MACHINE LEARNING

Computer Science Engineering

Università degli Studi di Roma ”La Sapienza”

Andrea Lombardo

Matricola 1893440

December 13, 2019

Contents

1 Introduction

Technological development has involved numerous changes, one of these concerns the classification of images and objects. The classification and numbering of objects is an hard and annoying job for humans in fact nowadays they 're helped by different models which train a machine. According the model used the machine has got a different value of accuracy on the classification function but in general we can say that machines are able to recognize an element and classify it with a little value of miss-classification in comparison with humans. This task helps people saving a lot of time spent in counting and classifying objects, without considering that an human is subjected to a miss-classification also for the constant and monotonous job and physic conditions like illness or more common tiredness. Indeed humans spent a lot of time than a machine for the job of classifying objects due to the fact everyday the continuous evolution and creation of new objects has made the action of classifying objects more important and difficult for us. In general we prefer nowadays using a well-trained machine than job's person also the engine doesn't require pauses.

A branch of Machine Learning that is responsible for this important issue takes the name of Deep Learning. In this report we could see the powerful of this new technology, how much could be useful and how it could work.
In detail in that branch we can talk to the deep convolutional neural networks (CNNs) that have shown a great success in image classification, indeed it is important to note that most real world images contain multiple labels, which could correspond to different objects, scenes, actions and attributes in an image, in particular we 're focus on weather condition recognition which is widely required in many areas.

This aim of the developed system is monitoring the sky and predicting the weather condition, then it is able to classify from a picture what's the behaviour of the sky and classify it according the different targets we've provided: rainy,sunny, snowy and haze. This is a real problem which is analyzed nowadays by experts. This homework will discuss two possible implementations of models that are able to detect the particular class own of an image belongs at the provided dataset, furthermore, the performances retrieved will be analyzed and discuss afterwards.

2 Dataset

The provided dataset is composed by different directories where we can select from them the training sets and a test sets available in this link. I decide to fix as train set that collection of 2000 photos which are included in the directory named as **MWI-Dataset-1.1_2000** composed by 4 sub-directory [rainy,sunny,snowy,haze].
Instead for the test-set I decide to analyze first the **TestSetWeather** provided

by the small company **SMART-I**;pictures of this collection are 3040 whose had caught weather condition using cameras in open space. Using this first test-set I decide to make the part of A of the homework: **define a CNN and train it from scratch,** the only constraints I put is adding a directory called 'haze' for avoiding error code in developing phase because TestSetWeather contains only 3 target('sunny','rainy','snowy'). After testing this with only a CNN I make the testing for the part A and B with a second test-set **MWI-Dataset-1.1_400** which is is a sub-set of the train-set.

3 Detail of development

In order to implement the learning algorithm, the parser and the validator for performance I decide to use **Python language**. As a framework I used **Keras**, which is a high-level neural network API written in Python. But Keras can't work by itself, it needs a backend for low-level operations so I use a dedicated software library **Google's TensorFlow**. Keras is used to extract features using a pre-trained neural network. As a development environment I decide to use **Google Colab** for performance reason. I used **Matplotlib library** for visualization of accuracy and loss performance. For network training and testing I used a delivered dataset of photos of street or place where we should focus on the climate. Finally I used plotlib to render the confusion matrix and the evolution of the value accuracy and loss value.

The goals of this homework are two: define from scratch a CNN and define a CNN using transfer learning with a pre-trained CNN.
In general each CNN could be split into two phases: the first defined feature extraction phase or convolutional part and after the first phase there is the feature learning phase defined as the fully connected neural network.

4 Neural Network

The model we are going to build for classifying images is the neural network which represents an Artificial Neural Network model that is composed by a collection of neurons organized in groups where each group takes the name of layer and identifies a particular level of the network. In each layer neurons concur to solve the same task. The first point of the homework is going to build a Convolutional Neural Network (CNN) which is a particular feedforward neural network where all information passes from input nodes to output through the network without having cycles among neurons. The neural newtworks have an excellent efficiency in classification problems but there are some problems with the expression of the results obtained. The goal of a CNN is to learn a set of parameters called weights (or kernels) and biases such that, at the end of the training phase, applying those filters to a new image, the network should be able to correctly classify it. The main advantage of this approach different to other image classification algorithms is that in CNNs filters are self-learned during the training phase regardless of the human hand or a priori knowledge. Nowadays one of the famous CNN is known as the covolutional neural network. This particular neural network is used for multiple tasks as:

- Analysis and application of filters to pictures
- classification of object into an image
- classification of picture
- Features extraction

5 Metrics

In Machine Learning, performance measurement is an essential task. Indeed when it comes to a classification problem we should measure in terms of:
Precision= $\frac{tp}{tp+fp}$
Recall= $\frac{tp}{tp+fn}$
F1-score=$\frac{2PR}{P+R}$
Accuracy=$\frac{tp+tn}{tp+tn+fp+fn}$
Where:

True Positive (tp) is when correctly classified as positive
True Negative (tn) is when correctly classified as negative
False Positive (fp) is when falsely classified as positive
False Negative (fn) is when falsely classified as negative

Another element of measure is **Loss** is the penalty for a bad prediction. A machine learning algorithm builds a model by examining many examples and attempting to find a model that minimizes loss. That is, loss is a number indicating how bad the model's prediction was on a single example. If the model's prediction is perfect, the loss is zero; otherwise, the loss is greater. How we can see in the experiments I've done we can analyze the evolution of accuracy and the loss value in the different epochs.

6 Description

The job that I should do is resumed in the following steps:

1-Examine and understand data we have in dataset in this part I identify how I can modify the photos for improving the result and during this phase I choose between dataset which is the trainset and the testset of my experiment.

2-Build an input pipeline in this phase I decide to modify the two dataset(trainingset and test set) according the following policies:

- Modifying the size of the picture in a 64x64 and then according to the experiment I decide to make a smoothing.
- Modifying the training-set with different filters in that way with ImageDataGenerator where I apply different flags like a zoom of 0.1 or turning into a part of the photo through **rotation_range=6, width_shift_range=0.1** and so on. This features help me to generate batches of tensor image data with real-time data augmentation.

3-Build the model in this part I decide to build a sequential model which is different for the added layers.

4-Train the model According the trainset I use 30 epochs for training the model build above

5-Test the model I test the testset with the previous train model I return the performance in metrics of accuracy and loss.

6-Improve the model and repeat the process in this step I should modify the weight in the same model and understanding where is the case to improve the weights for example remove some neuron into a layer .

Another important issue about the preprocessing photos is that I decide to modify the pictures to look like clearest to the model, for doing it I make a smoothing with PILLOW's API. In Image-Processing, smoothing an image reduces noises present in the image and produces less pixelated image. The smooth filters provided by Pillow are Box Filters, where each output pixel is the weighted mean of its kernel neighbours. Pillow provides a couple of smooth filters denoted by:

- ImageFilter.SMOOTH
- ImageFilter.SMOOTH_MORE

For this reason I implement a routine that according to 'type' string giving as input is able to understand which of two filter applied. I used multiple techniques to improve upon the current state of the art deep convolutional neural network based image classification pipeline. The main goal of this process it to add more image transformations to training data, add more transformations to generate additional predictions and see what is the behaviour of the model with these transformations. In the picture below we can see the Smoothing Filters:

(a) No Smoothing

(b) smoothing

(c) More Smoothing

Figure 1: Filters applied to an image

How can we see from a to c there's an evidence discarding of the pixels which is represented as the changing of the intensity of the colours of the image.

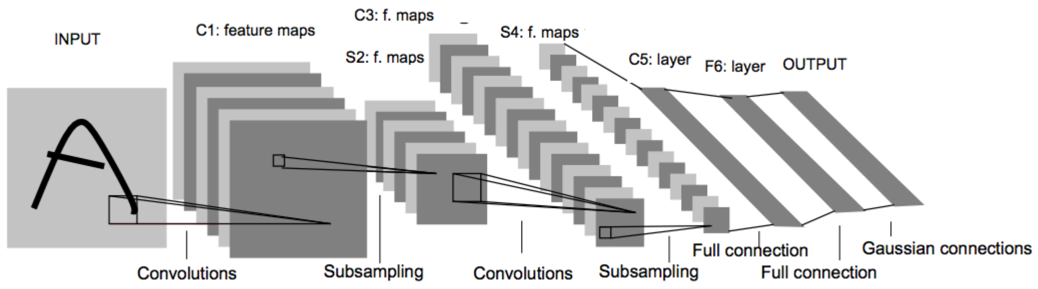
7 Models

In this sections we analyze the models used to build the system:

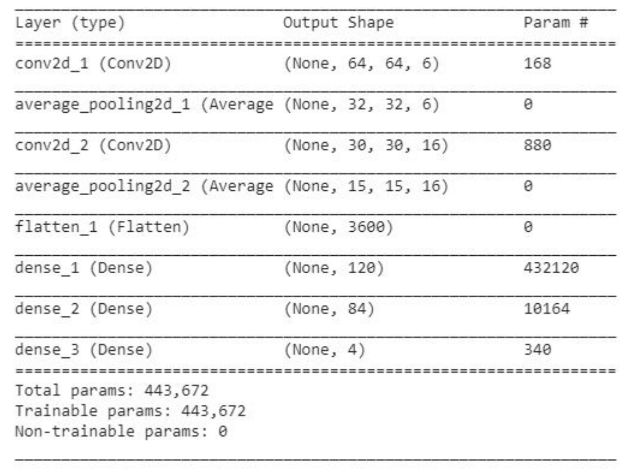
- in the first part I use a simplified LeNet net whose is composed by 5 layers without counting the input and output layer.
- in the second part I applied Transfer Learning from pre-trained model (VGG16).

7.1 LeNet-5 Architecture

LeNet-5 was used on large scale to automatically classify hand-written digits on bank cheques in the United States. This network is a convolutional neural network (CNN) that the real aim of this point of the homework.These networks are built upon 3 main ideas: local receptive fields, shared weights and spacial subsampling.



It's a Keras implementation of LeNet-5 network would be extremely readable and maintainable. We can change it and experiment with it with ease.It's composed by 7 layers as defined above, among which there are 3 convolutional layers (C1, C3 and C5), 2 sub-sampling (pooling) layers (S2 and S4), and 1 fully connected layer (F6), that are followed by the output layer.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 64, 64, 6)         168
_________________________________________________________________
average_pooling2d_1 (Average (None, 32, 32, 6)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 30, 30, 16)        880
_________________________________________________________________
average_pooling2d_2 (Average (None, 15, 15, 16)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3600)              0
_________________________________________________________________
dense_1 (Dense)              (None, 120)               432120
_________________________________________________________________
dense_2 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_3 (Dense)              (None, 4)                 340
=================================================================
Total params: 443,672
Trainable params: 443,672
Non-trainable params: 0
_________________________________________________________________
```

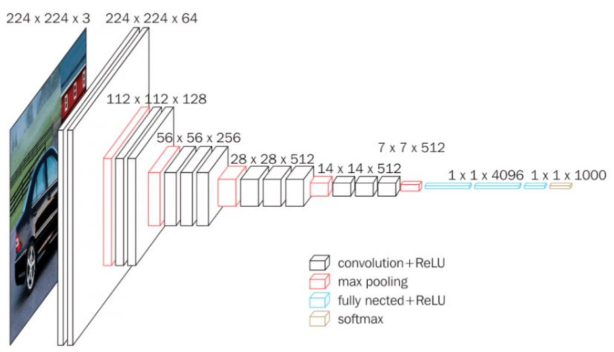
7.2 VGG-16 Architecture

In the second part I should use transfer learning which is a popular approach in deep learning where pre-trained models are used as the starting point. It's a technique where a model trained on one task is re-purposed on a second related task.A pre-trained model is a saved network pre-trained according to a dataset, typically it's used to a image-classification task. Types of most used pre-trained models for transfer learning:

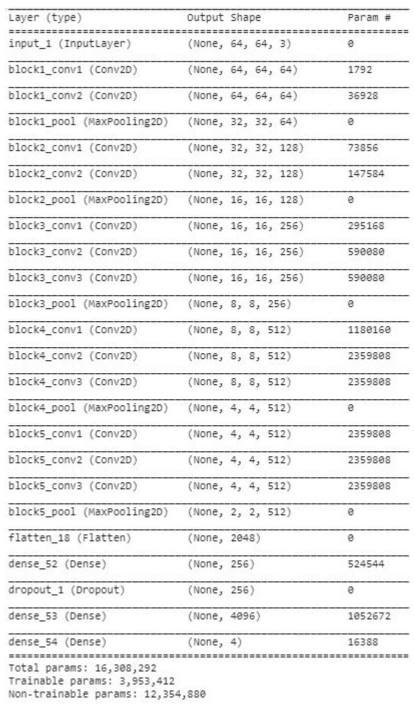
- AlexNet
- VGG-Net where we should divide it into two versions:16 and 19.
- RESNET

Now I decide to apply the VGG-Net 16 that is a simplified version of AlexNet with 3*3 kernal, stride=1 and padding= same. In this CNN we have:

- Convolutions layers (used only 3*3 size)
- Max pooling layers (used only 2*2 size)
- Fully connected layers at end with a total number of layers equal to 16



After build the model using transfer learning (VGG16) I freeze a number of layers starting from top and add some custom dense layers. In the icture behing we can see a summary of the layers.



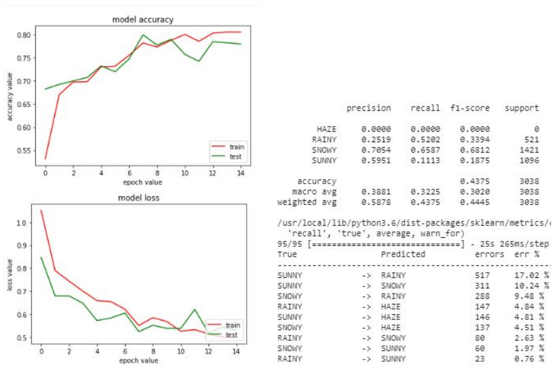
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 64, 64, 3)         0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 64, 64, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 64, 64, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 32, 32, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 32, 32, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 32, 32, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 16, 16, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 16, 16, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 8, 8, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 8, 8, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 4, 4, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 2, 2, 512)         0
_________________________________________________________________
flatten_18 (Flatten)         (None, 2048)              0
_________________________________________________________________
dense_52 (Dense)             (None, 256)               524544
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_53 (Dense)             (None, 4096)              1052672
_________________________________________________________________
dense_54 (Dense)             (None, 4)                 16388
=================================================================
Total params: 16,308,292
Trainable params: 3,953,412
Non-trainable params: 12,354,880
```

8 Experiments and Results

In this section we discuss about the experiments done. In the experiments from 1 to 3 I use only the(LeNet-5) with the pair training set and test set respectivly(**MWI-Dataset-1.1_2000** and **TestSetWeather**) where the differences between from test 1 to 3 is the amount of smoothing filter applied in each picture, so 2 in a smoothed with a slight value instead in the experiment 3 each picture is smoothed with an hard value. With the same pattern applied from 1 to 3 related on filtering and preproessing of the images I decide to make the same kind of experiments from 4 to 6 changing the pair of the dataset provided in input: in detail I modify only the dataset for the test set(**MWI-Dataset-1.1_400**) and using both the nets:(LeNet-5 and VGG-16).

8.1 Experiment 1

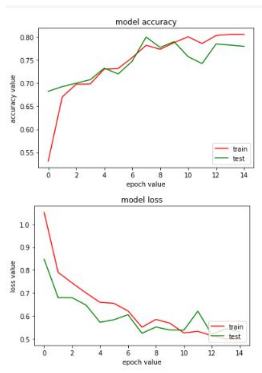
In the first experiment I choose to use the net: LeNet-5 with the same parameters:(15 epochs and with the same input shape 64x64). This kind of approach is in the case of testset with **TestSetWeather**.
LeNet-5

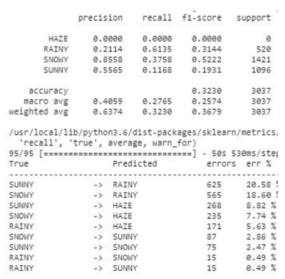


```
              precision    recall  f1-score   support

        HAZE     0.0000    0.0000    0.0000         0
       RAINY     0.2519    0.5202    0.3394       521
       SNOWY     0.7054    0.6587    0.6812      1421
       SUNNY     0.5951    0.1113    0.1875      1096

    accuracy                         0.4375      3038
   macro avg     0.3881    0.3225    0.3020      3038
weighted avg     0.5878    0.4375    0.4445      3038

/usr/local/lib/python3.6/dist-packages/sklearn/metrics/
  'recall', 'true', average, warn_for)
95/95 [==============================] - 25s 265ms/step
True                 Predicted          errors  err %
-------------------------------------------------------
SUNNY            ->  RAINY              517     17.02 %
SUNNY            ->  SNOWY              311     10.24 %
SNOWY            ->  RAINY              288     9.48 %
RAINY            ->  HAZE               147     4.84 %
SUNNY            ->  HAZE               146     4.81 %
SNOWY            ->  HAZE               137     4.51 %
RAINY            ->  SNOWY              80      2.63 %
SNOWY            ->  SUNNY              60      1.97 %
RAINY            ->  SUNNY              23      0.76 %
```

8.2 Experiment 2

In the second experiment I choose to modify the photos of the test set with a smoothing filtering, the **Smoothening(path,type)** is the routine that determine the value of the smoothing. In that routine I save in the same diretory and with the same name the photo modified. The net used is LeNet-5 with the same parameters:(15 epochs and with the same input shape 64x64). This kind of approach is in the case of testset with **TestSetWeather**.
LeNet-5

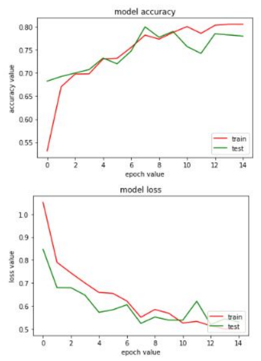


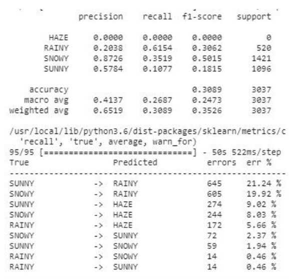


```
              precision    recall  f1-score   support

        HAZE     0.0000    0.0000    0.0000         0
       RAINY     0.2114    0.6135    0.3144       520
       SNOWY     0.8558    0.3758    0.5222      1421
       SUNNY     0.5565    0.1168    0.1931      1096

    accuracy                         0.3230      3037
   macro avg     0.4059    0.2765    0.2574      3037
weighted avg     0.6374    0.3230    0.3679      3037

/usr/local/lib/python3.6/dist-packages/sklearn/metrics,
  'recall', 'true', average, warn_for)
95/95 [==============================] - 50s 530ms/step
True                 Predicted         errors  err %
-------------------------------------------------------
SUNNY           ->  RAINY              625     20.58
SNOWY           ->  RAINY              565     18.60
SUNNY           ->  HAZE               268     8.82 %
SNOWY           ->  HAZE               235     7.74 %
RAINY           ->  HAZE               171     5.63 %
SNOWY           ->  SUNNY              87      2.86 %
SUNNY           ->  SNOWY              75      2.47 %
RAINY           ->  SNOWY              15      0.49 %
RAINY           ->  SUNNY              15      0.49 %
```

8.3 Experiment 3

In the third experiment I choose to modify the photos of the test set with a smoothing filtering that is higher than that applied in the experiment number 2, the **Smoothening(path,type)** is the routine that determine the value of the smoothing.This routine is implemented through glob library and the API's ImageFilter imported on PIL library. The net used is LeNet-5 with the same parameters:(15 epochs and with the same input shape 64x64). This kind of approach is in the case of testset with **TestSetWeather**.
LeNet-5

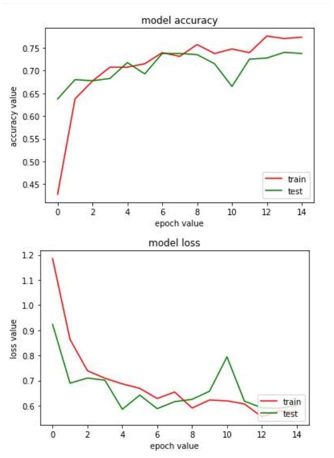




```
              precision    recall  f1-score   support

        HAZE     0.0000    0.0000    0.0000         0
       RAINY     0.2038    0.6154    0.3062       520
       SNOWY     0.8726    0.3519    0.5015      1421
       SUNNY     0.5784    0.1077    0.1815      1096

    accuracy                         0.3089      3037
   macro avg     0.4137    0.2687    0.2473      3037
weighted avg     0.6519    0.3089    0.3526      3037

/usr/local/lib/python3.6/dist-packages/sklearn/metrics/c
  'recall', 'true', average, warn_for)
95/95 [==============================] - 50s 522ms/step
True                 Predicted         errors  err %
--------------------------------------------------------
SUNNY           ->  RAINY              645     21.24 %
SNOWY           ->  RAINY              605     19.92 %
SUNNY           ->  HAZE               274     9.02 %
SNOWY           ->  HAZE               244     8.03 %
RAINY           ->  HAZE               172     5.66 %
SNOWY           ->  SUNNY              72      2.37 %
SUNNY           ->  SNOWY              59      1.94 %
RAINY           ->  SNOWY              14      0.46 %
RAINY           ->  SUNNY              14      0.46 %
```

8.4 Experiment 4

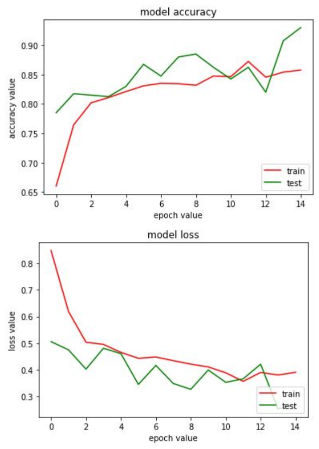
In the experiment number four the nets used are LeNet-5 and VGG-16 with the same parameters:(15 epochs and with the same input shape 128x128). This kind of approach is in the case of testset with **MWI-Dataset-1.1.1_400**.
LeNet-5



```
               precision    recall  f1-score   support

        HAZE       0.876     0.850     0.863       100
       RAINY       0.726     0.450     0.556       100
       SNOWY       0.623     0.660     0.641       100
       SUNNY       0.733     0.990     0.843       100

    accuracy                           0.738       400
   macro avg       0.740     0.738     0.725       400
weighted avg       0.740     0.738     0.725       400

13/13 [==============================] - 5s 396ms/step
True                Predicted          errors  err %
-----------------------------------------------------
RAINY            ->  SNOWY              34      8.50 %
RAINY            ->  SUNNY              16      4.00 %
SNOWY            ->  RAINY              15      3.75 %
SNOWY            ->  SUNNY              12      3.00 %
HAZE             ->  SUNNY              8       2.00 %
SNOWY            ->  HAZE               7       1.75 %
HAZE             ->  SNOWY              5       1.25 %
RAINY            ->  HAZE               5       1.25 %
HAZE             ->  RAINY              2       0.50 %
SUNNY            ->  SNOWY              1       0.25 %
```

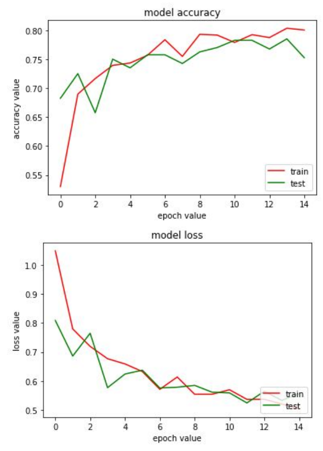
VGG-16

```
               precision    recall  f1-score   support

        HAZE       0.940     0.940     0.940       100
       RAINY       0.880     0.950     0.913       100
       SNOWY       0.923     0.840     0.880       100
       SUNNY       0.980     0.990     0.985       100

    accuracy                           0.930       400
   macro avg       0.931     0.930     0.930       400
weighted avg       0.931     0.930     0.930       400

True                Predicted          errors  err %
-----------------------------------------------------
SNOWY            ->  RAINY              10      2.50 %
SNOWY            ->  HAZE               5       1.25 %
HAZE             ->  SNOWY              4       1.00 %
RAINY            ->  SNOWY              3       0.75 %
HAZE             ->  RAINY              2       0.50 %
RAINY            ->  HAZE               1       0.25 %
RAINY            ->  SUNNY              1       0.25 %
SNOWY            ->  SUNNY              1       0.25 %
SUNNY            ->  RAINY              1       0.25 %
```



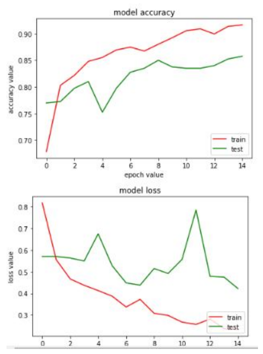
8.5 Experiment 5

In the experiment number five I choose to modify the photos of the test set with a smoothing filtering which is the same of the 2 experiment. The nets used are LeNet-5 and VGG-16 with the same parameters:(15 epochs and with the same input shape 128x128). This kind of approach is in the case of testset with **MWI-Dataset-1.1.1_400**.
LeNet-5



```
              precision    recall  f1-score   support

        HAZE     0.9011    0.8200    0.8586       100
       RAINY     0.5482    0.9100    0.6842       100
       SNOWY     0.7500    0.3300    0.4583       100
       SUNNY     0.9596    0.9500    0.9548       100

    accuracy                         0.7525       400
   macro avg     0.7897    0.7525    0.7390       400
weighted avg     0.7897    0.7525    0.7390       400

13/13 [==============================] - 11s 856ms/step
True                Predicted           errors   err %
-----------------------------------------------------------
SNOWY          ->   RAINY               61       15.25 %
HAZE           ->   RAINY               12       3.00 %
RAINY          ->   SNOWY               5        1.25 %
SNOWY          ->   HAZE                5        1.25 %
HAZE           ->   SNOWY               4        1.00 %
RAINY          ->   HAZE                3        0.75 %
HAZE           ->   SUNNY               2        0.50 %
SUNNY          ->   RAINY               2        0.50 %
SUNNY          ->   SNOWY               2        0.50 %
RAINY          ->   SUNNY               1        0.25 %
SNOWY          ->   SUNNY               1        0.25 %
SUNNY          ->   HAZE                1        0.25 %
```

VGG

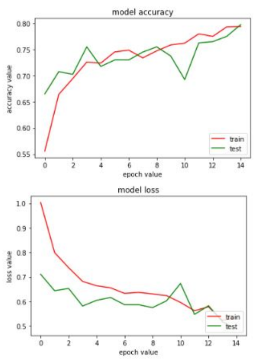


```
              precision    recall  f1-score   support

        HAZE      0.914     0.960     0.937       100
       RAINY      0.771     0.810     0.790       100
       SNOWY      0.789     0.710     0.747       100
       SUNNY      0.950     0.950     0.950       100

    accuracy                          0.858       400
   macro avg      0.856     0.857     0.856       400
weighted avg      0.856     0.858     0.856       400

True                Predicted           errors   err %
-----------------------------------------------------------
SNOWY          ->   RAINY               22       5.50 %
RAINY          ->   SNOWY               15       3.75 %
SNOWY          ->   HAZE                7        1.75 %
SUNNY          ->   SNOWY               4        1.00 %
RAINY          ->   SUNNY               3        0.75 %
HAZE           ->   RAINY               2        0.50 %
HAZE           ->   SUNNY               2        0.50 %
RAINY          ->   HAZE                1        0.25 %
SUNNY          ->   HAZE                1        0.25 %
```

8.6 Experiment 6

In the experiment number six I choose to modify the photos of the test set with a smoothing filtering value which is the same of the 3 experiment. The nets used are LeNet-5 and VGG-16 with the same parameters:(15 epochs and with the same input shape 128x128). This kind of approach is in the case of testset with **MWI-Dataset-1.1.1_400**.

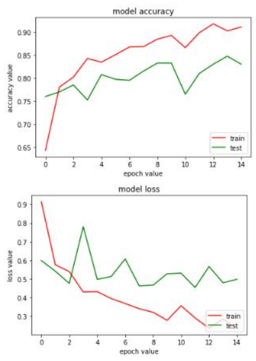
LeNet-5



```
                precision    recall  f1-score   support

        HAZE       0.8911    0.9000    0.8955       100
       RAINY       0.6525    0.7700    0.7064       100
       SNOWY       0.7260    0.5300    0.6127       100
       SUNNY       0.9167    0.9900    0.9519       100

    accuracy                           0.7975       400
   macro avg       0.7966    0.7975    0.7916       400
weighted avg       0.7966    0.7975    0.7916       400

13/13 [==============================] - 11s 865ms/ste
True                Predicted           errors  err %
------------------------------------------------------
SNOWY            ->  RAINY              37       9.25 %
RAINY            ->  SNOWY              15       3.75 %
SNOWY            ->  HAZE               7        1.75 %
HAZE             ->  RAINY              4        1.00 %
HAZE             ->  SNOWY              4        1.00 %
RAINY            ->  HAZE               4        1.00 %
RAINY            ->  SUNNY              4        1.00 %
SNOWY            ->  SUNNY              3        0.75 %
HAZE             ->  SUNNY              2        0.50 %
SUNNY            ->  SNOWY              1        0.25 %
```

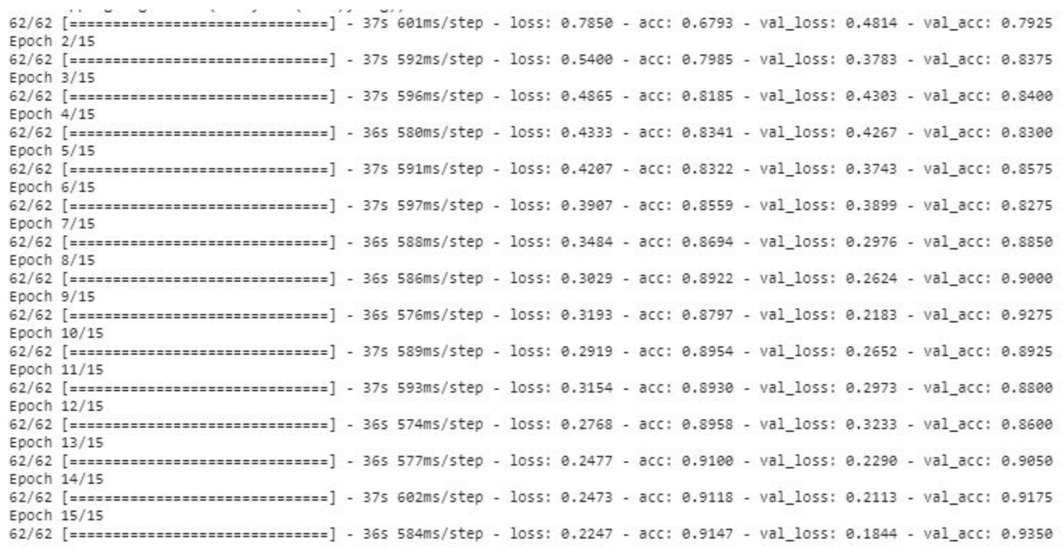
VGG-16

```
              precision    recall  f1-score   support

        HAZE      0.825     0.990     0.900       100
       RAINY      0.826     0.710     0.763       100
       SNOWY      0.750     0.690     0.719       100
       SUNNY      0.912     0.930     0.921       100

    accuracy                          0.830       400
   macro avg      0.828     0.830     0.826       400
weighted avg      0.828     0.830     0.826       400

True                Predicted           errors  err %
------------------------------------------------------
RAINY            ->  SNOWY              19       4.75 %
SNOWY            ->  HAZE               14       3.50 %
SNOWY            ->  RAINY              14       3.50 %
RAINY            ->  SUNNY              6        1.50 %
RAINY            ->  HAZE               4        1.00 %
SNOWY            ->  SUNNY              3        0.75 %
SUNNY            ->  HAZE               3        0.75 %
SUNNY            ->  SNOWY              3        0.75 %
HAZE             ->  SNOWY              1        0.25 %
SUNNY            ->  RAINY              1        0.25 %
                  model accuracy
```



9 Conclusions

How we an see from the different experiments the performances are quite clear: from the two models we prefer using that pre-trained so that used VGG-16 because how we can see in the experiment 4 it's able to have an accuracy value of 0.93 that is very high with a loss value near to 0. Indeed the percentages of errors are quite low. Instead about the filter applied we can notice how the system work with a filtering smoothing: we prefer more smoothed than simple smoothing, we can see it between performances from experiments 5 and 6, but in general we don't prefer using it because how we can see from 1 to 3 and 4 to 6 best performances are retrieved to the dataset where we don't apply that filtering. The best model from the two is that provided with this report: VGG-16. Behind we could find how is the evolution of the training phase epoch for epoch.



```
62/62 [==============================] - 37s 601ms/step - loss: 0.7850 - acc: 0.6793 - val_loss: 0.4814 - val_acc: 0.7925
Epoch 2/15
62/62 [==============================] - 37s 592ms/step - loss: 0.5400 - acc: 0.7985 - val_loss: 0.3783 - val_acc: 0.8375
Epoch 3/15
62/62 [==============================] - 37s 596ms/step - loss: 0.4865 - acc: 0.8185 - val_loss: 0.4303 - val_acc: 0.8400
Epoch 4/15
62/62 [==============================] - 36s 580ms/step - loss: 0.4333 - acc: 0.8341 - val_loss: 0.4267 - val_acc: 0.8300
Epoch 5/15
62/62 [==============================] - 37s 591ms/step - loss: 0.4207 - acc: 0.8322 - val_loss: 0.3743 - val_acc: 0.8575
Epoch 6/15
62/62 [==============================] - 37s 597ms/step - loss: 0.3907 - acc: 0.8559 - val_loss: 0.3899 - val_acc: 0.8275
Epoch 7/15
62/62 [==============================] - 36s 588ms/step - loss: 0.3484 - acc: 0.8694 - val_loss: 0.2976 - val_acc: 0.8850
Epoch 8/15
62/62 [==============================] - 36s 586ms/step - loss: 0.3029 - acc: 0.8922 - val_loss: 0.2624 - val_acc: 0.9000
Epoch 9/15
62/62 [==============================] - 36s 576ms/step - loss: 0.3193 - acc: 0.8797 - val_loss: 0.2183 - val_acc: 0.9275
Epoch 10/15
62/62 [==============================] - 37s 589ms/step - loss: 0.2919 - acc: 0.8954 - val_loss: 0.2652 - val_acc: 0.8925
Epoch 11/15
62/62 [==============================] - 37s 593ms/step - loss: 0.3154 - acc: 0.8930 - val_loss: 0.2973 - val_acc: 0.8800
Epoch 12/15
62/62 [==============================] - 36s 574ms/step - loss: 0.2768 - acc: 0.8958 - val_loss: 0.3233 - val_acc: 0.8600
Epoch 13/15
62/62 [==============================] - 36s 577ms/step - loss: 0.2477 - acc: 0.9100 - val_loss: 0.2290 - val_acc: 0.9050
Epoch 14/15
62/62 [==============================] - 37s 602ms/step - loss: 0.2473 - acc: 0.9118 - val_loss: 0.2113 - val_acc: 0.9175
Epoch 15/15
62/62 [==============================] - 36s 584ms/step - loss: 0.2247 - acc: 0.9147 - val_loss: 0.1844 - val_acc: 0.9350
```

Using the model provide I manage to predict that the pictures in BlindSet are divided in four kind of categories: RAINY **232**, SUNNY **398** SNOWY **499** HAZE **371**.

10 Improvements

In this section, we will briefly describe some possible improvements to our project. Through a new approach to this project, it could be interesting to do the same experiment using a specific model able to understand with an accuracy higher than 90%. Another important issue is try to test the model build in other pictures done with different devices as a drone or a phone by an human or a fixed camera in the traffic lights. Therefore, it could be more useful because in this way we also can see verify our system with pictures which has caught the instant weather with different prospective.
When the system is able to right guessing the classifier we can try to make a step forward, so try to prevent if could happen a violent situation in real-time by processing images in the cloud. This kind of research will prevent or mitigate hurricanes, earthquake or other natural disaster cause by the weather.

References

- Image Preprocessing
- keras-tutorial
- keas documentation
- convolutional-neural-network
- 2d-convolutions-keras
- confusion-matrix
- image classification
- training and loss
- tensorflow
- A Simple Deep Learning Baseline for Image Classification
- wikipedia rete convoluzionale
- convolutional-neural-network
- LeNet-5
- transfer-learning
- deep-learning-arhitectures-lenet
- vggnet vs resnet
- VGG16
- vgg-keras
- ML-pretrainedvggmodel