessing

2.1 detectAndCrop()

This first function detect the interesting part of the image and remove the background not useful for the analysis. The first step is to convert the image in grey scale using the cvtColor() method from OpenCV library, then the image is detected through the threshold() method of OpenCV library. The last two steps are crop step where the image is first rectangularly cropped using the boundingRect() method then is square cropped in order to obtain as output all square images. In the Figure 2.1 detectAndCrop() process: grey scale phase (a), threshold phase (b) and crop phase (c) illustrates the different phases.

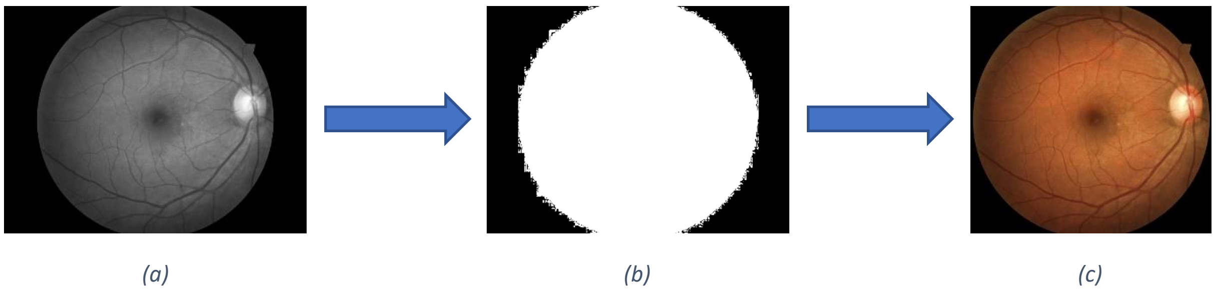


Figure 2.1 detectAndCrop() process: grey scale phase (a), threshold phase (b) and crop phase (c)

2.2 checkJPEG()

After the crop phase we notice errors on some images of the dataset so we start an analysis phase in order to find and remove the corrupted images. The function implemented is really easy and iterate on all the images using the verify() method of the Pillow library, then print the filename of the corrupted images in order to check if they are really corrupted and eventually delete them.

2.3 CLAHE

Additionally, we tried applying Contrast-Limited Adaptive Histogram Equalization (CLAHE) to the images. In this approach, images are divided into small blocks called "tiles" (tileSize is a required parameter that we set as 5x5). Then each of these blocks are histogram equalized as usual. This results in improved contrast and finer detail, as can be seen in Figure 2.2 CLAHE process: before (a) and after (b)

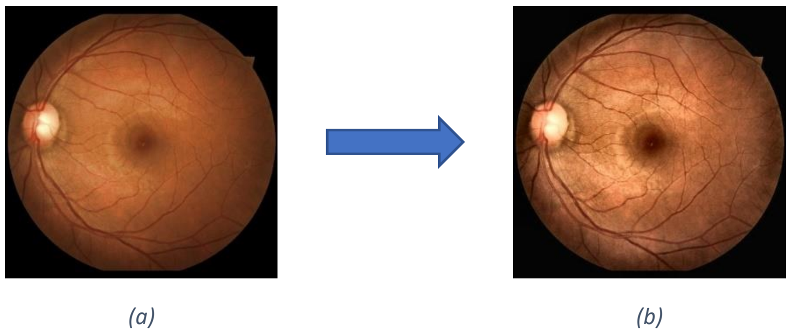


Figure 2.2 CLAHE process: before (a) and after (b)

2.4 Split the dataset in training and validation

In this phase we split the entire dataset in validation and training using 90%/10% ratio. The entire dataset was composed by 35124 and the number of elements in each class after the split were:

- validationSet/0 created with 2580 elements
- validationSet/1 created with 244 elements
- validationSet/2 created with 529 elements
- validationSet/3 created with 87 elements
- validationSet/4 created with 70 elements

- trainingSet/0 created with 23227 elements
- trainingSet/1 created with 2198 elements
- trainingSet/2 created with 4762 elements
- trainingSet/3 created with 785 elements
- trainingSet/4 created with 638 elements

These folders were created through createValidationSet() function implemented by us. This function utilize the shuffle() method, with the same seed, in order to take same random images and then the target images are moved by the move() method provided by the shutil library.

2.5 Undersampling e Oversampling

The balanceClass() function performs an oversampling procedure if the targeted class has less images then required. Otherwise it performs an undersampling if the targeted class has more images than required

- Oversampling: duplicate existing images belonging to a certain class
- Undersampling: drop exceeding images of a class

To further improve the algorithm and decrease the risks of overfitting, is possible to over and under sampled the different classes. For example, when more than 2000 labelled images were present, the class was under sampled. When less than 2000 labelled examples were present, the class was oversampled. In total, the results will contain 2000 images for each class. For the oversampled classes, some of the images may be repeated. Since all images are subjected to data augmentation, the probability to generate the same image is rare.

After the split we performed undersampling or oversampling on the classes. The results are showed below

- Start in folder: /content/trainingSet/1/in mode UNDERSAMPLING
  num samples in: /content/trainingSet/1/   2199 : 638
  num samples in: /content/trainingSet/1/   638
- Start in folder: /content/trainingSet/2/in mode UNDERSAMPLING
  num samples in: /content/trainingSet/2/   4763 : 638
  num samples in: /content/trainingSet/2/   638
- Start in folder: /content/trainingSet/3/in mode UNDERSAMPLING
  num samples in: /content/trainingSet/3/   786 : 638
  num samples in: /content/trainingSet/3/   638

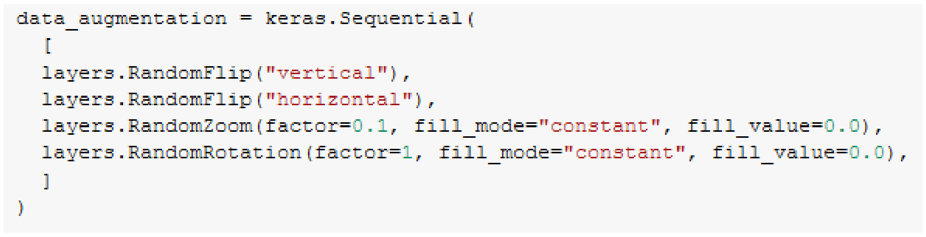
- Start in folder: /content/trainingSet/0/in mode UNDERSAMPLING
  num samples in: /content/trainingSet/0/   23228 : 638
  num samples in: /content/trainingSet/0/   638

As shown in the results all the classes are balanced to 638 instances, this choice is led by the time and resource constraints of Google Colab.

Also different models with a higher number of samples are tried but the results were not good and the computation requires a lot of time and due to Colab limitation we were disconnected and underprivileged GPU.

3. Online Data Augmentation

In this section is described the data augmentation layer inserted in all the network we developed. This operation may seem senseless as a reduction of the dataset was applied first and then this augmentation. It is necessary however to specify that the first reduction of the dataset has been made because of the limits of Google Drive and Colab that did not allow to work with the entire dataset composed from 35.000 images. After importing the reduced dataset, the online data augmentation was applied in order to avoid the model to see the exact same picture more than one time. This approach helps the model to generalize better.



```
data_augmentation = keras.Sequential(
  [
  layers.RandomFlip("vertical"),
  layers.RandomFlip("horizontal"),
  layers.RandomZoom(factor=0.1, fill_mode="constant", fill_value=0.0),
  layers.RandomRotation(factor=1, fill_mode="constant", fill_value=0.0),
  ]
)
```

Figure 3.1 Data augmentation layer

In the Figure 3.1 Data augmentation layer is represented. The main steps are the following:

- RandomFlip: the first operation is a random flip performed on the vertical axis, this operation does not require fill because simply flip the image keeping unchanged its dimension and shape. The random factor is referred to the application of the operation that is random.
- RandomFlip: the second operation is another random flip performed on the horizontal axis, this operation does not require fill because simply flip the image keeping unchanged its dimension and shape. The random factor is referred to the application of the operation that is random.
- RandomZoom: this operation performs a zoom of the input image, the zoom could be “in” or “out”. In case of zoom out the new pixels are filled with constant value set to 0 that correspond to black color (the images have already a black frame, so this operation does not alter the dataset). The parameter factor, set to 0.1, means that the zoom performed is selected random in the range [-10%, +10%]
- RandomRotation: this operation performs a random rotation on the input image, the rotation could be clockwise or counterclockwise. In particular case of rotation that is not multiple of 90° the new pixels are filled with constant values set to 0 that correspond to black color (the images has already a black frame so this operation does not alter the dataset). The parameter factor, set to 1, means that the rotation performed is selected random in the range [-100%, +100%]

4. CNN from scratch

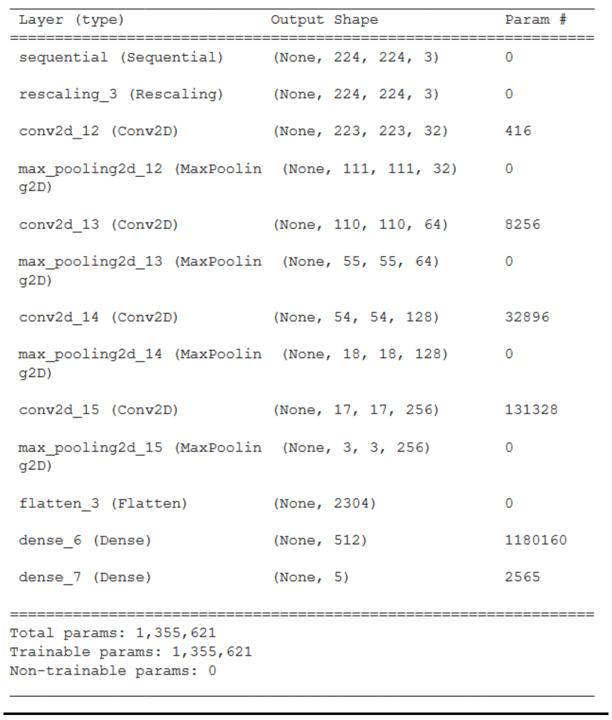
4.1 Experiments

We tried to define some architectures from scratch, to propose our own custom architecture for solving the problem. The trial-and-error approach was applied to find the best architecture, on which we then tried to perform further experiments to improve its performance.

Regarding the size of the input images, we decided to use 224×224 which seems to be a good trade-off between what the literature proposes, and the computational power provided by Google Colab. We chose to use 100 as batch size, which is the number of samples considered in each iteration. After the input layer, we performed a normalization with a Rescaling Layer, so that each pixel value was in the range [0,1]. Then we decided to use 4 convolutional layers, using the zero-padding technique, in order to give the same importance to each pixel of the image; in fact, even the borders are relevant having cropped the images as much as possible. For the stride we decided to use the default value of 1, to avoid penalties on the accuracy. As activation functions we used ReLU: $f(x)$ = max (0, x). As the size of the local receptive fields, we decided to use the value 2x2. In the first convolutional layer 32 filters are used, and for each convolutional layer the number of filters doubles. Max-pooling is applied after each convolutional level, so that small regions are summarized with a single value. Our architecture uses increasing size for max pooling, with the size of the pooling increasing in the final levels (2×2 in the first two layers, then 3×3 and 5x5 in the last two). This allowed us to decrease the number of network parameters while minimizing accuracy loss. The final levels change in the various experiments made. The only thing that does not change is the output layer, for which we used a dense layer with 5 neurons, exploiting a SoftMax function to perform the 5-class classification. We also introduced the dropout technique in some experiments for regularization purpose. Some callbacks were also used, in particular the ModelCheckpoint to store intermediate models in case of Google Colab crash and the EarlyStopping with a patience of 15 epochs (while monitoring the validation loss). The optimizer chosen is Adam with a learing rate set to 0.0001.

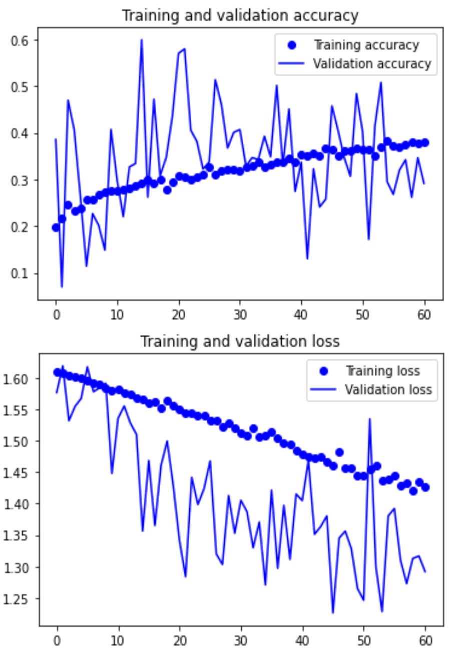
Experiment 1

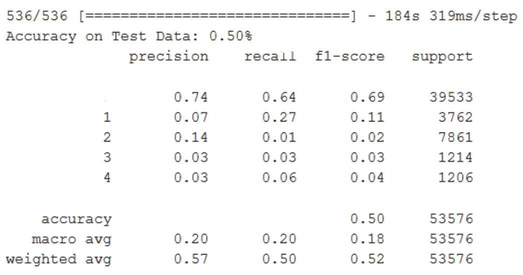
In this first experiment we tried to add a dense layer of 512 neurons without dropout and the results, shown below, were not so good.



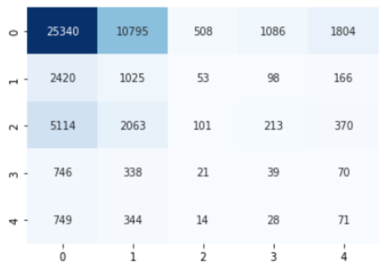
```
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 224, 224, 3)       0

 rescaling_3 (Rescaling)     (None, 224, 224, 3)       0

 conv2d_12 (Conv2D)          (None, 223, 223, 32)      416

 max_pooling2d_12 (MaxPoolin  (None, 111, 111, 32)     0
 g2D)

 conv2d_13 (Conv2D)          (None, 110, 110, 64)      8256

 max_pooling2d_13 (MaxPoolin  (None, 55, 55, 64)       0
 g2D)

 conv2d_14 (Conv2D)          (None, 54, 54, 128)       32896

 max_pooling2d_14 (MaxPoolin  (None, 18, 18, 128)      0
 g2D)

 conv2d_15 (Conv2D)          (None, 17, 17, 256)       131328

 max_pooling2d_15 (MaxPoolin  (None, 3, 3, 256)        0
 g2D)

 flatten_3 (Flatten)         (None, 2304)              0

 dense_6 (Dense)             (None, 512)               1180160

 dense_7 (Dense)             (None, 5)                 2565

=================================================================
Total params: 1,355,621
Trainable params: 1,355,621
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.1 Experiment 1 Model summary





```
536/536 [==============================] - 184s 319ms/step
Accuracy on Test Data: 0.50%
              precision    recall  f1-score   support

                   0.74      0.64      0.69     39533
           1       0.07      0.27      0.11      3762
           2       0.14      0.01      0.02      7861
           3       0.03      0.03      0.03      1214
           4       0.03      0.06      0.04      1206

    accuracy                           0.50     53576
   macro avg       0.20      0.20      0.18     53576
weighted avg       0.57      0.50      0.52     53576
```



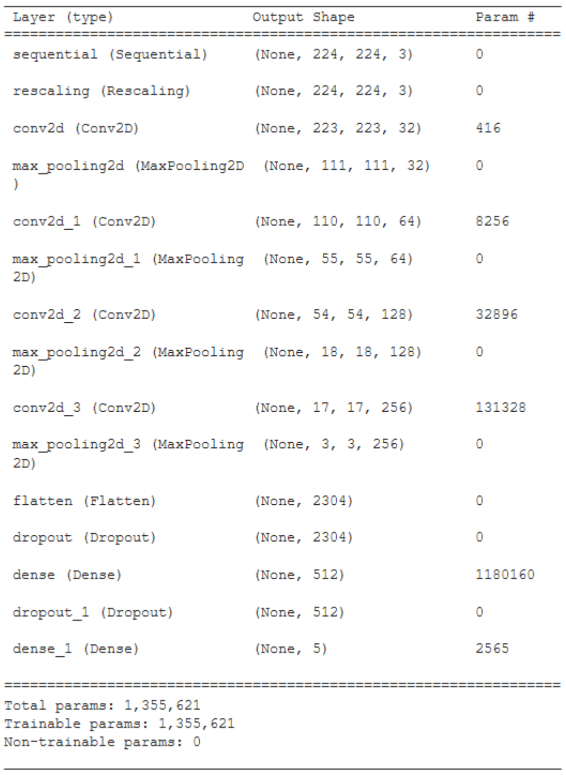
	0	1	2	3	4
0	25340	10795	508	1086	1804
1	2420	1025	53	98	166
2	5114	2063	101	213	370
3	746	338	21	39	70
4	749	344	14	28	71

Figure 4.2 Experiment 1 Results

The performance on the test set are mainly due to the class 0, showing how hard is to build a classifier that performs well on the minority classes. The validation accuracy and loss are also fluctuating a lot probably because of the different data distribution between training and validation set after down sampling in the training.

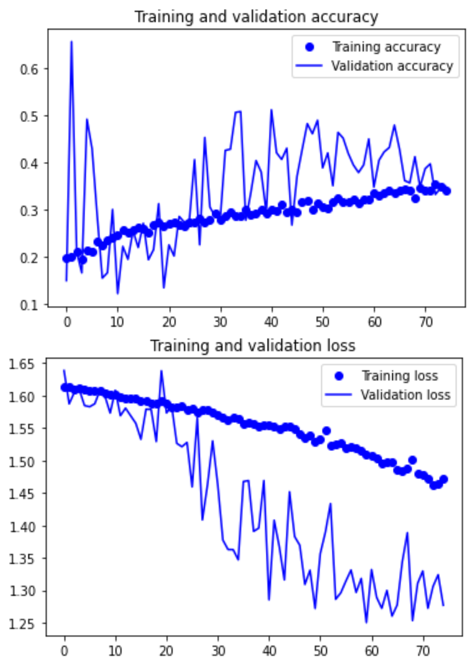
Experiment 2

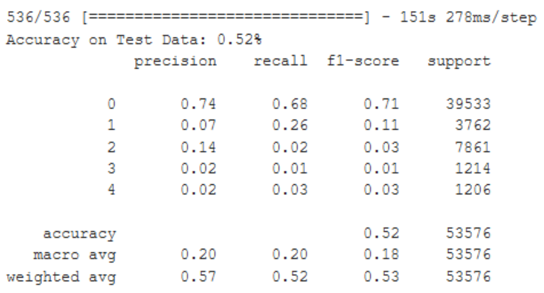
In this second experiment we added the dropout with a rate of 0.5 before and after the dense layer.



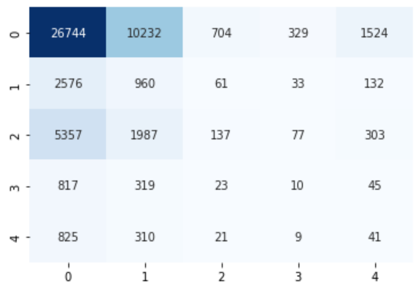
```
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 224, 224, 3)       0

 rescaling (Rescaling)       (None, 224, 224, 3)       0

 conv2d (Conv2D)             (None, 223, 223, 32)      416

 max_pooling2d (MaxPooling2D  (None, 111, 111, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 110, 110, 64)      8256

 max_pooling2d_1 (MaxPooling  (None, 55, 55, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 54, 54, 128)       32896

 max_pooling2d_2 (MaxPooling  (None, 18, 18, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 17, 17, 256)       131328

 max_pooling2d_3 (MaxPooling  (None, 3, 3, 256)        0
 2D)

 flatten (Flatten)           (None, 2304)              0

 dropout (Dropout)           (None, 2304)              0

 dense (Dense)               (None, 512)               1180160

 dropout_1 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 5)                 2565

=================================================================
Total params: 1,355,621
Trainable params: 1,355,621
Non-trainable params: 0
```

Figure 4.3 Experiment 2 Model summary





```
536/536 [==============================] - 151s 278ms/step
Accuracy on Test Data: 0.52%
               precision    recall  f1-score   support

           0       0.74      0.68      0.71     39533
           1       0.07      0.26      0.11      3762
           2       0.14      0.02      0.03      7861
           3       0.02      0.01      0.01      1214
           4       0.02      0.03      0.03      1206

    accuracy                           0.52     53576
   macro avg       0.20      0.20      0.18     53576
weighted avg       0.57      0.52      0.53     53576
```



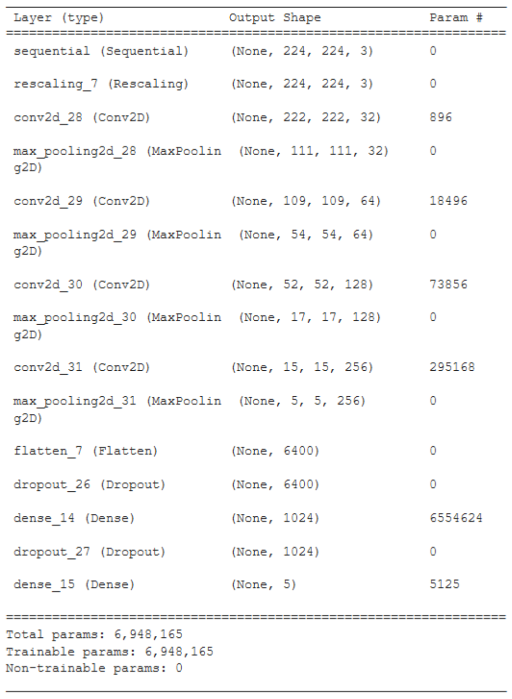
	0	1	2	3	4
0	26744	10232	704	329	1524
1	2576	960	61	33	132
2	5357	1987	137	77	303
3	817	319	23	10	45
4	825	310	21	9	41

Figure 4.4 Experiment 2 Results

The results achieved are a little bit better than the first experiment but the problems highlighted before are still present, so we decided to change something in our network for the third experiment to try to let the model generalize better.

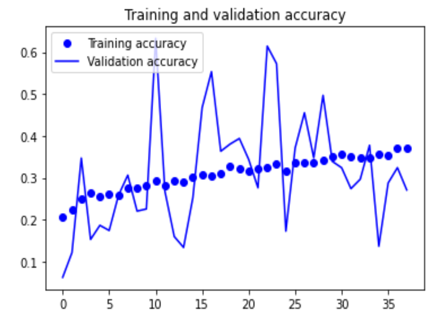
Experiment 3

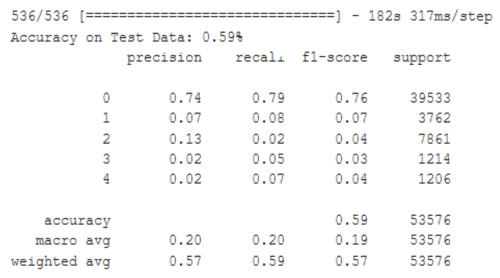
For this experiment we decided to insert a dense layer of 1024 neurons by keeping the dropout rate to 0.5. The size of the last MaxPooling is decreased from 5x5 to 3x3



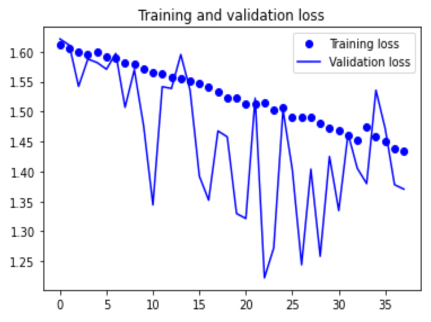
```
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 224, 224, 3)       0

 rescaling_7 (Rescaling)     (None, 224, 224, 3)       0

 conv2d_28 (Conv2D)          (None, 222, 222, 32)      896

 max_pooling2d_28 (MaxPoolin  (None, 111, 111, 32)     0
 g2D)

 conv2d_29 (Conv2D)          (None, 109, 109, 64)      18496

 max_pooling2d_29 (MaxPoolin  (None, 54, 54, 64)       0
 g2D)

 conv2d_30 (Conv2D)          (None, 52, 52, 128)       73856

 max_pooling2d_30 (MaxPoolin  (None, 17, 17, 128)      0
 g2D)

 conv2d_31 (Conv2D)          (None, 15, 15, 256)       295168

 max_pooling2d_31 (MaxPoolin  (None, 5, 5, 256)        0
 g2D)

 flatten_7 (Flatten)         (None, 6400)              0

 dropout_26 (Dropout)        (None, 6400)              0

 dense_14 (Dense)            (None, 1024)              6554624

 dropout_27 (Dropout)        (None, 1024)              0

 dense_15 (Dense)            (None, 5)                 5125

=================================================================
Total params: 6,948,165
Trainable params: 6,948,165
Non-trainable params: 0
_________________________________________________________________
```

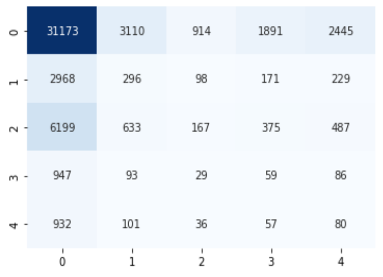
Figure 4.5 Experiment 3 Model summary





```
536/536 [==============================] - 182s 317ms/step
Accuracy on Test Data: 0.59%
              precision    recall  f1-score   support

           0       0.74      0.79      0.76     39533
           1       0.07      0.08      0.07      3762
           2       0.13      0.02      0.04      7861
           3       0.02      0.05      0.03      1214
           4       0.02      0.07      0.04      1206

    accuracy                           0.59     53576
   macro avg       0.20      0.20      0.19     53576
weighted avg       0.57      0.59      0.57     53576
```





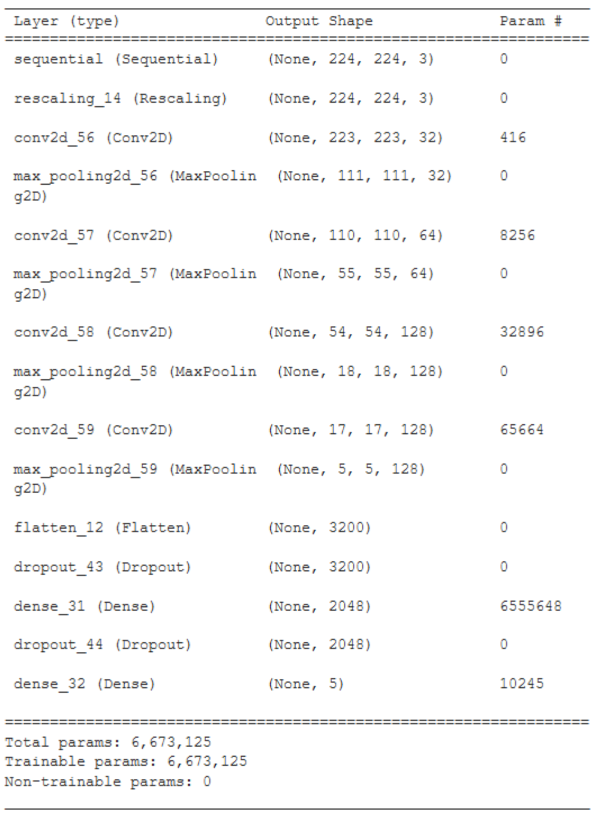
	0	1	2	3	4
0	31173	3110	914	1891	2445
1	2968	296	98	171	229
2	6199	633	167	375	487
3	947	93	29	59	86
4	932	101	36	57	80

Figure 4.6 Experiment 3 Results

With this third experiment we can notice an increase in terms of overall accuracy that reaches 59%, but it is always caused by the good performances for the class 0. As we can see from the two graphs of training and validation loss, the model is still not able to converge and it reaches the early stopping after 35 epochs (a lot earlier than the two experiments done before).

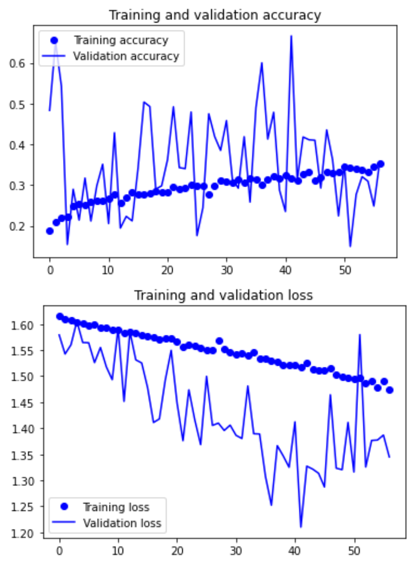
Experiment 4

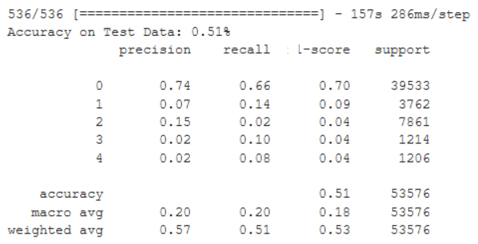
In the fourth experiment we introduced a dense layer of 2048 neurons



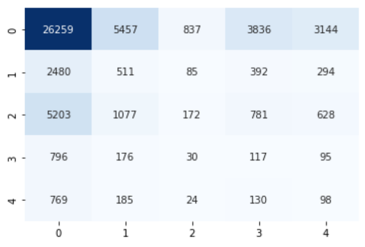
```
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 224, 224, 3)       0

 rescaling_14 (Rescaling)    (None, 224, 224, 3)       0

 conv2d_56 (Conv2D)          (None, 223, 223, 32)      416

 max_pooling2d_56 (MaxPoolin  (None, 111, 111, 32)     0
 g2D)

 conv2d_57 (Conv2D)          (None, 110, 110, 64)      8256

 max_pooling2d_57 (MaxPoolin  (None, 55, 55, 64)       0
 g2D)

 conv2d_58 (Conv2D)          (None, 54, 54, 128)       32896

 max_pooling2d_58 (MaxPoolin  (None, 18, 18, 128)      0
 g2D)

 conv2d_59 (Conv2D)          (None, 17, 17, 128)       65664

 max_pooling2d_59 (MaxPoolin  (None, 5, 5, 128)        0
 g2D)

 flatten_12 (Flatten)        (None, 3200)              0

 dropout_43 (Dropout)        (None, 3200)              0

 dense_31 (Dense)            (None, 2048)              6555648

 dropout_44 (Dropout)        (None, 2048)              0

 dense_32 (Dense)            (None, 5)                 10245

=================================================================
Total params: 6,673,125
Trainable params: 6,673,125
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.7 Experiment 4 Model summary





```
536/536 [==============================] - 157s 286ms/step
Accuracy on Test Data: 0.51%
              precision    recall  l-score   support

           0       0.74      0.66      0.70     39533
           1       0.07      0.14      0.09      3762
           2       0.15      0.02      0.04      7861
           3       0.02      0.10      0.04      1214
           4       0.02      0.08      0.04      1206

    accuracy                           0.51     53576
   macro avg       0.20      0.20      0.18     53576
weighted avg       0.57      0.51      0.53     53576
```



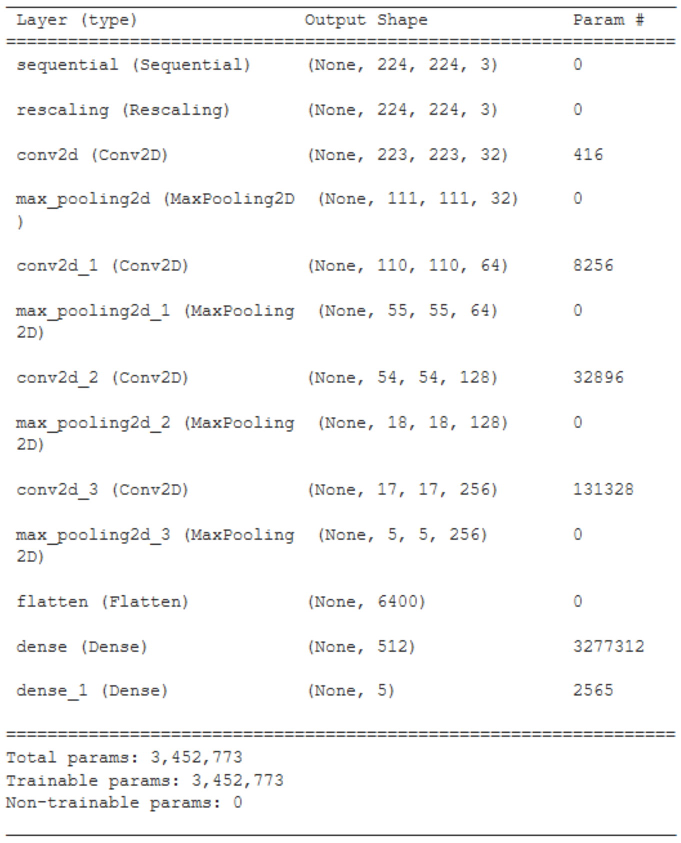
	0	1	2	3	4
0	26259	5457	837	3836	3144
1	2480	511	85	392	294
2	5203	1077	172	781	628
3	796	176	30	117	95
4	769	185	24	130	98

Figure 4.8 Experiment 4 Results

The results in this case didn’t improve and the accuracy came back to the initial values (around 50%). It is important to highlight the fact that, with this configuration, the network performs better than the other in detecting class 3 images (but still not sufficient performances) as shown in the confusion matrix above.

Experiment 5

Considering the poor performances achieved by the experiments 1-4, we decided to take all the imbalanced training dataset (obtained from the initial split 90-10), that has a similar distribution of data with respect to validation and test set, to train a network. We saw this approach in the literature and we tried to reproduce it to see if our results could have been better than the previous experiments. The summary of this experiment is shown below.



```
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 224, 224, 3)       0

 rescaling (Rescaling)       (None, 224, 224, 3)       0

 conv2d (Conv2D)             (None, 223, 223, 32)      416

 max_pooling2d (MaxPooling2D  (None, 111, 111, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 110, 110, 64)      8256

 max_pooling2d_1 (MaxPooling  (None, 55, 55, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 54, 54, 128)       32896

 max_pooling2d_2 (MaxPooling  (None, 18, 18, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 17, 17, 256)       131328

 max_pooling2d_3 (MaxPooling  (None, 5, 5, 256)        0
 2D)

 flatten (Flatten)           (None, 6400)              0

 dense (Dense)               (None, 512)               3277312

 dense_1 (Dense)             (None, 5)                 2565

=================================================================
Total params: 3,452,773
Trainable params: 3,452,773
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.9 Experiment 5 Model summary

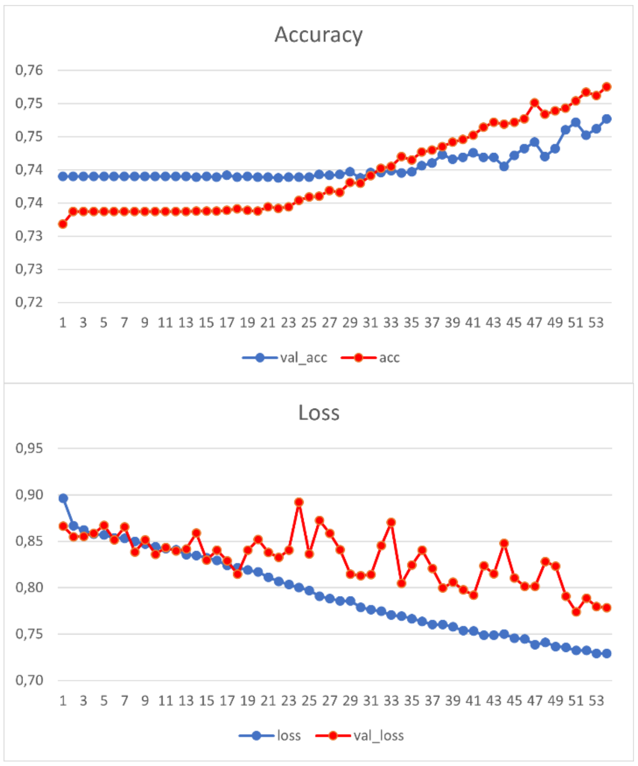


Figure 4.110 Training epoch 1-53

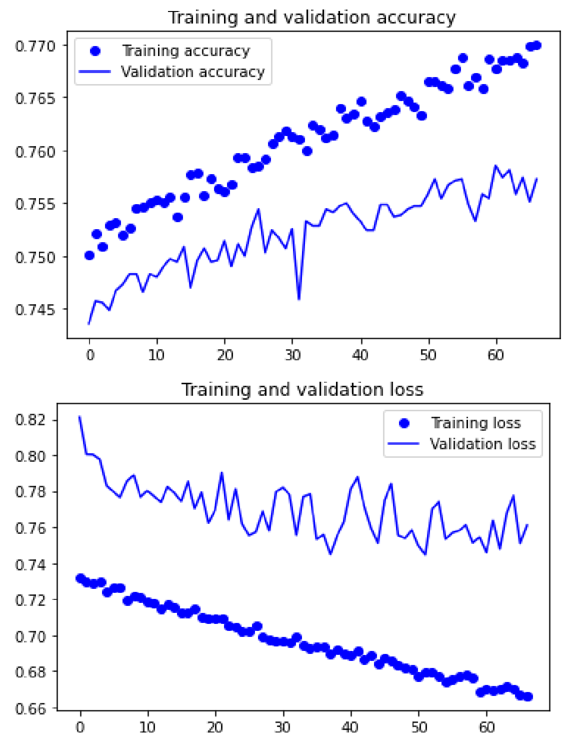
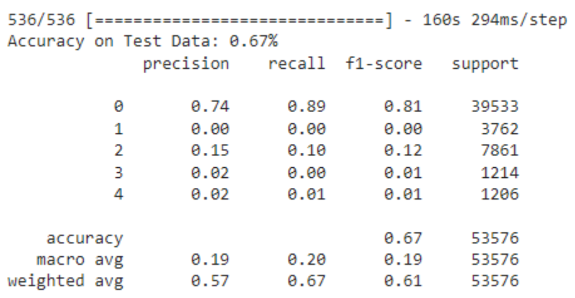
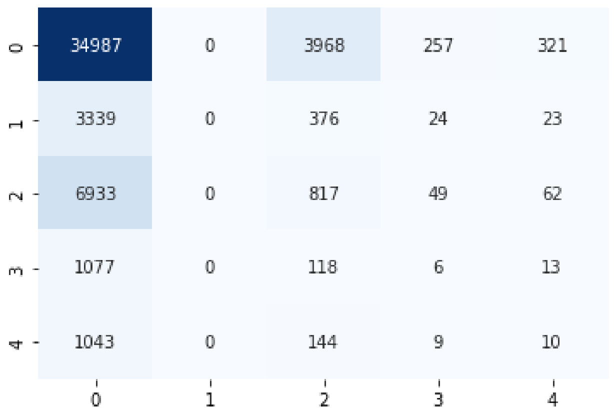


Figure 4.101 Training epochs 54-130



```
536/536 [==============================] - 160s 294ms/step
Accuracy on Test Data: 0.67%
              precision    recall  f1-score   support

           0       0.74      0.89      0.81     39533
           1       0.00      0.00      0.00      3762
           2       0.15      0.10      0.12      7861
           3       0.02      0.00      0.01      1214
           4       0.02      0.01      0.01      1206

    accuracy                           0.67     53576
   macro avg       0.19      0.20      0.19     53576
weighted avg       0.57      0.67      0.61     53576
```



	0	1	2	3	4
0	34987	0	3968	257	321
1	3339	0	376	24	23
2	6933	0	817	49	62
3	1077	0	118	6	13
4	1043	0	144	9	10

Figure 4.12 Experiment 5 Results

As we can see from the images above, we split the training in two phases because of a disconnection of the Colab environment due to the long training times (~3.20 hours of training for 50 epochs). Despite this fact, we can see that the curves of accuracy and loss for the validation and training set were more regular than the previous experiments. We can also see a similar behaviour with respect to the approaches found in literature [19] with the model initially stuck at a certain value for the accuracy (due to the correct classification only for class 0) and then it starts to learn how to correctly classify new instances of different classes. The big problem of this approach is the long training time required and due to Colab limitations we couldn't study in deep this experiment. Last but not least the model is not able to classify any instance of class 1 and only a few instances of class 3 and 4, but we think that going deeper into the training of this model (as seen in the literature) it would be possible to reach good results for all the classes considered.

5. Pre-trained models

The results obtained with CNN from scratch are already quite satisfactory. However, we think we can achieve even better performance by exploiting transfer learning. In fact, starting from the training set available, it is difficult for us to develop powerful networks that can generalize well. This task can be simplified and improved by using pre-trained networks. In particular we apply a trial-and-error approach for each of the networks mentioned below in order to have good results for our problem.

ResNet50, short for Residual Network with 50 layers, is a specific type of neural network that was introduced in 2015 by K. He, X. Zhang, S. Ren and J. Sun in their paper “Deep Residual Learning for Image Recognition” [7].

VGG16 is a convolutional neural network model proposed in 2014 by K. Simonyan and A. Zisserman from the University of Oxford in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition” [6].

MobileNetV2 builds upon the ideas from MobileNetV1, using depth wise separable convolution as efficient building blocks. However, V2 introduces two new features to the architecture: 1) linear bottlenecks between the layers, and 2) shortcut connections between the bottlenecks. [8]

InceptionV3 was proposed in the paper “Rethinking the Inception Architecture for Computer Vision”, published in 2015. Inception v3 mainly focuses on burning less computational power by modifying the previous Inception architectures. [9]

Since the challenge deals with medical images, which are very different from the images in ImageNet, the expectation to achieve good results were low. In particular, despite trying different combinations for the classifier and different regularization techniques, the model was only able to perform well on the training set, showing an inability to generalize

5.1 VGG16

VGG16 is a convolution neural net (CNN) architecture which was used to win ILSVR (ImageNet) competition in 2014, it is a pretty large network and it has about 138 million parameters. Most unique thing about VGG16 is that instead of having a large number of hyper-parameter they focused on having convolution layers of 3x3 filter with a stride 1 and always used same padding and max pool layer of 2x2 filter of stride 2. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end it has 2 fully connected layers followed by a SoftMax for output. The 16 in VGG16 refers to it has 16 layers that have weights.

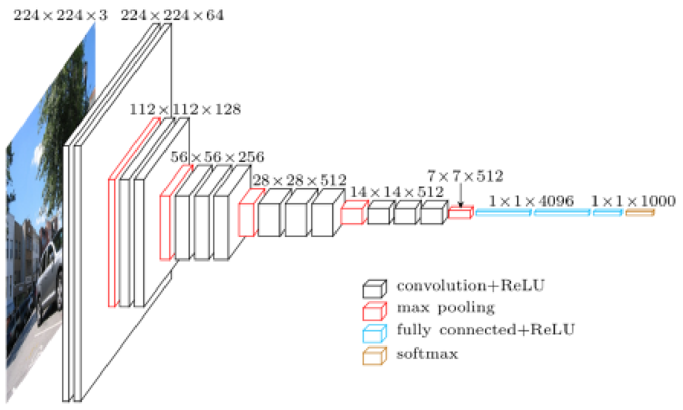


Figure 5.1 Architecture of VGG16

During training, the input to the convnets is a fixed-size 224 x 224 RGB image (3 channels). Subtracting the mean RGB value computed on the training set from each pixel is the only pre-processing done here. The image is passed through a stack of convolutional layers, where filters with a very small receptive field 3 × 3 are used. The convolution stride and the spatial padding of convolutional layer input is fixed to 1 pixel for 3 x 3 convolutional layers, which ensures that the spatial resolution is preserved after convolution. Five max-pooling layers, which follow some of the convolutional layers, helps in spatial pooling. Max-pooling is performed over a 2×2 pixel window, with stride 2. There are three Fully Connected layers that follow a stack of convolutional layers: the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks. [12][13]

5.1.1 Feature Extraction

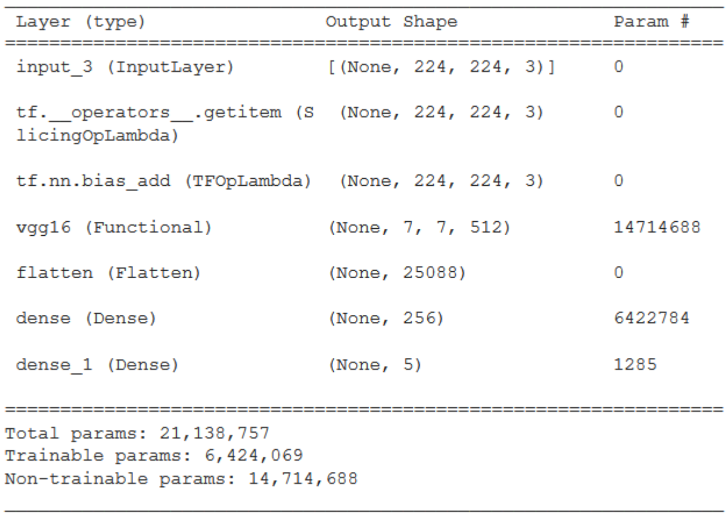
Our first approach was to use the convolutional base of the architecture only as a feature extractor. Since the image present in ImageNet are very different from the images contained in our dataset we did not expect to achieve very good results, and so it was. Although we have tried different combinations of layer the results obtained are not satisfying, in particular the model was not able to generalize well.

5.1.2 Feature Extraction Experiments

In all the following experiment the categorical cross entropy is utilized as loss function, it is designed to quantify the difference among probability distribution. Also, due to the limitation imposed by Google Colab, the function ModelCheckpoint is utilized to save the intermediate model after an improvement, in terms of val_loss, in order to restore the execution in case of Google Colab runtime disconnection. Therefore in the call-back functions is added an EarlyStop method which stop the training after 10 epochs without val_loss improvement. This choice is made because the time needed to train a single epoch is heavy in terms of duration.

Experiment 1

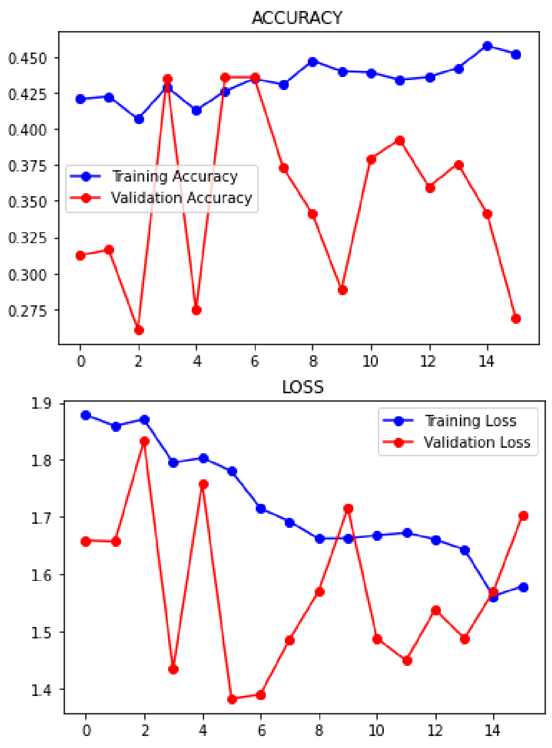
In this first experiment a Flatten layer is introduced between the convolutional base and the final dense layer (that utilize a SoftMax activation function). This layer is responsible of making the multidimensional input into one dimensional and is commonly used in the transition from convolutional layer to the full connected layer.

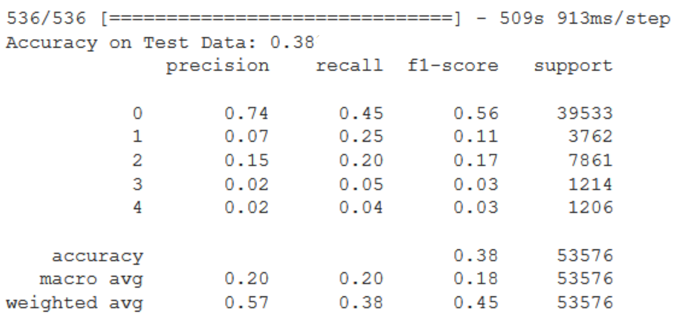


```
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.__operators__.getitem (S  (None, 224, 224, 3)      0
 licingOpLambda)

 tf.nn.bias_add (TFOpLambda)  (None, 224, 224, 3)      0

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 256)               6422784

 dense_1 (Dense)             (None, 5)                 1285

=================================================================
Total params: 21,138,757
Trainable params: 6,424,069
Non-trainable params: 14,714,688
_________________________________________________________________
```

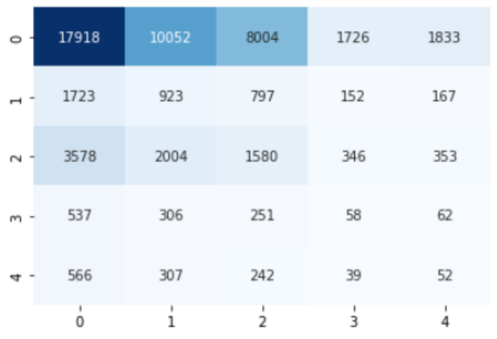
Figure 5.2 Experiment 1 Model summary

The model is discontinuous and computationally heavy, the results obtained in this case are not very satisfactory. As shown in Figure 5.3 in the final epochs is present also overfitting





```
536/536 [==============================] - 509s 913ms/step
Accuracy on Test Data: 0.38
              precision    recall  f1-score   support

           0       0.74      0.45      0.56     39533
           1       0.07      0.25      0.11      3762
           2       0.15      0.20      0.17      7861
           3       0.02      0.05      0.03      1214
           4       0.02      0.04      0.03      1206

    accuracy                           0.38     53576
   macro avg       0.20      0.20      0.18     53576
weighted avg       0.57      0.38      0.45     53576
```



	0	1	2	3	4
0	17918	10052	8004	1726	1833
1	1723	923	797	152	167
2	3578	2004	1580	346	353
3	537	306	251	58	62
4	566	307	242	39	52

Figure 5.3 Experiment 1 Results

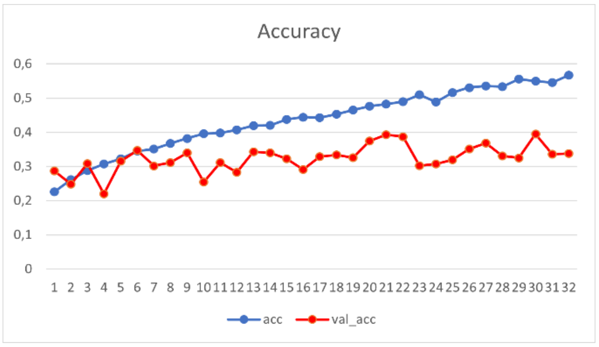
Experiment 2

To fight the overfitting present in the first experiment is introduced a regularization with a Dropout layer with a rate of 0.5. A fully connected layer of 128 neurons is introduced to involves weight biases and neurons calibration. This is used to classify images among different category by training. The Flatten layer is still used after the convolutional base.

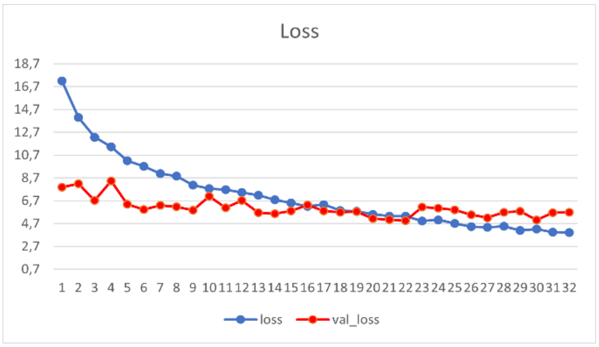
```
 Layer (type)                Output Shape              Param #
=================================================================
 input_5 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.__operators__.getitem_1   (None, 224, 224, 3)      0
 (SlicingOpLambda)

 tf.nn.bias_add_1 (TFOpLambd  (None, 224, 224, 3)      0
 a)

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten_1 (Flatten)         (None, 25088)             0

 dense_2 (Dense)             (None, 128)               3211392

 dropout (Dropout)           (None, 128)               0

 dense_3 (Dense)             (None, 5)                 645

=================================================================
Total params: 17,926,725
Trainable params: 3,212,037
Non-trainable params: 14,714,688
_________________________________________________________________
```

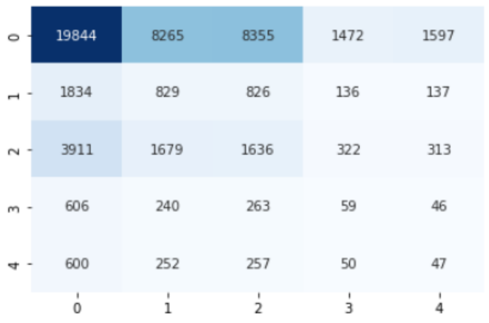
Figure 5.4 Experiment 2 Model summary

As the Figure 5.5 Experiment 2 Results below shows, the model does not produce tangible improvements even after the execution of more than 30 epochs, both the accuracy and the loss are rather static.



```
Accuracy on Test Data: 0.42
              precision    recall  f1-score   support

           0       0.74      0.50      0.60     39533
           1       0.07      0.22      0.11      3762
           2       0.14      0.21      0.17      7861
           3       0.03      0.05      0.04      1214
           4       0.02      0.04      0.03      1206

    accuracy                           0.42     53576
   macro avg       0.20      0.20      0.19     53576
weighted avg       0.57      0.42      0.48     53576
```



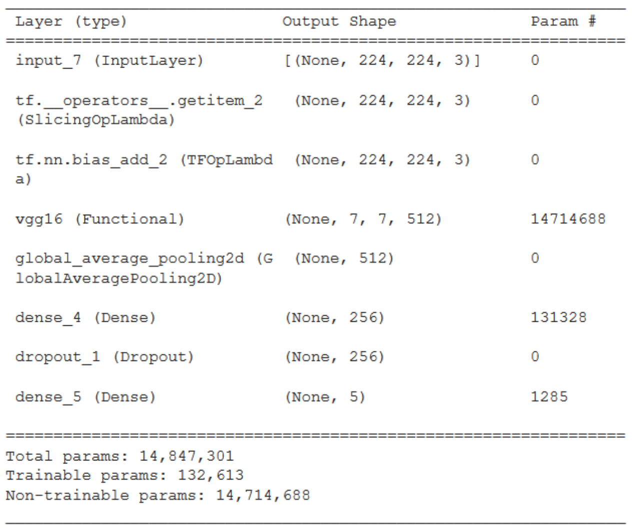


	0	1	2	3	4
0	19844	8265	8355	1472	1597
1	1834	829	826	136	137
2	3911	1679	1636	322	313
3	606	240	263	59	46
4	600	252	257	50	47

Figure 5.5 Experiment 2 Results

Experiment 3 (Adam)

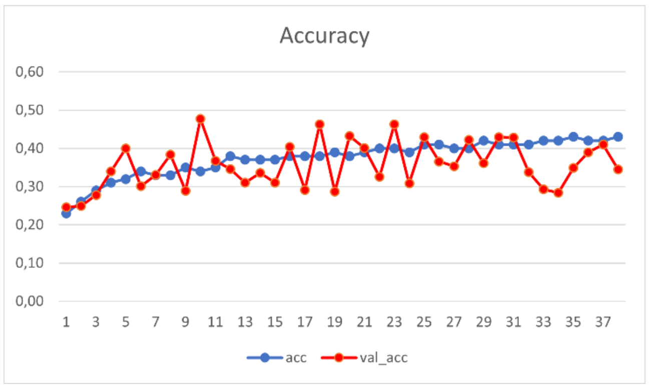
The improvement performed on this experiment is the substitution of the Flatten Layer with a GlobalAveragePooling2D layer. Using this techniques the output's dimension is reduced.

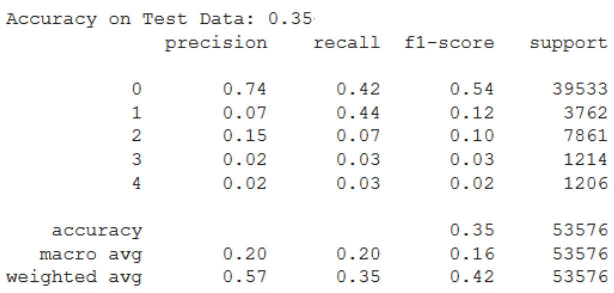


```
 Layer (type)                Output Shape              Param #
=================================================================
 input_7 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.__operators__.getitem_2   (None, 224, 224, 3)      0
 (SlicingOpLambda)

 tf.nn.bias_add_2 (TFOpLambd  (None, 224, 224, 3)      0
 a)

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 global_average_pooling2d (G  (None, 512)              0
 lobalAveragePooling2D)

 dense_4 (Dense)             (None, 256)               131328

 dropout_1 (Dropout)         (None, 256)               0

 dense_5 (Dense)             (None, 5)                 1285

=================================================================
Total params: 14,847,301
Trainable params: 132,613
Non-trainable params: 14,714,688
_________________________________________________________________
```

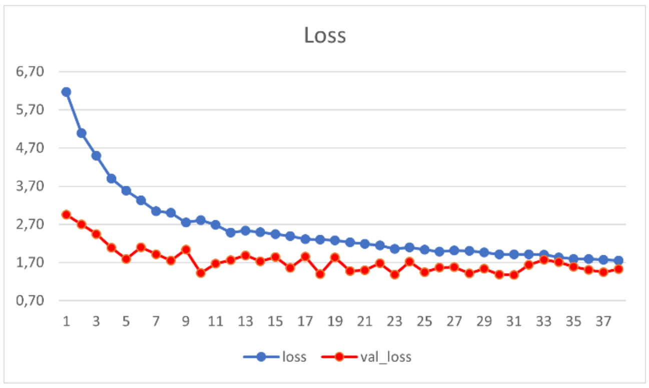
Figure 5.6 Experiment 3 (Adam) Model summary

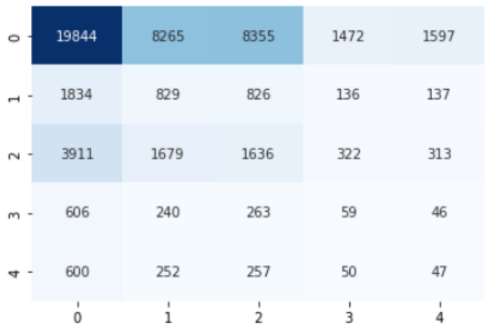
Validation results follow the training fairly well but the values obtained in terms of accuracy and loss are not satisfying.





```
Accuracy on Test Data: 0.35
              precision    recall  f1-score   support

           0       0.74      0.42      0.54     39533
           1       0.07      0.44      0.12      3762
           2       0.15      0.07      0.10      7861
           3       0.02      0.03      0.03      1214
           4       0.02      0.03      0.02      1206

    accuracy                           0.35     53576
   macro avg       0.20      0.20      0.16     53576
weighted avg       0.57      0.35      0.42     53576
```



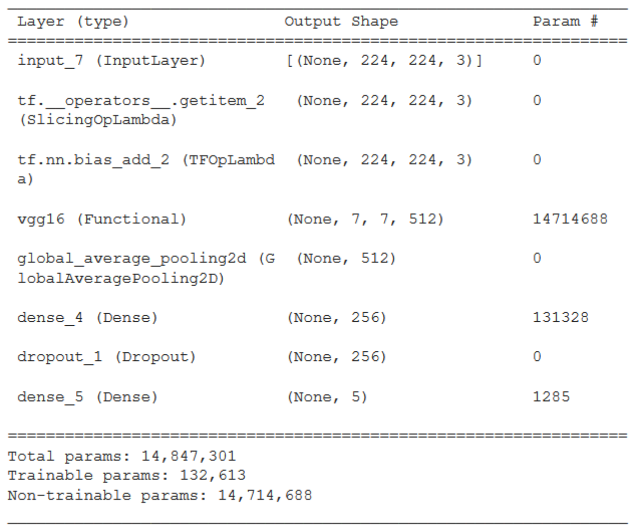


	0	1	2	3	4
0	19844	8265	8355	1472	1597
1	1834	829	826	136	137
2	3911	1679	1636	322	313
3	606	240	263	59	46
4	600	252	257	50	47

Figure 5.7 Experiment 3 (Adam) Results

Experiment 3 (SGD)

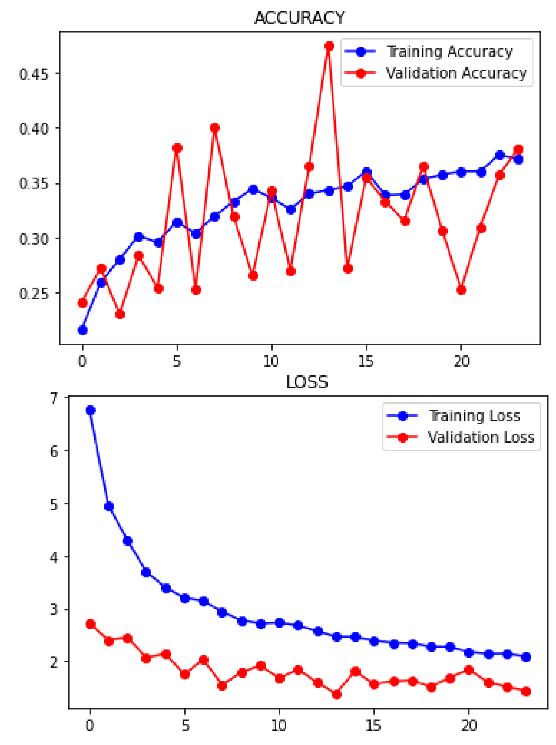
This experiment is equal to the previous one with the difference of using SGD optimizer instead of Adam optimizer

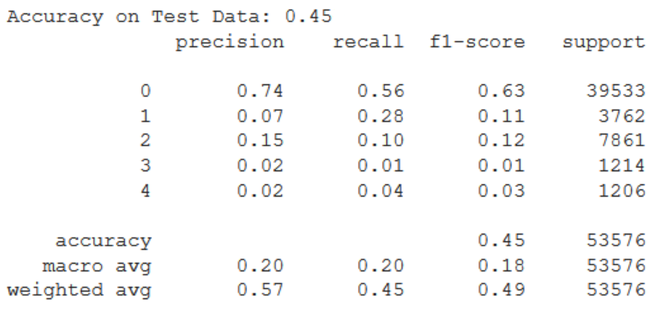


```
 Layer (type)                Output Shape              Param #
=================================================================
 input_7 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.__operators__.getitem_2   (None, 224, 224, 3)      0
 (SlicingOpLambda)

 tf.nn.bias_add_2 (TFOpLambd  (None, 224, 224, 3)      0
 a)

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 global_average_pooling2d (G  (None, 512)              0
 lobalAveragePooling2D)

 dense_4 (Dense)             (None, 256)               131328

 dropout_1 (Dropout)         (None, 256)               0

 dense_5 (Dense)             (None, 5)                 1285

=================================================================
Total params: 14,847,301
Trainable params: 132,613
Non-trainable params: 14,714,688
_________________________________________________________________
```

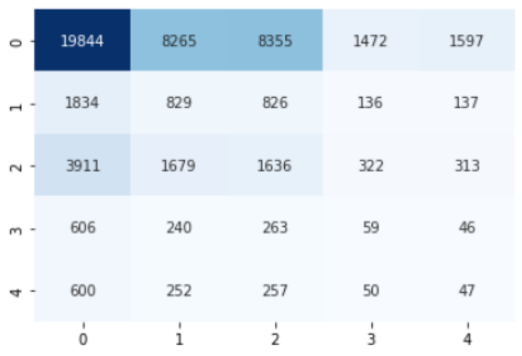
Figure 5.8 Experiment 3 (SGD) Model summary

No improvement in terms of the results are present after the change of the optimizer





```
Accuracy on Test Data: 0.45
              precision    recall  f1-score   support

           0       0.74      0.56      0.63     39533
           1       0.07      0.28      0.11      3762
           2       0.15      0.10      0.12      7861
           3       0.02      0.01      0.01      1214
           4       0.02      0.04      0.03      1206

    accuracy                           0.45     53576
   macro avg       0.20      0.20      0.18     53576
weighted avg       0.57      0.45      0.49     53576
```



	0	1	2	3	4
0	19844	8265	8355	1472	1597
1	1834	829	826	136	137
2	3911	1679	1636	322	313
3	606	240	263	59	46
4	600	252	257	50	47

Figure 5.9 Experiment 3 (SGD) Results

5.1.3 Fine Tuning

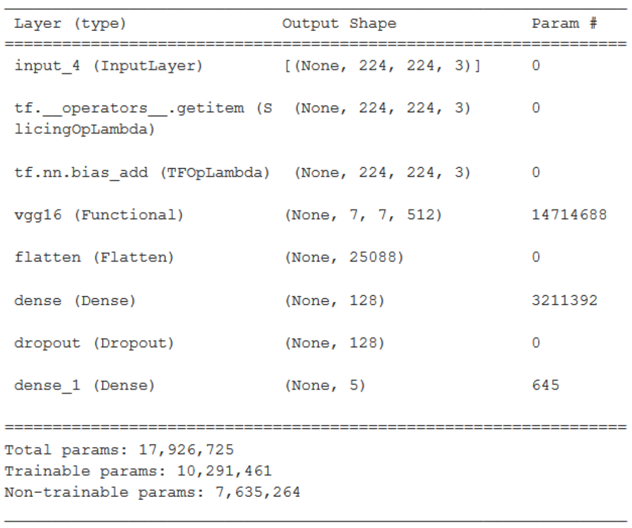
Finetuning is used to taking weights of a trained neural network and use it as initialization for a new model being trained on data from the same domain. It is used also to speed up the training and overcome small dataset size. There are various strategies, such as training the whole initialized network or "freezing" some of the pre-trained weights (usually whole initial layers). The following experiments are based on the unfreezing approach.

5.1.4 Fine Tuning Experiment

All the following experiments are performed on model 3 due to its better performances.

Experiment 4

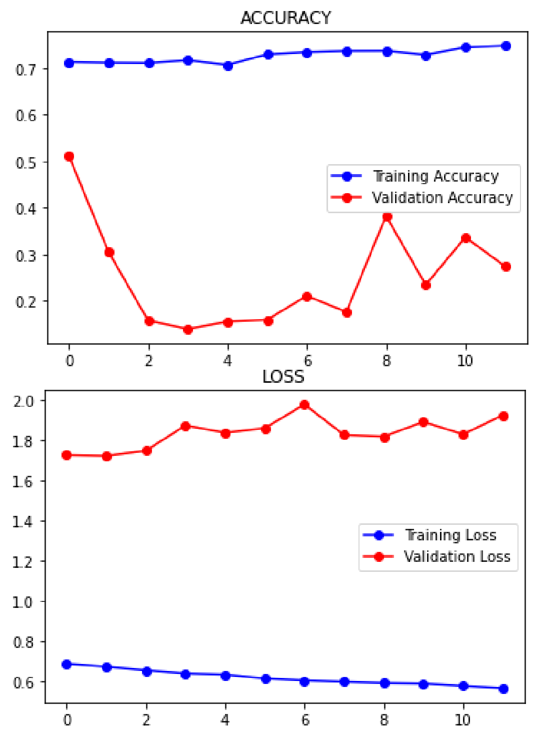
In this experiment all the blocks from the block5_conv1 are unfrozen to increase trainable variables.



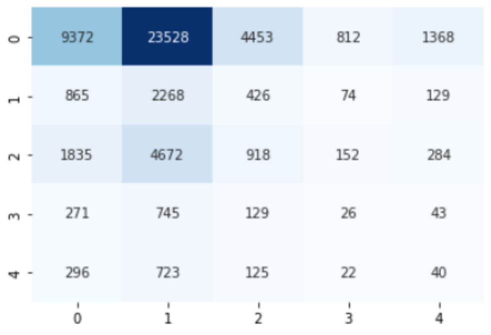
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.__operators__.getitem (S  (None, 224, 224, 3)      0
 licingOpLambda)

 tf.nn.bias_add (TFOpLambda)  (None, 224, 224, 3)      0

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 128)               3211392

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 5)                 645

=================================================================
Total params: 17,926,725
Trainable params: 10,291,461
Non-trainable params: 7,635,264
_________________________________________________________________
```

Figure 5.10 Experiment 4 Model summary

Unfreezing the last block the CNN does not learn well from the training set, both accuracy and loss has results not satisfying and the early stopping phase occur fast. Another possible experiment will consider increasing the patience but, due to the long duration of the epochs and the limitation of Google Colab, this was not tried.



```
Accuracy on Test Data: 23.56%
              precision    recall  f1-score   support

           0       0.74      0.24      0.36     39533
           1       0.07      0.60      0.13      3762
           2       0.15      0.12      0.13      7861
           3       0.02      0.02      0.02      1214
           4       0.02      0.03      0.03      1206

    accuracy                           0.24     53576
   macro avg       0.20      0.20      0.13     53576
weighted avg       0.58      0.24      0.29     53576
```

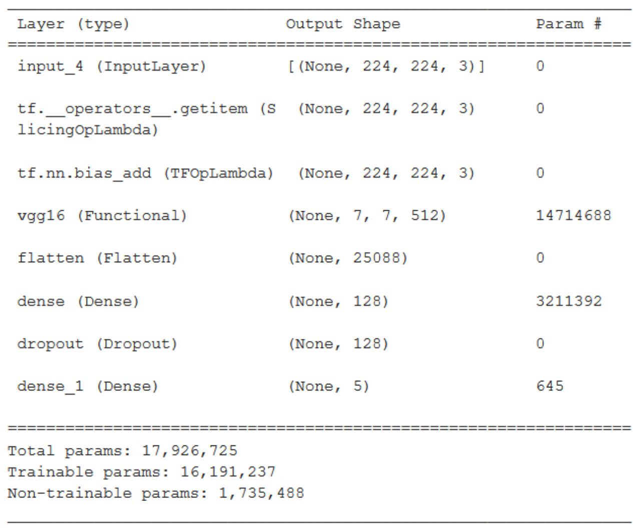


	0	1	2	3	4
0	9372	23528	4453	812	1368
1	865	2268	426	74	129
2	1835	4672	918	152	284
3	271	745	129	26	43
4	296	723	125	22	40

Figure 5.11 Experiment 4 Results

Experiment 5

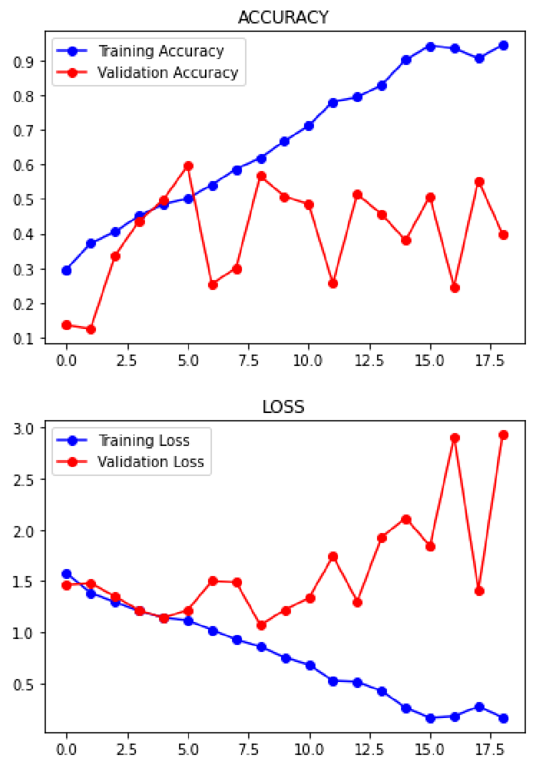
In this experiment all the blocks from the block4_conv1 are unfrozen to increase trainable variables.



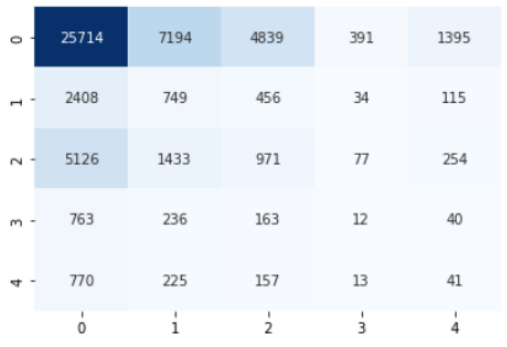
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.__operators__.getitem (S  (None, 224, 224, 3)      0
 licingOpLambda)

 tf.nn.bias_add (TFOpLambda)  (None, 224, 224, 3)      0

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 128)               3211392

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 5)                 645

=================================================================
Total params: 17,926,725
Trainable params: 16,191,237
Non-trainable params: 1,735,488
_________________________________________________________________
```

Figure 5.12 Experiment 5 Model summary

A better behaviour was encountered with this experiment but the results are still not satisfying, the validation and training are still far from each other and the experiment is not so good.



```
Accuracy on Test Data: 51.30%
              precision    recall  f1-score   support

           0       0.74      0.65      0.69     39533
           1       0.08      0.20      0.11      3762
           2       0.15      0.12      0.13      7861
           3       0.02      0.01      0.01      1214
           4       0.02      0.03      0.03      1206

    accuracy                           0.51     53576
   macro avg       0.20      0.20      0.20     53576
weighted avg       0.57      0.51      0.54     53576
```

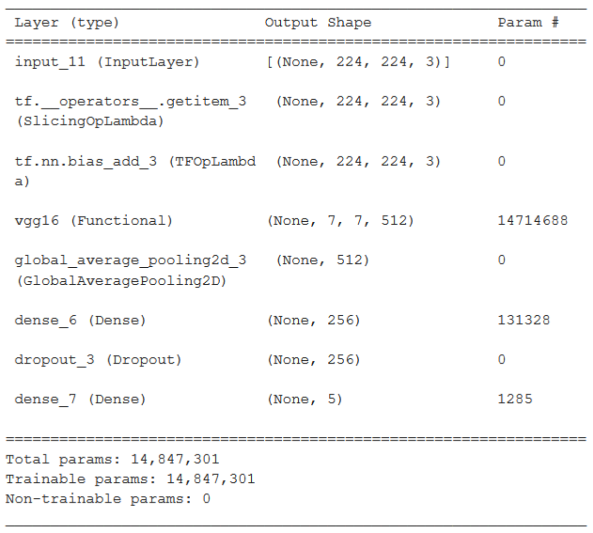


	0	1	2	3	4
0	25714	7194	4839	391	1395
1	2408	749	456	34	115
2	5126	1433	971	77	254
3	763	236	163	12	40
4	770	225	157	13	41

Figure 5.13 Experiment 5 Results

Experiment 6

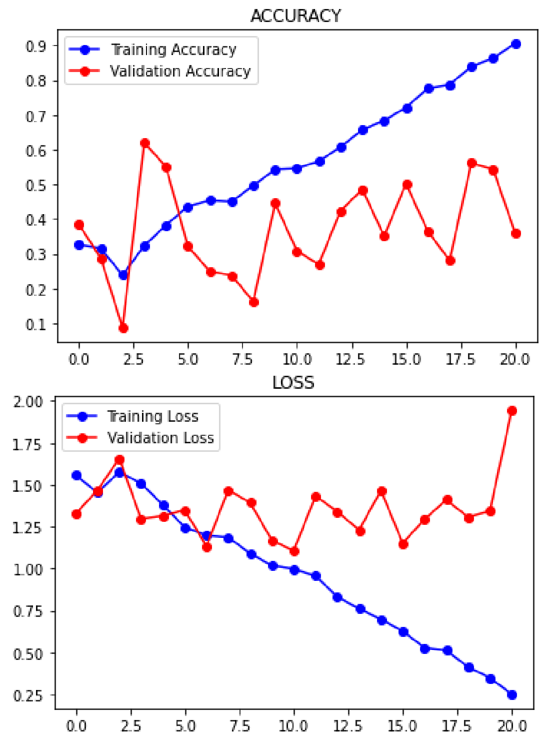
In this experiment all the blocks from the block1_conv1 are unfrozen to increase trainable variables.

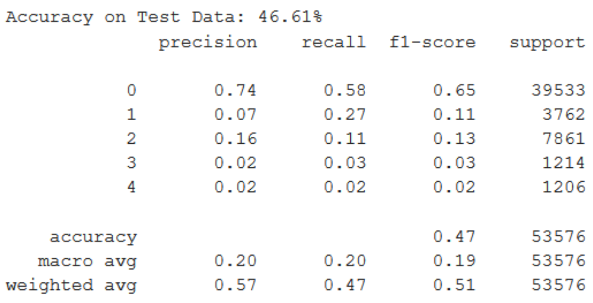


```
 Layer (type)                Output Shape              Param #
=================================================================
 input_11 (InputLayer)       [(None, 224, 224, 3)]     0

 tf.__operators__.getitem_3   (None, 224, 224, 3)      0
 (SlicingOpLambda)

 tf.nn.bias_add_3 (TFOpLambd  (None, 224, 224, 3)      0
 a)

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 global_average_pooling2d_3   (None, 512)              0
 (GlobalAveragePooling2D)

 dense_6 (Dense)             (None, 256)               131328

 dropout_3 (Dropout)         (None, 256)               0

 dense_7 (Dense)             (None, 5)                 1285

=================================================================
Total params: 14,847,301
Trainable params: 14,847,301
Non-trainable params: 0
_________________________________________________________________
```

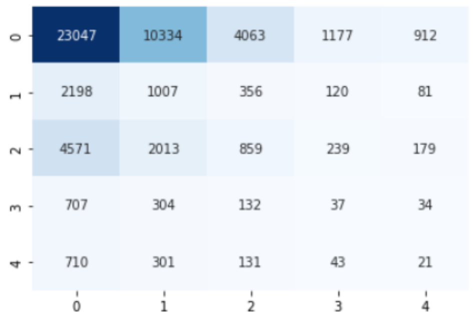
Figure 5.14 Experiment 6 Model summary

Instead all the layers are unfreeze the training speed does not increase so much with respect to the previous model and the validation has already not so good results.





```
Accuracy on Test Data: 46.61%
              precision    recall  f1-score   support

           0       0.74      0.58      0.65     39533
           1       0.07      0.27      0.11      3762
           2       0.16      0.11      0.13      7861
           3       0.02      0.03      0.03      1214
           4       0.02      0.02      0.02      1206

    accuracy                           0.47     53576
   macro avg       0.20      0.20      0.19     53576
weighted avg       0.57      0.47      0.51     53576
```



	0	1	2	3	4
0	23047	10334	4063	1177	912
1	2198	1007	356	120	81
2	4571	2013	859	239	179
3	707	304	132	37	34
4	710	301	131	43	21

Figure 5.15 Experiment 6 Model summary

5.2 MobileNetV2

MobileNetV2 is a convolutional neural network architecture that seeks to perform well on mobile devices. As a whole, the architecture, as we can see in Figure 5.16 MobileNetV2 architecture ,contains an initial fully convolution layer with 32 filters, followed by original basic blocks named bottleneck. These blocks are then followed by a 1×1 convolution layer with an average pooling layer. The last layer is the classification layer. ReLU is used as non-linearity because of its robustness when used with low-precision computation. Furthermore, kernel size 3×3 is always used as is standard for modern networks, and dropout and batch normalization were utilized during training. The output is a vector of 1000 probabilities corresponding to the 1000 classes in the ImageNet dataset. The input image is assigned to the class with the maximum probability. [14]

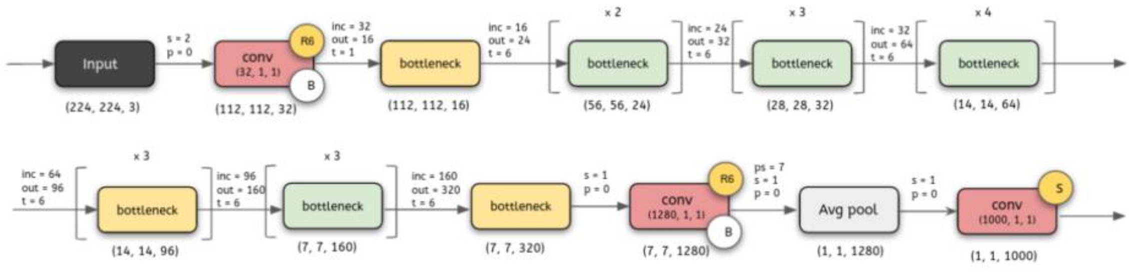


Figure 5.16 MobileNetV2 architecture

5.2.1 Feature Extraction

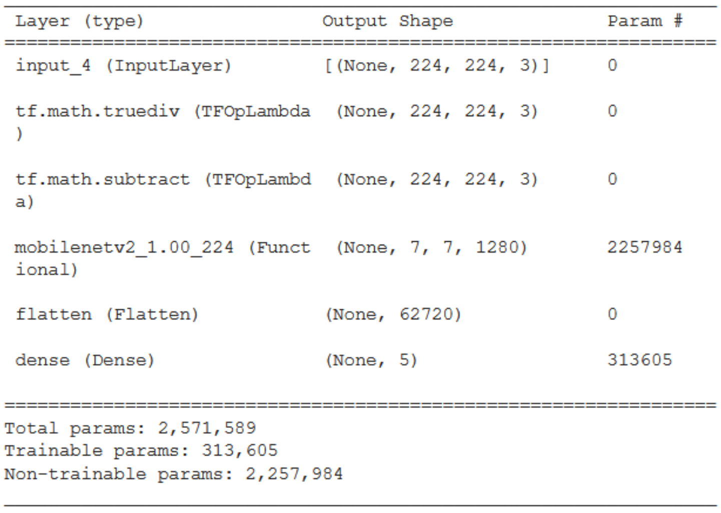
Our first approach was to use the convolutional base of the architecture only as a feature extractor. Since the image present in ImageNet are very different from the images contained in our dataset we did not expect to achieve very good results, and so it was. Although we have tried different combinations of layer the results obtained are not satisfying, in particular the model was not able to generalize well.

5.2.2 Feature Extraction Experiment

In all the following experiment the categorical cross entropy is utilized as loss function, it is designed to quantify the difference among probability distribution. Also, due to the limitation imposed by Google Colab, the function ModelCheckpoint is utilized to save the intermediate model after an improvement, in terms of val_loss, in order to restore the execution in case of Google Colab runtime disconnection. Therefore in the call-back functions is added an EarlyStop method which stop the training after 10 epochs without val_loss improvement. This choice is made because the time needed to train a single epoch is heavy in terms of duration.

Experiment 1

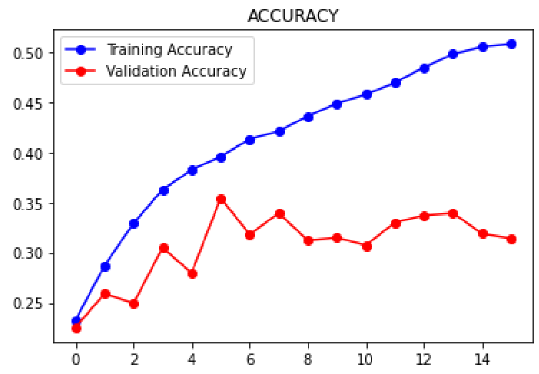
In this initial experiment is introduced a Flatten layer, used to make the multidimensional input into one dimensional. It is commonly used in the transition from convolution layer to the full connected layer. This layer was positioned between the convolutional base and the final dense layer with SoftMax activation function. The learning rate was set to 0.00001.

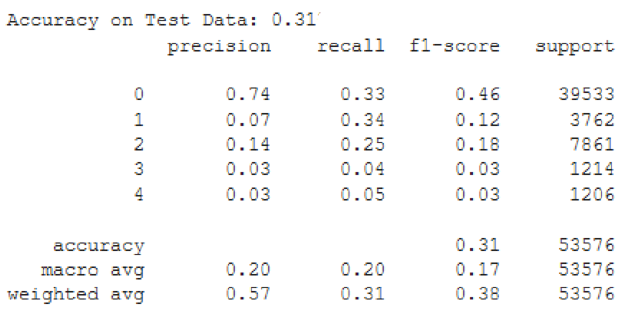


```
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.math.truediv (TFOpLambda  (None, 224, 224, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 224, 224, 3)      0
 a)

 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 flatten (Flatten)           (None, 62720)             0

 dense (Dense)               (None, 5)                 313605

=================================================================
Total params: 2,571,589
Trainable params: 313,605
Non-trainable params: 2,257,984
```

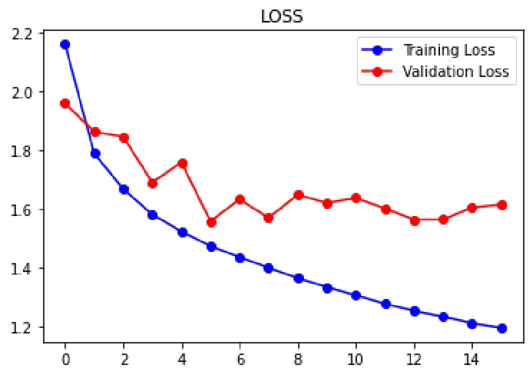
Figure 5.17 Experiment 1 Model summary

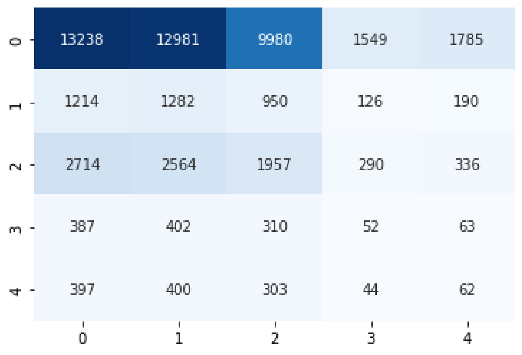
Figure 5.18 Experiment 1 Results describes respectively the accuracy, the loss and a summary of the results obtained from this model. In particular the graphs highlights that the model succeed to learn from training set and it does not perform well on validation set. For this reason the validation loss is very high.





```
Accuracy on Test Data: 0.31
              precision    recall  f1-score   support

           0       0.74      0.33      0.46     39533
           1       0.07      0.34      0.12      3762
           2       0.14      0.25      0.18      7861
           3       0.03      0.04      0.03      1214
           4       0.03      0.05      0.03      1206

    accuracy                           0.31     53576
   macro avg       0.20      0.20      0.17     53576
weighted avg       0.57      0.31      0.38     53576
```



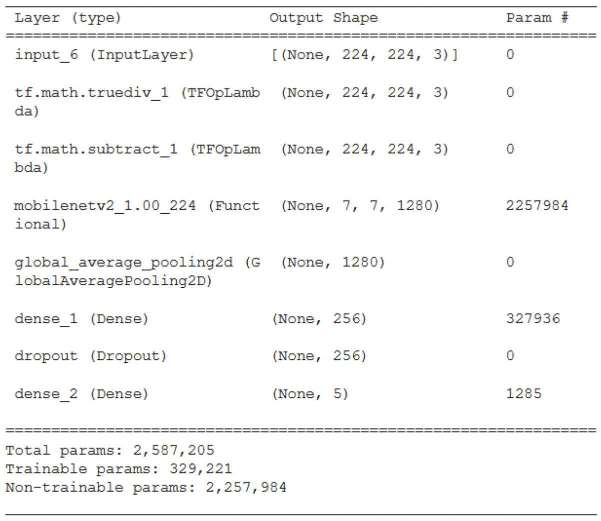


	0	1	2	3	4
0	13238	12981	9980	1549	1785
1	1214	1282	950	126	190
2	2714	2564	1957	290	336
3	387	402	310	52	63
4	397	400	303	44	62

Figure 5.18 Experiment 1 Results

Experiment 2

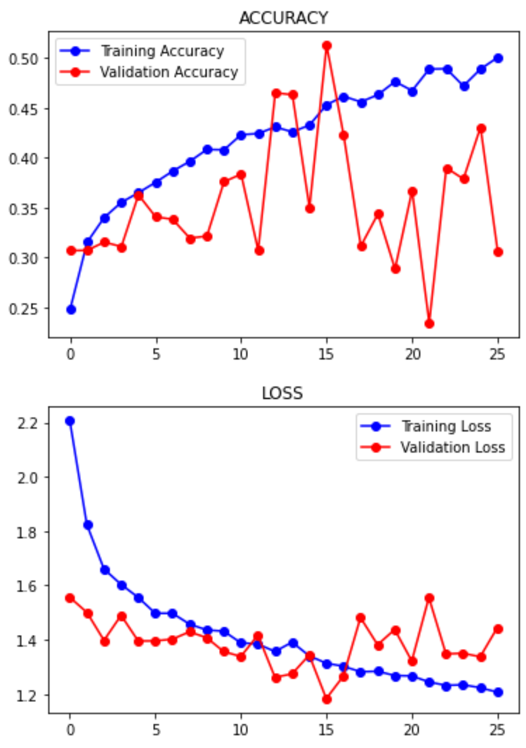
To fight the overfitting present in the first experiment is introduced a regularization with a Dropout layer with a rate of 0.5. A fully connected layer of 256 neurons is introduced to involves weight biases and neurons calibration. This is used to classify images among different category by training. Furthermore a GlobalAveragePooling2D layer substitute the Flatten layer.

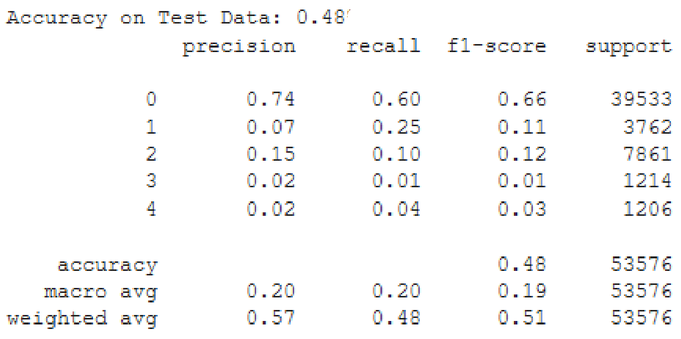


```
 Layer (type)                Output Shape              Param #
=================================================================
 input_6 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.math.truediv_1 (TFOpLamb  (None, 224, 224, 3)      0
 da)

 tf.math.subtract_1 (TFOpLam  (None, 224, 224, 3)      0
 bda)

 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dense_1 (Dense)             (None, 256)               327936

 dropout (Dropout)           (None, 256)               0

 dense_2 (Dense)             (None, 5)                 1285

=================================================================
Total params: 2,587,205
Trainable params: 329,221
Non-trainable params: 2,257,984
_________________________________________________________________
```

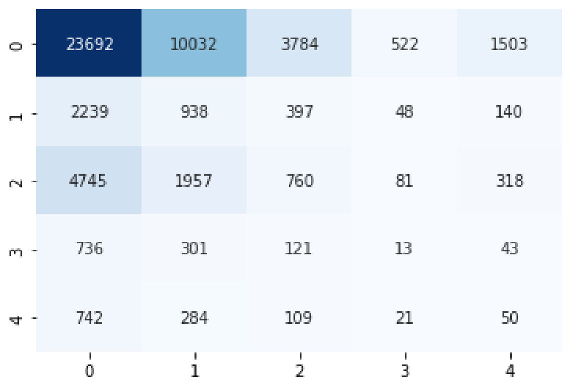
Figure 5.19 Experiment 2 Model summary

The regularizing technique reduced a little bit the validation loss as showed in Figure 5.20 Experiment 2 Results.





```
Accuracy on Test Data: 0.48
              precision    recall  f1-score   support

           0       0.74      0.60      0.66     39533
           1       0.07      0.25      0.11      3762
           2       0.15      0.10      0.12      7861
           3       0.02      0.01      0.01      1214
           4       0.02      0.04      0.03      1206

    accuracy                           0.48     53576
   macro avg       0.20      0.20      0.19     53576
weighted avg       0.57      0.48      0.51     53576
```

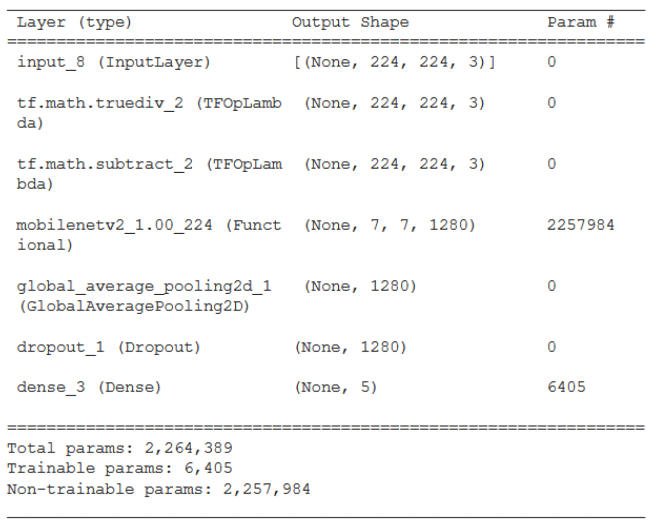


	0	1	2	3	4
0	23692	10032	3784	522	1503
1	2239	938	397	48	140
2	4745	1957	760	81	318
3	736	301	121	13	43
4	742	284	109	21	50

Figure 5.20 Experiment 2 Results

Experiment 3 (Adam)

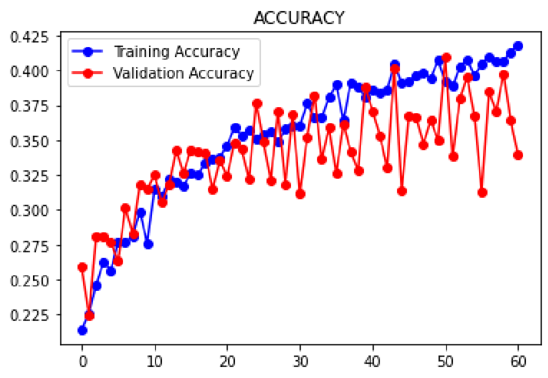
In this experiment the fully connected layer is removed to understand the behaviour of CNN.

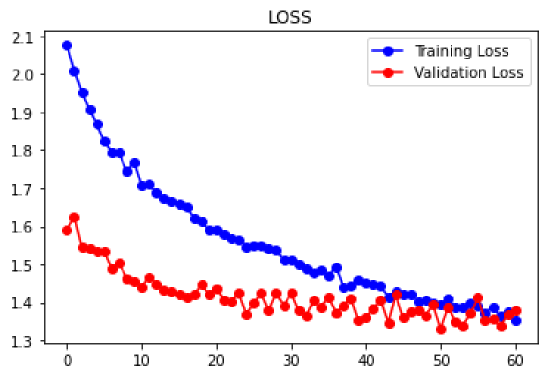


```
Layer (type)                 Output Shape              Param #
=================================================================
input_8 (InputLayer)         [(None, 224, 224, 3)]     0

tf.math.truediv_2 (TFOpLamb  (None, 224, 224, 3)       0
da)

tf.math.subtract_2 (TFOpLam  (None, 224, 224, 3)       0
bda)

mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)        2257984
ional)

global_average_pooling2d_1   (None, 1280)              0
(GlobalAveragePooling2D)

dropout_1 (Dropout)          (None, 1280)              0

dense_3 (Dense)              (None, 5)                 6405

=================================================================
Total params: 2,264,389
Trainable params: 6,405
Non-trainable params: 2,257,984
```

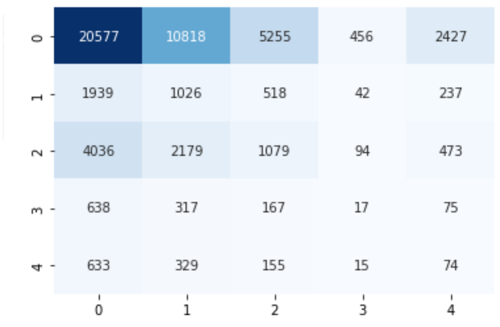
Figure 5.21 Experiment 3 (Adam) Model summary

In this experiment the Figure 5.22 Experiment 3 (Adam) Results, shows how the model needs many epochs to converge. The different optimizer changes the way in which the model comes to convergence by comparing the graphs of this model and the next one





```
Accuracy on Test Data: 0.43
              precision    recall  f1-score   support

           0       0.74      0.52      0.61     39533
           1       0.07      0.27      0.11      3762
           2       0.15      0.14      0.14      7861
           3       0.03      0.01      0.02      1214
           4       0.02      0.06      0.03      1206

    accuracy                           0.43     53576
   macro avg       0.20      0.20      0.18     53576
weighted avg       0.57      0.43      0.48     53576
```



	0	1	2	3	4
0	20577	10818	5255	456	2427
1	1939	1026	518	42	237
2	4036	2179	1079	94	473
3	638	317	167	17	75
4	633	329	155	15	74

Figure 5.22 Experiment 3 (Adam) Results

Experiment 3 (SGD)

This experiment differ from the previous one on the choiche of the optimizer, here the selected one is SGD instead Adam used in the previous experiment. This optimizatorimplements the stocastic gradient descent with a learning rate and momentum. The momentum is a method which helps to accelerate gradients vectors in the right direction thus leading to faster convergence.

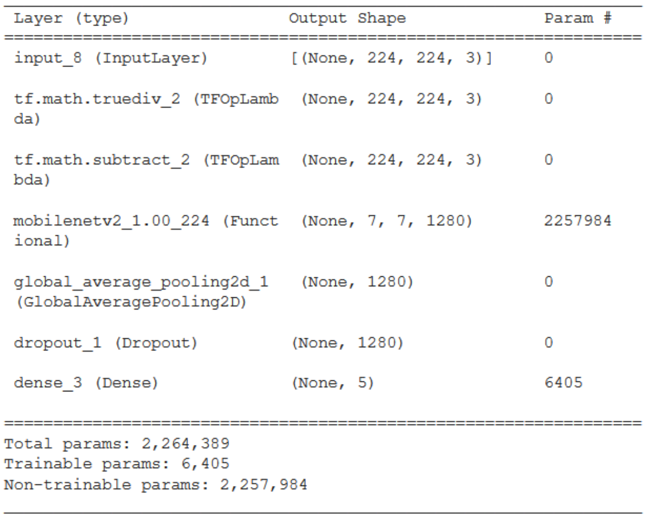
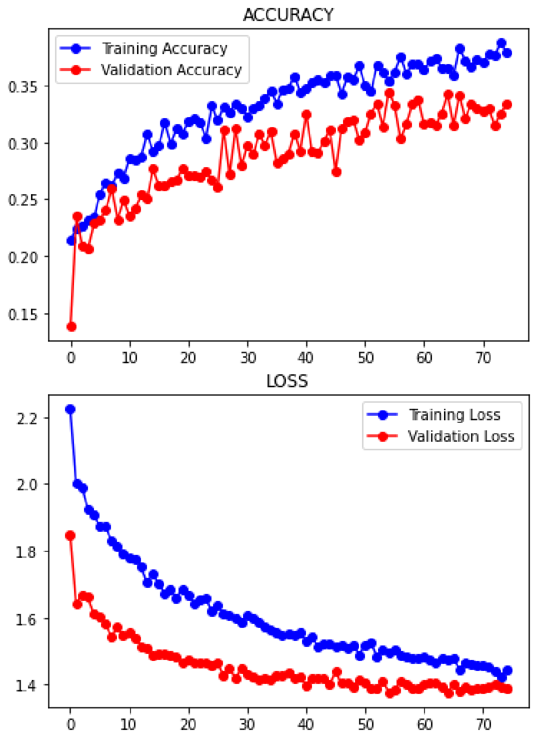
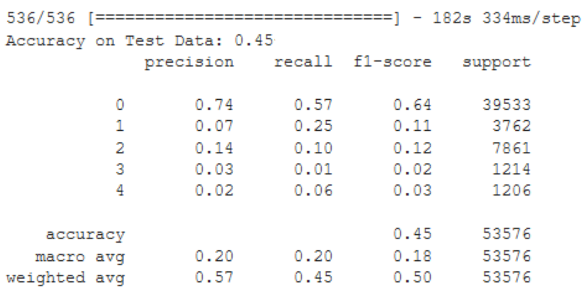
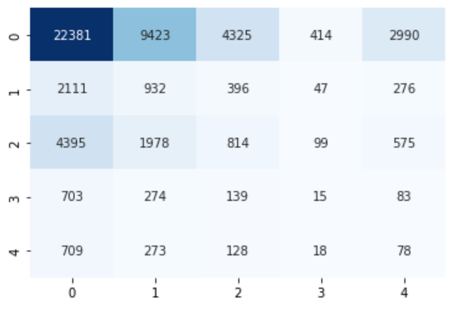

```
 Layer (type)                Output Shape              Param #
=================================================================
 input_8 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.math.truediv_2 (TFOpLamb  (None, 224, 224, 3)      0
 da)

 tf.math.subtract_2 (TFOpLam  (None, 224, 224, 3)      0
 bda)

 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 global_average_pooling2d_1   (None, 1280)             0
 (GlobalAveragePooling2D)

 dropout_1 (Dropout)         (None, 1280)              0

 dense_3 (Dense)             (None, 5)                 6405

=================================================================
Total params: 2,264,389
Trainable params: 6,405
Non-trainable params: 2,257,984
```

Figure 5.23 Experiment 3 (SGD) Model summary

Also this model is similar to the previous one with the difference in the choice of the optimizer. This model grows with a different trend. It is important to specify how there is a clear difference in the values of validation accuracy and the test accuracy.




```
536/536 [==============================] - 182s 334ms/step
Accuracy on Test Data: 0.45
              precision    recall  f1-score   support

           0       0.74      0.57      0.64     39533
           1       0.07      0.25      0.11      3762
           2       0.14      0.10      0.12      7861
           3       0.03      0.01      0.02      1214
           4       0.02      0.06      0.03      1206

    accuracy                           0.45     53576
   macro avg       0.20      0.20      0.18     53576
weighted avg       0.57      0.45      0.50     53576
```

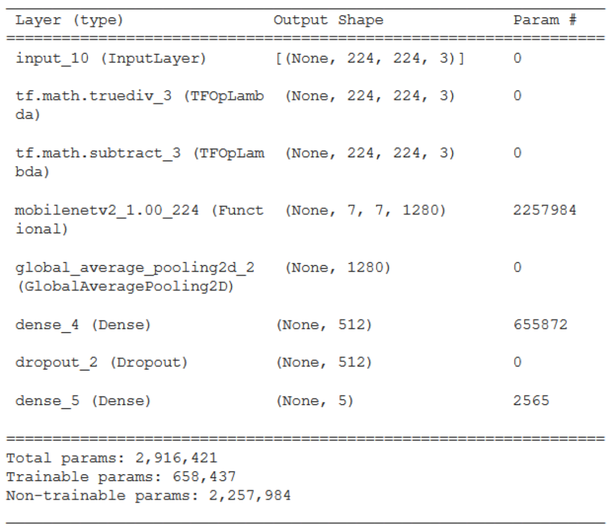


	0	1	2	3	4
0	22381	9423	4325	414	2990
1	2111	932	396	47	276
2	4395	1978	814	99	575
3	703	274	139	15	83
4	709	273	128	18	78

Figure 5.24 Experiment 3 (SGD) Results

Experiment 3.1

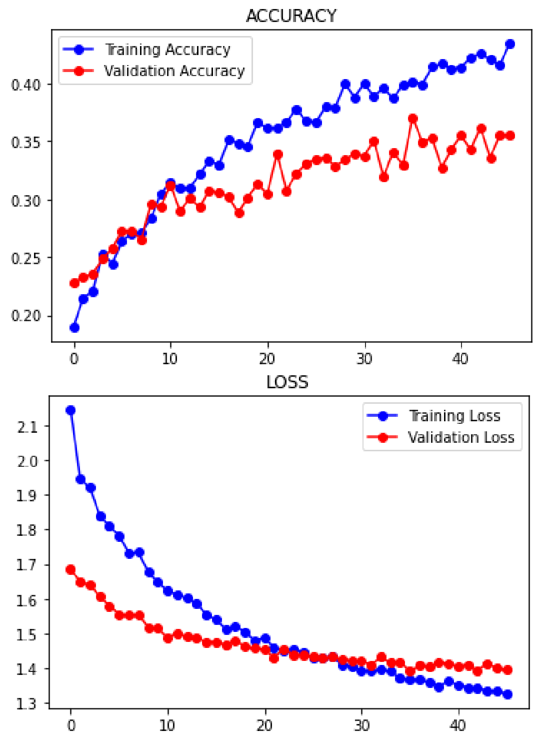
In this experiment the number of neurons of the fully connected layers is increased to 512 with ReLU as activation function. The ReLU is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. A model that uses it is easier to train and often achieves better performance.

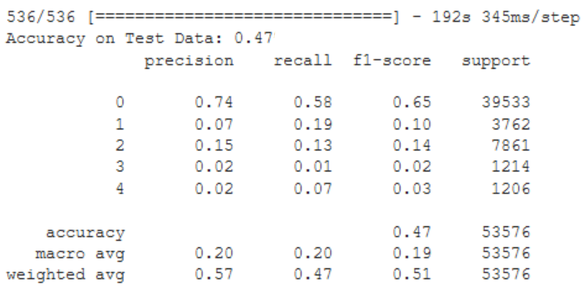


```
 Layer (type)                Output Shape              Param #
=================================================================
 input_10 (InputLayer)       [(None, 224, 224, 3)]     0

 tf.math.truediv_3 (TFOpLamb  (None, 224, 224, 3)      0
 da)

 tf.math.subtract_3 (TFOpLam  (None, 224, 224, 3)      0
 bda)

 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 global_average_pooling2d_2   (None, 1280)             0
 (GlobalAveragePooling2D)

 dense_4 (Dense)             (None, 512)               655872

 dropout_2 (Dropout)         (None, 512)               0

 dense_5 (Dense)             (None, 5)                 2565

=================================================================
Total params: 2,916,421
Trainable params: 658,437
Non-trainable params: 2,257,984
_________________________________________________________________
```

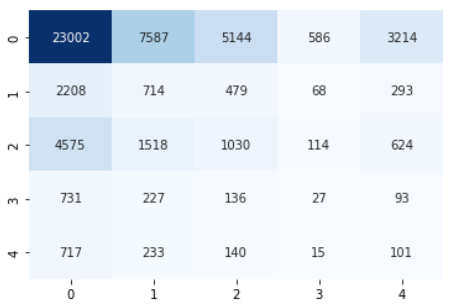
Figure 5.25 Experiment 3.1 Model summary

In the results obtained the validation follow the training both in accuracy and loss but after 40 epochs the early stop occurs. Increasing the patience better results could be obtained but observing the loss curve can be noticed as it begins to manifest overfitting.





```
536/536 [==============================] - 192s 345ms/step
Accuracy on Test Data: 0.47
              precision    recall  f1-score   support

           0       0.74      0.58      0.65     39533
           1       0.07      0.19      0.10      3762
           2       0.15      0.13      0.14      7861
           3       0.02      0.01      0.02      1214
           4       0.02      0.07      0.03      1206

    accuracy                           0.47     53576
   macro avg       0.20      0.20      0.19     53576
weighted avg       0.57      0.47      0.51     53576
```

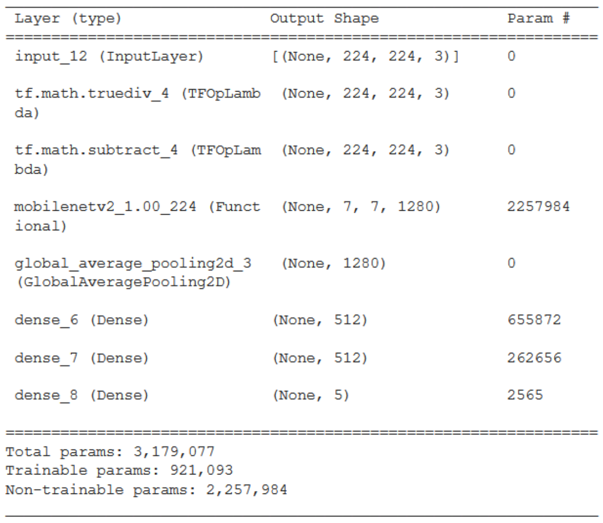


	0	1	2	3	4
0	23002	7587	5144	586	3214
1	2208	714	479	68	293
2	4575	1518	1030	114	624
3	731	227	136	27	93
4	717	233	140	15	101

Figure 5.26 Experiment 3.1 Results

Experiment 4 (Adam)

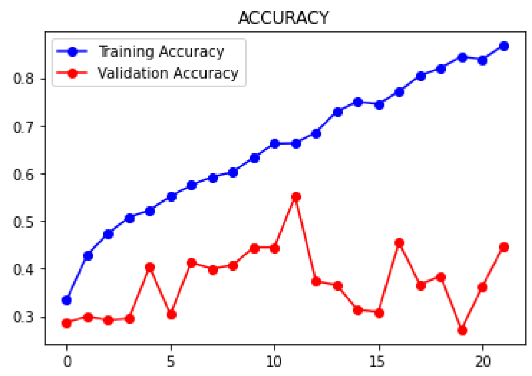
This experiment perform an high increase in terms of neurons of the two fully connected layers without regularization.

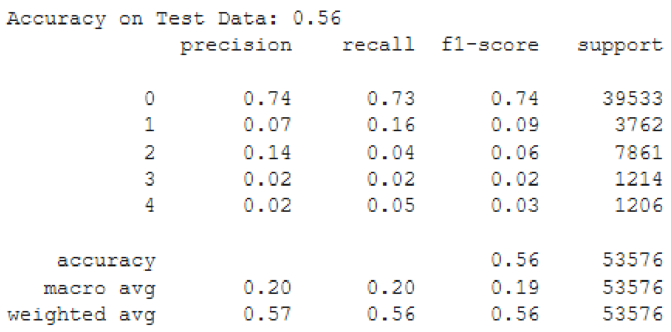


```
 Layer (type)                Output Shape              Param #
=================================================================
 input_12 (InputLayer)       [(None, 224, 224, 3)]     0

 tf.math.truediv_4 (TFOpLamb  (None, 224, 224, 3)      0
 da)

 tf.math.subtract_4 (TFOpLam  (None, 224, 224, 3)      0
 bda)

 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 global_average_pooling2d_3  (None, 1280)              0
  (GlobalAveragePooling2D)

 dense_6 (Dense)             (None, 512)               655872

 dense_7 (Dense)             (None, 512)               262656

 dense_8 (Dense)             (None, 5)                 2565

=================================================================
Total params: 3,179,077
Trainable params: 921,093
Non-trainable params: 2,257,984
```

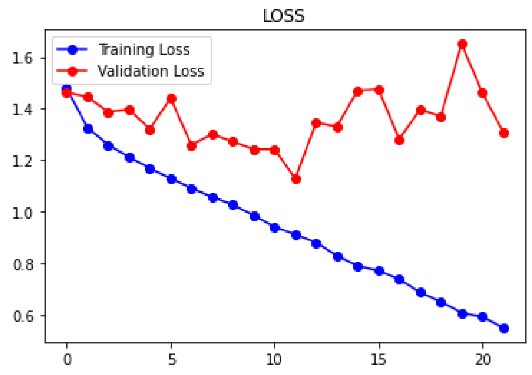
Figure 5.27 Experiment 4 (Adam) Model summary

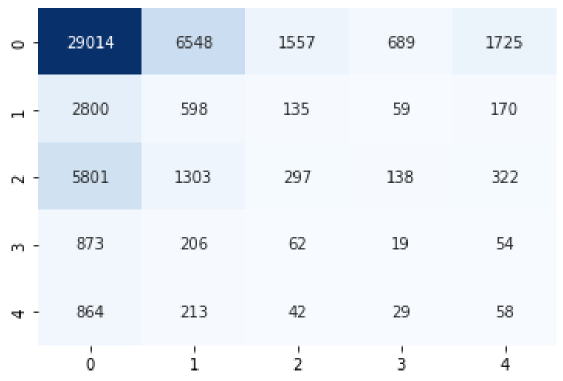
The graphs show the absence of a regularization technique and the trend of curves is not regular.





```
Accuracy on Test Data: 0.56
              precision    recall  f1-score   support

           0       0.74      0.73      0.74     39533
           1       0.07      0.16      0.09      3762
           2       0.14      0.04      0.06      7861
           3       0.02      0.02      0.02      1214
           4       0.02      0.05      0.03      1206

    accuracy                           0.56     53576
   macro avg       0.20      0.20      0.19     53576
weighted avg       0.57      0.56      0.56     53576
```



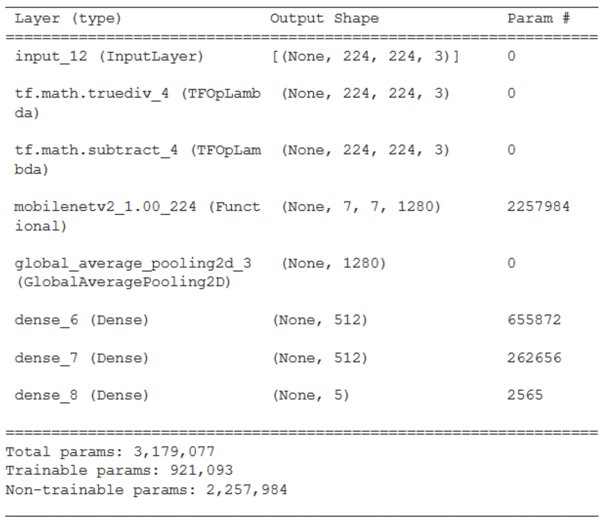


	0	1	2	3	4
0	29014	6548	1557	689	1725
1	2800	598	135	59	170
2	5801	1303	297	138	322
3	873	206	62	19	54
4	864	213	42	29	58

Figure 5.28 Experiment 4 (Adam) Results

Experiment 4 (SGD)

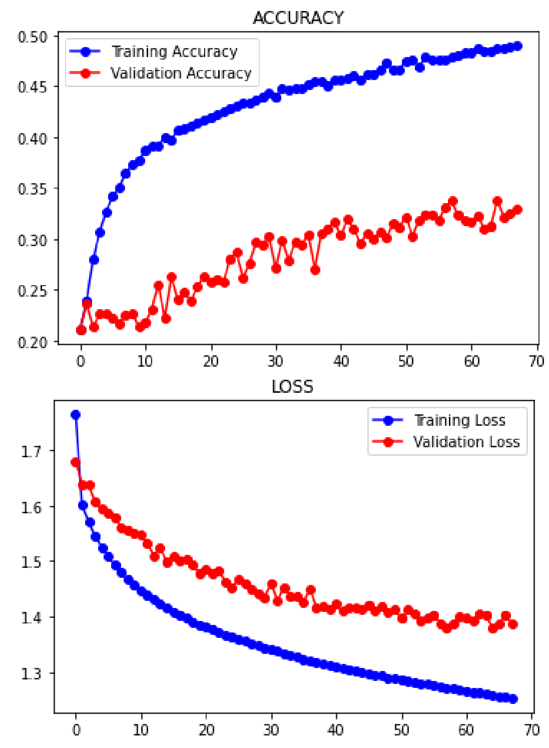
Also in this experiment the difference between the previous one is only in the choice of the optimizer. SGD is selected to converge faster to the optimal value.

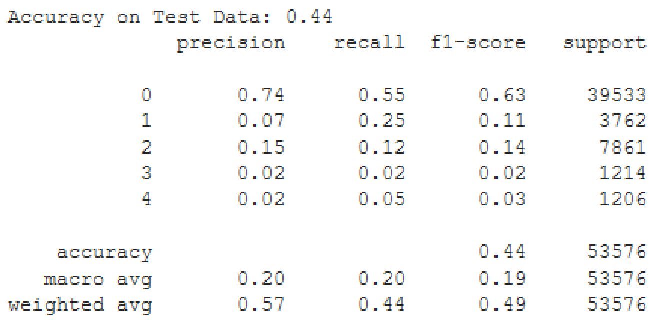


```
 Layer (type)                Output Shape              Param #
=================================================================
 input_12 (InputLayer)       [(None, 224, 224, 3)]     0

 tf.math.truediv_4 (TFOpLamb  (None, 224, 224, 3)      0
 da)

 tf.math.subtract_4 (TFOpLam  (None, 224, 224, 3)      0
 bda)

 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 global_average_pooling2d_3   (None, 1280)             0
 (GlobalAveragePooling2D)

 dense_6 (Dense)             (None, 512)               655872

 dense_7 (Dense)             (None, 512)               262656

 dense_8 (Dense)             (None, 5)                 2565

=================================================================
Total params: 3,179,077
Trainable params: 921,093
Non-trainable params: 2,257,984
_________________________________________________________________
```

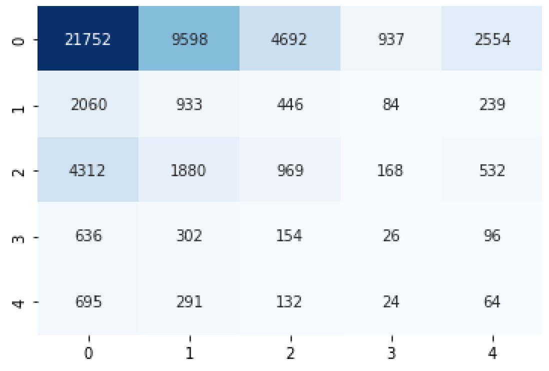
Figure 5.29 Experiment 4 (SGD) Model summary

After a very long computation lasting more than 60 epochs (~2 hours) the saved model continued to grow slowly, in relation to the presence of overfitting was considered the idea of not going on with the model.





```
Accuracy on Test Data: 0.44
              precision    recall  f1-score   support

           0       0.74      0.55      0.63     39533
           1       0.07      0.25      0.11      3762
           2       0.15      0.12      0.14      7861
           3       0.02      0.02      0.02      1214
           4       0.02      0.05      0.03      1206

    accuracy                           0.44     53576
   macro avg       0.20      0.20      0.19     53576
weighted avg       0.57      0.44      0.49     53576
```

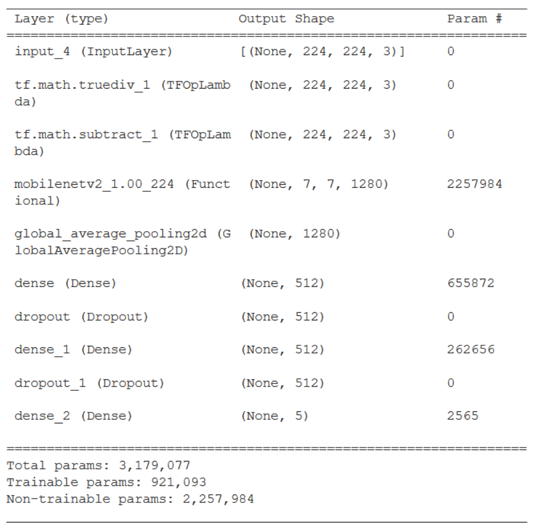


	0	1	2	3	4
0	21752	9598	4692	937	2554
1	2060	933	446	84	239
2	4312	1880	969	168	532
3	636	302	154	26	96
4	695	291	132	24	64

Figure 5.30 Experiment 4 (SGD) Results

Experiment 4.1

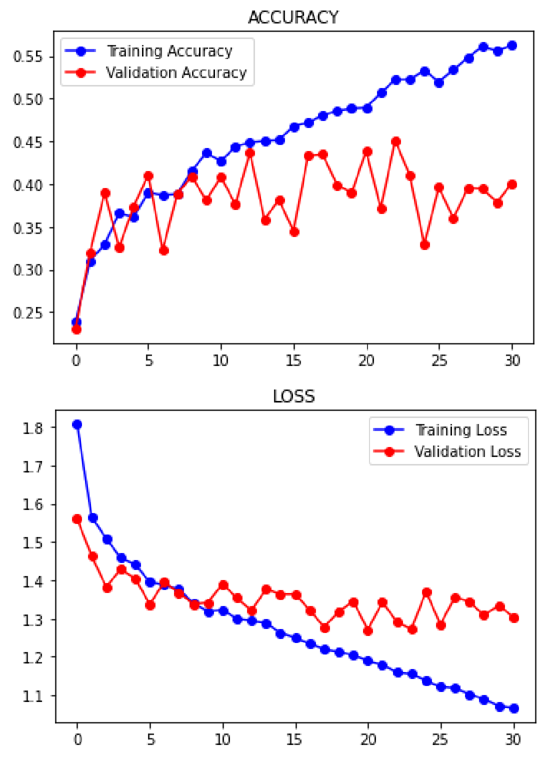
In this experiment a regularization technique was performed. Two dropout layers, with a rate fop 0.5, are added after both the dense layer.

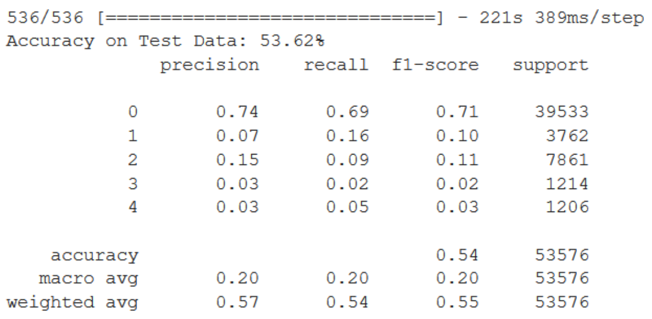


```
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.math.truediv_1 (TFOpLamb  (None, 224, 224, 3)      0
 da)

 tf.math.subtract_1 (TFOpLam  (None, 224, 224, 3)      0
 bda)

 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dense (Dense)               (None, 512)               655872

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 512)               262656

 dropout_1 (Dropout)         (None, 512)               0

 dense_2 (Dense)             (None, 5)                 2565

=================================================================
Total params: 3,179,077
Trainable params: 921,093
Non-trainable params: 2,257,984
_________________________________________________________________
```

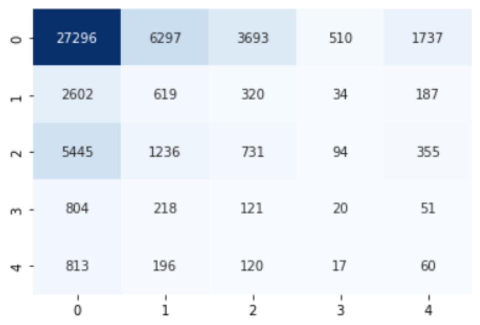
Figure 5.31 Experiment 4.1 Model summary

This experiment highlights how the network specializes heavily on training dataset and does not generalize bringing validation to early stopping.





```
536/536 [==============================] - 221s 389ms/step
Accuracy on Test Data: 53.62%
              precision    recall  f1-score   support

           0       0.74      0.69      0.71     39533
           1       0.07      0.16      0.10      3762
           2       0.15      0.09      0.11      7861
           3       0.03      0.02      0.02      1214
           4       0.03      0.05      0.03      1206

    accuracy                           0.54     53576
   macro avg       0.20      0.20      0.20     53576
weighted avg       0.57      0.54      0.55     53576
```



	0	1	2	3	4
0	27296	6297	3693	510	1737
1	2602	619	320	34	187
2	5445	1236	731	94	355
3	804	218	121	20	51
4	813	196	120	17	60

Figure 5.32 Experiment 4.1 Results

5.2.3 Fine Tuning

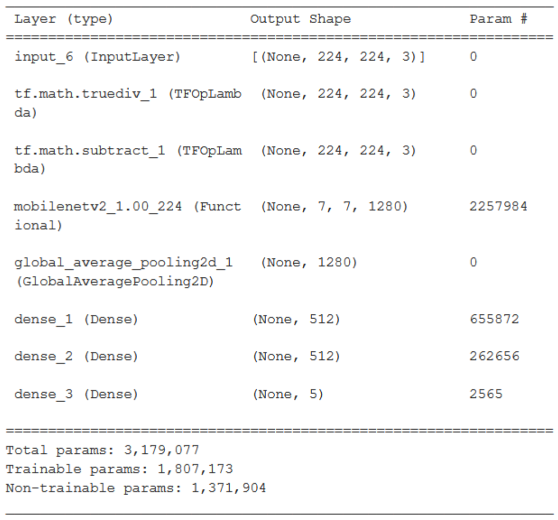
Finetuning is used to taking weights of a trained neural network and use it as initialization for a new model being trained on data from the same domain. It is used also to speed up the training and overcome small dataset size. There are various strategies, such as training the whole initialized network or "freezing" some of the pre-trained weights (usually whole initial layers). The following experiments are based on the unfreezing approach.

5.2.4 Fine Tuning Experiment

All the following experiment are performed on model 3.1 and model 4 due to their better performances.

Experiment 5 on Model 3.1

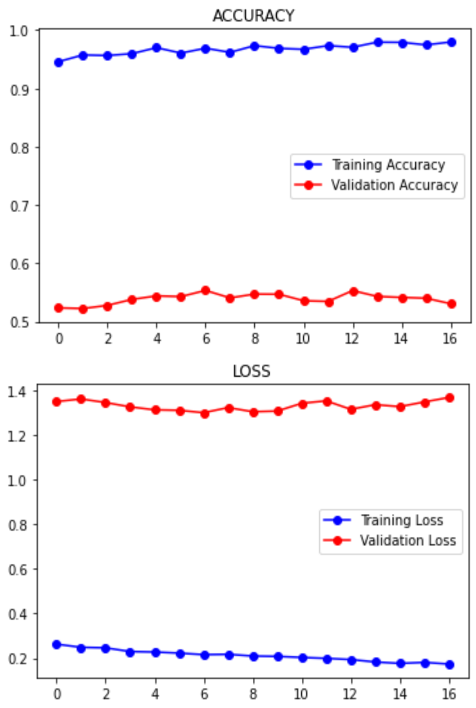
In this experiment the block 16 is unfrozen to increase the trainable variables.

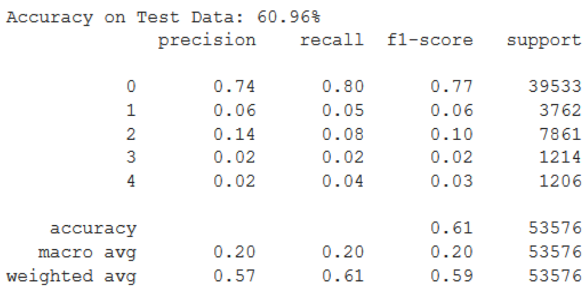


```
 Layer (type)                Output Shape              Param #
=================================================================
 input_6 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.math.truediv_1 (TFOpLamb  (None, 224, 224, 3)      0
 da)

 tf.math.subtract_1 (TFOpLam  (None, 224, 224, 3)      0
 bda)

 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 global_average_pooling2d_1   (None, 1280)             0
 (GlobalAveragePooling2D)

 dense_1 (Dense)             (None, 512)               655872

 dense_2 (Dense)             (None, 512)               262656

 dense_3 (Dense)             (None, 5)                 2565

=================================================================
Total params: 3,179,077
Trainable params: 1,807,173
Non-trainable params: 1,371,904
```

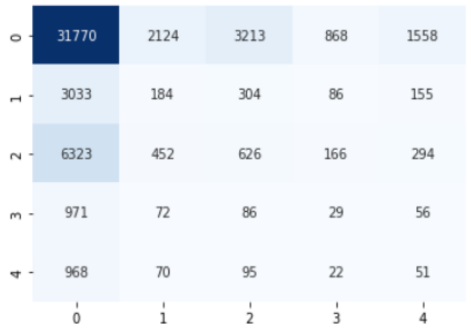
Figure 5.33 Experiment 5 (on Model 3.1) Model summary

Unfreezing the last block the CNN does not learn well from the training set and the early stopping phase occur fast. Another possible experiment will consider increasing the patience but, due to the long duration of the epochs and the limitation of Google Colab, this was not tried.





```
Accuracy on Test Data: 60.96%
              precision    recall  f1-score   support

           0       0.74      0.80      0.77     39533
           1       0.06      0.05      0.06      3762
           2       0.14      0.08      0.10      7861
           3       0.02      0.02      0.02      1214
           4       0.02      0.04      0.03      1206

    accuracy                           0.61     53576
   macro avg       0.20      0.20      0.20     53576
weighted avg       0.57      0.61      0.59     53576
```

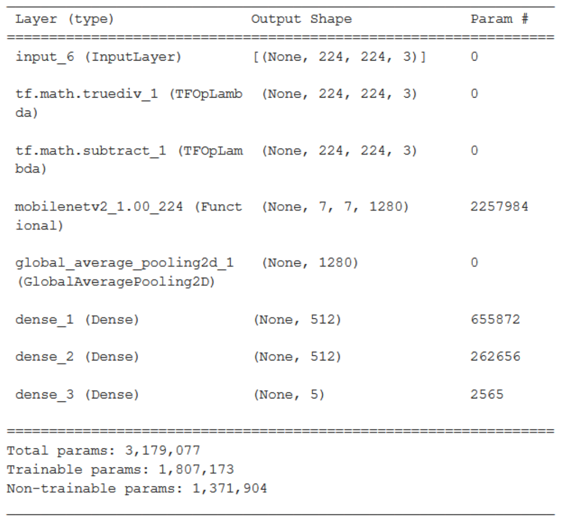


	0	1	2	3	4
0	31770	2124	3213	868	1558
1	3033	184	304	86	155
2	6323	452	626	166	294
3	971	72	86	29	56
4	968	70	95	22	51

Figure 5.34 Experiment 5 (on Model 3.1) Results

Experiment 5 on Model 4

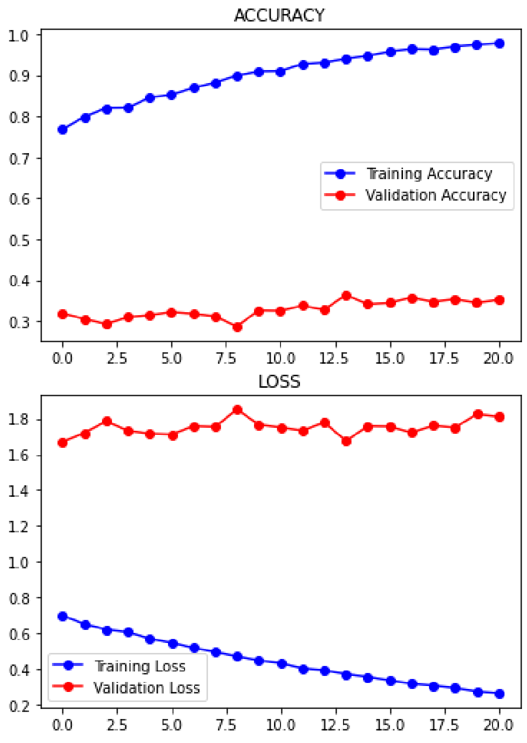
In this experiment the block 16 is unfrozen to increase the trainable variables.

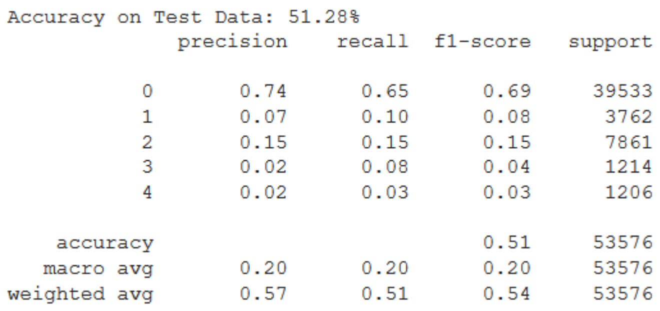


```
Layer (type)                Output Shape              Param #
=================================================================
input_6 (InputLayer)        [(None, 224, 224, 3)]     0

tf.math.truediv_1 (TFOpLamb  (None, 224, 224, 3)      0
da)

tf.math.subtract_1 (TFOpLam  (None, 224, 224, 3)      0
bda)

mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
ional)

global_average_pooling2d_1   (None, 1280)             0
(GlobalAveragePooling2D)

dense_1 (Dense)             (None, 512)               655872

dense_2 (Dense)             (None, 512)               262656

dense_3 (Dense)             (None, 5)                 2565

=================================================================
Total params: 3,179,077
Trainable params: 1,807,173
Non-trainable params: 1,371,904
_________________________________________________________________
```

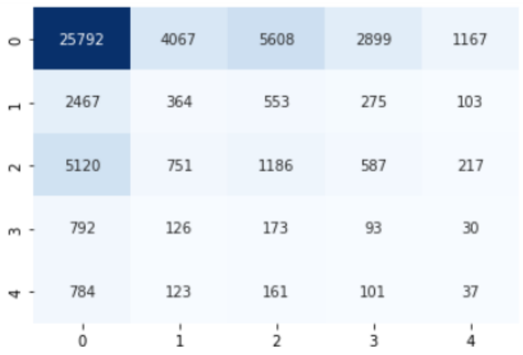
Figure 5.35 Experiment 5 (on Model 4) Model summary

A better behaviour was encountered on fine tuning performed on model 4 as shown in Figure 5.36 Experiment 5 (on Model 4) Results, in particular the network still learn from the training after some epochs, the same thing does not happen to the validation.





```
Accuracy on Test Data: 51.28%
              precision    recall  f1-score   support

           0       0.74      0.65      0.69     39533
           1       0.07      0.10      0.08      3762
           2       0.15      0.15      0.15      7861
           3       0.02      0.08      0.04      1214
           4       0.02      0.03      0.03      1206

    accuracy                           0.51     53576
   macro avg       0.20      0.20      0.20     53576
weighted avg       0.57      0.51      0.54     53576
```

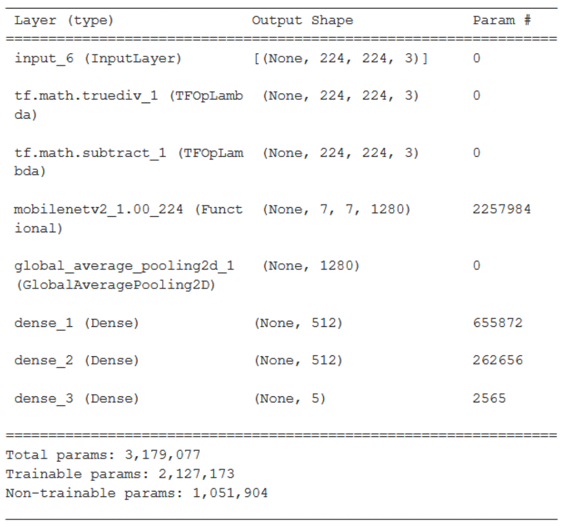


	0	1	2	3	4
0	25792	4067	5608	2899	1167
1	2467	364	553	275	103
2	5120	751	1186	587	217
3	792	126	173	93	30
4	784	123	161	101	37

Figure 5.36 Experiment 5 (on Model 4) Results

Experiment 6 on Model 4

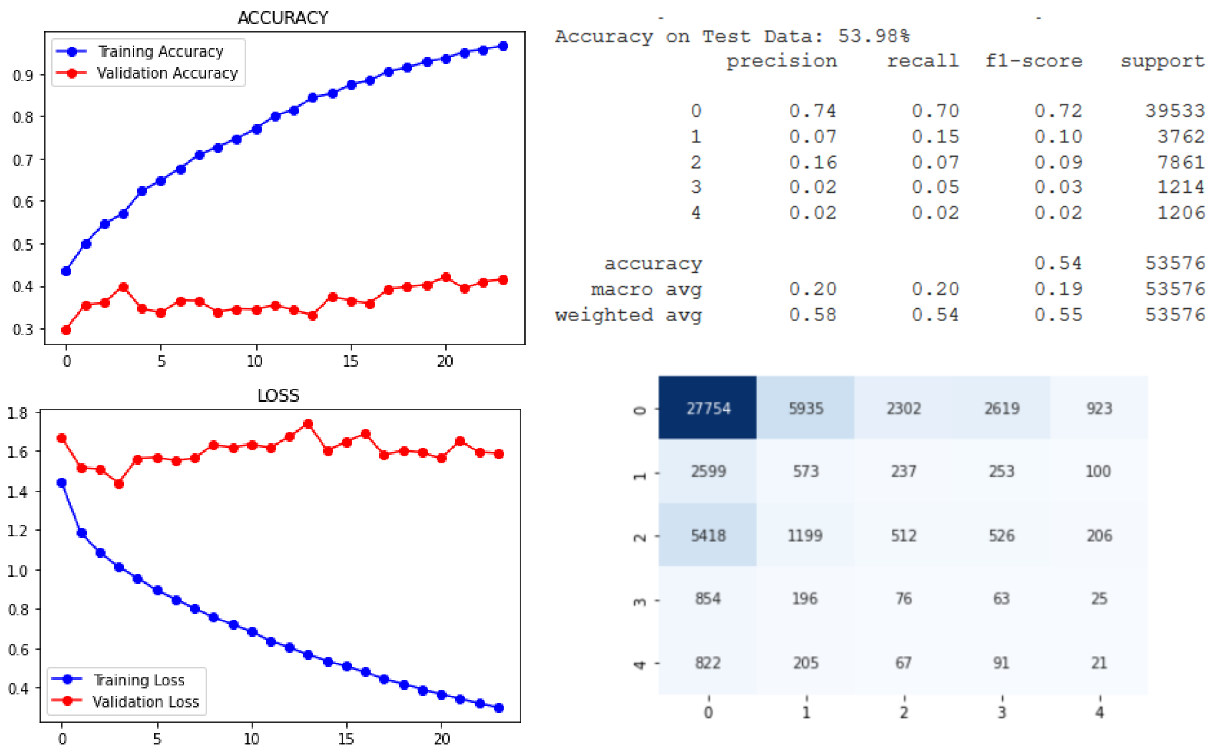
In this experiment all the layers from the block 15 are unfrozen to increase the trainable variables. Obviously the trainable parameters are more than the previous experiment.



```
Layer (type)                Output Shape              Param #
=================================================================
input_6 (InputLayer)        [(None, 224, 224, 3)]     0

tf.math.truediv_1 (TFOpLamb  (None, 224, 224, 3)      0
da)

tf.math.subtract_1 (TFOpLam  (None, 224, 224, 3)      0
bda)

mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
ional)

global_average_pooling2d_1   (None, 1280)             0
(GlobalAveragePooling2D)

dense_1 (Dense)             (None, 512)               655872

dense_2 (Dense)             (None, 512)               262656

dense_3 (Dense)             (None, 5)                 2565

=================================================================
Total params: 3,179,077
Trainable params: 2,127,173
Non-trainable params: 1,051,904
_________________________________________________________________
```

Figure 5.37 Experiment 6 (on Model 4) Model summary

In this approach after a low number of epoch the training still increase its accuracy instead of the validation that remain blocked on low value of accuracy. The Figure 5.38 Experiment 6 (on Model 4) Results, highlight how the network overfit training dataset.



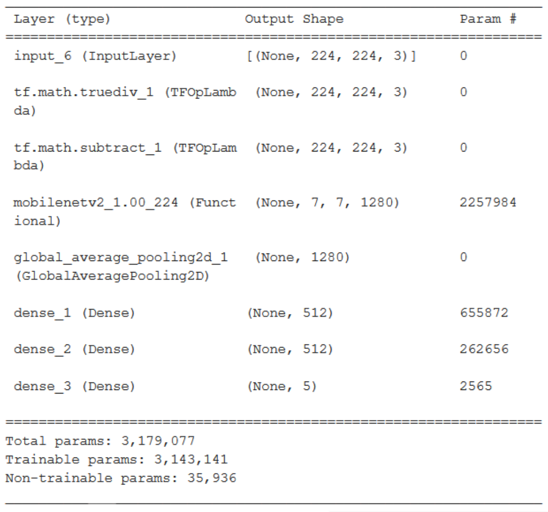
```
Accuracy on Test Data: 53.98%
                 precision    recall  f1-score   support

           0       0.74      0.70      0.72     39533
           1       0.07      0.15      0.10      3762
           2       0.16      0.07      0.09      7861
           3       0.02      0.05      0.03      1214
           4       0.02      0.02      0.02      1206

    accuracy                           0.54     53576
   macro avg       0.20      0.20      0.19     53576
weighted avg       0.58      0.54      0.55     53576
```

	0	1	2	3	4
0	27754	5935	2302	2619	923
1	2599	573	237	253	100
2	5418	1199	512	526	206
3	854	196	76	63	25
4	822	205	67	91	21

Figure 5.38 Experiment 6 (on Model 4) Results

Experiment 7 on Model 3.1

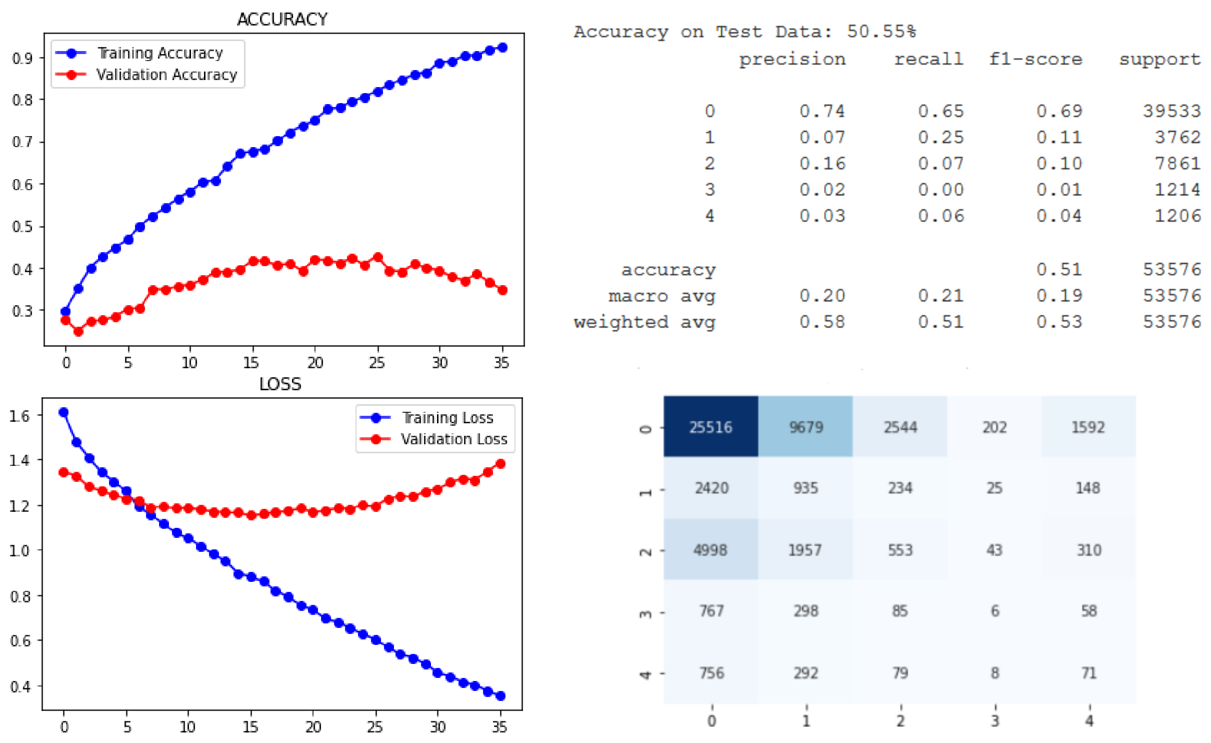
In this experiment all the layers from the block 1 are unfrozen. This severe approach was tried in order to improve the validation results but the desired behaviour was not encountered, only the training accuracy growth.



```
Layer (type)                Output Shape              Param #
=================================================================
input_6 (InputLayer)        [(None, 224, 224, 3)]     0

tf.math.truediv_1 (TFOpLamb  (None, 224, 224, 3)      0
da)

tf.math.subtract_1 (TFOpLam  (None, 224, 224, 3)      0
bda)

mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
ional)

global_average_pooling2d_1   (None, 1280)             0
 (GlobalAveragePooling2D)

dense_1 (Dense)             (None, 512)               655872

dense_2 (Dense)             (None, 512)               262656

dense_3 (Dense)             (None, 5)                 2565

=================================================================
Total params: 3,179,077
Trainable params: 3,143,141
Non-trainable params: 35,936
_________________________________________________________________
```

Figure 5.39 Experiment 7 (on Model 3.1) Model summary

After the unfreezing of all the layers form block 1 the training accuracy increase really fast from the earliest epochs, sadly the validation curve does not have the same behaviour.



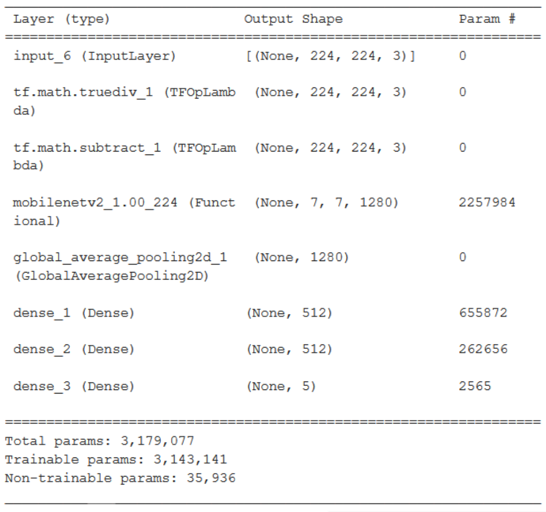
```
Accuracy on Test Data: 50.55%
              precision    recall  f1-score   support

           0       0.74      0.65      0.69     39533
           1       0.07      0.25      0.11      3762
           2       0.16      0.07      0.10      7861
           3       0.02      0.00      0.01      1214
           4       0.03      0.06      0.04      1206

    accuracy                           0.51     53576
   macro avg       0.20      0.21      0.19     53576
weighted avg       0.58      0.51      0.53     53576
```

	0	1	2	3	4
0	25516	9679	2544	202	1592
1	2420	935	234	25	148
2	4998	1957	553	43	310
3	767	298	85	6	58
4	756	292	79	8	71

Figure 5.40 Experiment 7 (on Model 3.1) Results

Experiment 7 on Model 4

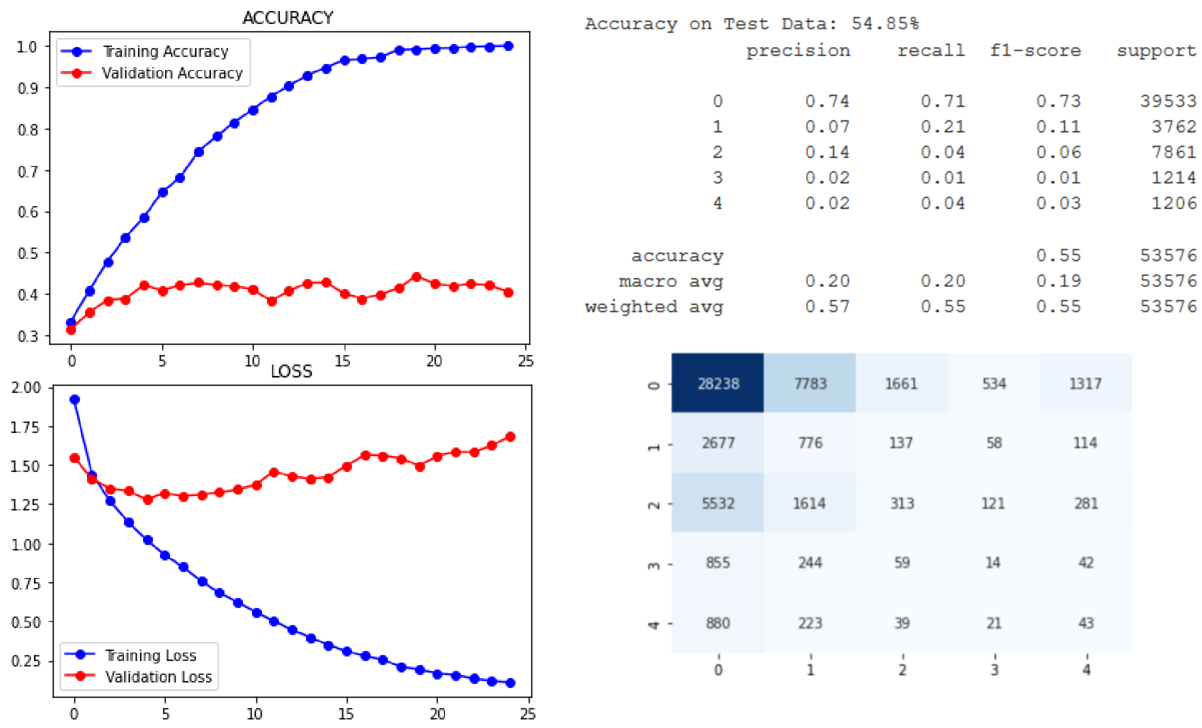
In this experiment all the layers from the block 1 are unfrozen. This severe approach was tried in order to improve the validation results but the desired behaviour was not encountered, only the training accuracy growth.



```
Layer (type)                Output Shape              Param #
=================================================================
input_6 (InputLayer)        [(None, 224, 224, 3)]     0

tf.math.truediv_1 (TFOpLamb  (None, 224, 224, 3)      0
da)

tf.math.subtract_1 (TFOpLam  (None, 224, 224, 3)      0
bda)

mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
ional)

global_average_pooling2d_1   (None, 1280)             0
 (GlobalAveragePooling2D)

dense_1 (Dense)             (None, 512)               655872

dense_2 (Dense)             (None, 512)               262656

dense_3 (Dense)             (None, 5)                 2565

=================================================================
Total params: 3,179,077
Trainable params: 3,143,141
Non-trainable params: 35,936
_________________________________________________________________
```

Figure 5.41 Experiment 7 (on Model 4) Model summary

After the unfreezing of all the layers form block 1 the training accuracy increase really fast from the earliest epochs, sadly the validation curve does not have the same behaviour.



```
Accuracy on Test Data: 54.85%
              precision    recall  f1-score   support

           0       0.74      0.71      0.73     39533
           1       0.07      0.21      0.11      3762
           2       0.14      0.04      0.06      7861
           3       0.02      0.01      0.01      1214
           4       0.02      0.04      0.03      1206

    accuracy                           0.55     53576
   macro avg       0.20      0.20      0.19     53576
weighted avg       0.57      0.55      0.55     53576
```

	0	1	2	3	4
0	28238	7783	1661	534	1317
1	2677	776	137	58	114
2	5532	1614	313	121	281
3	855	244	59	14	42
4	880	223	39	21	43

Figure 5.42 Experiment 7 (on Model 4) Results

5.3 Other Pre-trained models

More experiments was performed utilizing different pre-trained models, in particular 3 other different models are exploited: ResNet50, InceptionV3 and EfficientNet.

The results obtained are quite similar to the other models and, because of the short time available, the full deployment was not performed. For this 3 pre-trained models only 3 or 4 experiment were performed and the code is visible in the drive folder.

6. Ensemble

In this last part, model ensembling techniques are used to improve performances. In particular, three techniques have been considered:

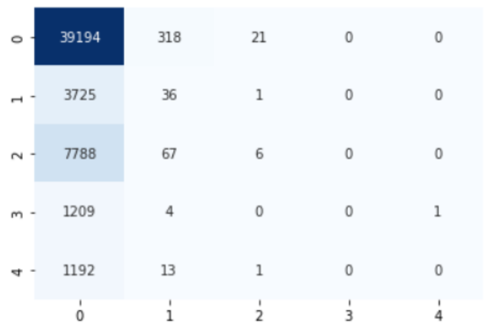
1. **Majority voting**: the assigned class to the sample is the class that receives the majority of votes from the predictions made by the base models.
2. **Averaging**: the output score of the classification is a weighted sum of the probabilities given to the sample by the base models. The weights are assigned according to the validation accuracy of the models. The assigned class is determined by the rounding of the output score.
3. **Model stacking**: a new model is built on the outputs coming from the base models. This new model learns how to correctly classify and interpret the probabilities provided by the base models.

In the majority voting and averaging all the chosen models are saved in a folder in order to load all the predictions. To reduce the computation time, all the prediction on test set of all models are saved in .npy format. In this way with a load_model_scores() function, all scores of models are loaded in a list of dictionaries.

6.1 Majority voting

In this first approach all the votes have the same weight. To fight the situation of tier votes, the predicted final class is the class which was voted by the model with the highest accuracy. This result is achieved with storing the best model in a list during voting phase.

```
Accuracy on Test Data: 73.23%
              precision    recall  f1-score   support

           0       0.74      0.99      0.85     39533
           1       0.08      0.01      0.02      3762
           2       0.21      0.00      0.00      7861
           3       0.00      0.00      0.00      1214
           4       0.00      0.00      0.00      1206

    accuracy                           0.73     53576
   macro avg       0.21      0.20      0.17     53576
weighted avg       0.58      0.73      0.63     53576
```



	0	1	2	3	4
0	39194	318	21	0	0
1	3725	36	1	0	0
2	7788	67	6	0	0
3	1209	4	0	0	1
4	1192	13	1	0	0

Figure 6.1 majority voting metrics and confusion matrix

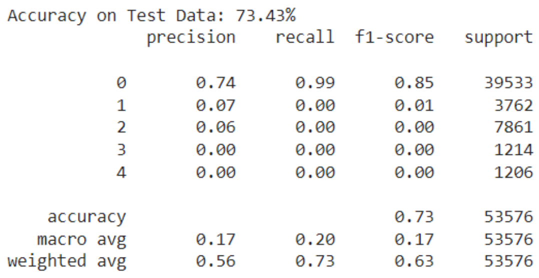
6.2 Averaging

With averaging mode two approach are chosen to evaluate weights.

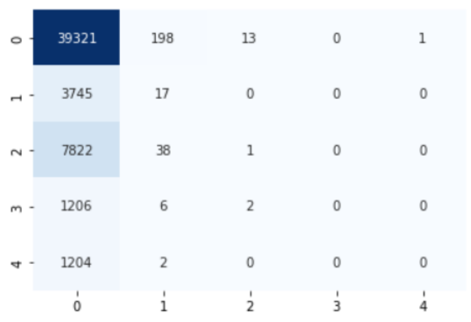
In the first approach the accuracy was choose as parameter to determine weights. In particular, the calculation is:

$$weightModel_1 = \frac{accuracyModel_1}{\sum accuracy}$$

The obtained results are shown in Figure 6.2 Averaging accuracy metrics and confusion matrix



```
Accuracy on Test Data: 73.43%
              precision    recall  f1-score   support

           0       0.74      0.99      0.85     39533
           1       0.07      0.00      0.01      3762
           2       0.06      0.00      0.00      7861
           3       0.00      0.00      0.00      1214
           4       0.00      0.00      0.00      1206

    accuracy                           0.73     53576
   macro avg       0.17      0.20      0.17     53576
weighted avg       0.56      0.73      0.63     53576
```



	0	1	2	3	4
0	39321	198	13	0	1
1	3745	17	0	0	0
2	7822	38	1	0	0
3	1206	6	2	0	0
4	1204	2	0	0	0

Figure 6.2 Averaging accuracy metrics and confusion matrix

In the second approach the weights for averaging were chosen by recall on dataset. Firstly, static weights on classes are assigned with the formula:

$$\frac{\sum training\_samples}{n\_classes} * training\_samples\_class\_i$$

Class 0	Class 1	Class 2	Class 3	Class 4
0.27219	2.87548	1.32743	8.04673	9.92203

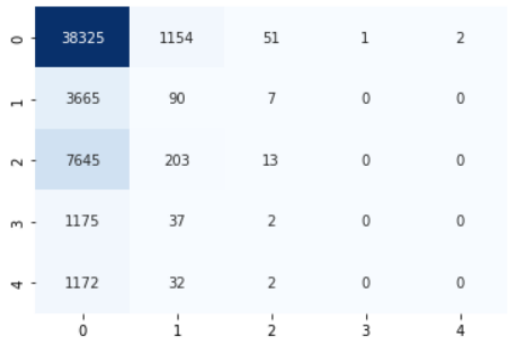
Table 3 Static class weights

After this assignment, the model weights are assigned by:

$$\sum(Class_i\_weight * Class_i\_recall) \qquad for\ all\ class$$

This approach was chosen to have greater coverage in the dataset.

```
Accuracy on Test Data: 71.73%
              precision    recall  f1-score   support

           0       0.74      0.97      0.84     39533
           1       0.06      0.02      0.03      3762
           2       0.17      0.00      0.00      7861
           3       0.00      0.00      0.00      1214
           4       0.00      0.00      0.00      1206

    accuracy                           0.72     53576
   macro avg       0.19      0.20      0.17     53576
weighted avg       0.57      0.72      0.62     53576
```



	0	1	2	3	4
0	38325	1154	51	1	2
1	3665	90	7	0	0
2	7645	203	13	0	0
3	1175	37	2	0	0
4	1172	32	2	0	0

Figure 6.3 averaging recall metrics and confusion matrix

The averaging approach can be extended further by training an entirely new model to learn how to best combine the contributions from each sub model. This method is used in stacked generalization and can result in better predictive performance than any single contributing model.

For the creation of model stacking 3 different models are chosen:

- pretrained_MobileNet_exp4_adam_638c_0001.h5
- pretrained_VGG16_exp5_onmodel3_638c_0001_Adam.h
- model3.hdf5

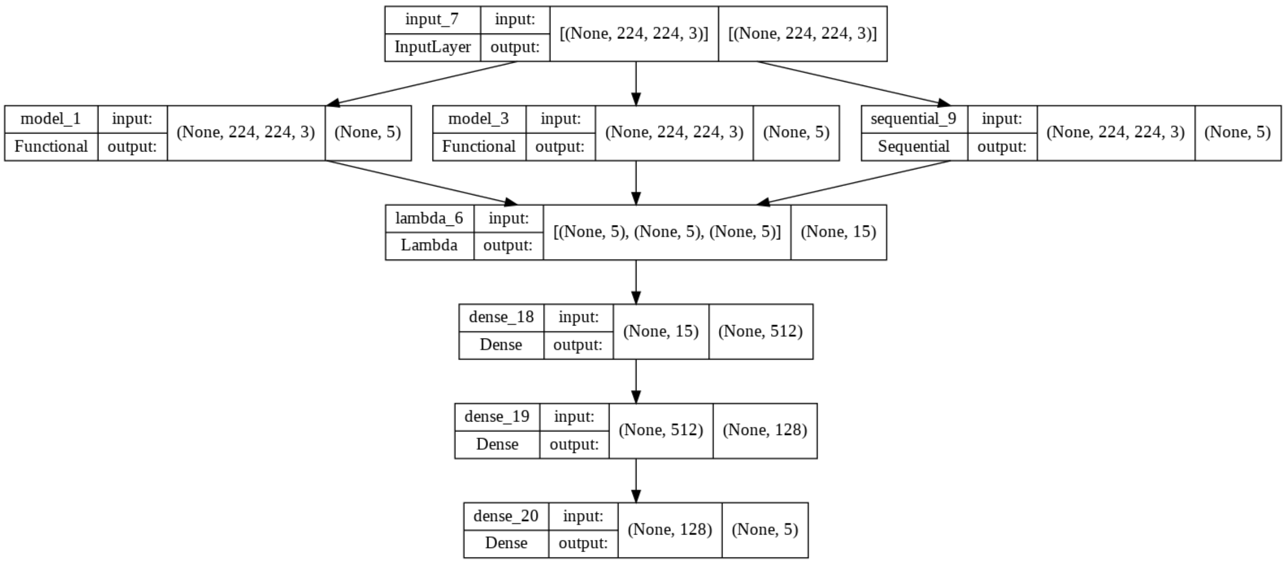
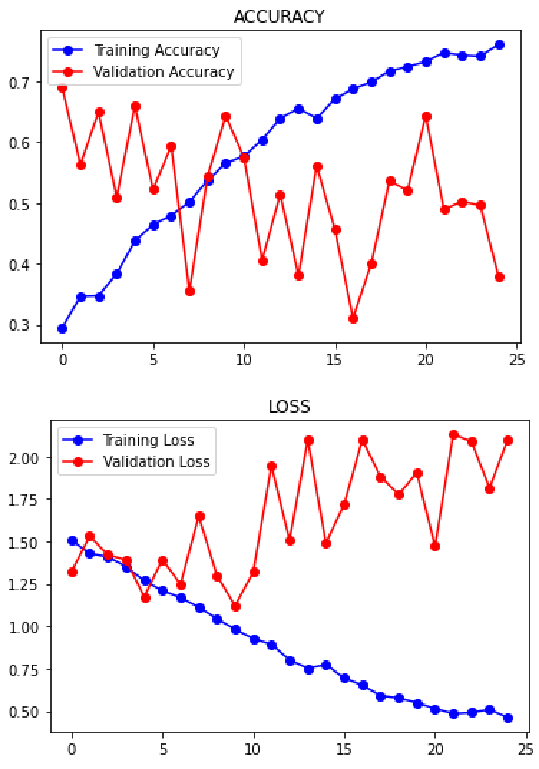
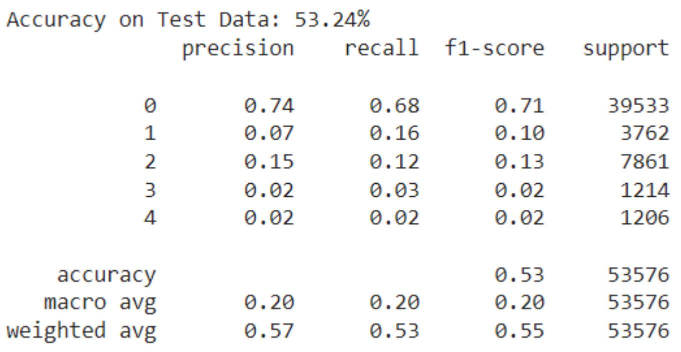

input_7 | InputLayer | input: | output: | [(None, 224, 224, 3)] | [(None, 224, 224, 3)]
model_1 | Functional | input: | output: | (None, 224, 224, 3) | (None, 5)
model_3 | Functional | input: | output: | (None, 224, 224, 3) | (None, 5)
sequential_9 | Sequential | input: | output: | (None, 224, 224, 3) | (None, 5)
lambda_6 | Lambda | input: | output: | [(None, 5), (None, 5), (None, 5)] | (None, 15)
dense_18 | Dense | input: | output: | (None, 15) | (None, 512)
dense_19 | Dense | input: | output: | (None, 512) | (None, 128)
dense_20 | Dense | input: | output: | (None, 128) | (None, 5)

Figure 6.4 Stacked Ensemble architecture

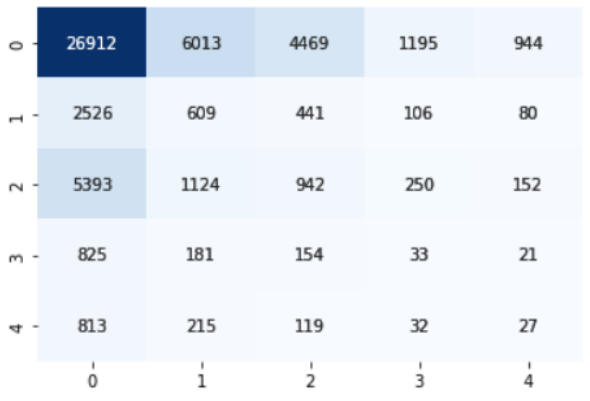
The first experiment with this approach was made with low neurons in the last two dense layer, but during the training the CNN doesn't recognize images of classes. The accuracy remains in 0.2

value. In the final experiment the last two layer was increased with 512 and 128 as shown in the architecture figure and the network after 25 epochs the training phase was blocked due the early stopping.





```
Accuracy on Test Data: 53.24%
              precision    recall  f1-score   support

           0       0.74      0.68      0.71     39533
           1       0.07      0.16      0.10      3762
           2       0.15      0.12      0.13      7861
           3       0.02      0.03      0.02      1214
           4       0.02      0.02      0.02      1206

    accuracy                           0.53     53576
   macro avg       0.20      0.20      0.20     53576
weighted avg       0.57      0.53      0.55     53576
```



	0	1	2	3	4
0	26912	6013	4469	1195	944
1	2526	609	441	106	80
2	5393	1124	942	250	152
3	825	181	154	33	21
4	813	215	119	32	27

Figure 6.5 Test model stacking

7. Explainability

In this section are explained two possible techniques that analyze how the network classifies the images. All the techniques are applied on the models of each network developed.

7.1 Intermediate Activations

The first premise to be made is that, in order to compare the models, the experiments were carried out with the same image.

In this section all the feature maps produced by the convolutional and pooling layers in the network are displayed and analysed.

7.1.1 CNN from scratch

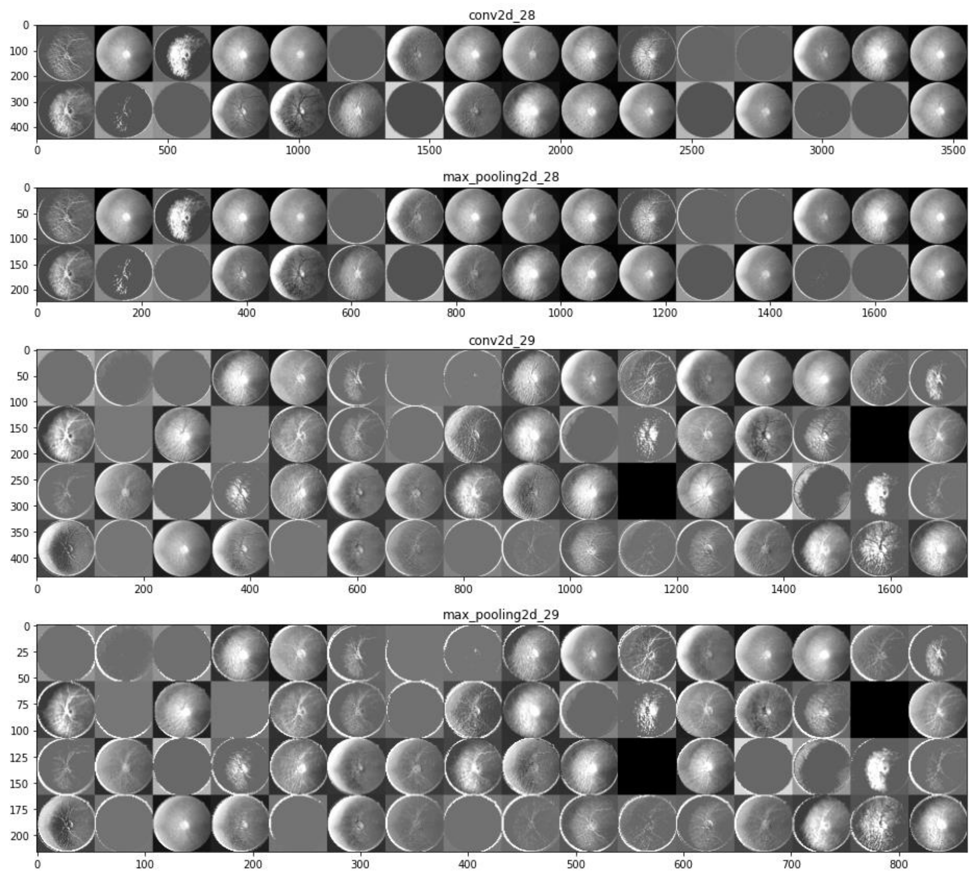


Figure 7.1 Intermediate Activations CNN from scratch

7.1.2 VGG16

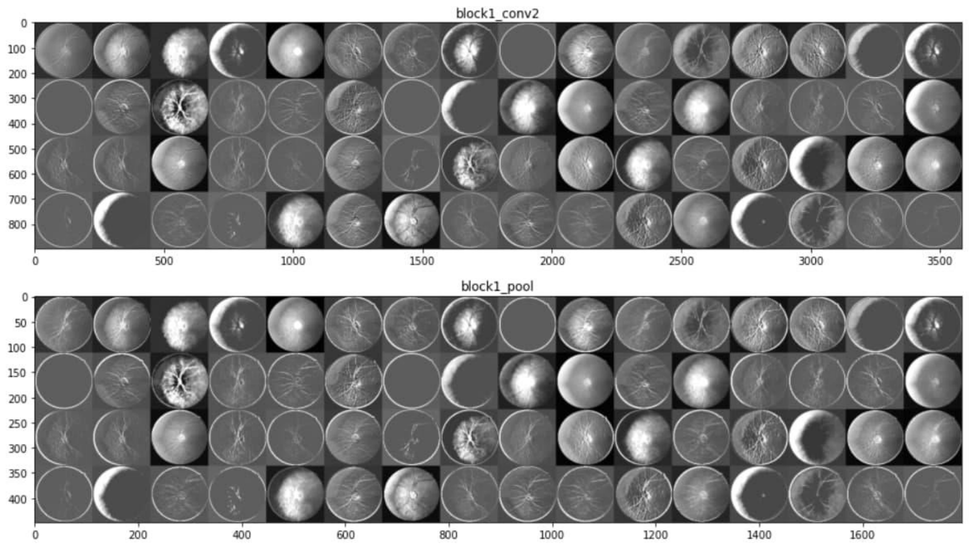


Figure 7.2 Intermediate Activations VGG16

7.1.3 MobileNetV2

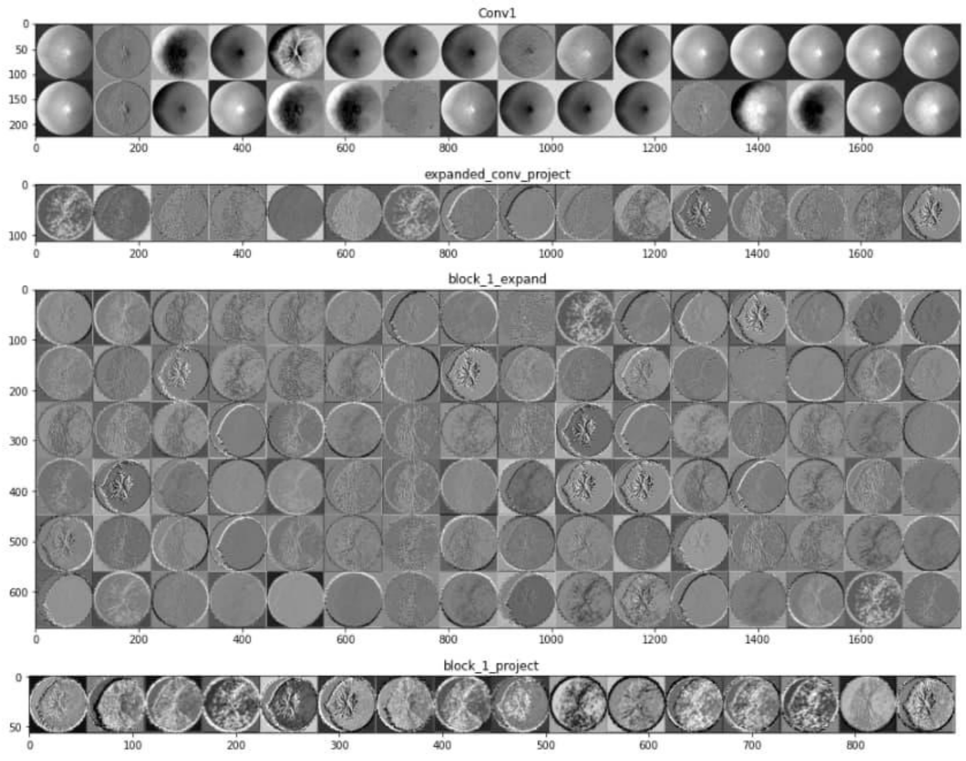


Figure 7.3 Intermediate Activations MobileNetV2

7.2 Heatmap of Class Activation

The first premise to be made is that, in order to compare the models, the experiments were carried out with the same image.

In this section will be analysed which part of the images was exploited in order to make the final classification.

7.2.1 Class 0

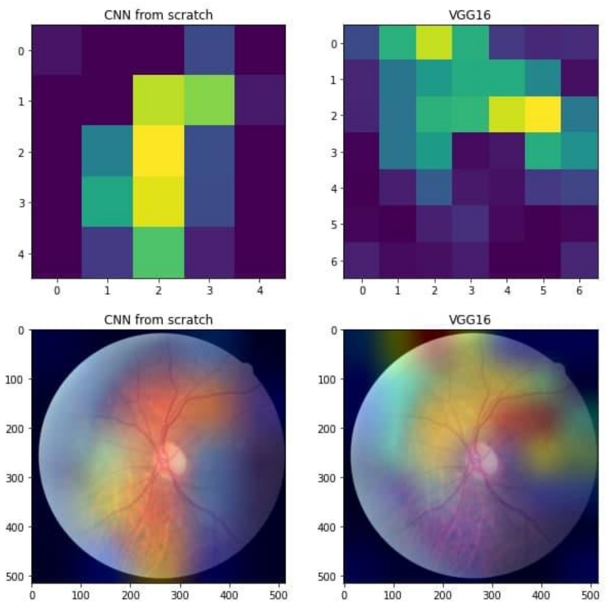


Figure 7.4 Class 0 image's heatmap

7.2.2 Class 1

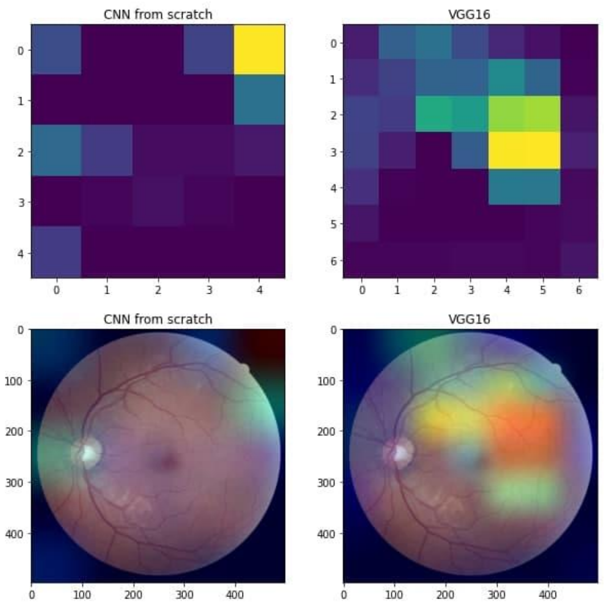


Figure 7.5 Class 1 image's heatmap

7.2.3 Class 2

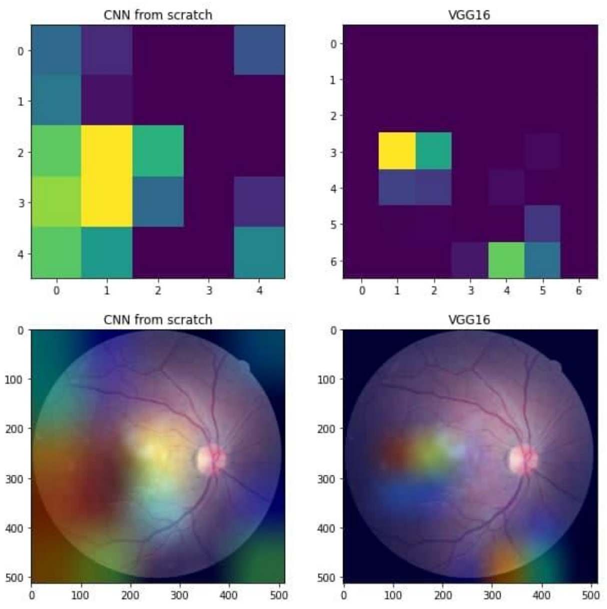


Figure 7.6 Class 2 image's heatmap

7.2.4 Class 3

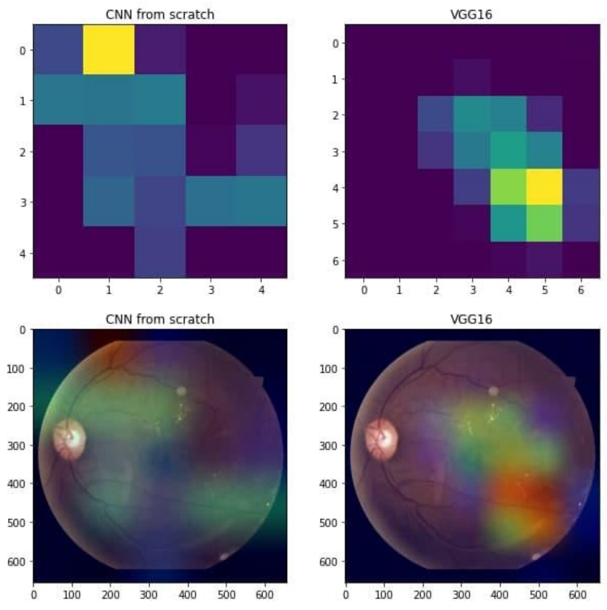


Figure 7.7 Class 3 image's heatmap

7.2.5 Class 4

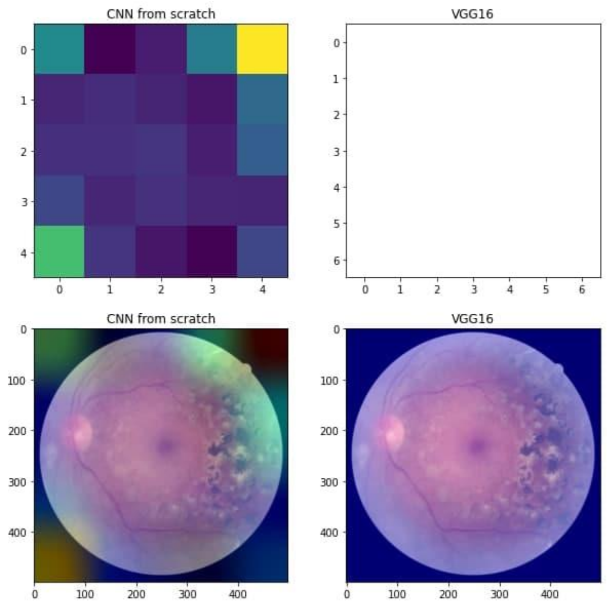


Figure 7.8 Class 4 image's heatmap

8. Conclusions

The results obtained are not satisfying, this condition is due to several factors that we have tried to analyze. The dataset taken in analysis was composed of 35126 images (+53576 images of the test set) unmanageable for resource constrains of Google Colab. The nature of the images is another penalizing factor as they present different levels of brightness and exposure in addition to loud noise. Moreover the images, being of medical nature, have not been classified in satisfactory way from the pre-trained networks as these last ones have been trained on ImageNet, born for different purposes. In addition to this, the challenge from which we downloaded the dataset had a large reward in terms of money, to confirm the complexity of the problem. The complexity of the problem is mainly due to the presence of 5 classes composed of images very similar to each other and not easily distinguishable from people that do not have experience in the medical field. Given the complexity, it was decided to start the project with an in-depth research on the state of the art. Given the popularity of the challenge, on the web many papers have been found and they can be divided into two categories. The first category consists of a minority of people who participated in the challenge and managed to achieve satisfactory results on the entire test set. Unfortunately, the solutions adopted in these papers are not public available. Other papers, as well as the majority, have results that appear to be relevant but that after careful analysis reveal excellent accuracy only on the majority class. Finally, given the restrictions on Google Colab Free accounts (and the numerous disconnections) our analysis could not last long which prevented us from achieving the desired results. Despite all the problems we encountered, we experienced different configurations, but we are sure that it is still possible to improve the analysis, for example by using hyperparameter optimization to define more accurate models.

9. References

1. https://en.wikipedia.org/w/index.php?title=Diabetic_retinopathy&oldid=1056390270
2. https://ieeexplore.ieee.org/abstract/document/8869883 (“Deep convolutional neural networks for diabetic retinopathy detection by image classification”)
3. https://www.researchgate.net/publication/328667007_Deep_convolutional_neural_networks_for_diabetic_retinopathy_detection_by_image_classification
4. Patel, S. (2020). Diabetic Retinopathy Detection and Classification using Pre-trained Convolutional Neural Networks. International Journal on Emerging Technologies, 11: 1082–1087
5. https://ieeexplore.ieee.org/abstract/document/8050011?casa_token=JZtAsDup90kAAAAA:KIY6Hytz-CQuzzNDQ7QkBSys9a4_N1rbREd-oml_bpjvEQX-lO1TjAa7ogpdVzQvkV7rC1X6RUQ (“Automatic Detection and Classification of Diabetic Retinopathy stages using CNN”)
6. https://www.mygreatlearning.com/blog/introduction-to-vgg16/
7. https://www.mygreatlearning.com/blog/resnet/
8. https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html
9. https://blog.paperspace.com/popular-deep-learning-architectures-resnet-inceptionv3-squeezenet/
10. https://cv-tricks.com/keras/understand-implement-resnets/
11. https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33
12. https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c
13. https://medium.com/@mygreatlearning/what-is-vgg16-introduction-to-vgg16-f2d63849f615
14. M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," 2019.
15. https://cloud.google.com/tpu/docs/inception-v3-advanced
16. https://en.wikipedia.org/wiki/Inceptionv3
17. https://paperswithcode.com/method/efficientnet
18. https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142
19. M. Mohsin Butt, Ghazanfar Latif, D.N.F. Awang Iskandar, Jaafar Alghazo, Adil H. Khan, “Multi-channel Convolutions Neural Network Based Diabetic Retinopathy Detection from Fundus Images”, 2019.