Multi-class Weather Classification

HW2 - Machine Learning - La Sapienza university of Rome

Manuel Ivagnes 1698903

30/11/19

1 Introduction

In the first homework, we understood how to apply some machine learning algorithms to the classification problem, but unfortunately, when it becomes much more complex and nonlinear, those methods are not vey useful.

The aim behind this homework is to understand how to solve a similar task with a more powerful technique, the *Artificial Neural Networks* (ANN). In particular, we want to use the **Convolutional Neural Network** (CNN) to classify weather conditions.

Neural Networks are computing systems with interconnected nodes inspired by the human brain. These can recognize hidden patterns and correlations in raw data.

The Convolutional neural networks are a specific type of NNs, which also contain at least one convolutional layer. These are widely used for image classification and object detection. However, CNNs have also been applied to other areas, such as natural language processing and forecasting.

2 Problem definition and Input data

The problem given is a supervised learning task, which requires to build a *multiclass classifier* to predict the weather conditions in a picture. The dataset contains 4000 pictures divided in 4 different folders, one for each class. The classes are:

- HAZE
- SUNNY
- SNOWY
- RAINY

The images are provided as jpg files, in different sizes.

For the implementation of the neural network, I will use *TensorFlow* 2.0 (GPU-version), which is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks [5]. Furthermore, it natively uses *Keras* as interface for this last task.

3 Data Preprocessing

Also in this case we need to do some data preprocessing. First of all, we need to deal with the input size, indeed the CNN needs to have all the pictures with the same size to perform the task.

The Keras interface provides a useful class for the image preprocessing, the *ImageDataGenerator* class, but unfortunately, the target_size option gave me some troubles, so I decided to use a simple script (contained in resize.py) to resize the images before mounting the folder. This also increased the computation's speed, executing the action only one time.

However, this class methods provide many other useful tools for the data augmentation, which helps to expose the classifier to a wider variety of lighting and coloring situations so as to make it more robust. Indeed, this makes possible to zoom, rotate, cut and apply many other changes to the pictures, just by setting some parameters.

By default, the modifications will be applied randomly while performing the fit_generator(), so not every image will be changed every time.

Note: only the training set should be augmented, the validation set should remain the same.

4 Brief overview on CNNs

While in the previous homework we were using the Vectorizer for the features engineering, here, we use the convolutional layers for the features extraction and then we process everything with the dense layers.

The role of the Convolutional Neural Network is to reduce the images into a form which is easier to process, without losing critical features for getting a good prediction. Obtained by sequentially repeating 2 main stages:

- **Convolutional stage** ⇒ Extract the high-level features from the input image. The layer's parameters consist of a set of learnable filters (or kernels), which are convolved across the width and height of the input volume, to compute the dot product between the entries of the filter and the input. This process produces a 2-dimensional activation map of the corresponding filters.

- **Pooling** ⇒ Responsible for reducing the spatial size of the Convolved Features. It is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model. Moreover, also performs as a Noise Suppressant.

Between the hidden layers there is the activation function: it decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

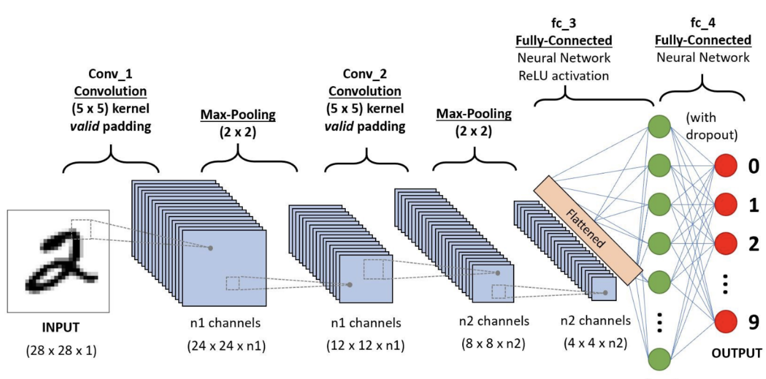


Figure 1: Example of Convolutional Neural Network

After these operations there is 'Flatten' which ransforms a two-dimensional matrix of features into a vector that can be fed into a fully connected neural network classifier.

5 Model from scratch Tests

In this section, I will expose some tests that I believe to be the most interesting to show how I built the solution, following an heuristic construction based on empirical observations. Here some parameters and assumptions used in all tests:

- The first 2 tests use 150x150 pixels images, then I started to use the 224x224 size, as for AlexNet. All the pictures had different format so, considering that, in this case it does not really influence the results, I decided to use something suitable for all pictures.
- Usually *Max Pooling* performs better than *Average Pooling*, but in this case we want to find the general atmosphere and the presence of light, so I preferred to use the less common one.
- The *ReLu function* (Rectified Linear unit) is used as activation function between each hidden layer, given that it is considered to be most efficient one. The next neuron is activated only if the output is positive.
- Given the fact that, we have a multi-class classification problem, the *softmax function* is used as activation function in the output layer.
- For the performance metrics, I will refer mostly to the concepts already explained during the previous homework. However, here we consider also the loss, which is a summation of the errors made for each example in training or validation sets.
- The loss function used in all the tests is the *categorical_crossentropy*, which compare the distribution of the predictions (the activations in the output layer, one for each class) with the true distribution, where the probability of the true class is set to 1 and 0 for the other classes. The true class is represented as a one-hot encoded vector, and the closer the model's outputs are to that vector, the lower the loss.

5.1 Test 1

This first test wants to be a demonstration of how, increasing the amount of images in the dataset, also the model increases the performance metrics. In fact, I have used a subsample of the dataset, containing only 400 images.

This gave me 320 images to use for the training and 80 for tests. The validation_split option of the ImageDataGenerator has been used to perform the division. There is no data augmentation.

The model is simply the basic one provided by the Tensorflow documentation to perform the convolution. It has been trained for 20 epochs.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896
_________________________________________________________________
average_pooling2d (AveragePo (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
average_pooling2d_1 (Average (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 64)        36928
_________________________________________________________________
flatten (Flatten)            (None, 73984)             0
_________________________________________________________________
dense (Dense)                (None, 64)                4735040
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 260
=================================================================
Total params: 4,791,620
Trainable params: 4,791,620
Non-trainable params: 0
_________________________________________________________________
```

Figure 2: model test 1

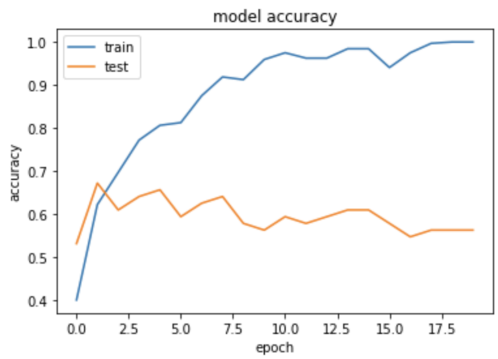
- **Loss** $\Rightarrow$ 1.730436
- **Accuracy** $\Rightarrow$ 0.550000

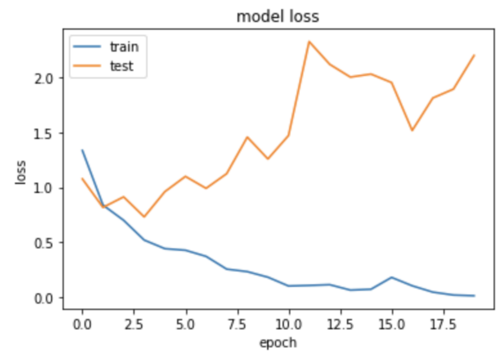
```
Loaded 80 test samples from 4 classes.
3/3 [==============================] - 0s 66ms/step
              precision    recall  f1-score   support

        HAZE      0.636     0.700     0.667        20
       RAINY      0.400     0.200     0.267        20
       SNOWY      0.310     0.450     0.367        20
       SUNNY      0.895     0.850     0.872        20

    accuracy                          0.550        80
   macro avg      0.560     0.550     0.543        80
weighted avg      0.560     0.550     0.543        80
```

Figure 3: Precision, Recall and F-score test 1





It is clear that this is not a good solution for the problem, the accuracy is too low and the loss too high.

The presence of a low quantity of images, without augmentation, does not provide enough samples. Moreover, the model does not provide systems to prevent the overfitting, so after few epochs the validation set performance metrics start to decrease.

```
[[14  0  5  1]
 [ 2  4 14  0]
 [ 4  6  9  1]
 [ 2  0  1 17]]

Confusion matrix analysis
True                 Predicted            errors  err %
-----------------------------------------------------------
RAINY            ->  SNOWY                14      17.50 %
SNOWY            ->  RAINY                6       7.50 %
HAZE             ->  SNOWY                5       6.25 %
SNOWY            ->  HAZE                 4       5.00 %
RAINY            ->  HAZE                 2       2.50 %
SUNNY            ->  HAZE                 2       2.50 %
HAZE             ->  SUNNY                1       1.25 %
SNOWY            ->  SUNNY                1       1.25 %
SUNNY            ->  SNOWY                1       1.25 %
```

Figure 5: Confusion Matrix and analysis

5.2 Test 2

Here the model is still the same, but in this case I have used all the dataset, with 4000 images. There are 3200 images to use for the training and 800 for tests. Moreover, the following augmentation options provided by Keras are applied to the dataset:

- rotation_range = 90,
- horizontal_flip = True,
- width_shift_range = [-10,10],
- height_shift_range = [-10,10],
- vertical_flip = True

Given the fact that, most of the difference between a sunny picture and the other pictures is given by the light, I decided to avoid the brightness parameter. It decreased the performances in some other tests.

Note: Trying to apply the augmentation also to the test set, the performance metrics seems to increase, but this can be explained by the overfitting.

```
Loaded 800 test samples from 4 classes.
25/25 [==============================] - 2s 95ms/step
              precision    recall  f1-score   support

        HAZE      0.766     0.705     0.734       200
       RAINY      0.642     0.485     0.553       200
       SNOWY      0.547     0.700     0.614       200
       SUNNY      0.785     0.820     0.802       200

    accuracy                          0.677       800
   macro avg      0.685     0.677     0.676       800
weighted avg      0.685     0.677     0.676       800
```

Figure 6: Precision, Recall and F-score test 2

- **Loss** ⇒ 0.789361
- **Accuracy** ⇒ 0.677500

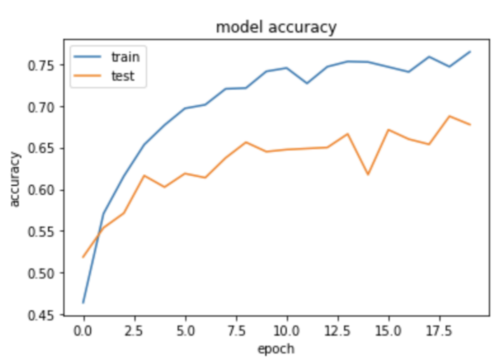
This solutions needs more epochs to reach the same metrics of the previous one, but after a while, instead of overfitting the model, it continues to raise the accuracy and reduce the loss.

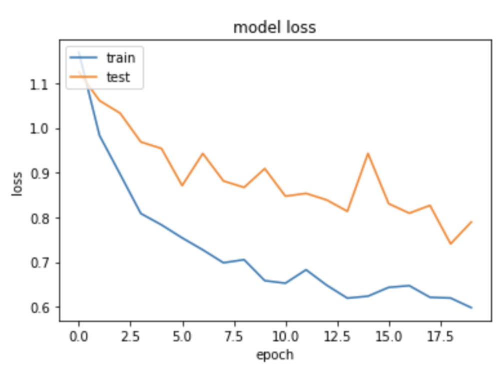
As seen before, this will start to overfit the model and it will never reach a good final scores, so it is time to update.

```
[[141  10  29  20]
 [ 15  97  75  13]
 [ 17  31 140  12]
 [ 11  13  12 164]]

Confusion matrix analysis
True                 Predicted            errors  err %
-------------------------------------------------------
RAINY            ->  SNOWY                75      9.38 %
SNOWY            ->  RAINY                31      3.88 %
HAZE             ->  SNOWY                29      3.62 %
HAZE             ->  SUNNY                20      2.50 %
SNOWY            ->  HAZE                 17      2.12 %
RAINY            ->  HAZE                 15      1.88 %
RAINY            ->  SUNNY                13      1.62 %
SUNNY            ->  RAINY                13      1.62 %
SNOWY            ->  SUNNY                12      1.50 %
SUNNY            ->  SNOWY                12      1.50 %
SUNNY            ->  HAZE                 11      1.38 %
HAZE             ->  RAINY                10      1.25 %
```

Figure 7: Confusion Matrix and analysis





5.3 AlexNet test

Find a good model is not easy and needs to try many different combinations, to start the construction I decided to use the well known structure of AlexNet.

```
Loaded 800 test samples from 4 classes.
25/25 [==============================] - 4s 172ms/step
              precision    recall  f1-score   support

        HAZE      0.750     0.030     0.058       200
       RAINY      0.431     0.500     0.463       200
       SNOWY      0.408     0.595     0.484       200
       SUNNY      0.642     0.860     0.735       200

    accuracy                          0.496       800
   macro avg      0.558     0.496     0.435       800
weighted avg      0.558     0.496     0.435       800
```

Figure 9: Precision, Recall and F-score AlexNet

- **Loss** $\Rightarrow$ 1.744001
- **Accuracy** $\Rightarrow$ 0.496250

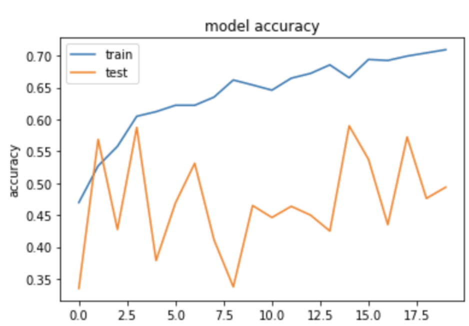
The structure behind AlexNet is too complex for this problem, indeed the dataset is too small to use so many layers and big kernels. The model does probably recognize objects in the pictures, instead of the general atmosphere.

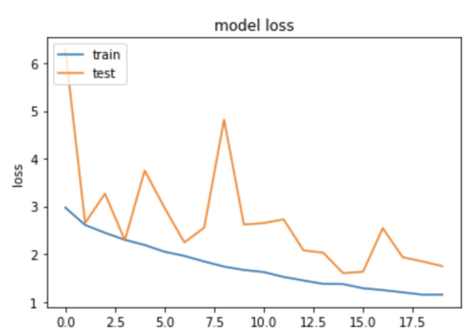
For instance, the presence of car could be associated to a particular class and fail all the other predictions.

```
[[  6  65 103  26]
 [  1 100  55  44]
 [  0  55 119  26]
 [  1  12  15 172]]

Confusion matrix analysis
True                 Predicted              errors   err %
-------------------------------------------------------------
HAZE                 -> SNOWY               103      12.88 %
HAZE                 -> RAINY               65       8.12 %
RAINY                -> SNOWY               55       6.88 %
SNOWY                -> RAINY               55       6.88 %
RAINY                -> SUNNY               44       5.50 %
HAZE                 -> SUNNY               26       3.25 %
SNOWY                -> SUNNY               26       3.25 %
SUNNY                -> SNOWY               15       1.88 %
SUNNY                -> RAINY               12       1.50 %
RAINY                -> HAZE                1        0.12 %
SUNNY                -> HAZE                1        0.12 %
```

Figure 10: Confusion Matrix and analysis





5.4 The final model (from scratch)

Even trying different combinations, based on different well known architectures, the first model, plus the addiction of the dropout to decrease the overfitting and 256 neurons, resulted to be the most efficient.

The test was initialized to run for 100 epochs, but even with the dropout, after a while it just started to overfit the model. While the accuracy on the validation set was remaining the same, the loss was decreasing in the training but increasing in the validation. To avoid this situation I decided to introduce the 'EarlyStopping' callback, monitoring the val_loss.

The model here uses the 224x224 pixels pictures, the augmentation performs only the horizontal flip and a 30 degrees rotation.

Here the results:

- **Loss** ⇒ 0.696015
- **Accuracy** ⇒ 0.753750

```
Loaded 800 test samples from 4 classes.
25/25 [==============================] - 6s 230ms/step
              precision    recall  f1-score   support

        HAZE      0.834     0.780     0.806       200
       RAINY      0.665     0.735     0.698       200
       SNOWY      0.708     0.680     0.694       200
       SUNNY      0.820     0.820     0.820       200

    accuracy                          0.754       800
   macro avg      0.757     0.754     0.755       800
weighted avg      0.757     0.754     0.755       800
```

Figure 12: Precision, Recall and F-score test final

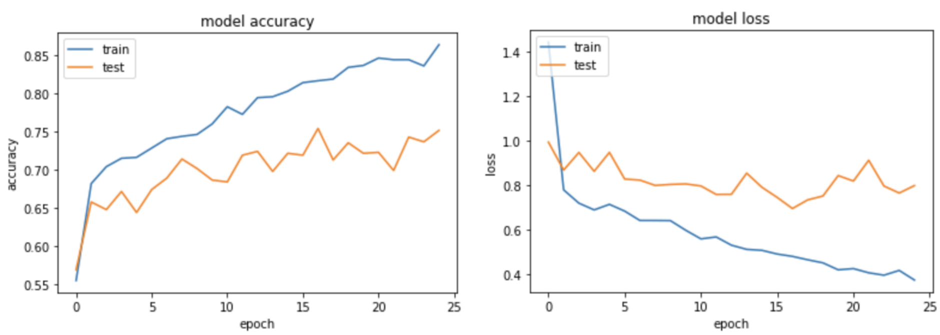
The network still finds hard to detect whether there is raining or snowing, but in general the performances are clearly better compared to the other models.

Trying to increase the metrics on this dataset, reduced the performances on the Smart-I dataset, showed in the next page.

```
[[156  18  11  15]
 [  9 147  34  10]
 [ 12  41 136  11]
 [ 10  15  11 164]]

Confusion matrix analysis
True                    Predicted          errors  err %
---------------------------------------------------------
SNOWY            ->   RAINY              41       5.12 %
RAINY            ->   SNOWY              34       4.25 %
HAZE             ->   RAINY              18       2.25 %
HAZE             ->   SUNNY              15       1.88 %
SUNNY            ->   RAINY              15       1.88 %
SNOWY            ->   HAZE               12       1.50 %
HAZE             ->   SNOWY              11       1.38 %
SNOWY            ->   SUNNY              11       1.38 %
SUNNY            ->   SNOWY              11       1.38 %
RAINY            ->   SUNNY              10       1.25 %
SUNNY            ->   HAZE               10       1.25 %
RAINY            ->   HAZE               9        1.12 %
```

Figure 13: Confusion Matrix and analysis



Here the results for the smart-I dataset:

- **Loss** $\Rightarrow$ 2.221850
- **Accuracy** $\Rightarrow$ 0.402261

The pictures provided by this dataset are very different from the pictures used for the training. Therefore, even trying to make the model more general as possible, it is very hard to have good results with so small training dataset.

```
TEST ON SMART I
Found 3038 images belonging to 4 classes.
95/95 [==============================] - 39s 405ms/step
              precision    recall  f1-score   support

        HAZE      0.000     0.000     0.000         0
       RAINY      0.230     0.645     0.339       521
       SNOWY      0.768     0.524     0.623      1421
       SUNNY      0.569     0.132     0.215      1096

    accuracy                          0.404      3038
   macro avg      0.392     0.325     0.294      3038
weighted avg      0.604     0.404     0.427      3038
```

```
[[   0    0    0    0]
 [129 336  30  26]
 [138 454 745  84]
 [ 82 674 195 145]]

Confusion matrix analysis
True                 Predicted            errors  err %
---------------------------------------------------------
SUNNY            ->  RAINY                674     22.19 %
SNOWY            ->  RAINY                454     14.94 %
SUNNY            ->  SNOWY                195     6.42 %
SNOWY            ->  HAZE                 138     4.54 %
RAINY            ->  HAZE                 129     4.25 %
SNOWY            ->  SUNNY                84      2.76 %
SUNNY            ->  HAZE                 82      2.70 %
RAINY            ->  SNOWY                30      0.99 %
RAINY            ->  SUNNY                26      0.86 %
```

6 Transfer Learning

In practice, very few people train an entire Convolutional Network from scratch (with random initialization), because it is relatively rare to have a dataset of sufficient size. Instead, it is common to pretrain a ConvNet on a very large dataset, and then use the ConvNet either as an initialization or a fixed feature extractor for the task of interest [8].

There are 2 main ways to use the transfer learning:

- *ConvNet as fixed feature extractor* ⇒ Take a pretrained ConvNet, remove the last fully-connected layer, then treat the rest of the ConvNet as a fixed feature extractor for the new dataset. This is the best option with small dataset, as in this case.

- *Fine-tuning the ConvNet* ⇒ Not only replace and retrain the classifier on top of the ConvNet on the new dataset, but also fine-tuning the weights of the pretrained network by continuing the backpropagation.

6.1 feature extractor

For these tests I have used the VGG16 pre-trained model with the ImageNet dataset, freezing the convolutional base to prevent weights from being updated during the training.

For the first attempt, I have just removed the pre-trained dense layers, then replaced them with only the new output layer.

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten_3 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_3 (Dense)              (None, 4)                 100356
=================================================================
Total params: 14,815,044
Trainable params: 100,356
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure 16: model transfer 1

- **Loss** ⇒ 0.280378

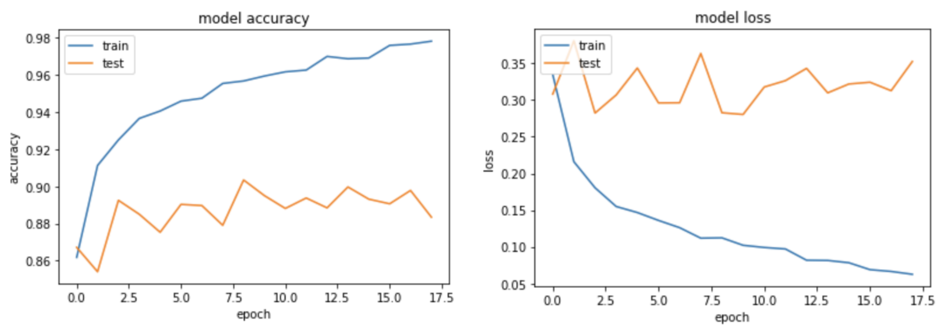
- **Accuracy** ⇒ 0.895000

```
Loaded 800 test samples from 4 classes.
25/25 [==============================] - 3s 122ms/step
              precision    recall  f1-score   support

        HAZE      0.751     0.890     0.815       200
       RAINY      0.779     0.810     0.794       200
       SNOWY      0.781     0.730     0.755       200
       SUNNY      0.857     0.720     0.783       200

    accuracy                          0.787       800
   macro avg      0.792     0.788     0.786       800
weighted avg      0.792     0.787     0.786       800
```

Figure 17: Precision, Recall and F-score transfer 1



The results are incredible, it does not have much problems anymore to distinguish between rain and snow.

The light conditions still makes recognizing the sunny pictures easier, but all the performance metrics in general are increased.

However, the graphs show that is still possible to improve to model to reduce the overfitting.

```
[[178   7   4  11]
 [  9 162  20   9]
 [ 21  29 146   4]
 [ 29  10  17 144]]

Confusion matrix analysis
True                 Predicted            errors  err %
------------------------------------------------------------
SNOWY            ->  RAINY                29      3.62 %
SUNNY            ->  HAZE                 29      3.62 %
SNOWY            ->  HAZE                 21      2.62 %
RAINY            ->  SNOWY                20      2.50 %
SUNNY            ->  SNOWY                17      2.12 %
HAZE             ->  SUNNY                11      1.38 %
SUNNY            ->  RAINY                10      1.25 %
RAINY            ->  HAZE                 9       1.12 %
RAINY            ->  SUNNY                9       1.12 %
HAZE             ->  RAINY                7       0.88 %
HAZE             ->  SNOWY                4       0.50 %
SNOWY            ->  SUNNY                4       0.50 %
```

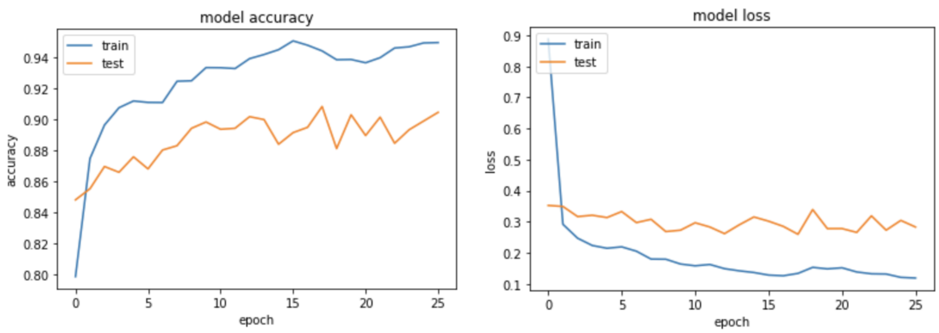
Figure 18: Confusion Matrix and analysis

For the second test, I have reused the simple dense layers structure of the final test (on scratch).

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 256)               6422784
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 1028
=================================================================
Total params: 21,138,500
Trainable params: 6,423,812
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure 20: model transfer 2

- **Loss** $\Rightarrow$ 0.259292
- **Accuracy** $\Rightarrow$ 0.908437



Loaded 800 test samples from 4 classes.
25/25 [==============================] - 12s 488ms/step

	precision	recall	f1-score	support
HAZE	0.819	0.885	0.851	200
RAINY	0.790	0.830	0.810	200
SNOWY	0.754	0.780	0.767	200
SUNNY	0.886	0.740	0.807	200
accuracy			0.809	800
macro avg	0.812	0.809	0.808	800
weighted avg	0.812	0.809	0.808	800

Figure 22: Precision, Recall and F-score transfer 2

The results are quite similar, but again this test shows that could still be possible to make some improvements.

The results on the smart-I dataset are quite similar too, specifically for this model:

- **Loss** $\Rightarrow$ 0.946717
- **Accuracy** $\Rightarrow$ 0.732547

```
[[177   7   5  11]
 [  7 166  19   8]
 [ 12  32 156   0]
 [ 20   5  27 148]]

Confusion matrix analysis
True                 Predicted            errors  err %
--------------------------------------------------------
SNOWY           ->   RAINY                32      4.00 %
SUNNY           ->   SNOWY                27      3.38 %
SUNNY           ->   HAZE                 20      2.50 %
RAINY           ->   SNOWY                19      2.38 %
SNOWY           ->   HAZE                 12      1.50 %
HAZE            ->   SUNNY                11      1.38 %
RAINY           ->   SUNNY                8       1.00 %
HAZE            ->   RAINY                7       0.88 %
RAINY           ->   HAZE                 7       0.88 %
HAZE            ->   SNOWY                5       0.62 %
SUNNY           ->   RAINY                5       0.62 %
```

Figure 23: Confusion Matrix and analysis

TEST ON SMART I
Found 3038 images belonging to 4 classes.
95/95 [==============================] - 70s 735ms/step

	precision	recall	f1-score	support
HAZE	0.000	0.000	0.000	0
RAINY	0.301	0.702	0.422	521
SNOWY	0.634	0.600	0.616	1421
SUNNY	0.811	0.137	0.234	1096
accuracy			0.450	3038
macro avg	0.436	0.360	0.318	3038
weighted avg	0.641	0.450	0.445	3038

```
[[   0    0    0    0]
 [  68 366   84    3]
 [ 173 364 852   32]
 [  53 485 408 150]]

Confusion matrix analysis
True                 Predicted            errors  err %
--------------------------------------------------------
SUNNY           ->   RAINY                485     15.96 %
SUNNY           ->   SNOWY                408     13.43 %
SNOWY           ->   RAINY                364     11.98 %
SNOWY           ->   HAZE                 173     5.69 %
RAINY           ->   SNOWY                84      2.76 %
RAINY           ->   HAZE                 68      2.24 %
SUNNY           ->   HAZE                 53      1.74 %
SNOWY           ->   SUNNY                32      1.05 %
RAINY           ->   SUNNY                3       0.10 %
```

Note: using Xception already trained with a set of images more suitable for this problem, the performance metrics could probably increase even more.

The fine-tuning operation requires to unfreeze some of the layers of the CNN, to make possible the partial retraining. In this case we don't have enough big dataset and this operation does actually just decrease the current performance metrics.

References

[1] Course slides

[2] Machine Learning, Tom Mitchell, McGraw Hill, 1997.

[3] Pattern Recognition and Machine Learning, Christopher M. Bishop, 2006

[4] https://scikit-learn.org/stable/index.htmll

[5] https://en.wikipedia.org/wiki/TensorFlow

[6] https://www.tensorflow.org/

[7] https://keras.io

[8] http://cs231n.github.io/transfer-learning/

[9] https://www.tensorflow.org/tutorials/images/transfer_learning

[10] https://arxiv.org/pdf/1409.1556.pdf