Weather Classification using CNN

G4

INTRODUCTION

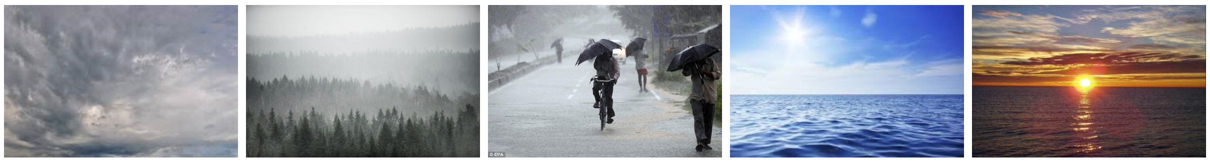
Weather plays a huge part in day to day life, from destroying crops to affecting people's commute it affects every individual by one way or another. Generally, in traditional methods we always make use of expensive sensors and human involvement to detect the weather. It often ends up giving results with low accuracy. Determining the weather from a single image is a simple task for an average human but designing a classifier that does this from a single image is a tall order. There's also a constraint on the availability of human resources and expensive instruments. Hence to save human efforts and instruments the new approach of classifying has been proposed. This approach involves use of convolutional neural networks to classify the weather condition based on the image. We can make use of surveillance cameras which are readily available to get the image of the outdoors and detect the weather based on the image. In this project, a CNN architecture has been proposed that classifies an image into a weather condition. As there are significant differences in the visual condition of every weather. It extracts the various information from the image and detects the weather condition based on the photometric property of the image such as color and texture of the image.

PROBLEM STATEMENT

To determine if a person needs to carry an umbrella by predicting the weather condition by utilizing Convolutional Neural Networks. If the weather is sunny or rainy, the CNN model should advise the user to carry an umbrella.

DATASET

The dataset features 5 different classes of weather collected from kaggle, however it's real life data so any system for weather classification must be able to handle this sort of images. Training set includes about 1500 labelled images including the validation images. Images are not of fixed dimensions and the photos are of different sizes. Images do not contain any border.



Cloudy Foggy Rain Shine Sunrise

Figure 1: Image from each weather class

The number of images of each class in the Weather dataset is almost the same. The distribution of these classes can be seen in following figure:

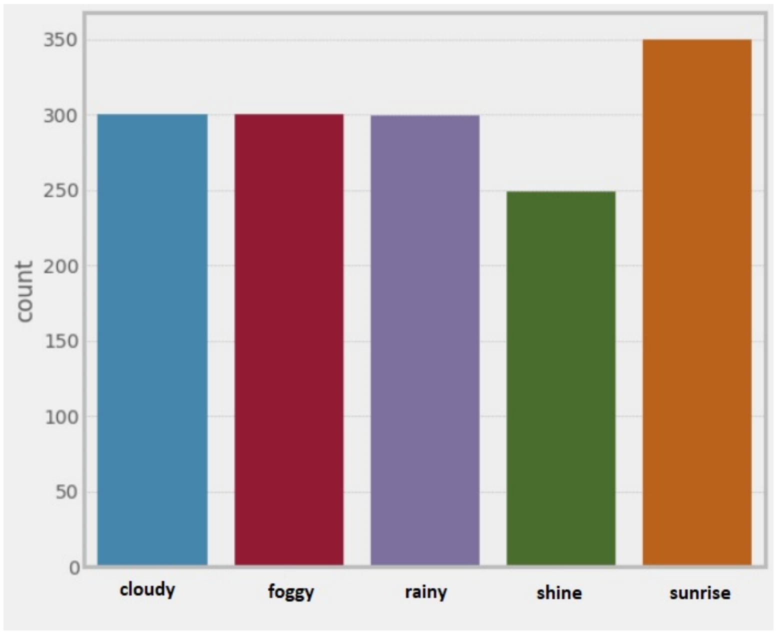


Figure 2: Distribution of image classes

CONVOLUTIONAL NEURAL NETWORK

A convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analysing visual imagery. CNNs are regularized versions of multilayer perceptrons (fully connected networks).

The name “convolutional neural network” indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of a series of convolutional layers that convolve with a multiplication or other dot product. The activation function is commonly a RELU layer, and is subsequently followed by additional convolutions such as pooling layers, fully connected layers and normalization layers, referred to as hidden layers because their inputs and outputs are masked by the activation function and final convolution. The final convolution, in turn, often involves backpropagation in order to more accurately weight the end product.

<u>Architecture of CNN</u>

Layers:

- Convolution: Convolutional layers convolve around the image to detect edges, lines, blobs of colours and other visual elements. Convolutional layers hyperparameters are the number of filters, filter size, stride, padding and activation functions for introducing nonlinearity.

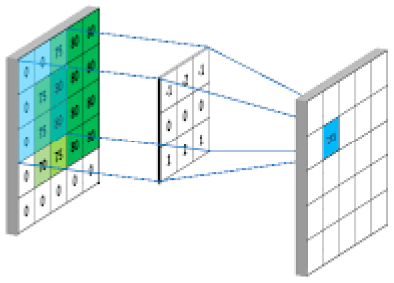


Figure 3: Convolutional layer

- MaxPooling: Pooling layers reduces the dimensionality of the images by removing some of the pixels from the image. MaxPooling replaces an nxn area of an image with the maximum pixel value from that area to downsample the image.

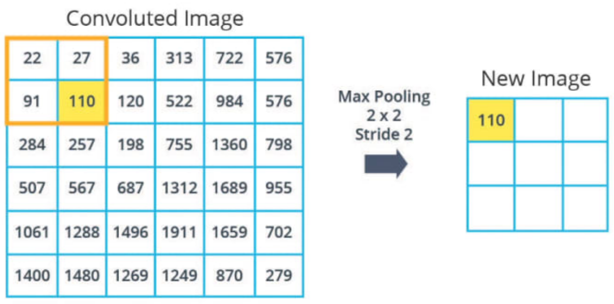


Figure 4: MaxPooling

- Dropout: Dropout is a simple and effective technique to prevent the neural network from overfitting during the training. Dropout is implemented by only keeping a neuron active with some probability p and setting it to 0 otherwise. This forces the network to not learn redundant information.
- Flatten: Flattens the output of the convolution layers to feed into the Dense layers.
- Dense: Dense layers are the traditional fully connected networks that maps the scores of the convolutional.
- Batch Normalization: Batch Normalization is a recently developed technique by Ioffe and Szegedy which tries to properly initialize neural networks by explicitly forcing the activations throughout a network to take on a unit gaussian distribution at the beginning of the training. In practice, we put the Batch Normalization layers right after Dense or convolutional layers. Networks that use Batch Normalization are significantly more robust to bad initialization. Because normalization greatly reduces the ability of a small number of outlying inputs to over influence the training, it also tends to reduce overfitting. Additionally, batch normalization can be interpreted as doing pre-processing at every layer of the network, but integrated into the network itself.

ARCHITECTURE OPTIMIZATION

Many neural networks have been sketched and tried out with the purpose of getting a high accuracy in the evaluation test. The development of the neural network began with inspiration from the VGG19 architecture. The VGG network architecture was introduced by Simonyan and Zisserman in their 2014 paper, Very Deep Convolutional Networks for Large Scale Image Recognition. The designed model had 16 convolution layers and the performance obtained was unsatisfactory. The validation accuracy was 0.26 and while testing, the model predicted every image as a sunrise.

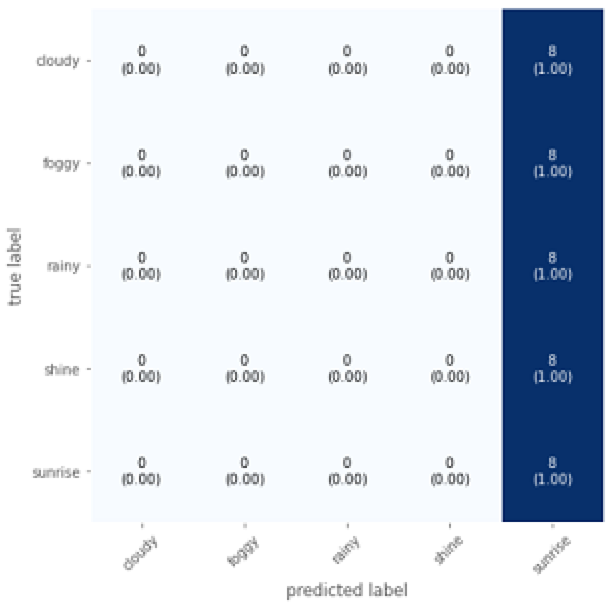


Figure 5: Confusion matrix of initial model

Clearly the model is not trained well. To train such a long model successfully, it would require a huge dataset.Since the available dataset had only around a few thousand images, the number of convolution layers were reduced. After reducing the number of convolution layers to 6, that is 3 blocks of 2 layers each, significant improvements were observed.

<u>Kernel size</u>

We optimized the model kernel size to (2, 2) and (1, 1) for the first and the second convolution layers respectively. For a larger Kernel size, the loss increased significantly and the model performed poorly. This may be due to the fact that for an image size of 150x150, a large kernel may often miss out on small details like rain droplets.

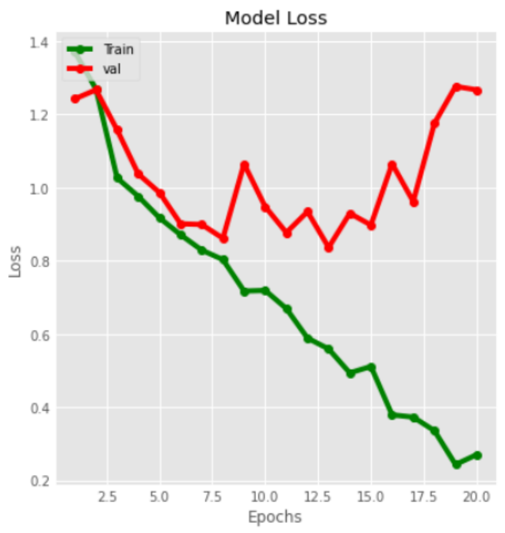


Figure 6: Large kernel size results in overfitting

Epochs

In terms of artificial neural networks, an epoch refers to one cycle through the full training dataset. Usually, training a neural network takes more than a few epochs. In other words, if we feed a neural network the training data for more than one epoch in different patterns,we hope for a better generalization when given a new "unseen" input (test data).With a neural network, the goal of the model is generally to classify or generate material which is right or wrong. Thus, an epoch for an experimental agent performing many actions for a single task may vary from an epoch for an agent trying to perform a single action for many tasks of the same nature. In reinforcement learning terminology, this is more typically referred to as an episode.

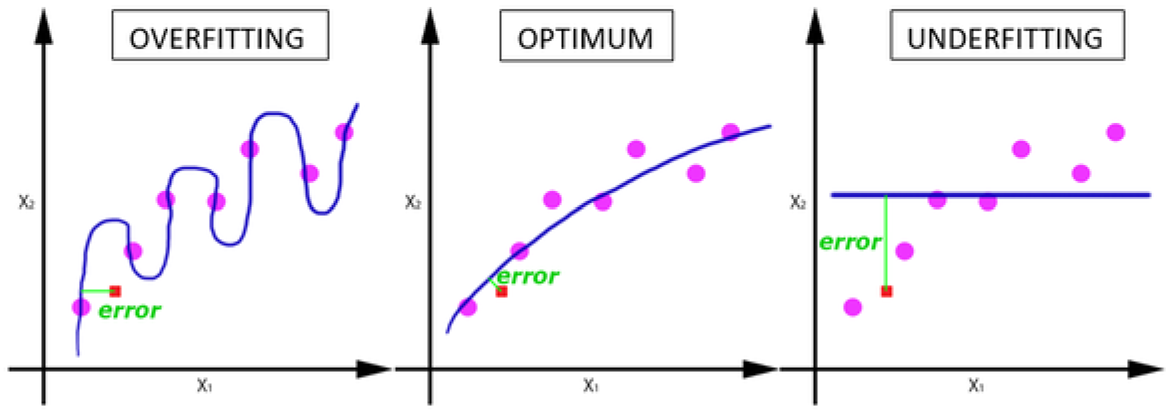


Figure 7: Epoch behaviour for different model fittings

Early Stopping: One of the simplest methods to prevent overfitting of a network is to simply stop the training before overfitting has had a chance to occur. It comes with the disadvantage that the learning process is halted.Too many epochs can lead to overfitting of the training dataset, whereas too few may result in an underfit model. Early stopping is a method that allows you to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a hold out validation dataset.

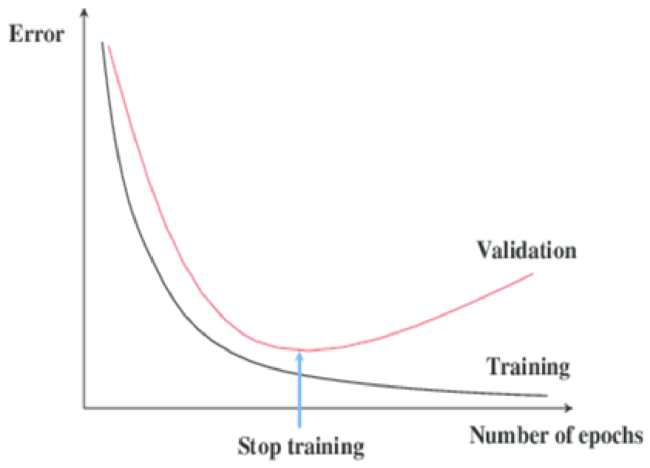


Figure 8: Early stopping graph

Also, our model struggled to differentiate between what is cloudy and what is foggy as they are interchangeable.

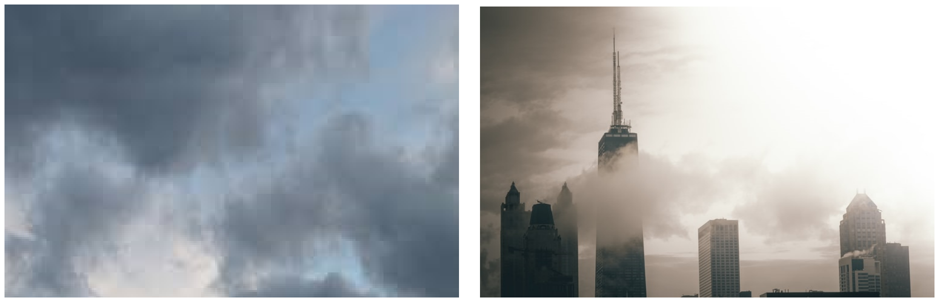


Figure 9: An example where the model fails to classify properly

This is one example where the foggy image was categorized as a cloudy image.

Hence, we used data augmentation to improve our model of cloudy and foggy conditions.

Data augmentation

Data augmentation in data analysis are techniques used to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data. It acts as a regularizer and helps reduce overfitting when training a machine learning model.

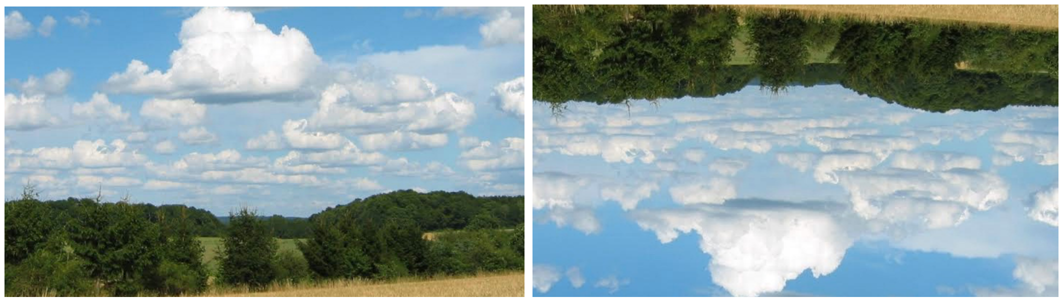
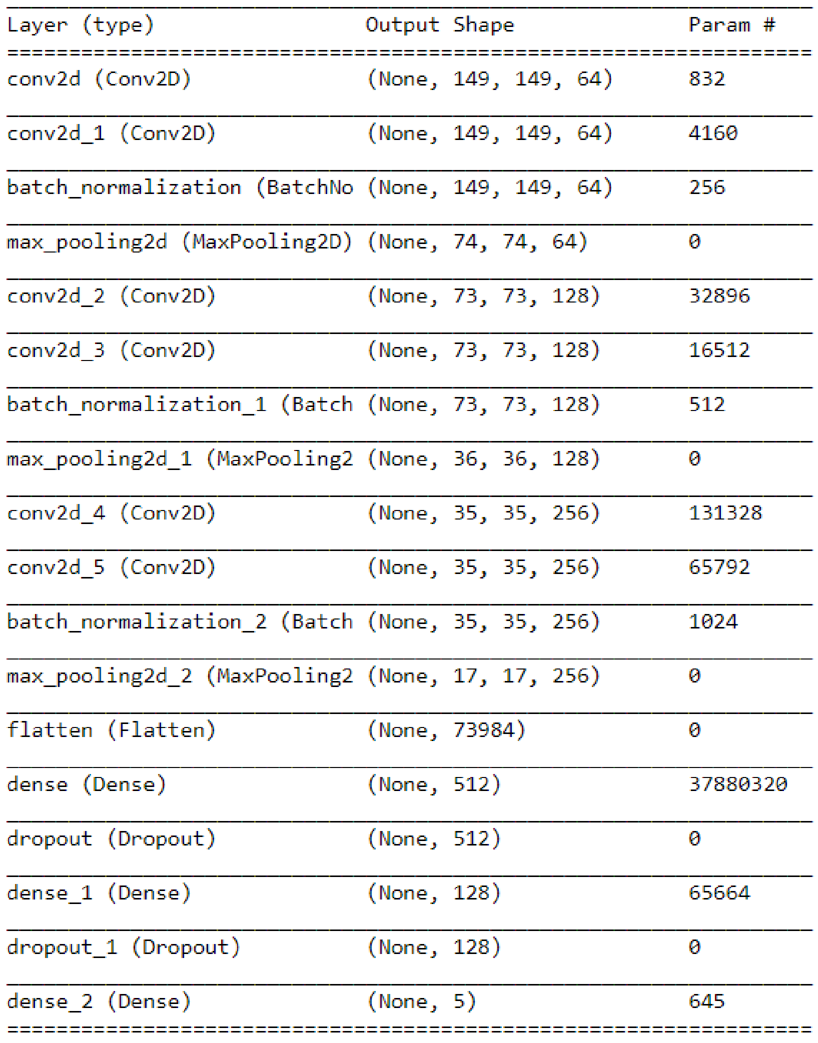


Figure 10: Image of a cloudy weather before and after augmentation

In order to overcome the model's impotence to differentiate cloudy and foggy images, we augmented cloudy and foggy images, increasing the number of images for better training.

FINAL MODEL

The final model consists of 3 blocks of 2 convolutional layers followed by batch normalization and max pooling layer. After that there is a flatten to introduce a full connection between the units, dropout layers to reduce overfitting and dense layers that slowly decrease the size of the array from 512 to 5.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 149, 149, 64)      832       
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 149, 149, 64)      4160      
_________________________________________________________________
batch_normalization (BatchNo (None, 149, 149, 64)      256       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 64)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 73, 73, 128)       32896     
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 73, 73, 128)       16512     
_________________________________________________________________
batch_normalization_1 (Batch (None, 73, 73, 128)       512       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 128)       0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 35, 35, 256)       131328    
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 35, 35, 256)       65792     
_________________________________________________________________
batch_normalization_2 (Batch (None, 35, 35, 256)       1024      
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 256)       0         
_________________________________________________________________
flatten (Flatten)            (None, 73984)             0         
_________________________________________________________________
dense (Dense)                (None, 512)               37880320  
_________________________________________________________________
dropout (Dropout)            (None, 512)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 128)               65664     
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 645       
=================================================================
```

Figure 11: CNN Layers of final model

An early stopping callback was also used in case overfitting happened. The growth of model accuracy and loss during training and validation is given below. The confusion matrix of model performance is also included below.

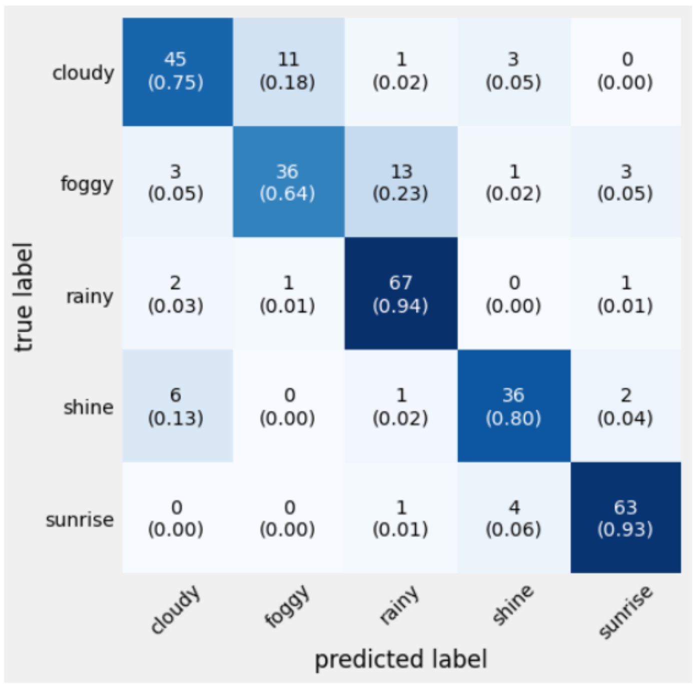


Figure 12: Confusion matrix of the final model

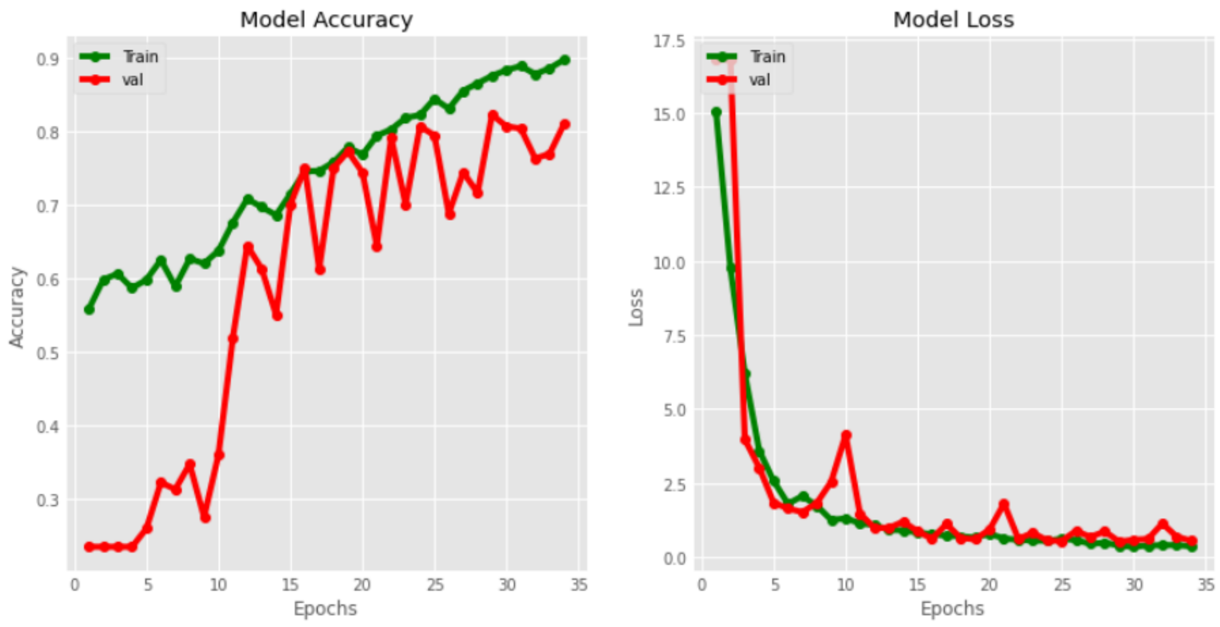
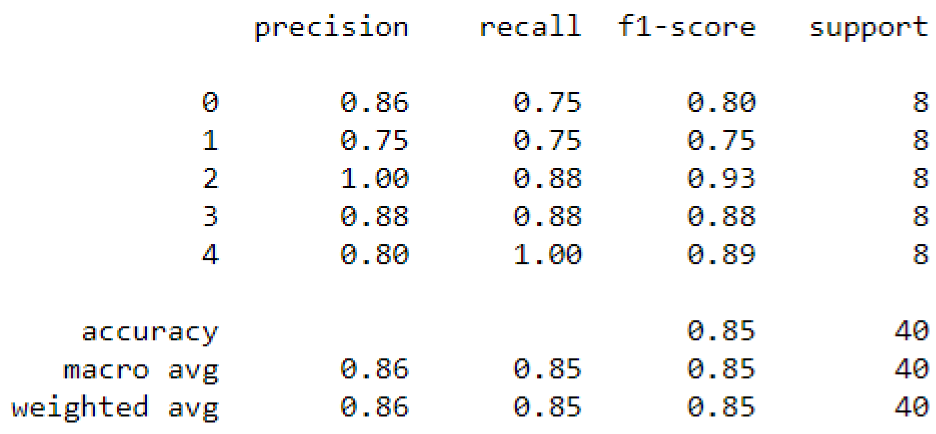


Figure 13: Growth of model accuracy and loss

We used this model to predict the output of 40 images gathered from the internet, 8 of each weather type.



```
              precision    recall  f1-score   support

           0       0.86      0.75      0.80         8
           1       0.75      0.75      0.75         8
           2       1.00      0.88      0.93         8
           3       0.88      0.88      0.88         8
           4       0.80      1.00      0.89         8

    accuracy                           0.85        40
   macro avg       0.86      0.85      0.85        40
weighted avg       0.86      0.85      0.85        40
```

Figure 14: Classification Report

The above classification report shows the accuracy, precision, f1 score for all classes along with macro average and micro average for each metric. A macro-average will compute the metric independently for each class and then take the average (hence treating all classes equally), whereas a micro-average will aggregate the contributions of all classes to compute the average metric. In a multi-class classification setup, micro-average is preferable if you suspect there might be class imbalance (i.e you may have many more examples of one class than of other classes).

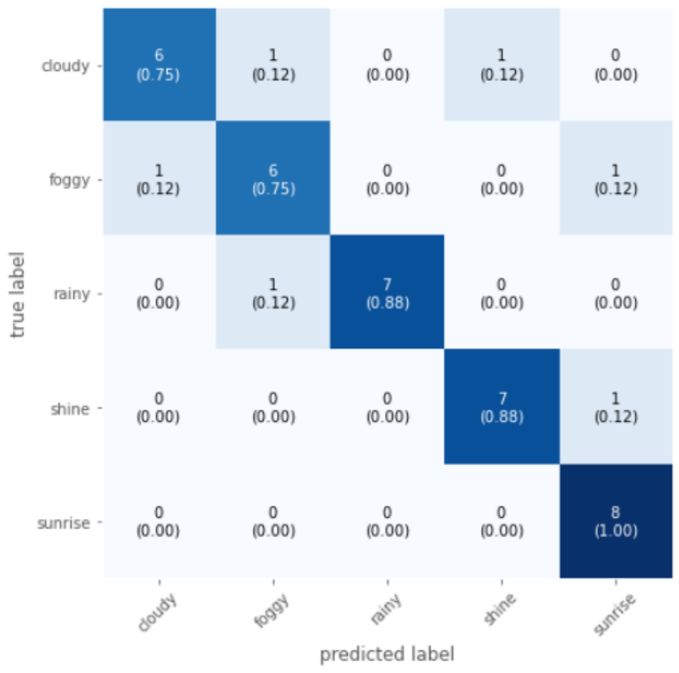


Figure 15: Confusion matrix of test data

CONCLUSION

Our objective was to develop a CNN model for weather detection with tensorflow. After the project we were able to fulfill the objective to develop a powerful for classifying images as rainy, cloudy, shiny, foggy and sunrise. The final model displayed an 85% accuracy when it was provided with 40 weather images, 8 of each class.

After testing our samples we made slight modifications in architecture like changing number of filters, activation function between other layers, optimizers and by varying the number of epochs, learning rate,batch size. All these modifications are implemented successfully. Finally, we plotted corresponding graphs for each modification and tabulated the changes occurring in the output.

Drawbacks: Even though the model predicts weather conditions like shine and sunrise with high accuracy, it still struggles to differentiate some cloudy, foggy and rainy images.

REFERENCES

https://poloclub.github.io/cnn-explainer/

https://www.researchgate.net/publication/280218749_Weather_classification_with_deep_convolutional_neural_networks

https://towardsdatascience.com/image-augmentation-for-deep-learning-histogram-equalization-a71387f609b2

https://www.tensorflow.org/

https://scikit-learn.org/

https://numpy.org/