Machine Learning Engineer

Capstone Project

Dated: 25/12/2019

Multi-Class Weather Classification

by

Vijay Kumar Gupta
(An aspiring Machine Learning Engineer)
(mech.vijayg@gmail.com)

Contents

I. Definition

1.1. Project Overview

Time and again unfortunate accidents due to inclement weather conditions across the globe have surfaced. Ship collision, train derailment, plane crash and car accidents are some of the tragic incidents that have been a part of the headlines in recent times. This grave problem of safety and security in adverse conditions has drawn the attention of the society and numerous studies have been done in past to expose the vulnerability of functioning of transportation services due to weather conditions.
In past, weather-controlled driving speeds and behaviour have been proposed. With the advancement in technology and emergence of a new field, intelligent transportation, automated determination of weather condition has become more relevant. Present systems either rely on series of expensive sensors or human assistance to identify the weather conditions. Researchers, in recent era have looked at various economical solutions. They have channelled their research in a direction where computer vision techniques have been used to classify the weather condition using a single image. The task of assessing the weather condition from a single image is a straightforward and easy task for humans. Nevertheless, this task has a higher difficulty level for an autonomous system and designing a decent classifier of weather that receives single images as an input would represent an important achievement.

The aim of this capstone project is to build a convolutional neural network that classifies different weather conditions while working reasonably well under constraints of computation. The work described in this report translates to two contributions to the field of weather classification. The first one is exploring the use of data augmentation technique, considering different Convolutional Neural Network (CNN) architectures for the feature extraction process when classifying outdoor scenes in a multi-class setting using general-purpose images. The second contribution is the creation of a new, open source dataset, consisting of images collected online that depict scenes of five weather conditions.

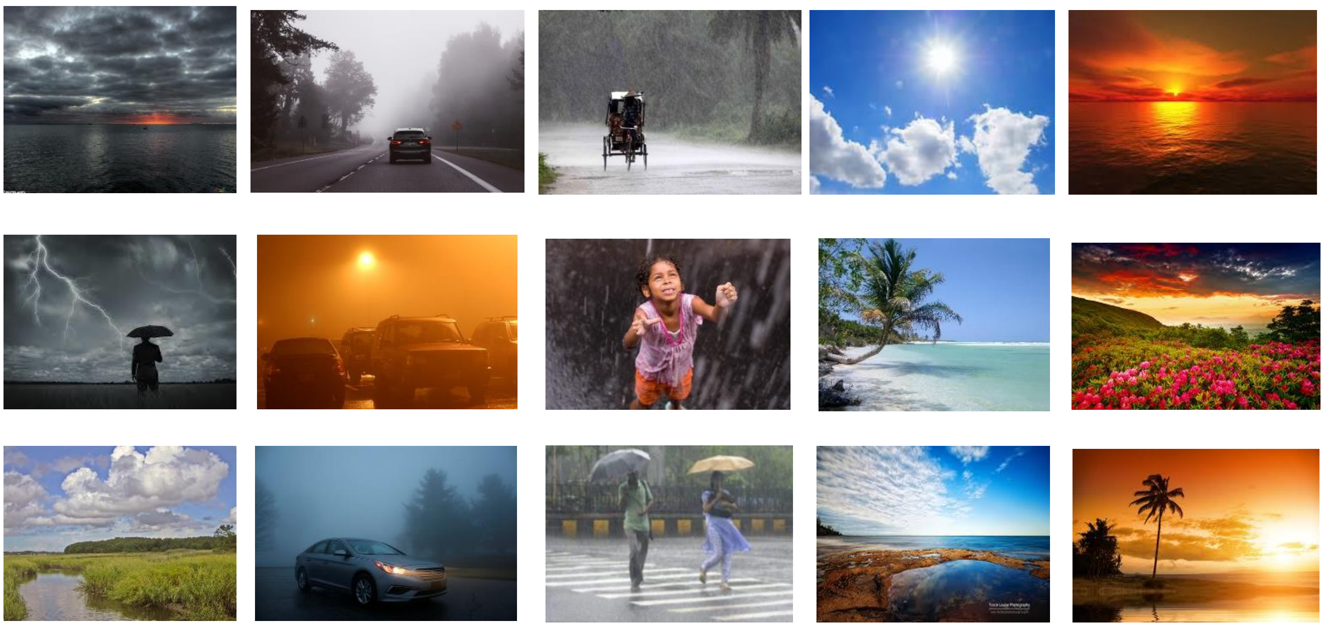

a) Cloudy b) Foggy c) Rainy d) Shine e) Sunrise

Fig. 1. Sample images of all five categories of Weather Dataset.

1.2. Problem Statement

The weather dataset was saved in local disk in respective folder labelled by identifying weather condition in the image such as cloudy, foggy, rainy, shine, sunrise.

The goal is to predict the likelihood that the weather condition is from a certain class from the provided classes, thus making it a multi-class classification problem in machine learning terms.

Five target classes are provided in this dataset: Cloudy, Foggy, Rainy, Shine, Sunrise.

The goal is to train a CNN that would be able to classify the weather condition into these five classes.

Deep-learning based techniques (CNNs) has been very popular in the last few years where they consistently outperformed traditional approaches for feature extraction to the point of winning ImageNet challenges. In this project, transfer learning along with data augmentation will be used to train a convolutional neural network to classify images of weather to their respective classes.

Transfer learning is referred as a machine learning method where a model developed for a task is reused as the starting point for a model on a second task or in other words, Transfer learning refers to the process of using the weights from pre-trained networks on large dataset. As the pre-trained networks have already learnt how to identify lower level features such as edges, lines, curves etc with the convolutional layers which is often the most computationally time-consuming parts of the process, using those weights help the network to converge to a good score faster than training from scratch.

To train a CNN model from scratch successfully, the dataset needs to be huge (which is definitely not the case here, the available dataset is very small, only 1500 images for training and validation) and machines with higher computational power is needed, preferably with GPU, which I don't have access to at this point. Fortunately, many such networks such as ResNet, Inception, VGG pretrained on ImageNet_challenge is available for use publicly. I'll be using all of them and comparing them for best accuracy.

1.3. Metrics

Here, I’m using the metric for this project is multi-class logarithmic loss (also known as categorical cross entropy)

$$logloss = \frac{1}{N} \sum_{i}^{N} \sum_{j}^{M} y_{ij} \log(p_{ij})$$

Here, each image has been labelled with one true class and for each image a set of predicted probabilities should be submitted. ***N*** is the number of images in the test set, ***M*** is the number of image class labels, ***log*** is the natural logarithm, y_{ij} is 1 if observation ***i*** belongs to class ***j*** and 0 otherwise, and p_{ij} is the predicted probability that observation ***i*** belongs to class ***j***.

A perfect classifier will have the ***log-loss*** of *0*.

Multiclass *log-loss* punishes the classifiers which are confident about an incorrect prediction. In the above equation, if the class label is 1(the instance is from that class) and the predicted probability is near to 1(classifier predictions are correct), then the loss is really low as $log(x) \rightarrow 0$ as $x \rightarrow 1$, so this instance contributes a small amount of loss to the total loss and if this occurs for every single instance(the classifiers is accurate) then the total loss will also approach 0.

On the other hand, if the class label is 1(the instance is from that class) and the predicted probability is close to 0(the classifier is confident in its mistake), as ***log(0)*** is undefined it approaches →∞ so theoretically the loss can approach infinity. In order to avoid the extremes of the log function, predicted probabilities are replaced with $max(min(p, 1\text{-}10^{15}),10^{15})$.
Graphically[^1] , assuming the i^{th} instance belongs to class ***j*** and y_{ij} = 1 , it's shown that when the predicted probability approaches 0, loss can be very large.

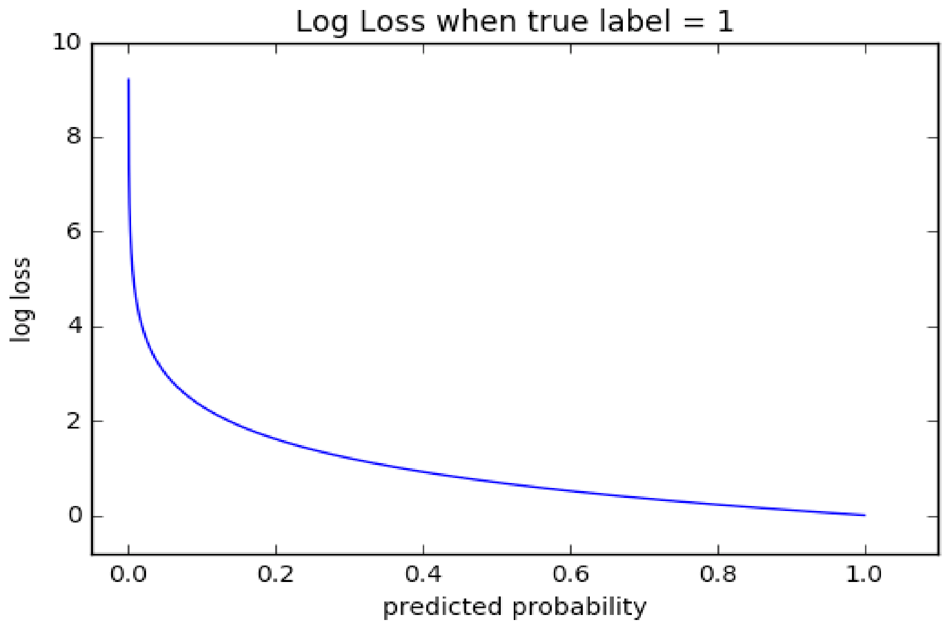


Fig. 2. Log-loss Curve

II. Analysis

2.1. Data overview

For the purposes of creating an open source dataset that can potentially contribute to future efforts in the field of computer vision, the images of interest containing Creative Commons license retrieved from Flickr, Unsplash and Pexels have been used. Different images are subject to different types of licenses; however, a common requirement is to give attribution to the author. This premise as a method to collect images is a fair way to build a dataset intended to be shared while respecting the producers of the content of interest. Unfortunately, the number of images with the Creative Commons licenses are considerably smaller than the number of images with reserved copyright, leading to potentially smaller datasets. The collection of images created as part of this work composes the Weather Dataset. Figure 1 shows a sample of the images contained in Weather Dataset.

The dataset was created, the various images are downloaded and saved in the local disk which contains weather depicting images. The dataset was labelled by identifying objects in the image such as cloudy, foggy, rainy, shine and sunrise.

The dataset features 5 different classes of weather collected from the above said different sources, 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.

Each image has only one weather category and are saved in separate folder as of the labelled class. The images are saved in folders as shown in following figure:

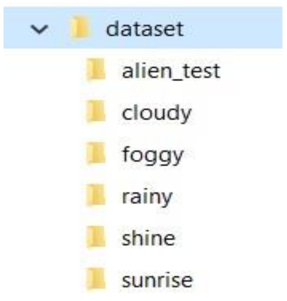


Fig. 3. Dataset containing respective class of folders

As the input is just raw images (3-dimensional arrays with height x width x channels for computers) it'd be important to pre-process them for classifying them into provided labels. However, the exact details of the pre-processing depend on our choice of the architecture to apply transfer learning.

A. Images of Cloud

The category for images showing cloudy outdoor scenes was built using terms in English, Hindi and French. The terms included cloudy and clouds, and returned a decent number of images that are included in the dataset. From the returned images, the 300 most representative images are included in the Weather Dataset.

B. Images of Fog

The task to look for images depicting foggy weather was relatively easy, as the different image hosting web sites have a decent collection of this type of pictures. However, the use of synonyms, as well as terms in other languages was required to find the total number images that compose the foggy category. The foggy category of the Weather Dataset contains 300 images.

C. Images of Rain

Collecting images of rainy weather condition was a challenge. Since many of the images which were tagged as “rainy” on the platforms were representation of raindrops on glass scene and indoor activities for rainy days. However, 300 quality images with landscapes and urban scenes of rainy days are represented in this category of the dataset.

D. Images of Shine

Collecting images of shiny weather condition was a little bit difficult. The terms included shine, shiny and beach, and returned a decent number of images that are included in the dataset. From the returned images, the 250 most representative images are included in the Weather Dataset.

E. Images of Sunrise

Collecting images of sunrise was relatively easy, as the different image hosting web sites have a decent collection of this type of pictures. The terms included sunrise, early morning and returned a decent number of images that are included in dataset. This category of the Weather Dataset contains 350 images

2.2 Exploratory Visualization

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

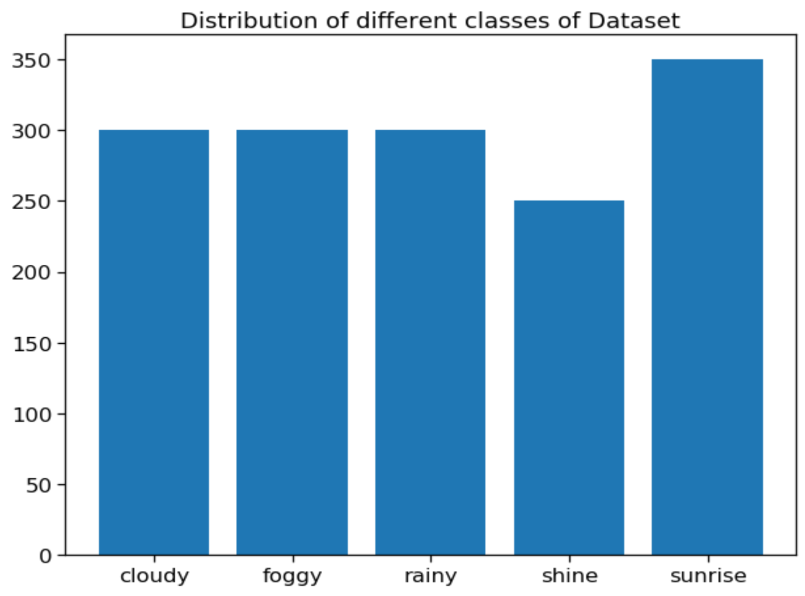
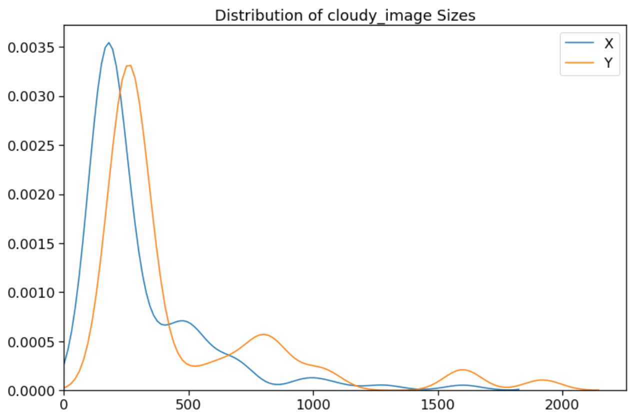
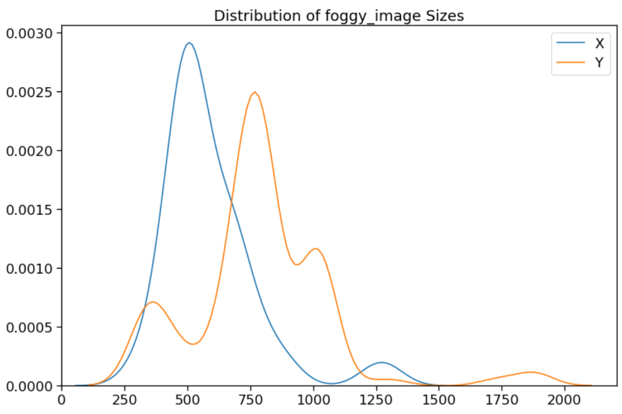


Fig. 4. Distribution of image classes

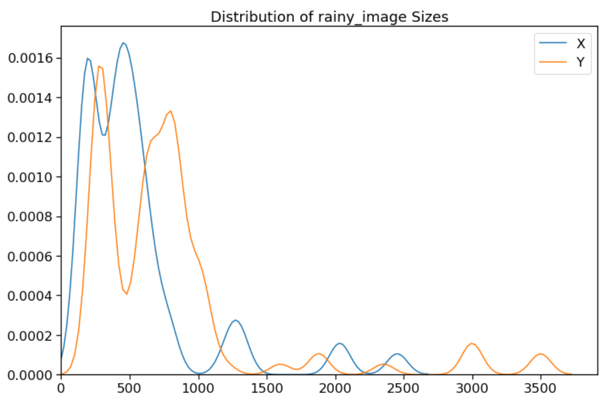
Kernel-density plot of image sizes: *As known as* Density Plots, Density Trace Graph.
A Density Plot visualises the distribution of data over a continuous interval or time period. This chart is a variation of a Histogram that uses kernel smoothing to plot values, allowing for smoother distributions by smoothing out the noise. The peaks of a Density Plot help display where values are concentrated over the interval.
An advantage Density Plots have over Histograms is that they're better at determining the distribution shape because they're not affected by the number of bins used (each bar used in a typical histogram). A Histogram comprising of only 4 bins wouldn't produce a distinguishable enough shape of distribution as a 20-bin Histogram would. However, with Density Plots, this isn't an issue.

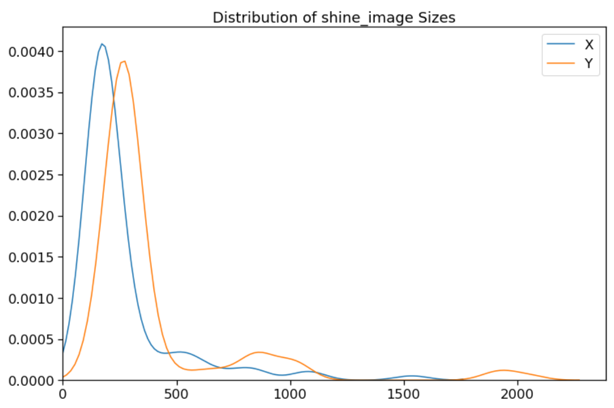
The size of images in Cloudy class varies from 250 to 1950 and the maximum of images lie in the range of 250x280. Similarly, maximum of the images in Foggy class lie in the range of 510x800.





We see here in Rainy image dataset that the images peak at around 510x480 in size and in Shine dataset the images peak at around 250x250.





Similarly, in Sunrise image dataset the images peak at around 250x250.

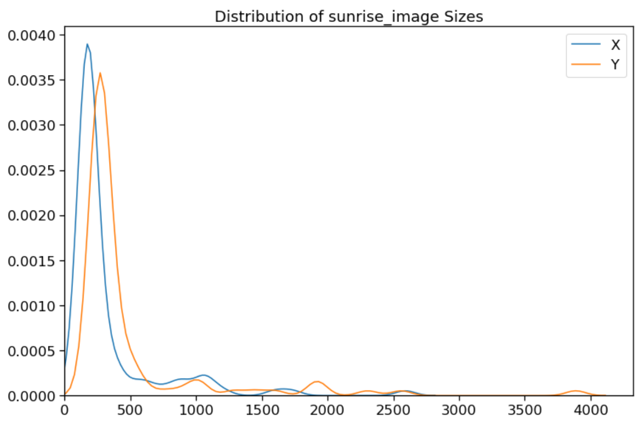


Fig. 5. Kernel density plot

So, for our input size I will choose the size as: 256x256.

2.3. Algorithms and Techniques

Transfer Learning:
Transfer learning refers to the process of using the weights of a pretrained network trained on a large dataset applied to a different dataset (either as a feature extractor or by finetuning the network). Finetuning refers to the process of training the last few or more layers of the pretrained network on the new dataset to adjust the weight. Transfer learning is very popular in practice as collecting data is often costly and training a large network is computationally expensive. Here, in this case the dataset is very small so weights from a convolutional neural network pretrained on ImageNet dataset is finetuned to classify weather condition.

Benchmark method:

- **CNN:** 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 perceptron (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.

  Convolutional: When programming a CNN, the input is a tensor with shape *(number of images) x (image width) x (image height) x (image depth)*. Then after passing through a convolutional layer, the image becomes abstracted to a feature map, with shape *(number of images) x (feature map width) x (feature map height) x (feature map channels)*. A convolutional layer within a neural network should have the following attributes:

  - Convolutional kernels defined by a width and height (hyper-parameters).
  - The number of input channels and output channels (hyper-parameter).
  - The depth of the Convolution filter (the input channels) must be equal to the number channels (depth) of the input feature map.

  Pooling: Convolutional networks may include local or global pooling layers to streamline the underlying computation. Pooling layers reduce the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, typically 2 x 2. Global pooling acts on all the neurons of the convolutional layer. In addition, pooling may compute a max or an average. Max pooling uses the maximum value from each of a cluster of neurons at the prior layer. Average pooling uses the average value from each of a cluster of neurons at the prior layer.

Fully Connected: Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron neural network (MLP). The flattened matrix goes through a fully connected layer to classify the images.

Receptive Field: In neural networks, each neuron receives input from some number of locations in the previous layer. In a fully connected layer, each neuron receives input from every element of the previous layer. In a convolutional layer, neurons receive input from only a restricted subarea of the previous layer. Typically, the subarea is of a square shape (e.g., size 5 by 5). The input area of a neuron is called its receptive field. So, in a fully connected layer, the receptive field is the entire previous layer. In a convolutional layer, the receptive area is smaller than the entire previous layer.

Weights: Each neuron in a neural network computes an output value by applying a specific function to the input values coming from the receptive field in the previous layer. The function that is applied to the input values is determined by a vector of weights and a bias (typically real numbers). Learning, in a neural network, progresses by making iterative adjustments to these biases and weights.
The vector of weights and the bias are called *filters* and represent particular features of the input (e.g., a particular shape).

Architecture of CNN: A 13-layered network is used to predict the weather condition of each class. The following layers are used in the ConvNet.

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 non-linearity.
- **MaxPooling:** Pooling layers reduces the dimensionality of the images by removing some of the pixels from the image. MaxPooling replaces a n x n area of an image with the maximum pixel value from that area to down sample the image.
- **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

Very Deep Convolutional networks (VGG): Winner of the ImageNet ILSVRC-2014 competition, VGGNet was invented by Oxford's Visual Geometry Group, The VGG architecture is composed entirely of 3x3 Convolutional and MaxPooling layers, with a fully connected block at the end. The pretrained model is available in Keras, TensorFlow, Caffe, Torch and many other popular DL libraries for public use.

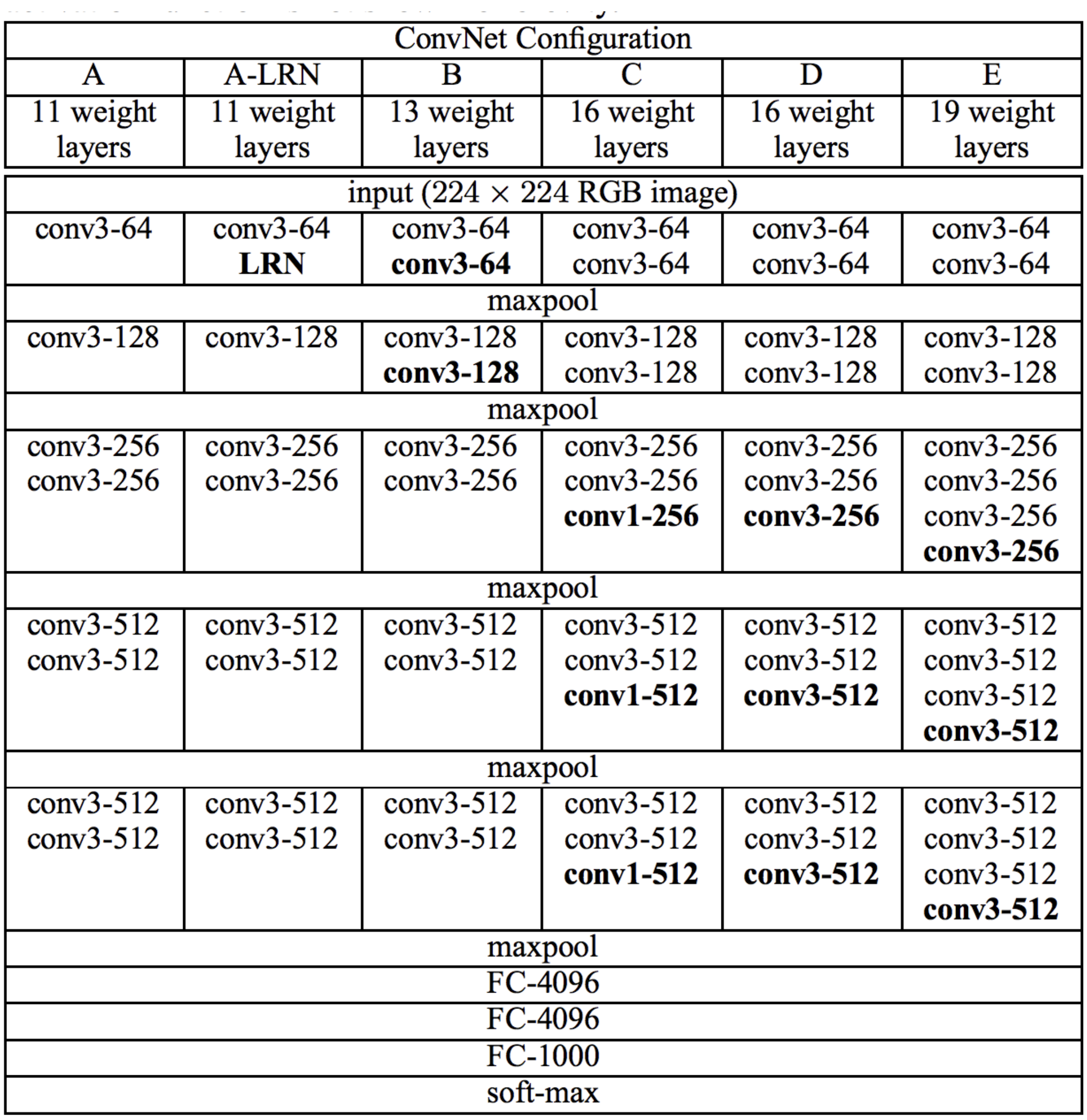
In the original paper, they have shown the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting. In the whole architecture they have used a very small (3x3) convolution filters, and shown a significant improvement on the prior-art configurations by pushing the depth to ***16-19*** weight layers. Their findings were the basis of ImageNet Challenge 2014 submission, where their team secured the first and the second places in the localisation and classification tracks respectively. They have made their two best-performing ConvNet models publicly available for further use.

Architecture of VGG:

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 non-linearity.
- **MaxPooling:** Pooling layers reduces the dimensionality of the images by removing some of the pixels from the image. MaxPooling replaces a n x n area of an image with the maximum pixel value from that area to down sample the image.
- **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 layers into the correct labels with some activation function (SoftMax used here)

Below is a table taken from the paper; note the two far right columns indicating the configuration (number of filters) used in the VGG-16 and VGG-19 versions of the architecture.



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Fig 6 Architecture of the VGG Convolutional Neural Network for Object Photo Classification (taken from the 2014 paper).

Residual Network (ResNet): Winner of the ImageNet ILSVRC & COCO-2015 competition, ResNet was invented by a group of researchers from Microsoft Research. The pretrained model is available in Keras, TensorFlow, Caffe, Torch and many other popular DL libraries for public use.

Their model had an impressive 152 layers. Key to the model design is the idea of residual blocks that make use of shortcut connections. These are simply connections in the network architecture where the input is kept as-is (not weighted) and passed on to a deeper layer, e.g. skipping the next layer.
A residual block is a pattern of two convolutional layers with ReLU activation where the output of the block is combined with the input to the block, e.g. the shortcut connection. A projected version of the input used via 1×1 if the shape of the input to the block is different to the output of the block, so-called 1×1 convolution. These are referred to as projected shortcut connections, compared to the unweighted or identity shortcut connections.

The authors start with what they call a plain network, which is a VGG-inspired deep convolutional neural network with small filters (3×3), grouped convolutional layers followed with no pooling in between, and an average pooling at the end of the feature detector part of the model prior to the fully connected output layer with a SoftMax activation function. The plain network is modified to become a residual network by adding shortcut connections in order to define residual blocks. Typically, the shape of the input for the shortcut connection is the same size as the output of the residual block.

Architecture of ResNet:

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 non-linearity.
- **MaxPooling:** Pooling layers reduces the dimensionality of the images by removing some of the pixels from the image. MaxPooling replaces a n x n area of an image with the maximum pixel value from that area to down sample the image.
- **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 layers into the correct labels with some activation function (SoftMax used here)

The image below was taken from the paper and from left to right compares the architecture of a VGG model, a plain convolutional model, and a version of the plain convolutional with residual modules, called a residual network.

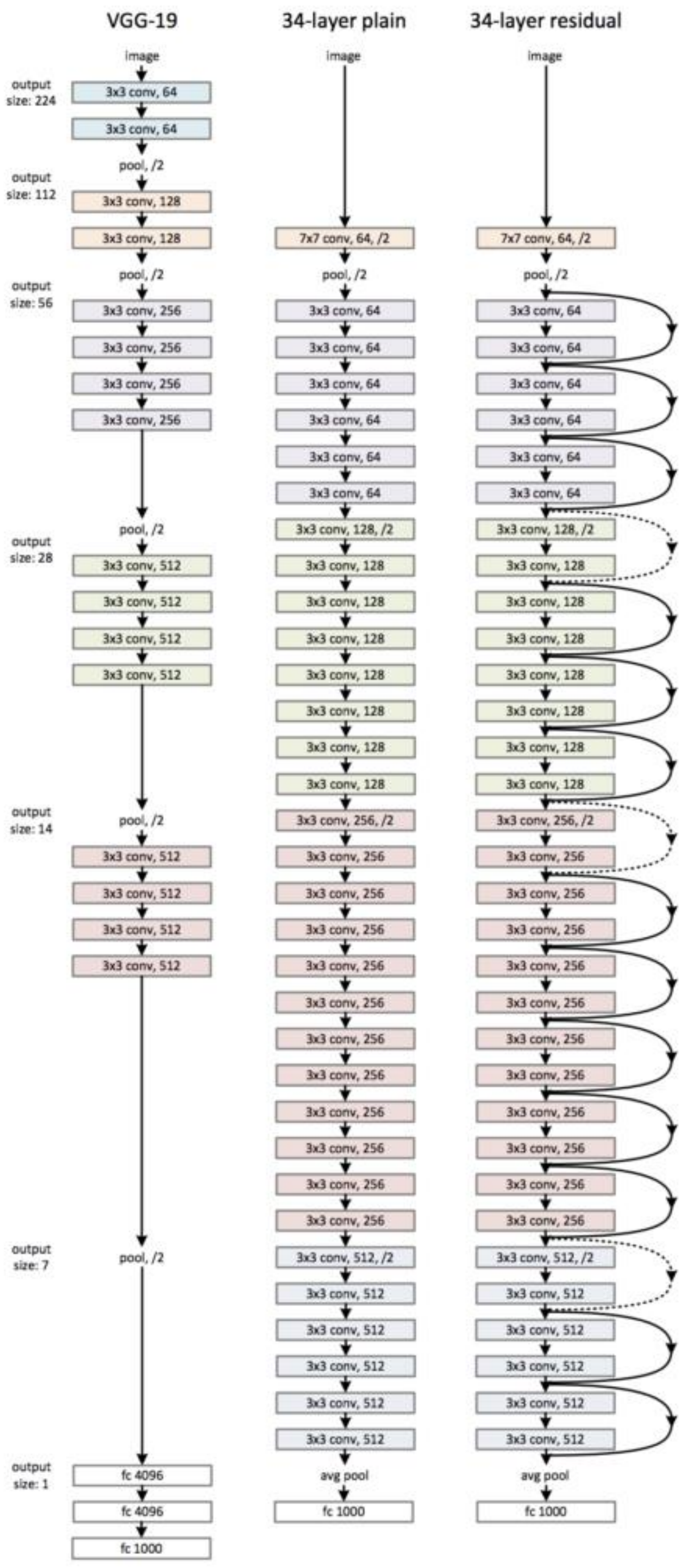


Fig. 7. Architecture of the Residual Network for Object Photo Classification (taken from the 2016 paper).

Activation functions: Activation layers apply a non-linear operation to the output of the other layers such as convolutional layers or dense layers.

- **ReLu Activation:** ReLu or Rectified Linear Unit computes the function $f(x)=max(0,x) to threshold the activation at 0.
- **SoftMax Activation:** SoftMax function is applied to the output layer to convert the scores into probabilities that sum to 1.

Optimizers:

- **Adam** (Adaptive moment estimation): is an update to RMSProp optimizer in which the running average of both the gradients and their magnitude is used. In practice Adam is currently recommended as the default algorithm to use, and often works slightly better than RMSProp. In my experiments Adam also shows general high accuracy while Adadelta learns too fast. I've used Adam in all the experiments because I felt having similar optimizer would be a better baseline for comparing the experiments.

Data Augmentation: Data augmentation is a regularization technique where we produce more images from the training data provided with random jitter, crop, rotate, reflect, scaling etc to change the pixels while keeping the labels intact. CNNs generally perform better with more data as it prevents overfitting.

Batch Normalization: Batch Normalization is a recently developed technique by Ioffe and Szegedy which tries to properly initializing 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 BatchNormalization 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.

2.4. Benchmark

CNN: This model predicts the probabilities for a weather class to belong to any class of the five classes for the naive benchmark. This model yields the log-loss of 0.927672 showing the best validation accuracy of 75.66%.

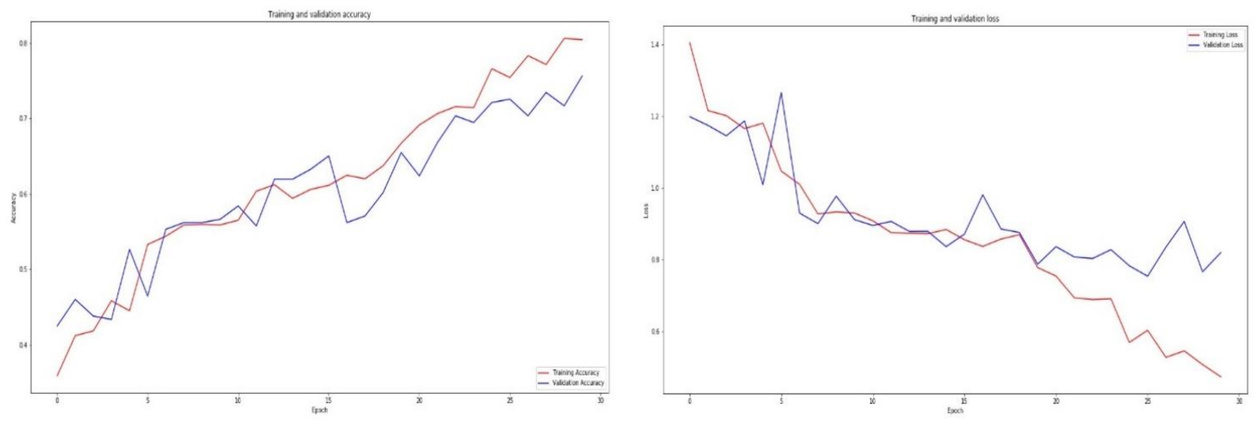


Fig. 8. Accuracy and Loss Curve of Conv Net (benchmark)

Since, after 16th epoch the validation accuracy is continuously lower than the training accuracy which indicates that the model has started to memorise the features i.e. the model is overfitting. The one and main reason for the overfitting of the model is the available dataset. Here, we have only 1274 images for training belonging to 5 different classes.

To overcome the problem of overfitting, we expand the training dataset artificially by creating modified versions of images. This can be achieved by data augmentation. This model yields the log-loss of 0.609387 and the best validation accuracy of 74.78%.

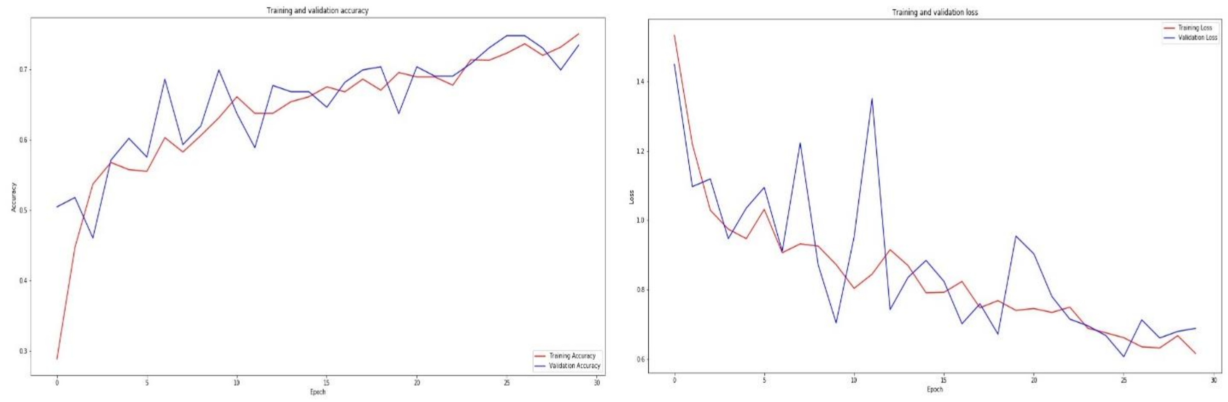


Fig. 9. Accuracy and Loss curve of ConvNet with Data Augmentation

A highly deep convolutional neural network with data augmentation should be able to beat the above model easily considering even the mode with augmentation clearly surpasses the initial benchmark of CNN. However, due to computational costs, it may not be possible to add more layers and run the model for sufficient number of epochs (more than 30) so that it may be able to converge.

So, the reasonable score for beating the CNN benchmark would be anything <0.609387 even if the difference is not large considering running the neural network longer would keep lowering the loss.

III. Methodology

3.1. Data Pre-processing

As per using ResNet, VGG like architecture for transfer learning, images are pre-processed as performed in the original architecture mentioned in paper. Creators of these original Networks took the different inputs for their respective Nets like in VGG16 they had subtracted the mean of each channel (R, G, B) first so the data for each channel had a mean of 0. Furthermore, their processing software expected input in (B, G, R) order whereas python by default expects (R, G, B), so the images had to be converted from RGB -> BGR.
For this purpose, a *preprocess_input* function, imported from *keras application* module, is used for respective network.
In this dataset input images also come in different sizes and resolutions, so they were resized to 256 x 256 x 3 to reduce size.
This dataset does not have any validation set, so it was split into a training set and a validation set for evaluation. Out of 1500 images, 1274 images are in the training set and the remaining (0.15% of all classes) are in the validation set. Note that instead of using *train_test_split* methods in *scikit-learn*, I randomly took 0.15% of each class from the dataset to the validation set while preserving the directory structure.

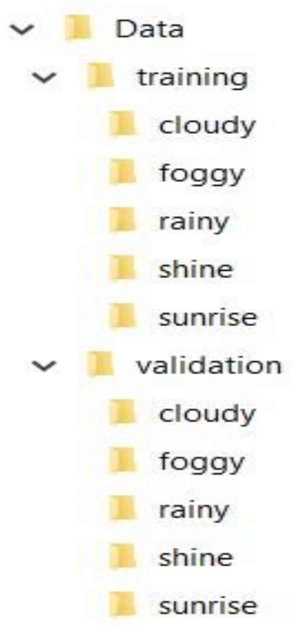


Fig. 10. Training and Validation dataset

It also preserves the distribution of the classes as visualized below.

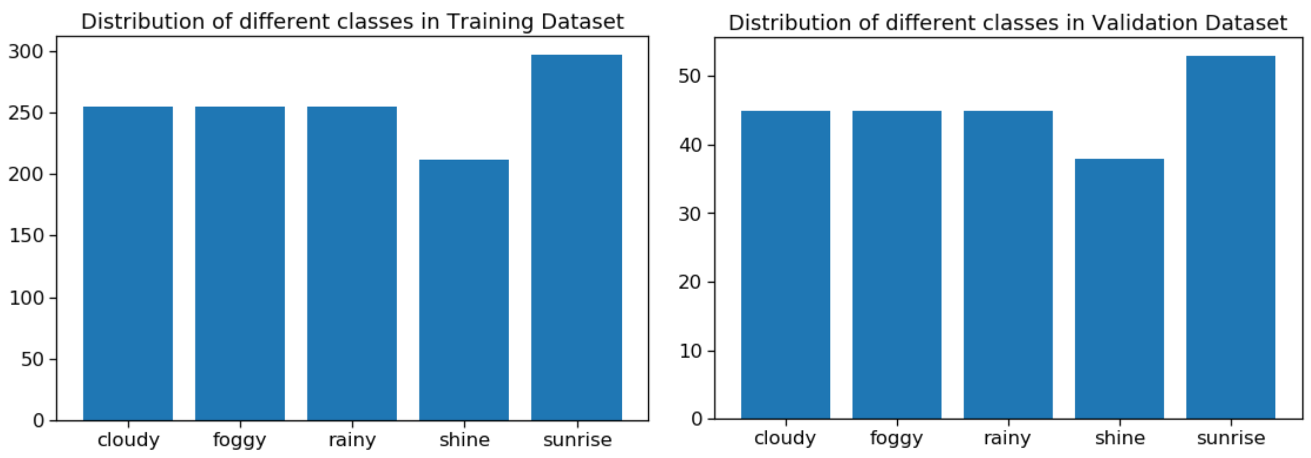


Fig. 11. Distribution of Training and Validation dataset

Data Augmentation: Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset.
Training deep learning neural network models on more data can result in more skilful models, and the augmentation techniques can create variations of the images that can improve the ability of the fit models to generalize what they have learned to new images.
Here, in our case the number of images is very less (only 1274 training images belonging to all class). So, Data-augmentation will help in increasing the number of images (creating variations of images) and supply the images to the model in batches. The images are not repeated in batches so it also helps in preventing the overfitting of model.

Keras ImageDataGenerator (only rotation_range, zoom_range and horizontal_flip was used) generate training data from the directories/NumPy arrays in batches and processes them with their labels. Training data was also shuffled during training the model, while validation data was used to get the validation accuracy and validation loss during training.

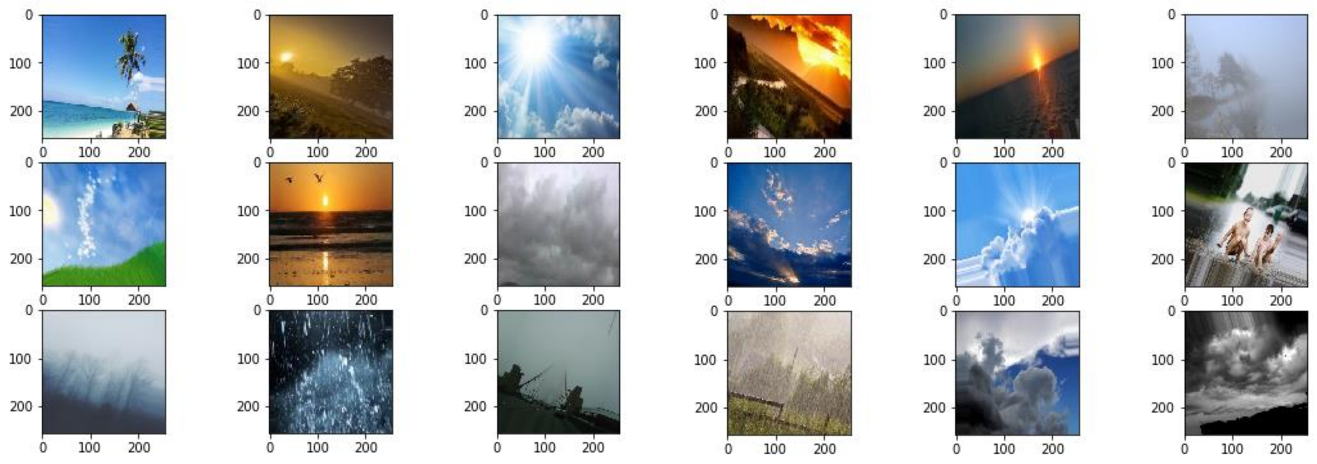


Fig. 12. A random sample of artificially generated data by Data Augmentation

3.2. Implementation

Initially the baselines with a deep ConvNet with and without augmentation were implemented for comparison. After that the models with transfer learning were used.
So, VGG and ResNet architecture without the last fully connected layers were used to extract the convolutional features from the pre-processed images.
On the extracted features (CNN codes), a small fully connected model was applied first but unfortunately it didn't have a good result. After that I applied dropout and batch normalization to the fully connected layer which beat the CNN benchmark by 17.50.

3.3. Refinement

I applied batch normalization in the model to prevent arbitrary large weights in the intermediate layers as the batch normalization normalizes the intermediate layers thus helping to converge well. Even in the model with batch normalization enabled during some epochs training accuracy was much higher than validation accuracy, often going near about 98-99%. Since the data set is small (only 1274 training images & 226 validation images) it's definitely plausible our model is memorizing the patterns. To overcome this problem, data augmentation was used. Data Augmentation alters our training batches by applying random rotations, zooming, flipping etc.

In the specific dataset, vertical flipping does not make sense because the camera would always be in a fixed position and the weather photos wouldn't be captured upside-down. I've added random rotation because it's possible the cameras are going to move from one corner to another to cover a broad area. I've also added horizontal flipping and random shifting because all these scenarios are likely.

Unfortunately, the model with data augmentation is computationally expensive and takes around 40 minutes per epoch on my machine, so I've trained the model for 30 epochs and the best validation accuracy is 95.13%. Log-loss for this model is 0.080738, which is an 86.75% decrease in log-loss. If I could train the data augmented model for a few more epochs or with greater filter size it'd probably lead the log-loss close to 0.

IV. Results

4.1 Model Evaluation, validation and justification:

For each experiment only the best model was saved along with their weights (a model only gets saved per epoch if it shows higher validation accuracy than the previous epoch)

To recap, the best model so far uses transfer learning technique along with data augmentation and batch normalization to prevent overfitting. To use transfer learning, I've collected pretrained weights for the VGG and ResNet architecture from the keras official GitHub page and used the similar architecture only with replacing the fully connected layers with different dropout and batch normalization. I had used less aggressive dropout in my models.

I've pre-processed all the images according to VGG and ResNet architecture directions (using the *preprocess_input* function from *keras_applications* module). For the final model I used the base model of ***ResNet101*** excluding the fully connected layers along with the pretrained weights, added two new Dense layers with dropout and batch normalization on top of it and on top of these two, a dense layer with *softmax* activation function to predict the final images. This model beats the CNN benchmark by 95.297% decrease and the CNN with augmentation model by 86.75% decrease of multi-class log-loss. A table with all the experiments performed is given below along with their results.

Model	**Multi-class log-loss score**
CNN (Benchmark)	0.927672
CNN with augmentation (Benchmark)	0.609387
VGG16 with Data augmentation + Dense Layers	1.998586
VGG16 with Data augmentation + Dense Layers + Dropout	1.383569
VGG16 with Data augmentation + Dense Layers + Dropout + BatchNormalization	0.317465
VGG19 with Data augmentation + Dense Layers + Dropout + BatchNormalization	0.936093
ResNet50 with Data augmentation + Dense Layers + Dropout + BatchNormalization	0.215166
ResNet101 + Data augmentation + Dense Layers + Dropout + BatchNormalizatio	**0.080738**
ResNet152 with Data augmentation + Dense Layers + Dropout + BatchNormalization	0.360206

In the test data (randomly downloaded from internet) out of 30 images, 29 images are classified accurately and 1 image is incorrect.
The confusion matrix(non-normalized) plot of the predictions on the test data is given below.

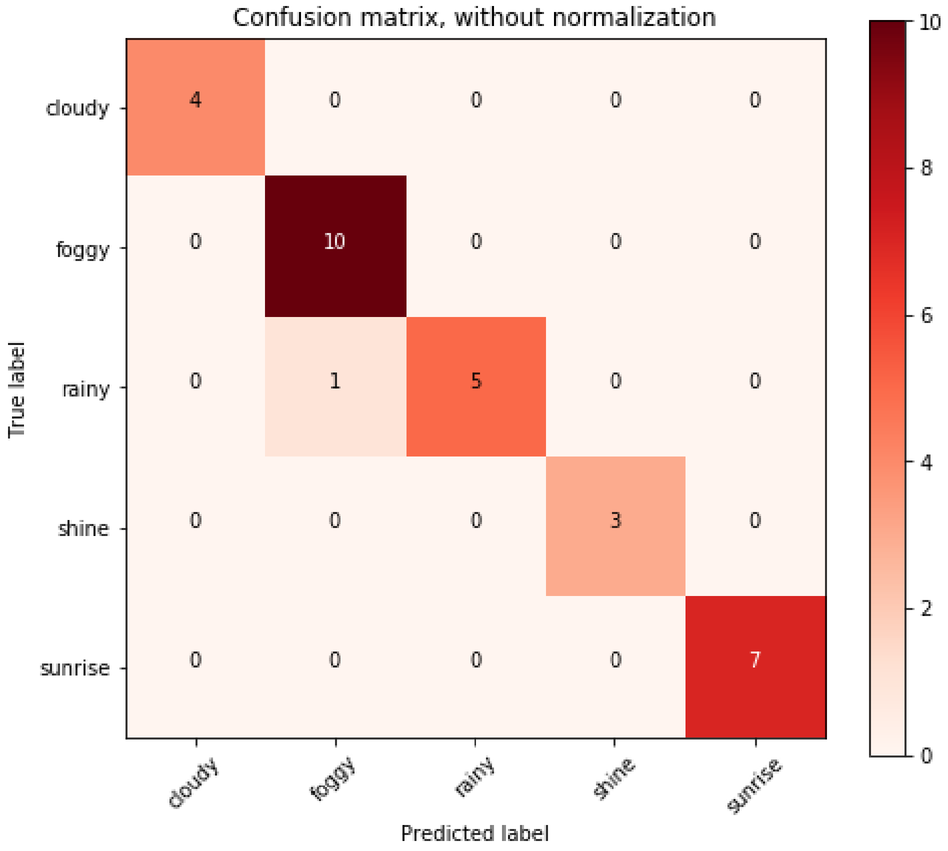


Fig. 13. Confusion Matrix of predicted test dataset

As seen from the confusion matrix, this model is really good at predicting all the classes. There is only 1 wrong prediction among 30 images.

The normalized confusion matrix plot of the predictions on the validation set is given here.

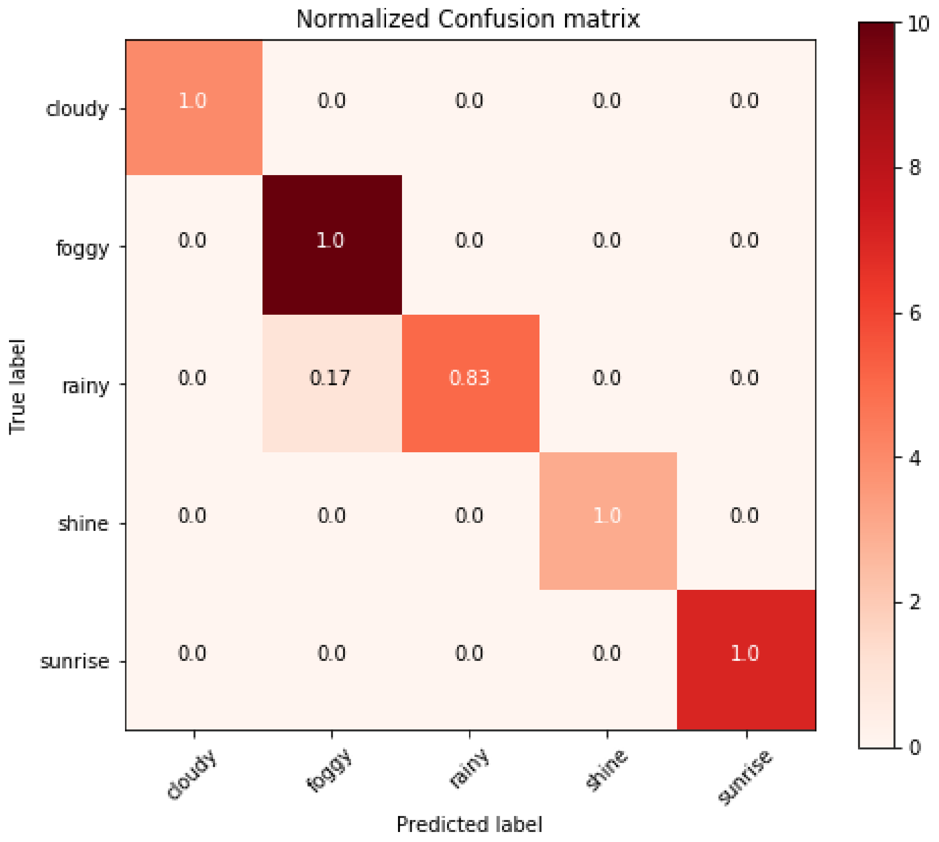


Fig. 14. Normalized Confusion Matrix of predicted test dataset

As data augmentation was used to train this model, it can also handle slight variations in the images such as horizontal flip, different illuminations, rotations and shifting up and down which are the scenarios of real-life images.
Data leakage is an issue in this problem because most images look very similar. On top of that, images were of different sizes, to overcome that issue, resizing each image was important. I'd have had to resize for feeding them into CNN in any case, however, resizing also was important to avoid data leakage issues.

V. Conclusion

5.1. Free Form Visualization

As I've recorded the accuracy and loss curve of the models per epoch, the final model's graph can be shown below. Here's the accuracy/loss graph of the model with data augmentation, dropout and batch normalization.

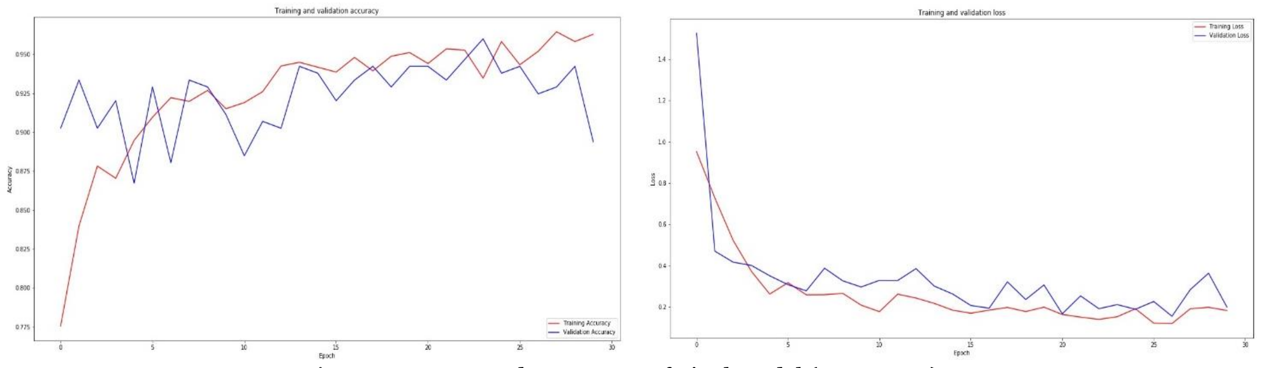


Fig. 15. Accuracy and Loss curve of Final model (ResNet101)

As we can see the training accuracy is crossing 96% in the diagram and the loss is close to 0.15. Similarly, the validation accuracy is also about 95.13% while the validation loss is around 0.17 near the end of the 24 epochs.

5.2. Reflection

For reaching into this end to end solution, I've tried to progressively use more complex models to classify the images. I've tried a baseline convolutional model as a good practice because I wanted to see how the model performs with a conv model with a good 13 number of layers and it gives training accuracy of 80.46% while the validation of 75.66%, clearly showing the sign of overfitting after 16th epoch. So, coming to the point of using Data Augmentation, the accuracy of the model was almost same while the loss was decreased (as discussed above section) and the problem of overfitting was resolved. I, then applied transfer learning and experiment with running different versions of VGG and ResNet models pretrained on ImageNet dataset. Only after applying batch normalization instead of the fully connected model I saw significant improvement, and so I used it with the VGG and ResNet architecture

The most difficult part for me was to get the experiments running on my local machine (with 4gb RAM & AMD A8 processor). Higher computational time results in lower number of experiments when it comes to neural networks, especially when I'm just figuring out what to do as it's my first experience with deep learning. However, as I feel more confident in my skills in using deep learning now, I'll definitely try to seek more computational resources from now on.

5.3. Improvement

Due to time and computational cost it was not possible for me to run more experiments using different known architectures other than VGG & ResNet such as Inception, EfficientNet for this dataset. For the Dense layer I could not use the larger filter size. It's definitely possible that a different architecture would be more effective.

References :

- https://creativecommons.org/
- https://unsplash.com/
- https://www.pexels.com/
- https://www.mendeley.com/?interaction_required=true
- https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
- https://github.com/keras-team/keras-applications/releases/