Machine Learning, HW2 - Multi-class Weather Classification

Riccardo Caprari, ID number 1743168

Fall 2019

Contents

1 Introduction

Aim of this work is to solve an image classification problem using convolutional neural networks (CNN) either trained from scratch or pre-trained. In particular the problem is to predict, given an image, weather conditions which can be {RAINY, SNOWY, SUNNY, HAZE}.

In general, a problem of this kind can be solved estimating a function $f : X \mapsto Y$, with $Y = \{C_1, ..., C_k\}$ which is the codomain containing the classes, in this case $k = 4$, using a data set $D = \{(x_n, t_n)_{n=1}^N\}$ such that $t_n \approx f(x_n)$. Moreover, using a neural network, it's necessary to learn a set θ of parameters and introduce $y = \hat{f}(x; \theta)$ so that $\hat{f}$ approximates f. A neural network is composed of units of different kinds, depending on the activation function, which simulate brain neurons and that belong to an hidden layer. The number of layers can vary from network to network and represents the depth of the actual network. As shown in Fig. 1 units of different layers are linked each other or directly connected with inputs or a final output. The objective is to find a set of weight parameters to be assigned to these links in order to obtain $y = \hat{f}(\sum_{j=0}^{m} w_j x_j)$ with m the number of units.

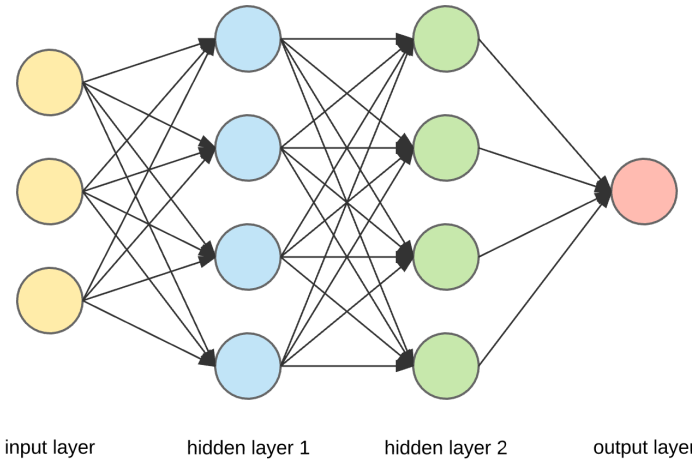


Figure 1: An artificial, fully connected, neural network.

The work is composed of a preprocessing part in which the images of the data set are remodeled and then other two parts concerning the development and training of a CNN from scratch and the use of a pre-trained CNN with transfer learning techniques. Last chapter (5) shows the results obtained

with the various neural networks, analyzing them in order to get a good comparison. Evaluation metrics used for this work are accuracy, precision, loss, recall and f1-score.

2 Datasets and image modeling

Two datasets has been used in order to proceed with the work. The first one is the Multi Weather Image (MWI) dataset which contains 4000 images divided into four classes mentioned in Chapter 1. The other one is the Smart-I (S-I) dataset composed of 3038 images obtained from Closed Circuit Television (CCTV) and that doesn't contain samples for HAZE class. In fact, this second dataset is highly unbalanced: 0 images for HAZE, 521 for RAINY, 1421 for SNOWY and 1096 for SUNNY class. It is also necessary to say that both datasets have some misclassified samples or, for example, images that may contain both rain and haze. Another *noisy* example of the many ones is having night images in the SUNNY category, lowering the final accuracy of the model. Nevertheless, it's been decided to leave the partitions of the four classes intact, making the training of the network more challenging. On the contrary, being the two datasets quite different, it's been decided to mix them in a precise manner. In fact, this notable difference leads to an higher accuracy on MWI images and a drastic performance on CCTV images. This problem could have been also solved training the neural network on the first dataset and then applying transfer learning with fine tuning on the second dataset. However, it's been preferred to create a new training set and test set mixing MWI and S-I as showed in Fig. 2.

Each class has been split in two sets in order to get two new mixed datasets for the training and the evaluation processes. These sets aren't splitted in a weighted manner but literally obtained dividing in two identical parts the initial class set and merging each of them in a new different dataset. HAZE set, instead, has been fully copied in the two datasets. This structure permits to obtain an higher accuracy of the NN as showed in Chapter 5.

With regard to image modeling, it has been used the *ImageDataGenerator* library imported from *Keras*. In particular, this library permits us to rescale, zoom, rotate, flip, shift and even fill image edges. All these techniques has been used to transform the images and permit the network to have a fixed input size. In fact, all the RGB images that will be used have a

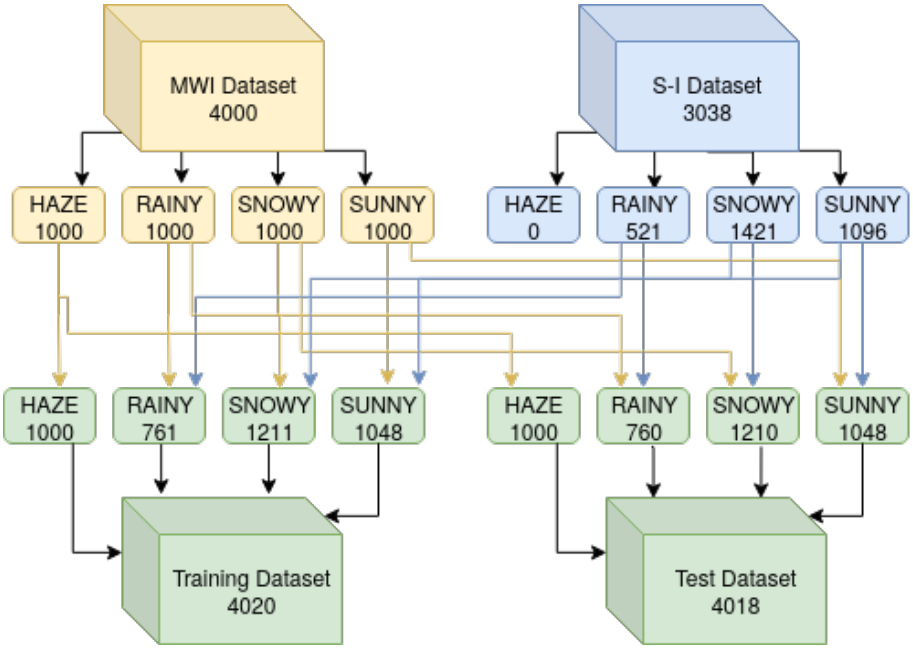


Figure 2: Structure of training and test datasets.

size of 136 x 136 x 3 pixels. This library has also been used for the real time *data augmentation* process which aim is to increase the size of the dataset by adding different transformations of the same image and actually permit the NN to train with different variants. *ImageDataGenerator* library has also a special parameter that can split the training set for the *cross validation* process. In particular, the size of the validation test is about 400 images, which is the 10% of the training dataset. The evaluation tests, which will compute the final scores, have been done, instead, using the test dataset and the *S-I* Dataset.

At the end, the training dataset has been augmented by also adding 40 images (10 for each class) that were collected for homework purposes.

3 CNN from scratch

Many neural networks have been sketched and tried out with the purpose of getting an high accuracy in the evaluation test. The development of the NN began with inspiration from networks like AlexNet and LeNet. However, none of those networks (or slightly modified ones) had a good performance. In fact, this classification problem does not focus on a single subject but focuses on a group of details that differentiate for example a sunny day to a snowy one. That could be one of the main reason why these networks perform bad. Another important reason is, of course, the misclassification problem of the dataset and noise images. All these details have led back to a careful selection of layers and parameters, focusing on features extraction. At the end, after many attemps, a good convolutional neural network has been found: RickyNet.

3.1 RickyNet

Architecture design

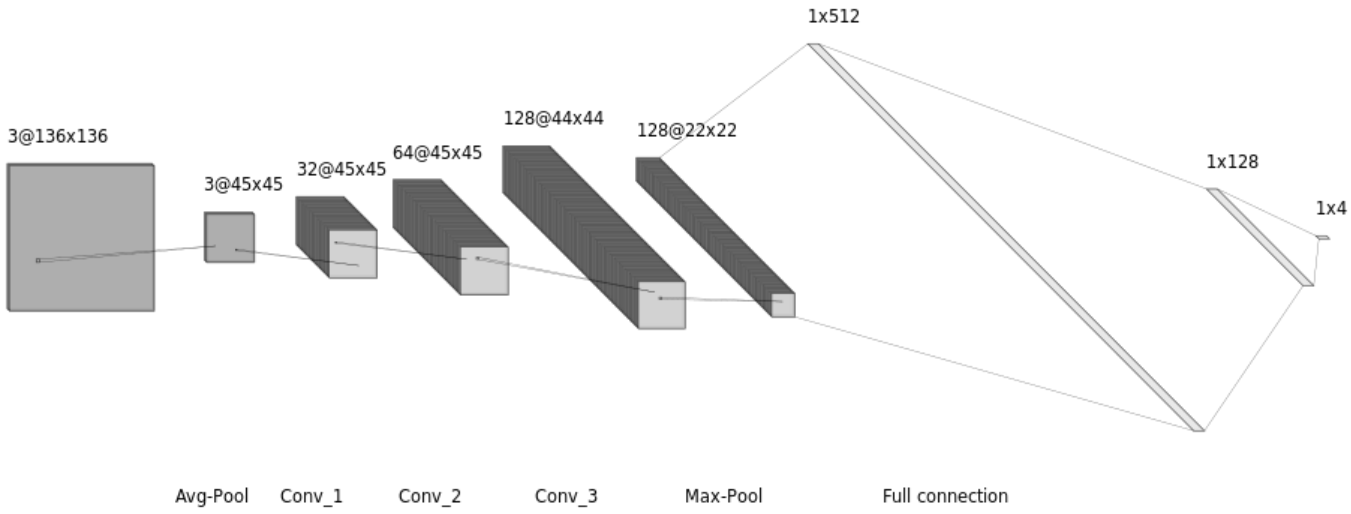


Figure 3: RickyNet architecture.

RickyNet is composed of eight layers as shown in Fig. 3. This is the

actual list of layers with regard to their type, output shape and number of parameters[1]:

Layer (type)	Output Shape	Param #
average_pooling2d (AvgPool2d)	(None, 45, 45, 3)	0
conv2d_1 (Conv2D)	(None, 45, 45, 32)	128
conv2d_2 (Conv2D)	(None, 45, 45, 64)	2112
conv2d_3 (Conv2D)	(None, 44, 44, 128)	32896
max_pooling2d (MaxPooling2D)	(None, 22, 22, 128)	0
flatten (Flatten)	(None, 61952)	0
dropout (Dropout)	(None, 61952)	0
dense (Dense)	(None, 512)	31719936
dropout_1 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 128)	65664
dense_2 (Dense)	(None, 4)	516

Total params: 31,821,252
Trainable params: 31,821,252
Non-trainable params: 0

This network has a different approach in comparison with other NNs like AlexNet which use sequences of convolutional layers and max pooling layers. In fact, in RickyNet there's an initial average pooling to reduce the size of the image, three convolutional layers that step by step increase depth and a single max pooling to reduce height and width. After this last layer there's a flatten to introduce a full connection between the units, a dropout to reduce overfitting and three dense that slowly decrease the size of the array from 512 to 4.

[1]Further details about activation, kernel-size, padding, etc. can be found in the code.

Training process

The model has been compiled using *adam* optimizer instead of *SGD*, *rmsprop*, *adadelta* and many others because of its great accuracy level. All the results are shown in the last chapter. The loss function is the *categorical crossentropy* which works good for a multi-class classification problem. The network has been trained using a variable number of epochs which have been chosen from the *EarlyStopping* callback which gets as parameters a *patience* level and a value to monitor (e.g. validation accuracy): training will stop as soon as validation accuracy stops increasing for *patience* times. Regarding the size of the batch, it actually changed during all the trainings, but was usually set from 6 to 64.

4 Transfer learning

Instead of sketching and training a neural network from scratch it's usually convenient apply a technique called *transfer learning* using a pre-trained model. Its approach is to "freeze" a set of layers of a pre-trained network that will be the starting point of another model which can be trained with another data set. The process to replace the ouput layer and take advantage of feature extraction is called *fine tuning*.

In order to get advantage of a pre-trained model some of the most famous NN has been tested out: ResNet, InceptionV3, VGG16, VGG19. The first two networks are quite heavy (50 and 42 layers) while the VGG ones are smaller (16 and 23 layers) and actually performed better in this work. VGG neural networks are usually used for object recognition and have a good performance on ImageNet dataset. For this work VGG networks have been preferred because of their good performance level on the weather images dataset.

4.1 VGG[16/19]-TL

Both VGG16 and VGG19 have sequences of convolutional layers and max pooling. This strategy seems to be quite good for the feature extraction process. Fine tuning with this network has been done including all the layers before the final flatten() after the block_5. After this layer, other four has

been added, in particular for VGG16:

```
_________________________________________________________________
average_pooling2d (AveragePo (None, 1, 1, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
batch_normalization (BatchNo (None, 512)               2048
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 200)               102600
_________________________________________________________________
batch_normalization_1 (Batch (None, 200)               800
_________________________________________________________________
dropout_1 (Dropout)          (None, 200)               0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               20100
_________________________________________________________________
batch_normalization_2 (Batch (None, 100)               400
_________________________________________________________________
batch_normalization_3 (Batch (None, 100)               400
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 404
=================================================================
Total params: 14,841,440
Trainable params: 2,484,736
Non-trainable params: 12,356,704
_________________________________________________________________
```

and VGG19:

```
_________________________________________________________________
average_pooling2d (AveragePo (None, 1, 1, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
batch_normalization (BatchNo (None, 512)               2048
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 200)               102600
_________________________________________________________________
batch_normalization_1 (Batch (None, 200)               800
_________________________________________________________________
dropout_1 (Dropout)          (None, 200)               0
```

```
-----------------------------------------------------------------
dense_1 (Dense)              (None, 100)               20100
-----------------------------------------------------------------
batch_normalization_2 (Batch (None, 100)               400
-----------------------------------------------------------------
batch_normalization_3 (Batch (None, 100)               400
-----------------------------------------------------------------
dense_2 (Dense)              (None, 4)                 404
=================================================================
Total params: 20,151,136
Trainable params: 2,484,736
Non-trainable params: 17,666,400
-----------------------------------------------------------------
```

This structure has been adopted for both the networks and it's composed of one average pooling layer and three dense layers for the full connection on the units. Moreover, some batch normalizations and dropouts have been added to reduce the overfitting in the training process. There's a difference of about 5.5 million parameters between the two models and, of course, the same number of trainable parameters obtained by adding new layers. However, VGG16 had a better performance in terms of accuracy and loss using this sequence of layers. All the comparisons are shown in the next chapter.

5 Results

This last chapter reports the performance outcomes of the developed neural networks, in particular: RickyNet, VGG16-TL and VGG19-TL. All the results are obtained training the models on the new dataset explained in Chapter 2 and testing them on the Smart-I dataset[2]. Training and test processes are done using this environment:

```
GPU: GeForce GTX 1650 4GB
Python 3.7.5, TensorFlow2, CUDA 10.1, CUDNN 7.6.5
```

[2]Also called *TestSet_Weather* dataset.

5.1 Accuracy and loss

Let's now proceed with a comparison of the models in terms of accuracy and loss.

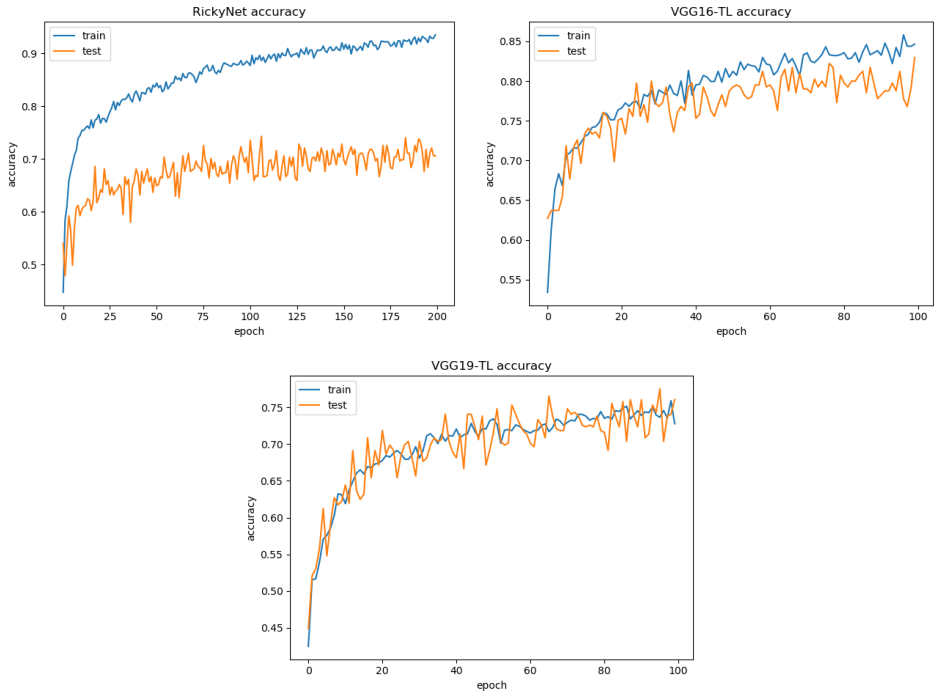


Figure 4: Accuracy level of the models during the training.

As it's possible to notice in Fig. 4, there's a considerable overfitting in RickyNet while in the other networks it's near to zero. In particular, it even may happen that VGG19-TL goes in underfitting.

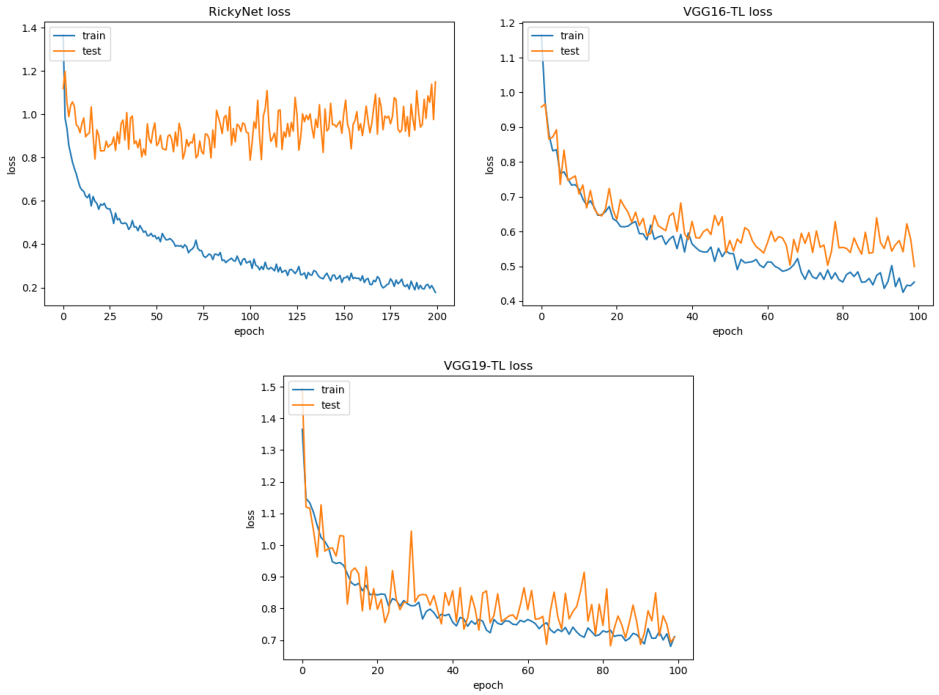
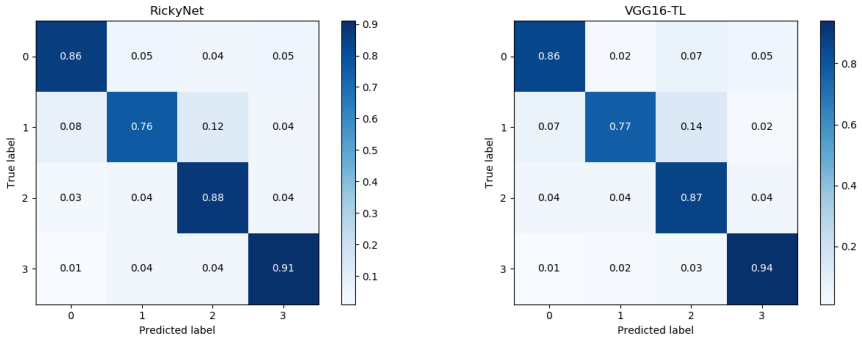


Figure 5: Loss level of the models during the training.

Also for the loss (5), RickyNet has an increasing behavior that differs from VGGs one which is more stable. However, it must be taken into account that the number of epochs is different and we don't know exactly the trend of the function when $epoch \to \infty$.



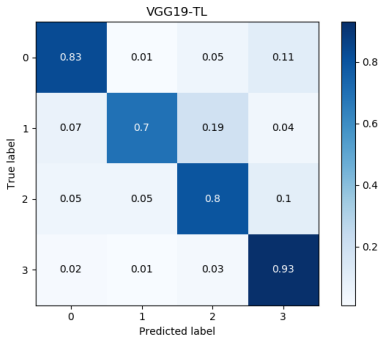


Figure 6: Confusion matrix of the models.

Regarding the confusion matrix of the models in Fig. 6, it's evident that the value of recall on the diagonals is almost the same in each NN. Anyway, VGG16-TL is the one performing better.

5.2 Precision and recall

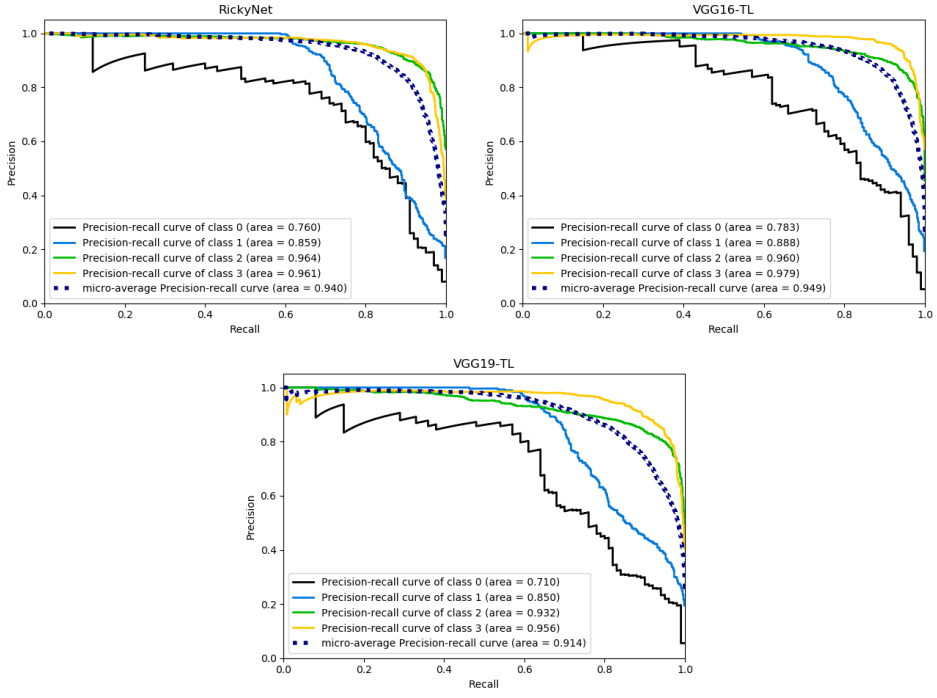


Figure 7: Precision-Recall graph of the models.

Precision and recall metrics are represented in Fig. 7 using the precision-recall curve. In particular it's possible to do a comparison through AUC (area under curve) of the models. Each class has a different curve and they are labeled in the following way: class 0 [HAZE], class 1 [RAINY], class 2 [SNOWY], class 3 [SUNNY]. A quick look suggests that the best model is VGG16-TF, thanks to his bigger AUCs.

5.3 Tests on Smart-I dataset

This chapter concludes with the scores of all the metrics in testing the Smart-I dataset which is the one containing images from CCTVs. Concretely, this last comparison is done using also the f1-scores which are obtained through precision and recall values, giving us a good term to base with.

RickyNet

```
Test loss: 0.484739
Test accuracy: 0.870300
```

	precision	recall	f1-score	support
HAZE	0.480	0.860	0.616	100
RAINY	0.780	0.756	0.768	521
SNOWY	0.918	0.885	0.901	1421
SUNNY	0.917	0.907	0.912	1096
accuracy			0.870	3138
macro avg	0.774	0.852	0.799	3138
weighted avg	0.881	0.870	0.874	3138

VGG16-TL

```
Test loss: 0.348587
Test accuracy: 0.877226
```

	precision	recall	f1-score	support
HAZE	0.446	0.860	0.587	100
RAINY	0.828	0.766	0.796	521
SNOWY	0.914	0.870	0.891	1421
SUNNY	0.930	0.942	0.936	1096
accuracy			0.877	3138
macro avg	0.779	0.859	0.802	3138
weighted avg	0.890	0.877	0.881	3138

VGG19-TL

```
Test loss: 0.473054
Test accuracy: 0.831103
```

	precision	recall	f1-score	support
HAZE	0.381	0.830	0.522	100
RAINY	0.819	0.704	0.757	521
SNOWY	0.887	0.799	0.841	1421
SUNNY	0.857	0.932	0.893	1096
accuracy			0.831	3138
macro avg	0.736	0.817	0.754	3138
weighted avg	0.849	0.831	0.835	3138

After an accurate analysis of the performance of the three convolutional neural networks it is right to conclude that VGG16-TF is the best performing model and it will be chosen for the final *blind test* of the work.