Machine Learning for Multi-Class Weather Classification

Engineering on Computer Science

Daniele Cantisani 1707633

December 2019

Contents

1 Introduction

The objective of the report is to describe the methods, tests and results in using Machine Learning for Multi-Class Weather Classification. Two different techniques will be used, Convolutional Neural Network (CNN) and Transfer Learning, to achieve good accuracy, and we will try to compare methods by describing their use in detail.

2 Image Multi-Weather Classification

The "Image Classification" problem, is the task of assigning an input image one label from a fixed set of categories.
Now, an image can be represented in a greyscale in which each pixel represents the brightness of a pixel. In this case the image is seen from the computer as a 2D matrix where each pixel (a matrix cell) may be a number between 0 (white) and 255 (black).
The other way to represent an image is when this is colored. So we have a 3D matrix where each dimension represents one channel of the image: the channels are the three primary color Red, Green and Blue (RGB).
Now we have the following challenges:

- **Viewpoint variation** A single instance of an object can be oriented in many ways with respect to the camera.
- **Scale variation** Visual classes often exhibit variation in their size (size in the real world, not only in terms of their extent in the image).
- **Deformation** Many objects of interest are not rigid bodies and can be deformed in extreme ways.
- **Occlusion** The objects of interest can be occluded. Sometimes only a small portion of an object (as little as few pixels) could be visible.
- **Illumination conditions** The effects of illumination are drastic on the pixel level.
- **Background clutter** The objects of interest may blend into their environment, making them hard to identify.
- **Intra-class variation** The classes of interest can often be relatively broad, such as chair. There are many different types of these objects, each with their own appearance.

So, the task in Image Classification is to take a matrix of pixels that represents a single image and assign a label to it.
Now, the main steps to perform can be formalized as follows:

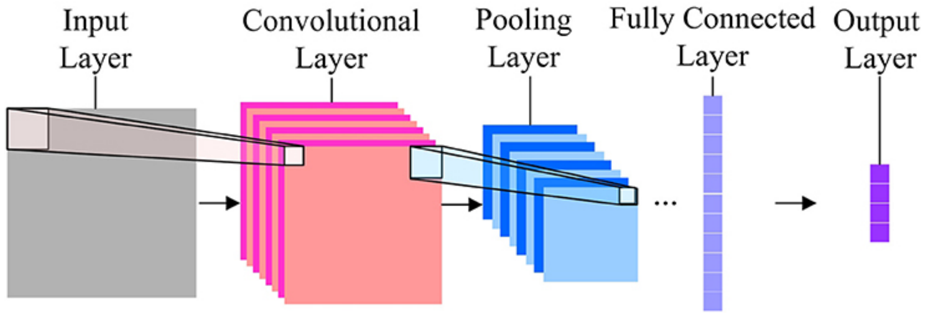
- **Input**: take as input a dataset of N images, each labeled with one of K different classes. This is the training set.

- **Learning**: use the training set to learn what every one of the classes looks like. In this way we train the classifier.
- **Evaluation**: in the end, we evaluate the quality of the classifier by asking it to predict labels for a new set of images (testing set). We will then compare the true labels of these images to the ones predicted by the classifier.

To achieve our goal we can use the so called Convolutional Neural Networks.

3 Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) is a class of deep neural networks. CNNs are applied too much to analyze visual patterns.
CNNs use a variation of multilayer perceptrons designed to require minimal preprocessing. A CNN algorithm, spite of other image classification algorithms, use little image pre-processing and in this way the network learns the filters indipendently from prior knowledge and human effort in feature design and takes advantage from this.
Neural Networks receive an input, and transform it through a series of hidden layers. Each hidden layer is made up of a set of neurons, where neurons in a single layer function are completely independently and do not share any connections. The last fully-connected layer is called the "output layer" and in classification settings it represents the class scores.



Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way: the layers of a CNN have neurons arranged in the 3 known dimensions: width, height, depth.
This CNN arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a CNN transforms the 3D input volume to a 3D output volume of neuron activations. As we said before, for the 3D input we have 3 channels: red, green, blue. Let's now analyze in a specific way how the convolutional layer works: there are three most important stages.

3.1 Convolution Stage

The Convolutional layer is the core building block of a Convolutional Network that does most of the computational work.
Let's first discuss what the convolution stage computes. It takes as parameters a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. During the next pass, it convolves each filter across the width and height of the input volume and compute products between the entries of the filter and the input at any position. We will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position. Now, we will have an entire set of filters in each convolutional layer, and each of them will produce a separate 2-dimensional activation map. We will stack these activation maps along the depth dimension and produce the output volume.
Three hyperparameters control the size of the output volume: the depth, stride and zero-padding.

- First, the depth of the output volume is a hyperparameter: it corresponds to the number of filters we would like to use, each learning to look for something different in the input.
- Second, we must specify the stride with which we slide the filter. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2 then the filters jump 2 pixels at a time as we slide them around. This will produce smaller output volumes spatially.
- Sometimes it will be convenient to pad the input volume with zeros around the border. The size of this zero-padding is a hyperparameter. The nice feature of zero padding is that it will allow us to control the spatial size of the output volumes.

In this stage, we follow the "parameters sharing" rules, that is: learn only one set of paraters shared to all the units. In this way we can control the number of the parameters.

3.2 Detector Stage

In this stage some non-linear functions will be applied on the input images. We apply the rectifier function to increase non-linearity in the CNN.
Indeed images are made of different objects that are not linear to each other. Without applying this function the image classification will be treated as a linear problem while it is actually a non-linear one.
A rectifier function is an activation function which takes the positive part of its argument x, the input of a neuron.
A unit that uses the rectifier is also called a rectified linear unit or **ReLU**.

3.3 Pooling Stage

The pooling stage down-samples the previous layers feature map.
Pooling layer follows a sequence of one or more convolutional layers and are intended to consolidate the features learned and expressed in the previous layers feature map. As such, pooling may be considered a technique to compress or generalize feature representations and generally reduce the overfitting of the training data by the model.
They too have a receptive field, often much smaller than the convolutional layer. Also, the stride or number of inputs that the receptive field is moved for each activation is often equal to the size of the receptive field to avoid any overlap. Pooling layers are often very simple, taking the average or the maximum of the input value in order to create its own feature map.

4 Transfer Learning

Transfer learning is a machine learning technique where a model trained on one task is re-purposed on a second related task. Transfer learning is related to problems such as multi-task learning and concept drift and is not exclusively an area of study for deep learning. Nevertheless, transfer learning is popular in deep learning given the enormous resources required to train deep learning models or the large and challenging datasets on which deep learning models are trained. There are two methods to use Transfer Learning:

- Develop Model Approach

We must select a related predictive modeling problem with an abundance of data where there is some relationship in the input data, output data, and/or concepts learned during the mapping from input to output data. Next, we must develop a skillful model for the first task. The model must be better than a naive model to ensure that some feature learning has been performed. The model fit on the source task can then be used as the starting point for a model on the second task of interest. This may involve using all or parts of the model, depending on the modeling technique used. Optionally, the model may need to be adapted or refined on the input-output pair data available for the task of interest.

- Pre-trained Model Approach

A pre-trained source model is chosen from available models. The model pre-trained model can then be used as the starting point for a model on the second task of interest. This may involve using all or parts of the model, depending on the modeling technique used. Optionally, the model may need to be adapted or refined on the input-output pair data available for the task of interest.
We will use this second model approach.

5 DataSet

The dataset provided is a set of folders full of images of the four classes to be identified (HAZE, RAINY, SNOWY and SUNNY). During all the following tests all the folder images were used as training_set except for MWI_DATASET_2.1 which was used for the test_set.

6 Models

During our tests, we tested two different applications, AlexNet and Resnet50. After that, we switched to an already trained application using the Transfer Learning technique.

6.1 AlexNet Model

The first application, AlexNet, is the very first work that popularized CNN in computer vision. AlexNet contains eight layers: the first five are convolutional layers, some of them followed by max-pooling layers, and the last three are fully connected layers. Here its layers:

1. **First Layer**: the input for AlexNet is a 227x227x3 RGB image which passes through the first convolutional layer with 96 feature maps or filters having size 11x11 and a stride of 4. The image dimensions changes to 55x55x96.
   Then the AlexNet applies maximum pooling layer or sub-sampling layer with a filter size 3x3 and a stride of two. The resulting image dimensions will be reduced to 27x27x96.

2. **Second Layer**: next, there is a second convolutional layer with 256 feature maps having size 5x5 and a stride of 1.
   Then there is again a maximum pooling layer with filter size 3x3 and a stride of 2. This layer is same as the second layer except it has 256 feature maps so the output will be reduced to 13x13x256.

3. **Third, Fourth, Fifth Layers**: the third, fourth and fifth layers are convolutional layers with filter size 3x3 and a stride of one. The first two used 384 feature maps where the third used 256 filters. The three convolutional layers are followed by a maximum pooling layer with filter size 3x3, a stride of 2 and have 256 feature maps.

4. **Sixth Layer**: the convolutional layer output is flatten through a fully connected layer with 9216 feature maps each of size 1x1.

5. **Seventh and Eighth Layers**: next is again two fully connected layers with 4096 units.

6. **Output Layer**: finally, there is a softmax output layer $\hat{y}$ with 1000 possible values.

6.2 ResNet50 Model

The Residual Neural Network is the network we rely most on. It's based on the thought of going deeper into training. However, this could lead to problems that the network is able to avoid by implementing a skip option that allows it not to block itself to an unsatisfactory result. In fact, by combining the input of the underlying layer you can avoid stagnation and perform satisfactorily. ResNets can easily gain accuracy from greatly increased depth, producing results which are better than previous networks.
ResNet is inspired by VGG net but is simpler. To fully understand it, it is best to compare it with a plain network.
A plain network has convolutional layers that have 3x3 filters and follow two rules: the first is that for the same output feature map, the layers have the same number of filters. The second is that if the size of the features map is halved, the number of filters is doubled to preserve the time complexity of each layer.
For the residual network, using the same structure, a shortcut connection is inserted. The shortcut performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no additional parameter. The identity shortcuts can be directly used when the input and output are of the same dimensions.
His implementation is this: the image is resized with its shorter side randomly sampled in [256,480] for scale augmentation. A 224x224 crop is randomly sampled from an image or its horizontal flip, with the per-pixel mean subtracted.

6.3 Transfer Learning

To implement the Transfer Learning we rely on Keras that provides access to a series of pre-formed models of the highest level that have been developed for the activities of image recognition and classification. We use the VGG-16 available in the API. Keras will download the weights of the required models. The last layers of the VGG16 model are fully connected before the output layer.
The building of a train model is a 3 step process:

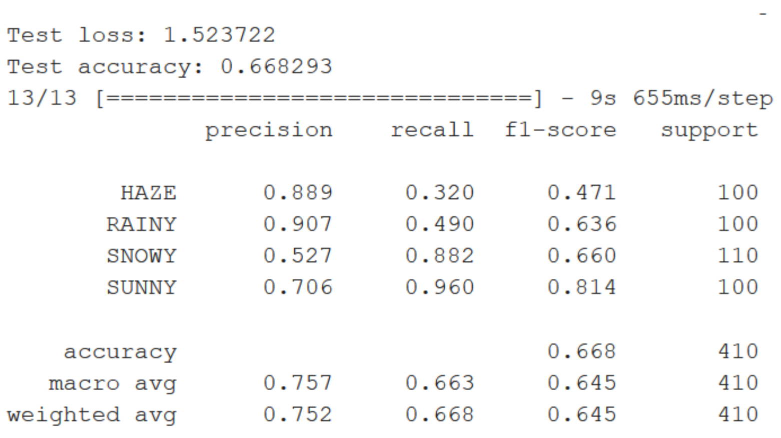
- Importing the pre-trained model and adding the dense layers.
- Loading train data into ImageDataGenerators.
- Training and Evaluating model.

ImageDataGenerators are inbuilt in keras and help us to train our model. We just have to specify the path to our training data and it automatically sends the data for training, in batches.

7 Evaluation

After creating models with different algorithms and predicting a previously defined test size, we can analyze the results. The classification-report() function (parameters has both the solutions and the predictions), prints to us the values of Accuracy, Precision, Recall and F1-Score for each element to be predicted.
Accuracy is the most intuitive performance measure and it is a ratio of correctly predicted observation to the total observations. Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. Recall is the ratio of correctly predicted positive observations to the all observations in actual class. Finally F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into consideration. Furthermore, in the case there is a big difference between the values of Recall and Precision, so it shows that it is not a good implementation for the problem.

7.1 AlexNet Model



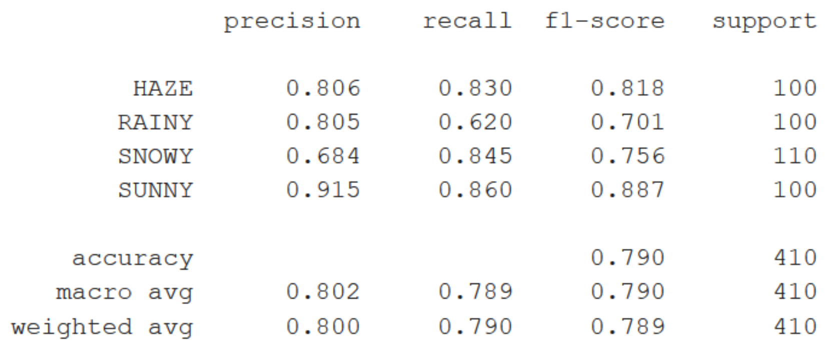
```
Test loss: 1.523722
Test accuracy: 0.668293
13/13 [==============================] - 9s 655ms/step
              precision    recall  f1-score   support

        HAZE      0.889     0.320     0.471       100
       RAINY      0.907     0.490     0.636       100
       SNOWY      0.527     0.882     0.660       110
       SUNNY      0.706     0.960     0.814       100

    accuracy                          0.668       410
   macro avg      0.757     0.663     0.645       410
weighted avg      0.752     0.668     0.645       410
```

Figure 1:

As we can see using the Alex Net model we reach an Accuracy of 0.66. The values Precision and Recall are very far away showing that the validity of the test is not satisfactory at all. In general it is not terrible but we are not completely satisfied and we expect from the next model a higher Accuracy.

7.2 ResNet50 Model

With the Resnet50 model, we can see a lot of improvements. Accuracy of 0.79 and very close values such as Precision, Recall and F1-Score.



```
              precision    recall  f1-score   support

        HAZE      0.806     0.830     0.818       100
       RAINY      0.805     0.620     0.701       100
       SNOWY      0.684     0.845     0.756       110
       SUNNY      0.915     0.860     0.887       100

    accuracy                          0.790       410
   macro avg      0.802     0.789     0.790       410
weighted avg      0.800     0.790     0.789       410
```

Figure 2: Classification Report for Resnet50

Another important measurement to see if a model works correctly is the confusion matrix.
A confusion matrix is a table that is used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. The confusion matrix shows the ways in which our classification model is confused when it makes predictions. It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made. The confusion matrix is an excellent method to confirm the values of Precision, Recall and F1- Score. We can read the confusion matrix in this way. The columns are the predictions and the rows must therefore be the actual values. The main diagonal gives the correct predictions. That is, the cases where the actual values and the model predictions are the same.

```
True                  Predicted          errors  err %
---------------------------------------------------------
RAINY             ->  SNOWY              30      7.32 %
SNOWY             ->  HAZE               9       2.20 %
HAZE              ->  SNOWY              8       1.95 %
SNOWY             ->  RAINY              7       1.71 %
SUNNY             ->  HAZE               6       1.46 %
HAZE              ->  RAINY              5       1.22 %
RAINY             ->  HAZE               5       1.22 %
SUNNY             ->  SNOWY              5       1.22 %
HAZE              ->  SUNNY              4       0.98 %
RAINY             ->  SUNNY              3       0.73 %
SUNNY             ->  RAINY              3       0.73 %
SNOWY             ->  SUNNY              1       0.24 %
```

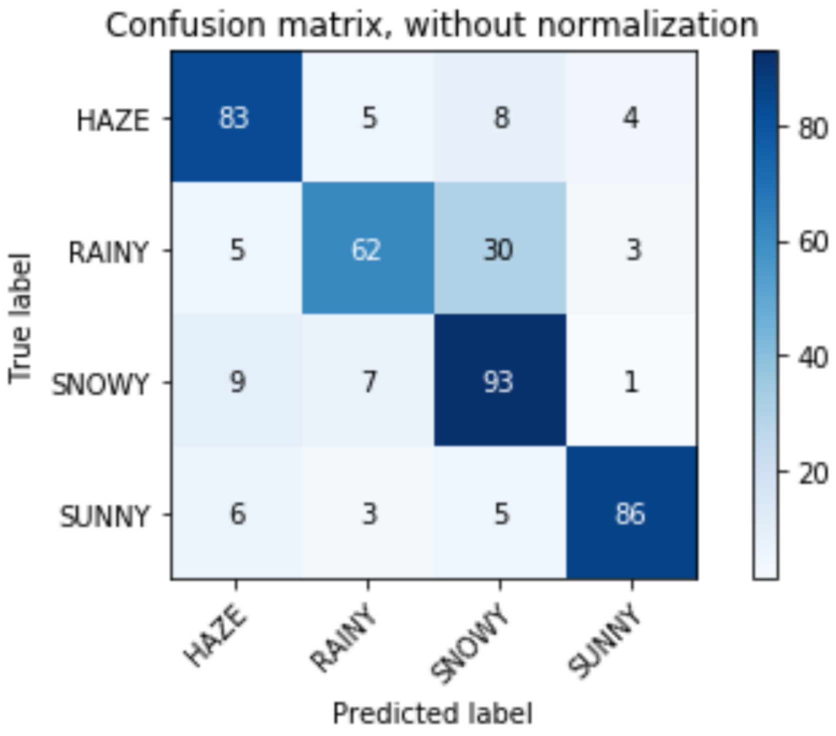
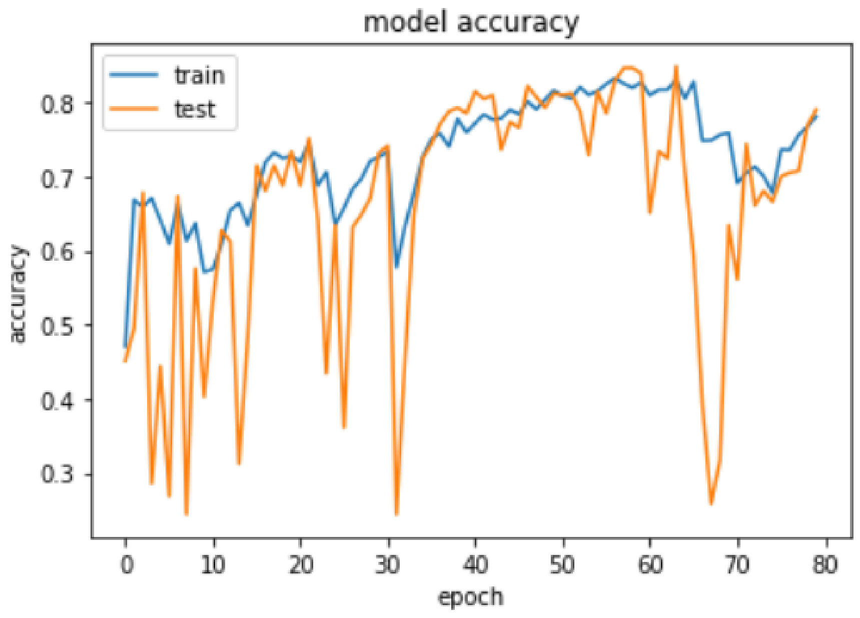


Figure 3: Confusion Matrix ResNet50

In the analysis of the results of ResNet50 it is very interesting to note that in the 8% of the images taken as test_set, the same error occurred. The model confuses the rain thinking that it is snow. On a test_set of 400 images there are 30 such errors. In front of such an analysis an excellent enhancement of the model would be to go to the deep search for the differences between rain and snow and train the model from that point of view.

These results were obtained after a training of 80 epochs as shown in the next graph showing the evolution of Accuracy and Loss over time.



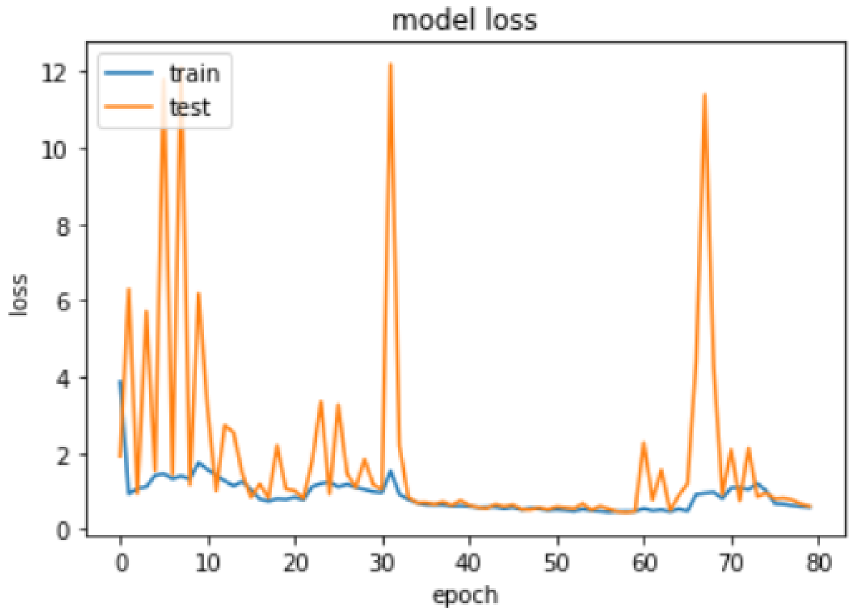


Figure 4: Model Accuracy and Loss ResNet50

,

7.3 Transfer Learning

Also for the test of the Transfer Learning model we have chosen to have 80 epochs.
As expected, the pre-driven Transfer Learning model has a better Accuracy than the CNNs models. Accuracy of 0.84 is the best found during our testing. In some cases (Sunny-Rainy and Sunny-Snowy) we can even have zero errors.

```
Test loss: 0.441832
Test accuracy: 0.858537
13/13 [==============================] - 6s 472ms/step
              precision    recall  f1-score   support

        HAZE      0.931     0.810     0.866       100
       RAINY      0.871     0.880     0.876       100
       SNOWY      0.840     0.764     0.800       110
       SUNNY      0.811     0.990     0.892       100

    accuracy                          0.859       410
   macro avg      0.863     0.861     0.858       410
weighted avg      0.863     0.859     0.857       410
```

Figure 5: Classification Transfer Learning

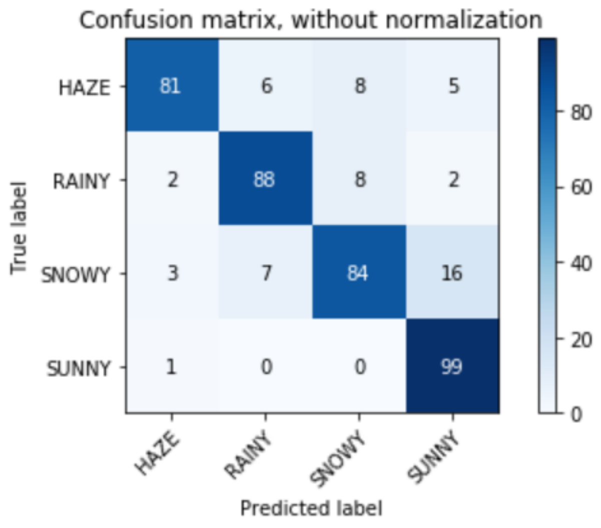


Figure 6: Confusion Matrix Transfer Learning