Report Homework 2 - Machine Learning

Sveva Pepe

Chapter 1

Project Overview

1.1 Project Statement

The goal is to classify images in two different ways:

1. Define a CNN and train it from scratch
2. Apply transfer learning and fine tuning from a pre-trained model, this means using some models trained by someone else and checking if that model produces good accuracy even on the dataset we are going to use.

The d ataset supplied to us presents 4 classes: HAZE, RAINY, SNOWY AND SUNNY.
Our model can be evaluated: with MWI dataset that we consider or with a test set provided, Weather_dataset.
In our case, both ways of evaluating the model were considered. Both with its own CNN and with the pre-trained model the first evaluation was made taking into consideration the dataset of 2000 images, which was split into 20% test and 80% train.
the second evaluation was instead made assuming to have as train set the whole dataset of 2000 images and as test set the Weather_dataset. When the Weather Dataset was used as a test set, an extra empty "haze" folder had to be added, so that both the train set and the test set had the same number of classes.

1.2 Load Data

The first thing done was to use a keras library, *ImageDataGenerator* to take the images from the dataset and preprocess them.
In particular, this library of Keras provides the ability to use data augmentation automatically when training a model.
First, the class may be instantiated and set up for the configuration of data augmentation specified by the arguments to the class constructor.

1.2.1 Evaluation with MWI

In particular we used a parameter, *validation_split*, to divide our dataset of 2000 images into 80% train and 20% test.
Then we go to use the *flow_from_directory* function to take the dataset we want to consider, in particular here we specify the **target_size**, the dimensions to which all images will be resized, in this case was choosen (118,124), and the **batch_size**, which corresponds to the size of the batches of data, which for a matter of performance has been set to 64. We apply this function to create in one case a train_generator and in onther case validation_generator.
When we use *flow_from_directory* function in the train_generator we would have a parameter that identifies that that is just the train generator: **subset = 'training'**; instead in the validation_generator we would have the same parameter but this time set to **'validation'**.

Furthermore we would also have the shuffle parameter which will be different, in the train_generator it will be equal to true instead in the validation_generator it will be equal to false, this because in the validation_test we want to sort the data in alphanumeric order.
In the end I get my train set of 1604 images and the test set of 401 images.

1.2.2 Evaluation with Weather_Dataset

Instead, in the second evaluation no "validation_split" was applied since the entire MWI dataset is considered as a train set.
In this case as in the previous one we have the function flow_from_directory to always create the train_generator and the validation_generator. In this case we have that the train_generator and validation_generator directory is no longer the same because we are not splitting the dataset. No, in this case the train_generator takes as a directory the MWI of 2000 images and the validation_generator takes the directory of the Weather_dataset.
We obtain that in the train set we have 2005 images and in the test set 2887 images.

1.2.3 Metrics

The metrics that we used for the classifications are the accuracy and the loss function.
The chosen loss function is the **categorical crossentropy**, alogarithmic function that increases when the prediction is far from the label,so it must be kept as lower as possible.

Chapter 2

Define CNN And Train It From Scratch For Image Classification

2.1 Model

A CNN is composed of a hierarchy of levels. The input level is directly connected to the pixels of the image (input_shape), the last levels are generally fully connected, while in the intermediate levels local connections and shared weights are used.
A CNN-type neural network is created so that the various layers will be Convolutional. The most important thing is that in the first convolutional there is input shape which is in the size of the images considered plus the factor 3 because color is 'rgb'. For example in our case (118,124,3) is the input shape.
In the convolutional layer it was decided to use the ReLU activation function which flattens the response to all negative values to zero, while leaving everything unchanged for values equal to or greater than zero.
This simplicity, combined with the fact of drastically reducing the problem of vanishing gradient, so it is used here,in intermediate layers, where the quantity of steps and calculations is important. Calculating the derivative is in fact very simple: for all negative values it is equal to zero, while for the positive ones it is equal to 1. In the angled point in the origin the derivative is indefinite but is still set to zero by convention.
Moreover after a convolutional layer it is convenient to insert a pooling to decrease the size of the matrix, because otherwise you would go to have a huge amount of data that you would not know how to manage.
With MaxPooling we are going to reduce the size of the matrix, for example, after the second convolutional layer of our neural network we considered a MaxPooling of size (2x2) and the idea is that if we found something interesting in one of the four entrance cards that make up each square of the 2x2 grid, we'll just keep the bit more interesting. This reduces the size of a matrix while keeping the most important bits.
In our case a MaxPooling is inserted after the second convolutional layer because for reasons related to performance, decreasing the size of the output of the convolutional and removing a MaxPooling the performance of our network increased in terms of accucy and loss function.
The matrix of the last MaxPooling, which in my case is the third, is fed to another neural network, Fully Connected Neural Network, to which we can add other dense layers followed also by dropuot (these are set to prevent overfitting) so as to make the network as general as possible.
The important thing is that the last dense layer must have exactly the number of classes of your dataset as the unit number and that the activation function must be softmax because in this case we have a multiclass problem.
As optimizer rmsprop was chosen with a learning rate of 0.0001 as after a series of tests it was the one that provided the best performance.
The model ends when the compile function is executed, in which it goes to configure the model which will then go to make the fit. As already anticipated the categorical_crossentropy is used as

a loss and the metric on which the model will be evaluated is the accuracy.

```
Model: "SvevaNet"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 114, 220, 15)      1140
_________________________________________________________________
activation_6 (Activation)    (None, 114, 220, 15)      0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 110, 216, 20)      7520
_________________________________________________________________
activation_7 (Activation)    (None, 110, 216, 20)      0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 55, 108, 20)       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 53, 106, 30)       5430
_________________________________________________________________
activation_8 (Activation)    (None, 53, 106, 30)       0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 26, 53, 30)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 41340)             0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               5291648
_________________________________________________________________
activation_9 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 96)                12384
_________________________________________________________________
activation_10 (Activation)   (None, 96)                0
_________________________________________________________________
dropout_3 (Dropout)          (None, 96)                0
_________________________________________________________________
dense_5 (Dense)              (None, 4)                 388
_________________________________________________________________
activation_11 (Activation)   (None, 4)                 0
=================================================================
Total params: 5,318,510
Trainable params: 5,318,510
Non-trainable params: 0
```

Figure 2.1: SvevaNet

2.2 Fit

As for the fit, the function fit_generator is used, which trains the model on data generated batch-by-batch.
It takes as input various parameters including the **train set**, the number of *epochs* (ie the number of iterations), *steps_per_epochs* (total number of steps to yield from generator before declaring one epoch finished and starting the next epoch. It should typically be equal to train_generator.n // train_generator.batch_size, where n are the number of samples).
We have added 2 other parameters which are the *validation_data* and the *validation_steps*.
Validation_data corresponds to the validation_generator that we found previously, instead the validation_steps corresponds to total number of steps to yield from validation_data generator before stopping at the end of every epoch (ie validation_generator.n // validation_generator.batch_size +1).
These two parameters have been added in such a way as to be able to see precisely the behavior of the model being applied on the train comparing it with the test.

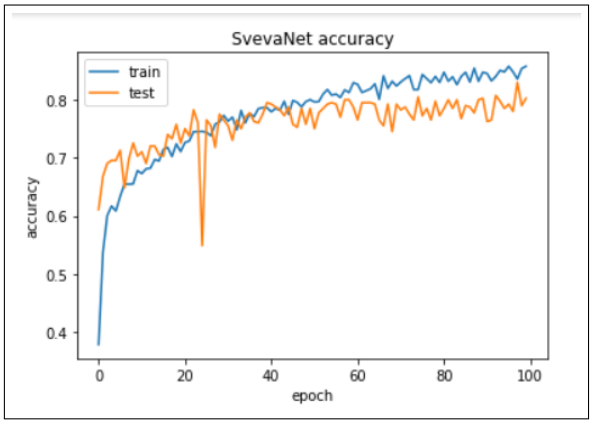
It means that I can observe from the accuracy values, which are generated at each epoch, whether or not the overfitting problem arises. We have overfitting when the difference between train and test accuracy is very high.
In our case, the number of epochs taken into consideration is 100.

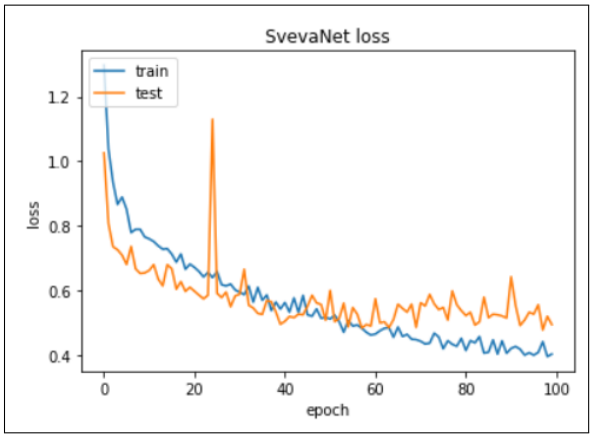
2.3 Evaluation

The evaluation of the model was done using the classification_report in which the values of precision, recall, F-score and accuracy are reported.
Furthermore the behavior of the model with the train is better analyzed and with the test it comes through the plt library. Also to better observe the presence of overfitting.

2.3.1 Evaluation with MWI dataset

	precision	recall	f1-score	support
HAZE	0.879	0.861	0.870	101
RAINY	0.689	0.840	0.757	100
SNOWY	0.760	0.570	0.651	100
SUNNY	0.924	0.970	0.946	100
accuracy			0.810	401
macro avg	0.813	0.810	0.806	401
weighted avg	0.813	0.810	0.806	401

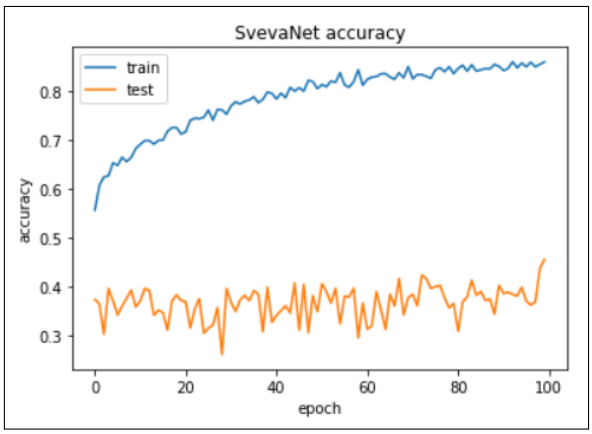


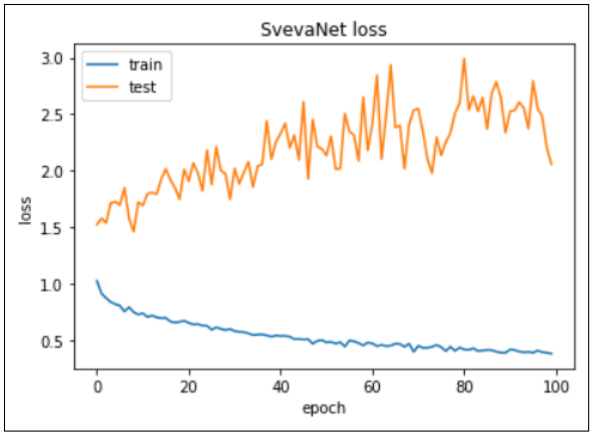


How can we observe the accuracy is not bad, 0.810. The model is quite good, but from how we can see from the plots of 100 epoches we have a slight overfitting.
This makes us understand that the model is not perfect but that in this case, seeing that the trend of both accuracy and loss is more or less stable towards the last half of the ages, we can still consider this model good.
What is not very nice is the fact that at around 25 epochs we have a peak of min in the accuracy and max in the loss but it is a single case, then for the rest both the train and the test have a rather similar trend , not perfect but not so bad.

2.3.2 Evaluation with Weather_dataset

	precision	recall	f1-score	support
HAZE	0.000	0.000	0.000	0
RAINY	0.197	0.446	0.273	350
SNOWY	0.731	0.716	0.723	1429
SUNNY	0.665	0.116	0.198	1108
accuracy			0.453	2887
macro avg	0.398	0.320	0.299	2887
weighted avg	0.641	0.453	0.467	2887





In this case we can observe how our model does not produce such good performances, but this does not depend only on our model but also on the dataset. Which contains noisy images, that is the images are difficult to classify as they are taken by cam and are not well understood. This greatly increases the difficulty in finding a neural network that can fit this dataset.
After all our model is not bad but it will certainly never give a rather high level of security, so we conclude that in this case there is the presence of overffing due mainly to the presence of noisy images in the weather dataset.
We note that ad HAZE we have the whole line zero, this because the weather test that was given to us did not contain the folder having haze, so to try not to have problems with number of classes an empty folder called 'haze' was added in the dataset , as said previously.

Chapter 3

Apply Transfer Learning and fine tuning from a pre-trained model for images classification.

We need to use transfer learning, that is, transfer learning refers to a process where a model is trained in one way or another, so we would use a pre-trained model from someone else on our train to evaluate our test set.
In this case we are going to take a pre-trained model from someone else, adding in the case of dropout and dense layers, concluding with the usual last layer of 4 units (which are precisely the 4 classes) with the softmax activation function.
The ResNet101V1 model, which contains 377 layers, it was chosen because the performances found were better than the other models that keras provided.
Furthermore we must also apply fine tuning, ie we are not training the entire network but only a small part.
The thing we're going to do is freeze some layers, for example we decided to freeze the first 376 layers so as to have only the last one not freezed, so the last one is the only trainable.
Other layers have been added to the model to try to make the model better for this type of problem.
In addition to the addition of 3 dense layers, 2 + 1 output, it was considered right to consider putting a two Droupout to prevent the model from having an overfitting, or trying to decrease its presence.
Furthermore, BatchNormalization layers were added as they greatly improved the model's performance compared to the datasets provided.

3.1 Fit

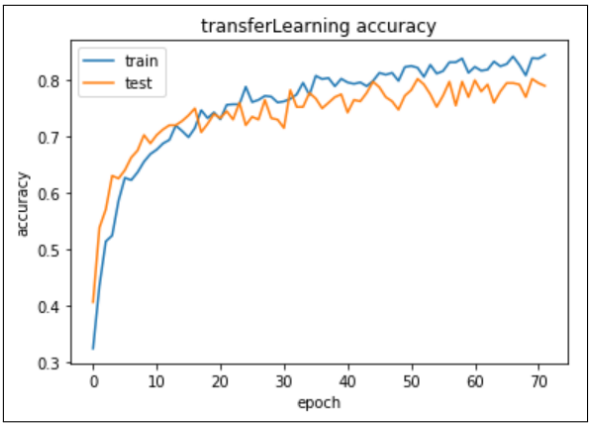
The fit is similar to the one made in the previous classification, as the fit_generator function is used on the train_generator using the parameters seen previously.
The only difference is that in this case it was considered to add a callback function as a parameter of the fit.
The callback function is useful as with it you avoid making many more iterations that don't go to improve the performance of the model.
This callback takes 2 parameters as input: *"monitor"* and *"patience"*.
Monitor is the parameter that deals with the quantity to be monitored, in this case we have chosen monitor = val_acc, ie the accuracy of the test.
Patience consists of the number of epochs that produced the monitored quantity with no improvement after which training will be stopped, in our case 20 have been considered since we make a fit on 100 epochs.

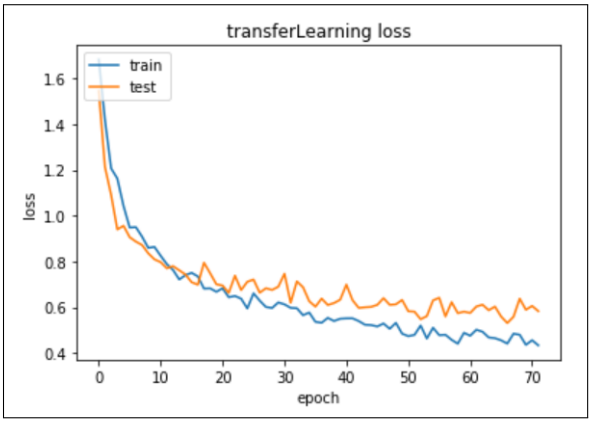
3.2 Evaluation

The evaluation as in the previous problem is analyzed by looking at the function of the classification report and the results shown by the accuracy and loss plot in order to analyze the overfitting if it is present.

3.2.1 Evaluation with MWI dataset

	precision	recall	f1-score	support
HAZE	0.780	0.842	0.810	101
RAINY	0.800	0.720	0.758	100
SNOWY	0.726	0.610	0.663	100
SUNNY	0.805	0.950	0.872	100
accuracy			0.781	401
macro avg	0.778	0.780	0.776	401
weighted avg	0.778	0.781	0.776	401

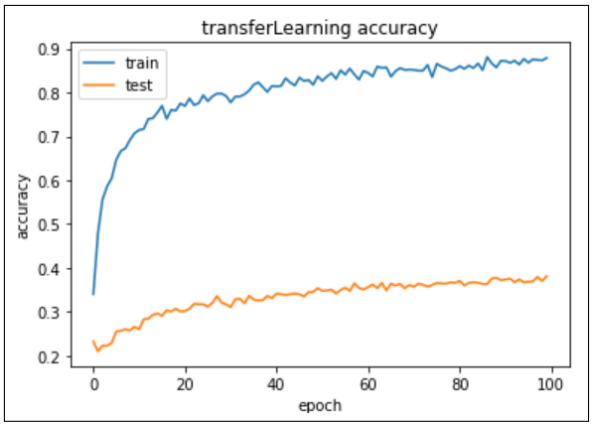


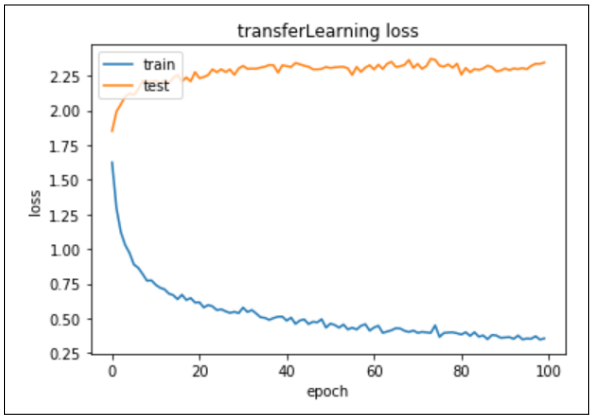


The model turns out to be quite good, certainly we can immediately notice that it does a slight overfitting but the performance of the train and the test is very similar and this bodes well that it is very good for this type of problem and behaves very positively with this dataset.
Other models had been tried among which DenseNet and Xception but both did not allow to reach a rather high level of accuracy without generating a remarkable overfitting.
We note how with the callback function we did not do all the 100 epochs we had done in the previous problem but only 72, this because after 20 consecutive epochs the test set had no improvements and therefore the execution was terminated.

3.2.2 Evaluation with Weather_dataset

	precision	recall	f1-score	support
HAZE	0.000	0.000	0.000	0
RAINY	0.178	0.394	0.246	350
SNOWY	0.822	0.493	0.616	1429
SUNNY	0.541	0.209	0.302	1108
accuracy			0.372	2887
macro avg	0.385	0.274	0.291	2887
weighted avg	0.636	0.372	0.451	2887





We can observe how even in this case, as in the case of the neural network I built, we find that the accuracy produced on this test is rather low. This is because, as mentioned previously, the images are all different and difficult to recognize.
However, the trend does not present any exaggerated peaks, indeed on both the train and the test we note that it has a rather constant trend of growth. The loss is not very positive but as said this depends a lot on the test we have available.
Also in this case we used a callback function to end our train before the 100 epochs, in particular after 20 consecutive epochs in which the accuracy of the test was not going to improve.

Chapter 4

Conclusion

We can say that regarding the evaluation on the Weather_dataset the neural network we built is better because it was created in such a way that it could try to fit in with that particular dataset but this does not lead to remarkable results due to the images that make up the dataset.
Moreover CNN created by us is also good for the spit made with the 2000 images MWI dataset, because it produces a high accuracy not producing a rather high overfitting.
The pre-trained model has the particularity of having a trend, both with regard to the accuracy and with regard to the most stable loss compared to our neural network, this because it was created in such a way that it could try to fit on several types of datasets.
In fact, when I consider the MWI dataset and I test it, at performance level it is slightly worse than our CNN, but overall it's better with regards to the stability it has during all the iterations it does.
Same thing regarding the evaluation on the Weathe_dataset, also it has a stable trend even if it does not lead to the desired results.