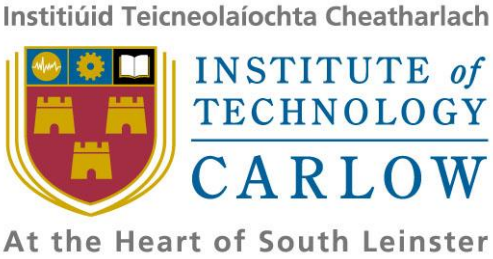


Using Deep Learning to Classify and Study the Formation and Meso-scale Organisation of Stratocumulus Clouds

Master of Science in Data Science

Course Code: CW_SRDAT_M

Year: 2019-2020

Supervised By: Dr Greg Doyle

Submitted By: Abhishek Mandal

Student ID: C00253536

Date of Submission: 12th August 2020

Word Count: 15120

ACKNOWLEDGEMENT

I would like to thank Dr Greg Doyle for his supervision and guidance for this project. This project wold not be possible without his help. I thank the scientists and researchers at TU Munich, Max Planck Institute for Meteorology, Hamburg and Sorbonne University, Paris for curating and labelling the satellite images of stratocumulus clouds that has been the cornerstone of this project and making the data publicly available.

I would also like to thank the National Aeronautics and Space Administration for providing the Earth observation data from their satellites to the public for free. I could not have done this project without it.

I would like to thank all the professors at IT Carlow for teaching me all the latest data science tools and technologies.

Lastly I thank all my friends and family members for their constant encouragement and wishes.

DECLARATION

• I declare that all material in this submission e.g. thesis/essay/project/assignment is entirely my/our own work except where duly acknowledged.

• I have cited the sources of all quotation, paraphrases, summaries of information, tables, diagrams or other material; including software and other electronic media in which intellectual property rights may reside.

• I have provided a complete bibliography of all works and sources used in the preparation of this submission.

• I understand that failure to comply with the institute's regulations governing plagiarism constitutes a serious offence.

Student Name : Abhishek Mandal

Student Number : C00253536

Signature: _____

Date: _____

ABSTRACT

Climate change has thrown many challenges before mankind today. They range from rising sea levels to unpredictable weather events causing huge human and material losses. As such, creating better and more accurate weather models is more important than ever before. However current weather models often fail to accurately predict weather events, mostly due to gaps in these models. One such gap is the understanding and representation of stratocumulus clouds. Stratocumulus clouds are shallow clouds that occur at about 2000m height and play an important role in maintaining the Earth's radiation and heat balance. Yet they have not been properly understood and represented in weather models. The first attempt at classifying these clouds was done only in 2018. Scientists classified stratocumulus clouds into four categories: 'Sugar', 'Fish', 'Flower' and 'Gravel'.

However, this classification was done manually and as such was limited to a few hundred satellite images. This project aims to do this on a global scale using deep learning algorithms involving thousands of such satellite images. In the past few years, deep learning algorithms like Convolutional Neural Networks have shown tremendous promise in the field of image recognition, reaching human-level accuracy in many cases.

This project has three major aims. The first aim is to create a deep neural network that can classify satellite images of stratocumulus clouds into the four aforementioned categories. Latest deep learning models such as UNet, ResNet and Xception has been used for this purpose. The second aim is to see if there is a relationship between any of these cloud categories and weather events like cyclones. Visualisation technologies such as Altair and Matplotlib has been used for this part. The third aim is to study the effect of rising global temperatures caused by climate change and the formation and meso-scale organisation of these stratocumulus clouds. The aforementioned visualisation technologies have been used for visual analysis along with time-series models such as ARIMA, which has been used for forecasting formation of clouds in the near future.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ACF	Auto-Correlation Function
API	Application Programming Interface
ARIMA	Auto Regressive Integrated Moving Average
AUC	Area Under Curve
AWS	Amazon Web Service
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
ELU	Exponential Linear Unit
FN	False Negative
FP	False Positive
GPU	Graphics Processing Unit
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IOU	Intersection Over Union
NASA	National Aeronautics and Space Administration
PR CURVE	Precision-Recall Curve
RADAM	Rectified Adaptive Moment Estimation
RELU	Rectified Linear Unit
RLE	Run-Length Encoding
ROC	Receiver Operating Characteristics
TN	True Negative
TP	True Positive
TPU	Tensor Processing Unit

Chapter 1. Introduction

Climate change is one of the biggest challenges faced by humankind today. Due to the activities of humans post the industrial age, huge amounts of greenhouse gasses like carbon dioxide, methane, and water vapour have been emitted into the Earth's atmosphere. This has significantly altered the Earth's chemical and physical processes. This has resulted in rising sea levels, increasing acidity of oceans, melting of glaciers and polar ice caps among many other things. These changes have led to unpredictable weather conditions like flash floods and torrential rains and frequent and bigger cyclones. As a result, creating accurate weather models has become difficult and weather predictions tend to deviate from real events. An example would be the recent storm 'Elsa' which hit Galway and Met Éireann was criticized for not giving warnings before (O'Brien, 2019). This shows how difficult weather modelling has become, mostly due to changing weather patterns.

Stratocumulus clouds are shallow clouds forming below 2000 meters and occur in groups of large masses following a wave or line-like pattern (International Cloud Atlas, 2019). These clouds play an important role in maintaining the Earth's radiation balance and temperature. They act as cooling agents of the Earth's lower atmosphere by reflecting solar radiation. However, they are not yet properly represented in weather models (Rasp et al., 2019).

In 2018, to properly understand stratocumulus clouds, scientists decided to classify them according to their grouping characteristics (Stevens et al., 2019). They came up with four categories: sugar, gravel, fish, and flower. About 900 satellite images were classified by a team of six scientists.

Further, in 2019, scientists at TU Munich, Max Planck Institute for Meteorology, Hamburg and Sorbonne University, Paris created a crowd source project to classify 10,000 satellite images by volunteers (Rasp et al., 2019).

Recently scientists at the California Institute of Technology, USA published a paper in which, they predicted that stratocumulus clouds would vanish completely in the near future due to rise in global temperature (Schneider, Kaul and Pressel, 2019). If this was to happen, it would further increase the global mean surface temperature, thereby contributing to runaway global warming.

This project attempts to carry further the work done until now and build upon the work done by these scientists.

1.1. Scope and Rationale

The process of identification and classification of clouds from images needs to be done on a large scale in order to generate large enough dataset. As manually doing this would be extremely time consuming, we would be using artificial intelligence to automate this process. Recent advancements in deep learning has made neural networks good enough to make human level understanding of images possible. In some cases, neural network based models have even surpassed humans in this task (Tsang, 2019).

The aim of this project is three folds and is divided into three parts. They are explained below.

- The first aim which makes up the first part is to create a deep neural network that can classify stratocumulus clouds into four categories: fish, flower, sugar and gravel. These categories were created by the scientists at TU Munich and Max Planck Institute of Meteorology. The categories are based on how clouds organise themselves upon formation in the Earth's lower atmosphere. This is also called meso-scale organisation.
- The second aim is to study the relationship between this meso-scale organisation of stratocumulus clouds and weather events such as tropical cyclones. Due to time constraints, we will only consider tropical Atlantic cyclones.
- The third part aims to study the effect of global warming on these cloud formations.

The findings from this experiment may aid scientists in better understanding of an important but poorly understood weather phenomenon. This, in turn, can help in creating better and more accurate weather models which can lead to more accurate weather predictions and thus minimise human and material losses in these changing and uncertain times.

My main contributions to this project are: (1) An exhaustive research of current state-of-the-art image processing and computer vision algorithms. (2) Creating two deep neural networks based on segmentation model architecture: UNet-ResNet and UNet-Xception. (3) Training, testing and evaluating the models. (4) Collecting image data from the NASA worldview website for part 2 and 3 of the project. (5) Using the better performing model for identifying and classifying clouds from the gathered images. (6) Performing exploratory data analysis for the second and third part of the project. (7) Creating a time-

series data from the classified images. (8) Creating an ARIMA model and forecasting cloud occurrences for the next ten years. (9) Drawing conclusions from the analysis and summarising the results.

The structure of this paper is as follows: Chapter 1 is the introduction which gives a brief overview of the project listing out the work done over the course. Chapter 2 provides a review of the literature that was studied in preparation for the project, detailing out the background and various technologies used in the project. Chapter 3 is laid out in a way showing the progress of the work and the methodology followed to achieve the objectives. It shows the working and construction of convolutional neural networks, segmentation models, and other algorithms and tools used. The pseudocode for most of the relevant code is also provided. A few results that were necessary for making decisions such as, model performance are also provided. Chapter 4 discusses the findings and results. Most of the exploratory data analysis is discussed here along with the outcomes of each part. Chapter 5 concludes the project, discusses the limitations, and gives recommendations for future work. Chapters 3, 4, and 5 are further divided into three parts, each dealing with the respective part of the project.

1.2. Research Objectives and Questions

The research objectives as discussed above are summarised here along with their research questions.

Part 1: Research Objective 1: To create a convolutional neural network based classifier, which can identify and classify stratocumulus clouds from satellite images.
Research Question 1: Is it possible to create a deep learning model that can identify and classify stratocumulus clouds from satellite images?

Part 2: Research Objective 2: To study the relationship between stratocumulus cloud categories and tropical cyclones.
Research Question 2: Do tropical cyclones affect the formation and meso-scale organisation of stratocumulus clouds?

Part 3: Research Objective 3: To study the effect of a rise in global temperature and the formation and meso-scale organisation of stratocumulus clouds.

Research Question 3: Does the rise in global temperature affect the formation and meso-scale organisation of stratocumulus clouds?

Chapter 2. Literature Review

2.1. Stratocumulus Clouds

Stratocumulus clouds are large dark coloured clouds forming in patterns of lines and waves generally forming at or below a height of 2000 meters. They are generally found over polar seas and tropical oceans (Arakawa and Schubert, 1974). They play an important role in maintaining the Earth's radiation balance. In the subtropical region, they form in the region between 30 and 35 degrees latitude and block the Sun's radiation. This also reduces the amount of solar energy being absorbed by the ocean, thus reducing average ocean temperature. Overland, they reduce the local temperature and create a shadow region creating a 'dull weather' (Stratocumulus (Sc) | International Cloud Atlas, 2020).

(Bretherton, 2015) studied the mesoscale organisation of shallow cumuli and the effect of warming on this. However, they are not yet properly represented in weather models (Rasp et al., 2019). The effect of global warming on the organisation and formation of stratocumulus clouds is also not properly understood and is under active study (Bony, Schulz, Vial, and Stevens, 2020).

2.2. Classification of Stratocumulus Clouds

The first attempt at categorising stratocumulus clouds was done way back in 1974 by Arakawa and Schubert (Arakawa and Schubert, 1974). However, the creation of proper categories and labels for them was only done in 2018 (Stevens et al., 2019). The four categories chosen to classify the clouds were: sugar, gravel, fish, and flower.

'Sugar' are large distributions of fine cumulus over a large area which not much reflective. Sometimes they form thin vein-like structures called 'Feathers'. They have been previously labelled as 'dendritic' clouds. 'Flowers' are circular cloud formations with the diameter in the range of 52 to 200 kilometers. They have large cloud free spaces between them. 'Fish' are long skeletal cloud formations spanning longitudinally up to 1,000 kilometres. 'Gravel' are circular or arc-like clouds with each arc about 20 kilometres in length (Rasp et al., 2019).

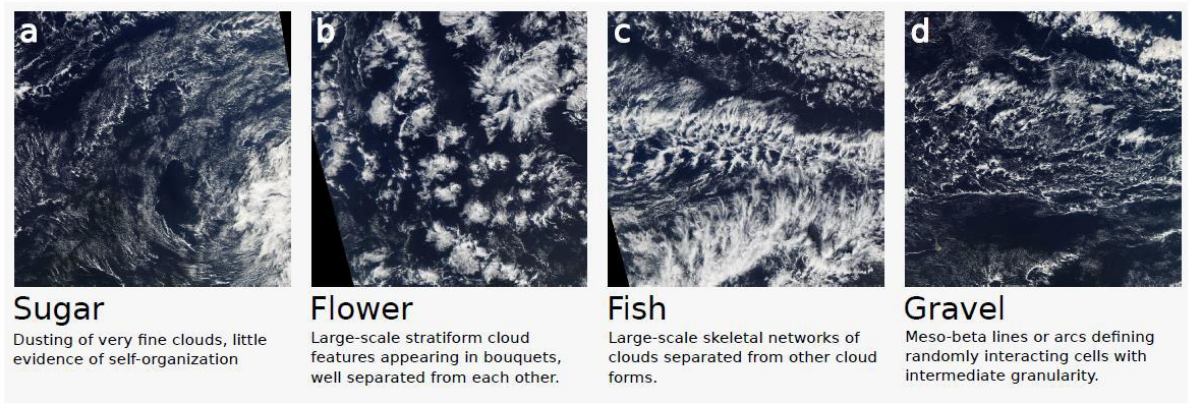


Fig 2.1: Categories of clouds (Rasp et al., 2019)

Around 900 images were manually classified by scientists in 2018. In order to properly understand stratocumulus clouds, this is needed to be done on a large scale which is not feasible if done manually. In 2019, scientists decided to use machine learning to classify about 10,000 images. The use of machine learning to identify patterns in clouds will be discussed in the following section.

2.3. Deep Learning

(Deng and Yu, 2014) describes deep learning as

"A class of machine learning techniques that exploit many layers of non-linear information processing for supervised or unsupervised feature extraction and transformation, and for pattern analysis and classification."

(Claude and Webb, 2017) defines machine learning as Computer systems which can perform a specific task without using explicit instruction but using patterns and inference instead. Machine learning systems create a mathematical model based on the training data and use it to base future predictions on. Deep learning is a further specialization of machine learning.

Deep learning, like all other machine learning algorithms, converts a higher dimensional vector i.e. input to a lower dimensional vector i.e. output. The difference between other machine learning algorithms and deep learning is that deep neural networks (neural networks employed in deep learning) extract the features of the data themselves while for other algorithms, this has to be done manually (Saha, 2019).

Neural networks are computing systems or units modelled on the neural networks found in human brains. Neural networks are made up of artificial neurons, modelled on the

biological neuron, which receive inputs from other neurons, process them and pass on to other neurons. Their operation is similar to biological neuron. Neural networks used in deep learning are called deep neural networks or deepnets (Wang, 2003).

These deep neural networks have revolutionised industries, transformed businesses and opened new vistas of possibilities not conceivable barely a decade ago. It has made cars self-driving, allowed blind people to read, defeated world champions in Go and created music of its own.

2.4. Deep Learning in the field of Image Processing and Computer Vision

An area where deep learning has had a profound effect is computer vision and image processing. Advancements in this area made certain things like self-driving cars, facial recognition possible which were earlier in the realms of science fiction. From an engineering perspective, computer vision aims to automate the tasks that humans do using their sight. Although work in this field began as early as 1960, practical applications could only be possible in the 21st century. The reason behind this is the advancement of deep learning algorithms particularly convolutional neural networks. Convolutional Neural Networks (CNN) have consistently broken records on various benchmarks computer vision datasets (Das, 2019)

Convolutional Neural Networks (CNN) are a class of deep neural networks that uses convolution in at least one of their layers (Goodfellow et al., 2016). Convolution is a linear mathematical operation in which two functions produce a third function which tells how the shape of one function is affected by the other. A CNN typically contains an input layer, an output layer, several convolutional layers, several pooling layers and RELU layers. Convolutional layers carry out the convolution operations and pass the output on to the next layer. Pooling layers combine the output of several layers and reduce the dimensions of the data. The RELU layer works as an activation function.

In the past few years, CNNs have consistently won every image recognition benchmarks. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC), which evaluates image detection and classification algorithms hosts annual image recognition competitions. CNNs have won this competition every time since 2012 (Russakovsky et al., 2015). Even the improvements in CNN models themselves have been astounding. ILSVRC measures object detection accuracy by mean average precision (mAP) and the

increase in mAP from ILSVRC2013 to ILSVRC2014 was 43.9%. By ILSVRC2017, the winner used a CNN called Squeeze-and-Excitation Network (SENet) which had a classification error of 0.023% (Tsang,

2019). This shows that CNNs have reached human level accuracy in object detection and classification.

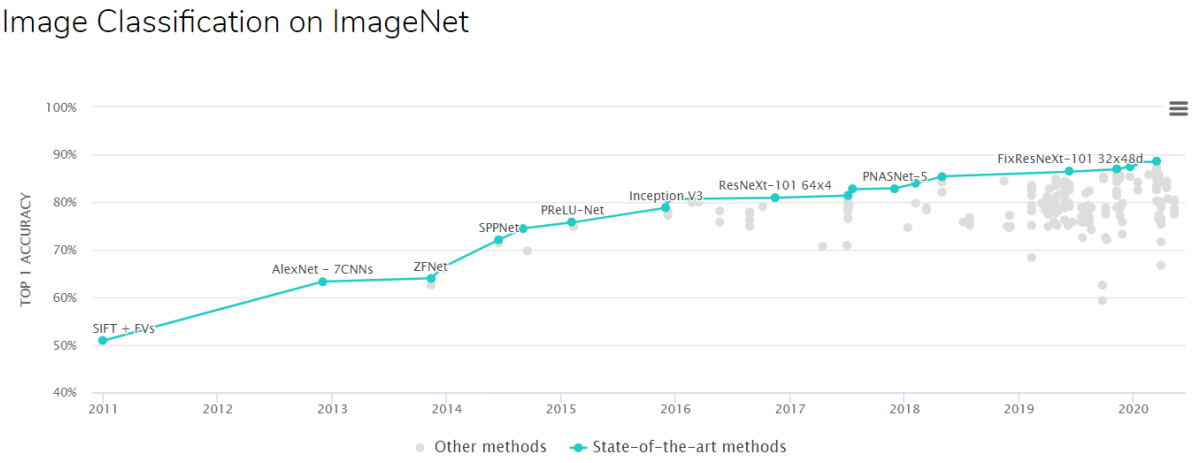


Fig 2.2: Top1 Accuracy of best ImageNet challenge winners from 2011-2020 (Papers with Code - ImageNet Leaderboard, 2020)

So, has deep learning reached a level where it can replace humans for vision related tasks? As per (Hosseini et al., 2017), deep learning networks like CNNs require a very large dataset to train. These datasets can lead to biases and can cause the network to fail when given images outside of the dataset. The researchers tested some of the benchmark setting CNNs like LeNet, Inception, etc. on negative images of some very popular datasets like CIFAR-10 and MNIST and compared the accuracy of the networks with their accuracy with the original images for image classification. The results were staggering. LeNet-5 had an accuracy of 99.21% on the MNIST dataset while the accuracy drops to 34.65% for the negative images of the same dataset. MVCG-8 network had an accuracy of 98.54% on the GTSRB-color dataset while the accuracy drops to 12.66% for the negative images of the same dataset. Humans for instance had an accuracy of 98.48% and 97.31% for original and negative images of the GTSRB-color dataset respectively. The reason behind this massive drop in accuracy is a fundamental problem of deep learning networks. Deep learning networks do not ‘understand’ the data, rather they ‘memorize’ the data. So, when presented with an image which they have not ‘memorized’, in this case, the negative images, they fail badly. Humans, on the other hand, possess an incredible ability to recognize unfamiliar objects. Humans try to understand an unfamiliar object by looking at its shape, size, color, orientation, etc. and try to infer from them. CNNs lack this ability.

Also, CNNs are highly sensitive to any changes in the data (Yuille and Liu, 2019). CNNs fail when presented with photoshopped images. When researchers (Wang et al., 2018) presented a CNN with a photoshopped image of a monkey holding a guitar, it promptly classified the monkey as a person. It would be difficult to fool a human in such a way. A solution to these types of problems can be to create a training set of different scenes by picking objects from an object dictionary. But the number of such combinations may be infinite and as such, the size of the training dataset will increase infinitesimally. This raises the problem of combinatorial explosion (Yuille and Liu, 2019). However this problem of combinatorial explosion is limited to certain scenarios like self-driving-cars where anything can be present on the road ranging from humans, different vehicles, animals etc. but not so in the case of medical imaging (e.g. the heart is always close to the lungs and all the organs present in the human body are known). This is one of the reasons why deep learning is so successful in the field of medical imaging, but we are yet to see fully functional self-driving-cars on the streets.

2.5. Using Convolutional Neural Networks to Classify Images of Clouds

(Rasp et al., 2019) first explored the possibility of applying deep learning for classifying stratocumulus clouds in 2019. First, they created an online crowd sourcing program to manually classify the satellite images on Zooniverse.org (sugar, flower, fish or gravel, 2020). Then a team of 67 researchers at the Max Planck Institute of Meteorology, Hamburg, Germany and at the Laboratoire de Meteorologie Dynamique in Paris, France classified the images into four categories – flower, fish, gravel and sugar (Rasp et al., 2019).

With more than 10,000 human labelled images, they set to automate the process of classifying further images. They used two deep learning models, one for object detection and another for semantic segmentation. For object detection, they used a model called Retinanet which is based on a Resnet50 backbone. The original images of 2100x1400 pixels were downsized to 1050x700 pixels resolution for fitting the input to the Resnet50 into batch sizes of 4 (Combining crowd-sourcing and deep learning to understand meso-scale organization of shallow convection, 2020).

For the semantic segmentation part, each human classification, i.e. area marked by a user as belonging to a particular cloud type, was converted to a mask. Where two or more such areas coincided, the value of the smaller area was chosen for the mask. For the segmentation model, a U-Net model architecture with Resnet50 backbone was chosen.

The images were downscaled to 700x466 pixels to fit the model with batch size of 6 (Combining crowd-sourcing and deep learning to understand meso-scale organization of shallow convection, 2020).

2.6. Survey of Convolutional Neural Network Architectures for Multi-Class Classification of Images

Convolutional Neural Networks or CNNs takes image as input and assign weights and biases in order to classify images from one another. The first step to do this involves feature detection and extraction. This step involves recognising features from images such as edges, boundaries and colour. This, the CNN achieves by performing a series of non-linear transformations such as convolution (Wei et al., 2016). These features form the basis for classification. CNNs have shown extraordinary ability in classifying images (Russakovsky et al., 2015).

Several recent works (Girshick, Donahue, Darrell and Malik, 2016), (Wei et al., 2016) have shown that CNN models trained on a variety of data such as Resnet and Imagenet, can be used for feature extraction on other datasets without needing much training data. This technique is called transfer learning.

In this project, we will use transfer learning extensively for creating the deep learning segmentation models. As creating a full network and training it is a very time consuming and resource intensive process, most machine learning practitioners use transfer learning. In deep learning, transfer learning is the process of training a model on a particular problem and using it to solve a similar problem. As such the information gained during the previous training is re-used for the latter problem. A few layers of the original pre-trained model are changed, and the parameters tweaked as required. Then the modified model is retrained on the particular dataset, provided for the problem. This reduces the number of training iterations considerably. This approach is very popular in computer vision problems.

A number of deep learning models have been developed for and tested at the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). These models have been trained on millions of images across thousands of categories. As such, they learn useful information about various real life images. This include spatial information such as position of body parts of humans and other animals (head is always connected to the neck), colour information such as plants are generally green in colour among many other similar features.

While learning these features, the weights of the layers are optimised and hence while reusing the model, these weights can serve as a starting point for further optimisations. These pre-trained models are available as a part of standard libraries in most of the popular deep learning frameworks such as Keras (Keras documentation: Keras Applications, 2020). Although these models can be used in standalone way with or without any modifications, they can also be used for a lot of other functions. These include using the models for feature extraction and weight initialisation for a different model. For using a pre-trained model for feature extraction, either the model or part of it is used to extract the features and pass the extracted features onto a different model, or the model is integrated into a different model with only the feature extracting layers are used and rest of the layers are frozen. They can also serve as weight initialisers for a network, where the weights of a pre-trained model are taken, and the values given to the layers of a different network. This can only work if both the networks have the same architecture (Brownlee, 2020). Both these methods of using transfer learning has been used in this project.

In this section a few of these popular pre trained models are explored. These include AlexNet, ResNet, Inceptionv3 and Xception.

AlexNet: AlexNet is a convolutional neural network designed by (Krizhevsky, Sutskever and Hinton, 2017) that participated in the Large Scale Visual Recognition Challenge (ILSVRC 2012) and won the first place. They used Graphical Processing Units (GPU) provided by Nvidia for training the model. The architecture contained five convolution and three fully connected layers. It used ReLU (Rectified Linear Unit) for activation instead of the more commonly used tanh or sigmoid functions. AlexNet's input consisted of RGB images of 256x256 pixels resolution. All the images in the training set were downscaled to this resolution. The grayscale images were converted to RGB by reproducing the single channel to get a 3 channel RGB image.

AlexNet is based on the Yann LeCun's work (Lecun, Bottou, Bengio and Haffner, 1998) which first explored the possibility of using deep neural networks for object recognition. AlexNet extracts features from an image using multiple convolutional kernels called filters. Each convolutional layer consists of multiple kernels. The first convolutional layer of AlexNet consists of 96 such filters of size 11x11x3 where 11x11 represents the height and width of the filter and 3 is the depth.

The first two convolutional layers are connected to a max pooling layer, which downsamples the height and width of the tensor, while the rest of the convolutional layers are directly connected. The last convolutional layer is connected to a max pooling layer followed by a softmax classifier.

Each layer is followed by a ReLU activation function. The authors of AlexNet used data augmentation (using mirror images of training image set) and dropout (dropping neurons from a network randomly) to reduce overfitting.

AlexNet was a pioneer in using convolutional neural networks for image classification and computer vision. The authors were able to get a top 5 error rate (incorrect labelling in top 5 predictions of the network) of 15.3%. This caused a seismic shift in the computer vision industry and CNNs have won every single iteration of ILSVRC ever since (Russakovsky et al., 2015). CNNs have become the go-to algorithm for computer vision tasks.

ResNet: Residual Neural Network or ResNet, developed by (He et al, 2015) won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC 2015). Inspired by the pyramidal cells of the human visual cortex, it is a 152 layers deep neural network that utilises skip connections known as residual connections to avoid information loss during training. ResNet was the first network to use deep networks.

The idea behind skipping layers is to avoid the problem of vanishing gradients (information loss between layers) by using the activation of one layer in the next layer. This ensures that the performance of the network does not get saturated as the network goes deeper. It also makes training the network faster (An Overview of ResNet and its Variants, 2020).

The model by (He et al, 2015) that won the ILSVRC 2015 was 1001 layer deep and had an accuracy of 93% (An Overview of ResNet and its Variants, 2020). ResNet architecture is shown in fig 2.3. ResNet is used as the model backbone in one of the models.

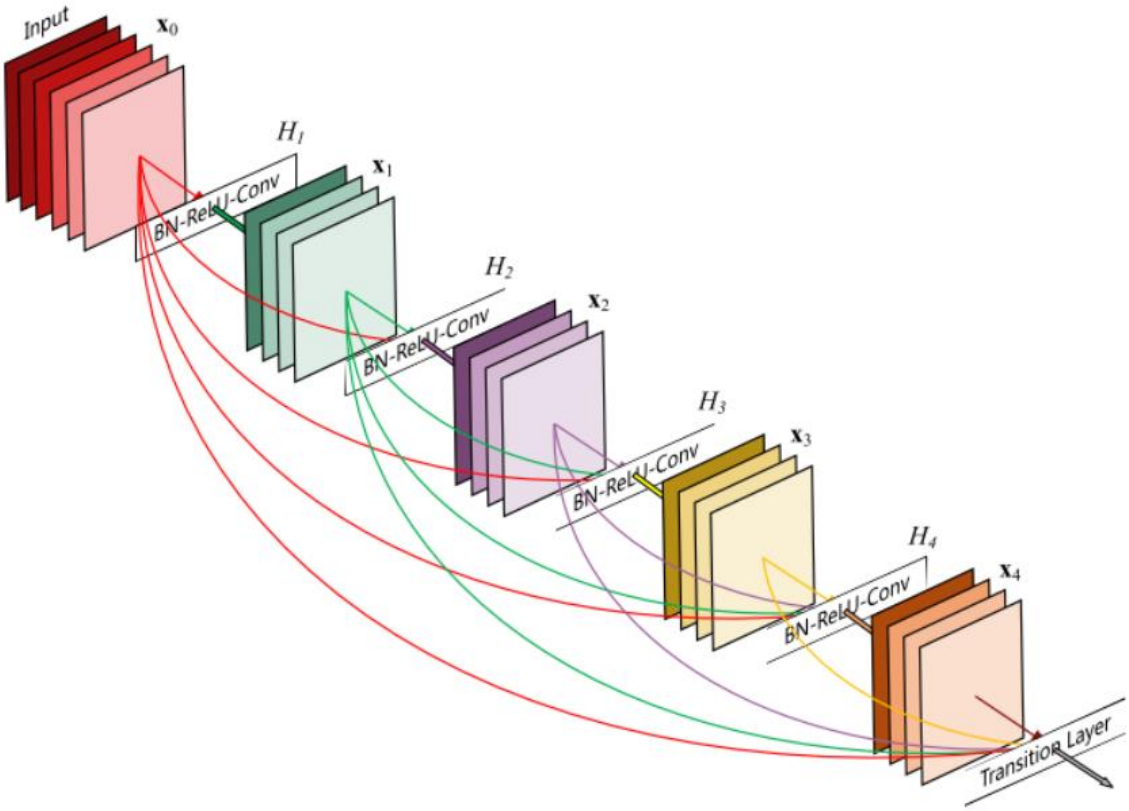


Fig 2.3: ResNet architecture (Fung, 2020)

U-Net: (Ronneberger, Fischer and Brox, 2015) created U-Net to use data augmentation in order to use annotated training samples more efficiently. Based on the sliding widow setup of (Ciresan et al, 2012) to predict the class of each pixel by localising the region around it and taking that as input. This technique increases the training data by a huge margin. This approach however, had a few drawbacks. The network had to be trained on all the localised regions (patches) and overlapping patches created redundancy. This made the training process very slow. The authors tried to address this problem by using a fully convolutional network.

The U-Net comprises of two arms in the shape of U. The left arm consists of a contracting path and expanding path on right side. The left side has a repeating application of 3x3 unpadded convolutions, each followed by ReLU and a 2x2 max pooling for downsampling. At each downsampling, the number of features channels are doubled. The In the expansive path, the features are upsampled followed by a 2x2 convolutions, with each convolution followed by a ReLU. A 1x1 convolution in the final layer maps the component feature vector to the classification class labels. This brings the total number of convolution layers to 23.

For data augmentation, the authors generated smooth deformations using random displacement vectors on a 3x3 coarse grid as well as dropout layers at the end of the left arm.

The U-Net model was able to achieve 92% IOU (Intersection Over Union) in the ISBI cell tracking challenge[1]. UNet architecture is shown in fig 2.4. UNet is the segmentation model architecture used for all the models in this project.

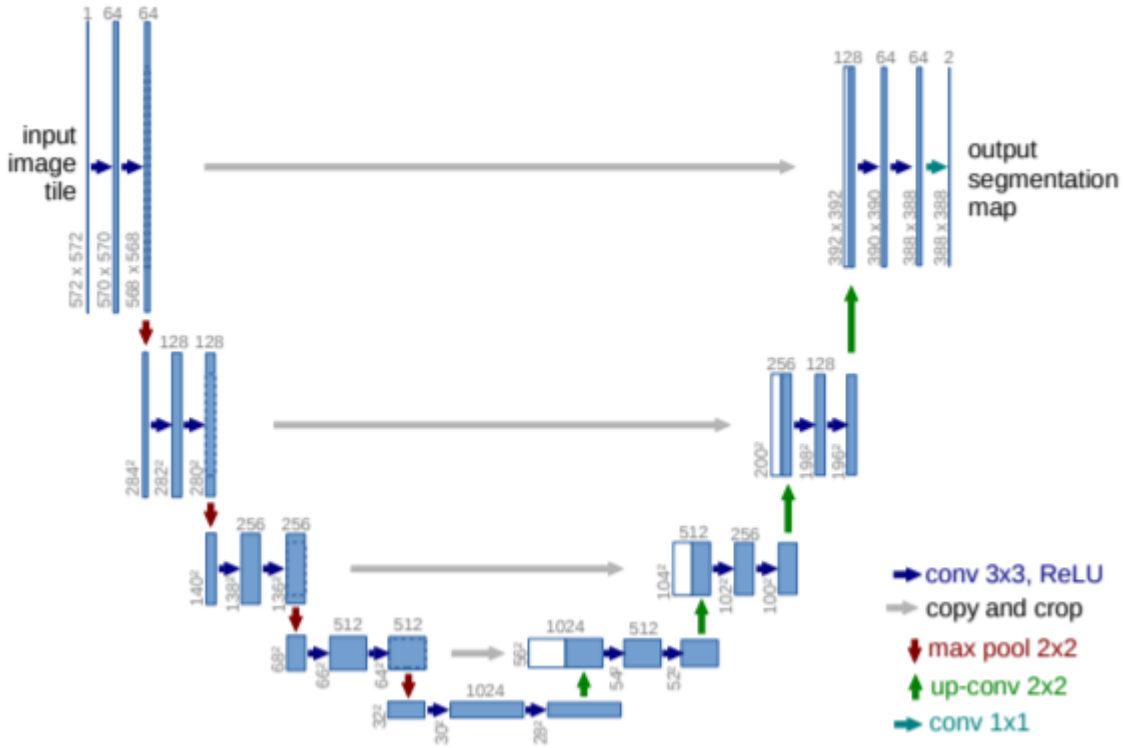


Fig 2.4: UNet architecture (Ronneberger, Fischer and Brox, 2015)

Inceptionv3: Inceptionv3 emerged as the 1st runner up of ILSVRC 2015. Developed by Google, it is a 42 layers deep convolutional neural network and an improvement over Inceptionv2. It replaces the 5x5 convolution layer of Inception2 by a series of two 3x3 convolution layers. The input image data is first convolved through the aforementioned 3x3 convolution window, followed by 3x1 and 1x3 convolutions. Then the data flows to an auxiliary classifier, which comprises of a 1x1 convolution layer followed by a fully connected layer. This auxiliary layer sits on top of the last 17x17 layer. While the Inceptionv2 used 2 auxiliary classifiers, Inceptionv3 uses just one. This along with the modified convolution design reduces the number of parameters considerably. In place of the traditional technique of using max pooling for feature map downsizing, Inceptionv3

[1] http://celltrackingchallenge.net/

uses grid reduction to create the final feature map (Tsang, 2020). The architecture of Inceptionv3 is given in fig 2.5. It is used for weight initialisation in one of the models in this project.

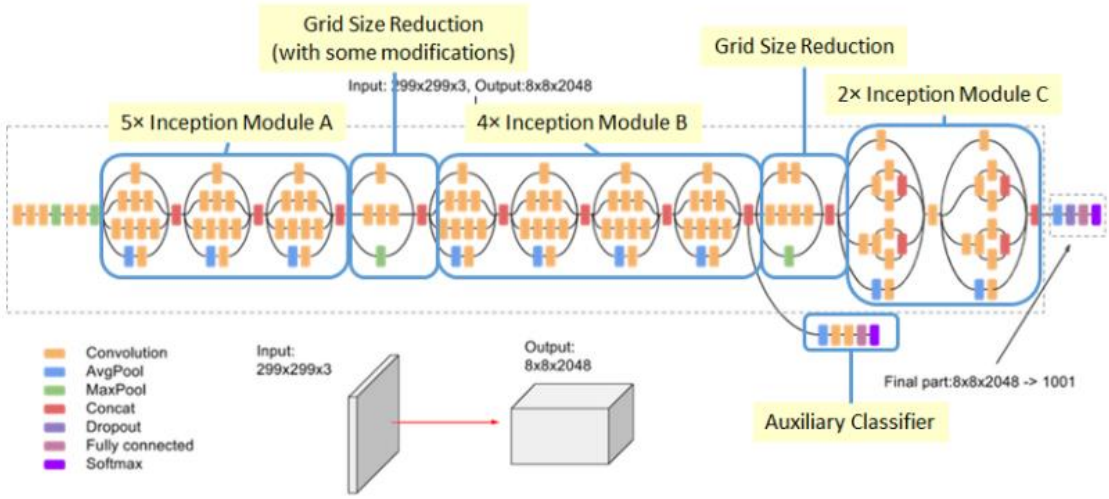


Fig 2.5: Inceptionv3 model architecture (Tsang, 2020).

Xception: Developed by (Chollet, 2017) for Google, Xception is based on the Inception family of convolutional neural networks and ranks among the most powerful CNN architectures for computer vision today. The Xception architecture is based on the concept of depth wise separable convolution layers. It aims to decouple the mapping of cross-channels correlations of convolutional neural networks and spatial correlations in feature maps.

The architecture consists of 36 convolutional layers structured in 14 modules with each layer having linear residual connections among them. The extreme layers do not have these connections.

The data in Xception goes through three flows: Entry flow, Middle flow and Exit flow.

The entry flow section contains 3x3 convolutional and separable convolutional layers. The convolutional layers are followed by ReLU and the separable convolutional layers are preceded by ReLU. The separable convolutional layers are followed by a max pooling layer. The entry flow generates a feature map of 19x19x728.

The middle flow consists of three sets of separable convolutional layers preceded by ReLU. This setup is repeated over eight times to generate a feature map of 19x19x728.

The exit flow consists of separable convolutional layers, preceded by ReLU and followed by max pooling layer. The last layer is a global average pooling layer. Xception outperformed popular models like ResNet and Inception v3 in the ImageNet challenge achieving a top-5% accuracy of 0.945 beating the second best model by a margin of 0.004. The data flow diagram of Xception is shown in fig 2.6. It is used as a model backbone in one of the models used in this project.

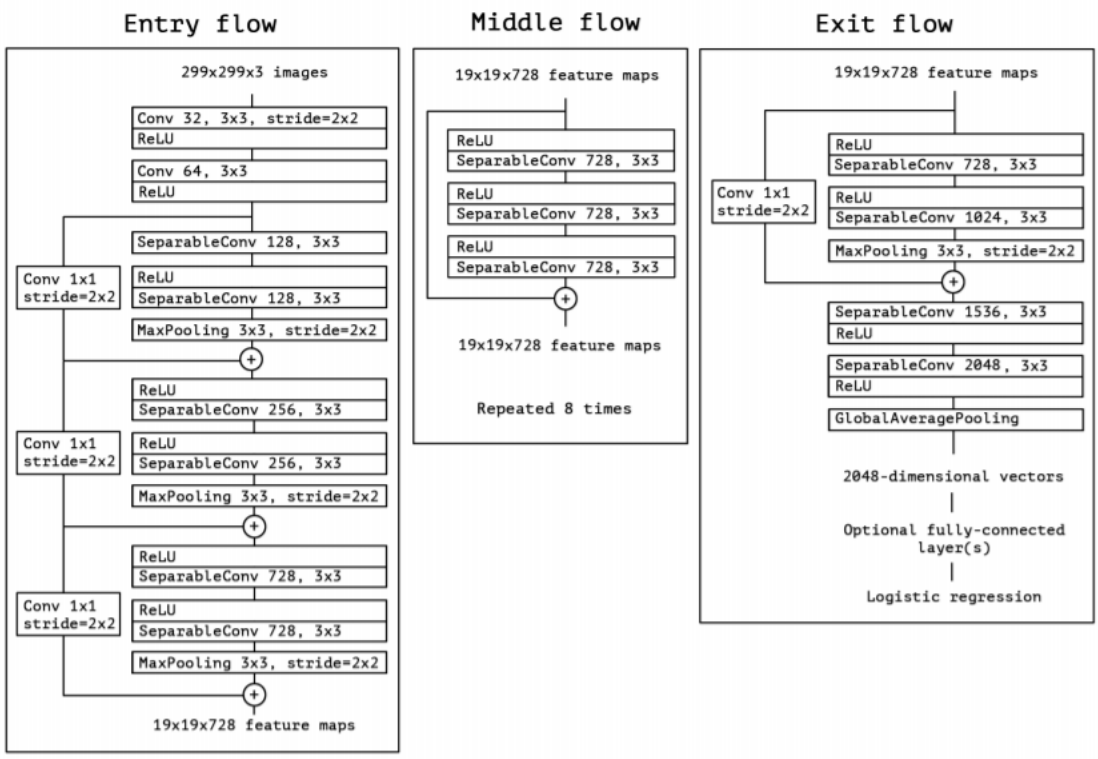


Fig 2.6: Xception data flow diagram (Chollet, 2017)

2.7. Survey of Deep Learning Frameworks and Technologies

A number of frameworks are available today which provides users with access to various deep learning technologies. In this section, a few popular frameworks will be explored.

TensorFlow: Developed by Google Brain team, TensorFlow is a machine learning framework capable of working at large scale and across environments. It can support multiple multicore CPUs, multiple GPUs as well as application specific integrated circuits called TPUs (Tensor Processing Units). It enables users to deploy machine learning models across clusters, workstations and dedicated deep learning accelerators. Different versions of TensorFlow are available to suit specific needs such as TensorFlow Extended for pipelining and TensorFlow.js for JavaScript environments.

TensorFlow provides stable APIs (Application Programming Interface) for Python and C as well as other APIs for many other programming languages such as Java, Go and Swift.

TensorFlow also provides stable CUDA (Compute Unified Device Architecture) support which enables the program to run on CUDA enabled GPUs.

Some popular applications of TensorFlow include DeepSpeech (Natural Language Processing), Nsynth (artificial music generation), RankBrain (Google's ranking algorithm for search) and Inception image classifier (Computer Vision) (Abadi et al., 2016).

PyTorch: Based on Torch library and developed by Facebook, PyTorch is an open source machine learning framework mainly developed for Python. It is mainly used for computer vision and natural language processing tasks. It makes use of Python's vast ecosystem and provides bidirectional compatibility between most popular Python libraries such as Pandas and NumPy. For example, it allows seamless bidirectional convertibility between NumPy arrays and PyTorch tensors. This gives users flexibility in data transformations.

It also provides native support for CUDA libraries such as cuDNN and cuBLAS. This enables users to run their models on GPUs. PyTorch is written in C++ which provides high performance and uses YAML based meta files for Python binding. This also allows for multi-thread execution of programs without the need of a global Python interpreter lock. This enhances performance considerably.

PyTorch is also supported on most cloud platforms such as AWS (Amazon Web Service) and Google Cloud. Some popular applications of PyTorch include Uber's Pyro, HuggingFace's Transformers and Catalyst (Paszke et al., 2019).

Keras: Keras is a deep learning platform built on top of TensorFlow. From TensorFlow 2.0, Keras uses TensorFlow backend by default. It provides numerous deep learning tools for creating neural networks such as layers, optimisers and activation functions. It allows low level tensor operations on multiple platforms like GPUs, CPUs and TPUs. It also allows high scalability supporting up to many thousands of GPUs and CPUs. Keras based deep learning models can also run on smartphones and on the web (Keras documentation: About Keras, 2020) .

Keras APIs include Models (for model management), Layers (for creating neural network layers), Callbacks (for managing network training, saving and deploying), Data Preprocessing (for handling specific data types such as images or text), Optimizers (for

model compilation), Metrics (for measuring model performance), Losses (for model training), Built-in datasets (for training models) and Keras Applications (pre trained models like Xception and DenseNet) (Keras documentation: Keras API reference, 2020).

CUDA: Compute Unified Device Architecture is a parallel computing platform developed by Nvidia which enables users to run programs and applications on CUDA supported Graphic Processing Units. Originally developed for running computer games, CUDA has found extensive use in deep learning. The CUDA Toolkit comprises of multiple GPU accelerated libraries. These include Math libraries such as cuBLAS for linear algebra, cuFFT for fourier transform, cuRAND for random number generation, cuSPARSE for solving sparse matrices, cuSPARSE for dense and sparse direct solvers and cuTensor for tensor operations. Other libraries include nvGRAPH and Thrust for parallel processing, image and video libraries such as nvJPEG, Nvidia Video SDK and Nvidia Optical Flow SDK and communication libraries such as NVSHEM and NCCL. For deep learning, CUDA Toolkit provides cuDNN for deep neural networks, TensorRT for high performance deep learning, Jarvis for NLP, DeepStream for real time AI analytics and DALI for image and video based deep learning.

2.8. Survey of Visualisation Technologies

Visualisations such as bar charts, histograms, pie charts and line charts help data scientists perform exploratory data analysis. This is a way of finding patterns and correlations in data. A lot of visualisation libraries are available today, to aid data scientists in making meaningful insights from raw data. A few of the libraries used in this project are discussed in brief below.

Matplotlib: Matplotlib is a Python based library that provides APIs for creating and embedding plots, charts and graphs into various applications as well as creating standalone versions of those charts. Most statistical plots be created using matplotlib. These include bar charts, point charts, area charts, histograms among many others (Matplotlib, 2020).

Altair: Altair is a statistical visualisation library for Python. It helps users to create various charts such as histogram, bar chart, dot plot, area chart among many others with minimal lines of code. It allows users to create interactive versions of statistical charts without the need for writing extensive JavaScript code (Overview — Altair 4.1.0 documentation, 2020).

Chapter 3. Methodology

This project is divided into three parts, each part fulfilling one of the three research objectives. The first part is dedicated to creating a convolutional neural network based image classifier that will classify the images of stratocumulus clouds in four categories – 'Fish', 'Flower', 'Sugar' and 'Gravel'. The second part consists of using the classifier to classify images of clouds taken before and after a natural weather event such as tropical cyclone and study the relationship between the cloud types and the weather event. In the third part, cloud images taken over a period of time will be classified and analysed leading to the study of the effect of climate change and stratocumulus cloud formation.

3.1. The Basic Theory

3.1.1. Computer Vision

Computer vision is a field of artificial intelligence which aims to enable computers to perform tasks that humans do using their sight. Computer vision tasks include image recognition and classification, pattern recognition in images, object and scene recognition in videos, noise removal from images and videos, medical imaging and many others. It is an interdisciplinary field involving computer science, neurology, electrical engineering and information sciences. Applications of computer vision include self-driving cars, facial recognition, video and image analysis and image restoration (Das, 2019).

The field of computer vision has seen huge advancements in the last decade. This is mostly due to advancements in deep learning, most notably convolutional neural networks (Russakovsky et al., 2015). In this section, we will see the application of convolutional neural networks for classifying and analysing satellite images of stratocumulus clouds.

3.1.2. Convolutional Neural Networks

Convolutional neural networks or CNNs have been described in detail in the previous section. In this section, we shall see the working, implementation and the mathematics behind CNNs.

The Convolution Function

Convolution is a mathematical function that expresses a change when one function overlaps another. In other words, it blends two different functions. Convolution of two functions ***f*** and ***g*** is given by:

$$(f * g)(t) \triangleq \int_{-\infty}^{\infty} f(\tau) g(t-\tau)\, d\tau.$$

When expressed in time domain, it becomes:

$$(f * g)(t) = \int_{0}^{t} f(\tau) g(t-\tau)\, d\tau \quad \text{for } f, g : [0, \infty) \to \mathbb{R}.$$

Here, function ***g*** gets reversed and slides along the horizontal axis and intersects ***f***. The area of intersection at a particular point is the convolution value. This can be visualised by the following diagram (Convolution -- from Wolfram MathWorld, 2020).

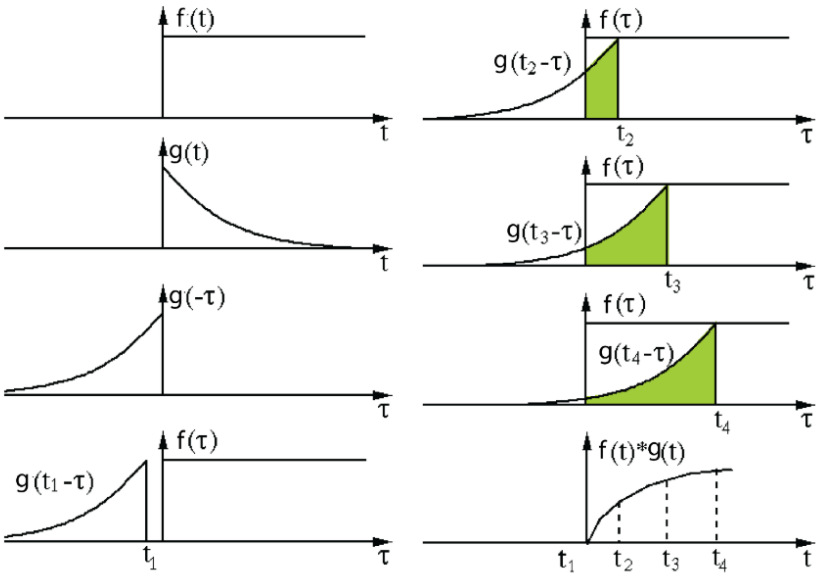


Fig 3.1: Visualising convolution (Bai, 2020)

3.1.3. Convolution in Deep Learning

In convolutional neural networks, convolution is used to extract different features from the images such as edges and vertices. It is done using filters (the function ***g***). The weights are adjusted during training and the learnt features are combined at the end. CNNs also learn about the spatial relationship between these features such as a person's head is above

the body. There are various types of convolutions used in CNNs (Bai, 2020). Some of them are explained below.

Single-channel convolution: An image is represented in terms of *w* x *h* x *c*, where *w* is the width, *h* is the height and *c* is the channel. Channel is the colour gradient of the image. As such, a single channel convolution means extracting a single colour feature as in grayscale images. In this case, the filter slides over the input image, doing element wise multiplication and addition and outputting a single number per instance, and creating a matrix output (Bai, 2020).

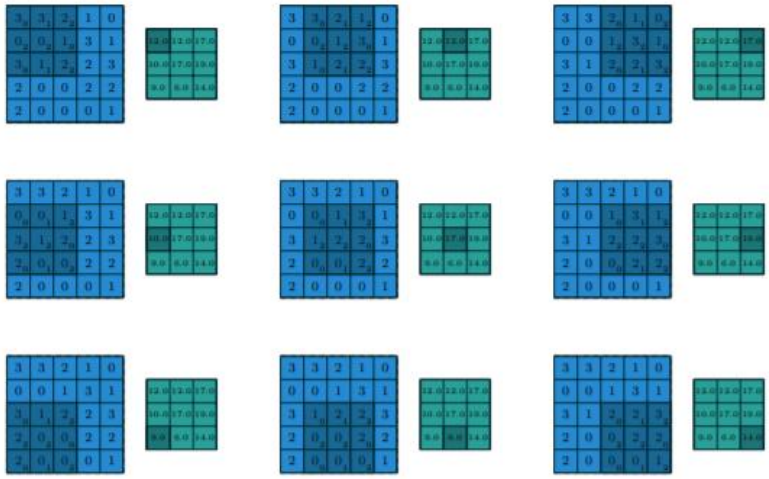


Fig 3.2: Single channel convolution. The shaded area is the filter (Bai, 2020).

Multi-channel convolution: If the input image is in colour, generally RGB (Red, Blue and Green) configuration, the input is filtered out as per the colour gradient or channel. Then the convolution operation is done separately as explained above. This results in three different matrices which are then added element-wise to generate a final matrix. This is called multi-channel convolution (Bai, 2020).

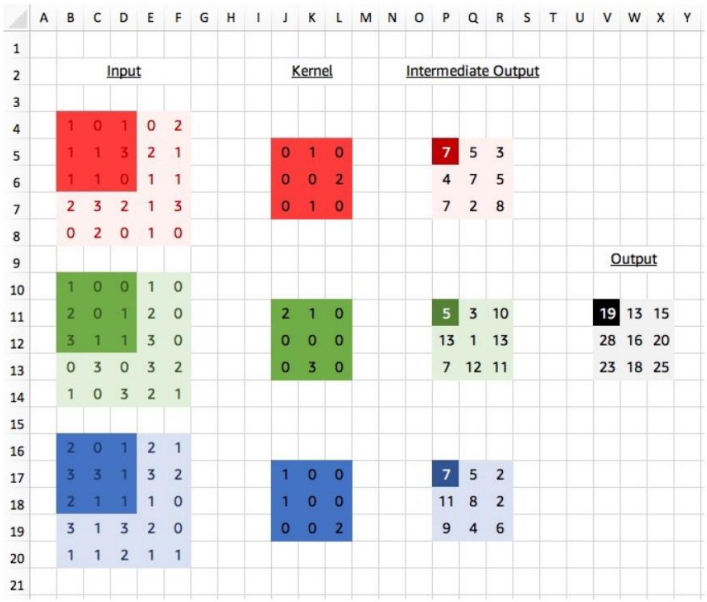


Fig 3.3: Multi-channel convolution (Bai, 2020).

3-D convolution: 3D convolution is similar to the multi-channel convolution process. Just as in multi-channel convolution, the filter moves along the channel matrices but depth-wise also. The depth size is specified. This results in volumetric sweeping of the 3D matrix. The output of this operation is a 2D matrix of a single colour. This is then further convolved to simpler matrices through a series of steps to arrive at a single matrix similar to that of the output of normal convolution (Bai, 2020).

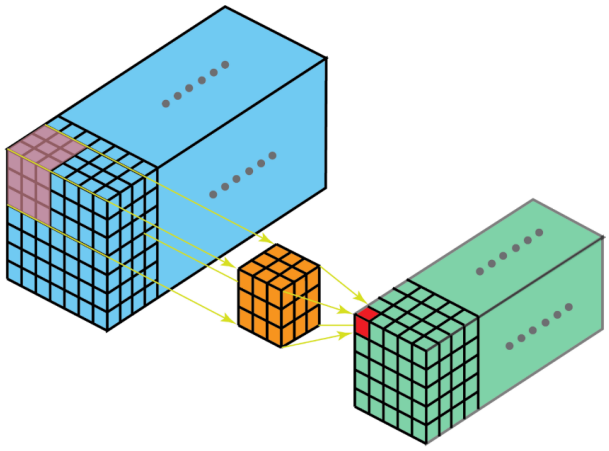


Fig 3.4: First step of 3D convolution. Showing the volumetric sweep (Bai, 2020).

3.1.4. Structure and working of Convolutional Neural Networks

Convolutional Neural Networks consist of an input layer, an output layer and several hidden layers. The hidden layers contain multiple series of convolutional layers which perform convolution as explained in the previous section. The technology for implementing the CNN in this project is Keras using TensorFlow backend. The architecture for segmentation models is UNet and the backbone is ResNet34. The CNN consists of different layers of neurons, modelled after the neurons in the human brain. In this project, the class of structure followed is Keras Sequential. This means that the neural network will train on one sample per epoch which results in better multi-processing of the data. The various layers of a CNN and their working is discussed below.

The Convolution Layer: This is generally the first layer which does the most computation heavy part. The data is convolved as the convolution operation is performed using filters. This results in a simpler matrix. In this project, Keras convolution layers are used in creating the convolution layers. Different convolution layers such as Conv1D and Conv2D have been used. Generally, parameters such as filters (specifying size of output), kernel_size (specifying the dimensions of convolution window, strides (how much the kernel moves), and activation (specifying the activation function). The activation function determines whether a neuron will fire or not. Commonly used activation functions in CNNs include Rectified Linear Unit or ReLU, Exponential Linear Unit or ELU, Hyperbolic Tangent or tanh and Sigmoid.

Pooling Layer: Pooling layers reduce the dimensions of the output of convolution layers. This is achieved by combining the output of several neurons. It is done locally and globally. In local pooling, the output of small clusters, generally 2x2 are combined. Global pooling works on all the neurons of the convolution layer. Max pooling utilises the maximum value at the end of each cluster. Average pooling does the same by taking average values from clusters. Several pooling layers from the Keras Layers API are used in this project. These include MaxPooling layers which perform max pooling on 1D, 2D and 3D convolution layers, GlobalMaxPooling layers which perform max pooling on all the neurons of that convolution layer and GlobalAveragePooling layer which performs average pooling in the same way.

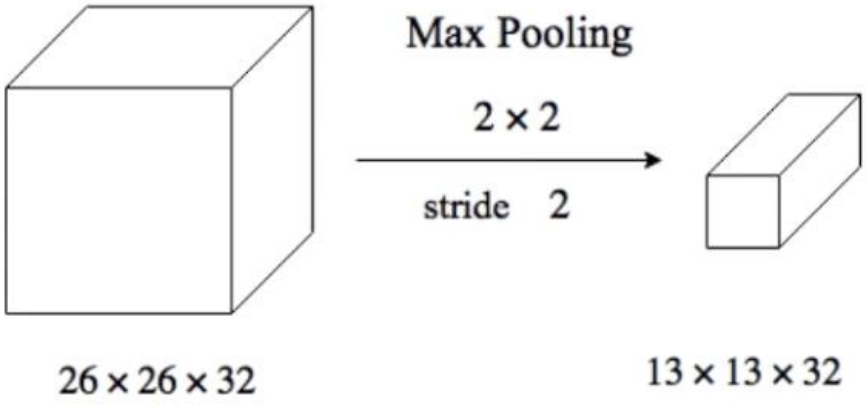


Fig 3.5: Dimensionality reduction using MaxPooling (Saha, 2020).

Fully Connected Layer: The fully connected layer or dense layer connects the neurons from one layer to another. In this layer, the output matrix of the convolution layer is flattened i.e. converted to a 1 dimensional tensor. This output is then fed to the network for further processing.

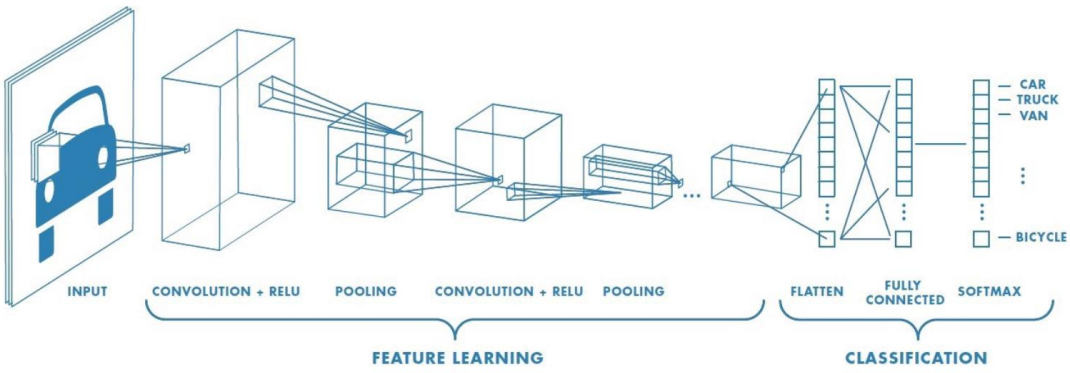


Fig 3.6: Structure of a CNN (Saha, 2020).

Apart from these main layers, several other layers (which often comprise these layers) are used in CNNs. A few are discussed below.

Dropout Layer: This layer sets random input to zero to prevent overfitting. This results in regularisation of the data. Inputs that are not set to zero are scaled up so that the overall sum of inputs stays the same. It is a part of the regularisation layers of Keras.

Batch Normalization Layer: This layer is generally used at the end of each layer or at the beginning. This layer scales the input in such a way that the standard deviation of the data becomes 1. This results in normalisation of the data.

Activation Layer: Activation layer applies the specified activation function to the output of the previous layer. However, using this layer standalone in CNNs is not done nowadays. Instead, the activation function is specified in the definition of the convolution layer itself.

3.1.5. Segmentation Models

Image segmentation is the process of analysing and classifying every pixel of an image to a particular class. This is very useful while working on real life scenarios where one image contains various objects. This can be explained from the image below (Jordan, 2020).

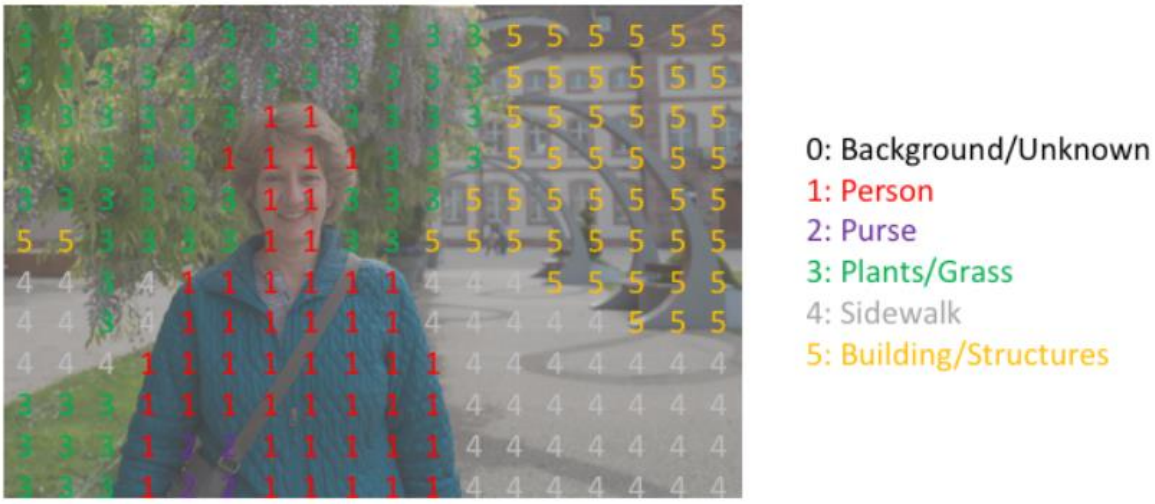


Fig 3.7: Image segmentation (Jordan, 2020).

As can be seen from this picture, one image has multiple areas belonging to different classes. Most real life images are like this. The segmentation model used in this project is UNet. The way it is achieved is called transpose convolution. While a normal convolution performs downsampling i.e. take the dot product of the values in the filter's view and produce a single output, transpose convolution does the opposite i.e. upsampling. In upsampling, a single value is taken and multiplied by the weights in the filter to create a higher dimension output which is then added on to the feature map. The figure below shows a simplified version of upsampling (Jordan, 2020).

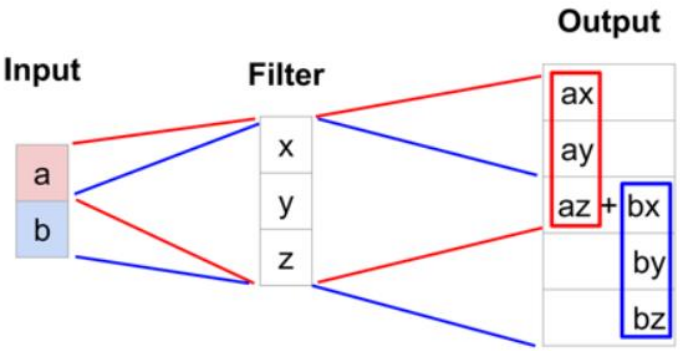


Fig 3.8: Transpose convolution upsampling (Jordan, 2020).

We will see the use of transpose convolution in later sections.

3.1.6. Classification Problems

Segmentation is, after all, a type of classification and as such to evaluate the performance of the segmentation model, we need to use classification metrics. In this subsection, we will get a brief overview of the classification metrics and the mathematics behind them.

Classification Accuracy: Accuracy in simple terms is the ratio of correct predictions to the total number of predictions made.

Precision: Precision or Positive Predictive Value, is the ratio of true positive to the sum of true positive and false positive (Precision and recall, 2020).

$$\text{Precision} = \frac{tp}{tp + fp}$$

Where,

tp: True Positive,

fn: False Negative,

fp: False Positive.

Recall: Recall or Sensitivity or True Positive Rate is given as the ratio of true positive to the sum of true positive and false negative (Precision and recall, 2020).

$$\text{Recall} = \frac{tp}{tp + fn}$$

F1 Score: It is defined as the harmonic mean of precision and recall. Mathematically, it is given as (F1 score, 2020):

$$F_1 = \frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Support: It is the number of occurrences of data belonging to a particular class. It is mostly used for analysing the evaluation process (Kohli, 2020).

		True condition	
	Total population	Condition positive	Condition negative
Predicted condition	Predicted condition positive	**True positive**	**False positive**, Type I error
	Predicted condition negative	**False negative**, Type II error	**True negative**
		True positive rate (TPR), Recall, Sensitivity, probability of detection, Power = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$

Fig 3.9: Confusion Matrix (Precision and recall, 2020).

Precision-Recall Curve: Precision-Recall Curve or PR Curve is used to determine the skill of the classifier. The baseline is fixed to ROC (Receiver Operating Characteristics) and the curve is plotted with the ratio of positives and negatives. The perfect skill is represented at (1,1) and a good classifier has a curve that bows towards that point (Brownlee, 2020).

Receiver Operating Characteristics: Receiver Operating Characteristics or ROC is a plot of false positive rate vs the true positive rate i.e. precision vs recall. It has a threshold value between 0.0 to 0.1 (Brownlee, 2020).

3.2. Part 1: Creating the classifier

3.2.1. Data Collection

The training data is taken from the dataset created by Max Planck Institute of Meteorology. The dataset consists of 10,000 images taken from NASA worldview website and labelled as a part of crowdsourcing project, where scientists from Max Planck Institute of Meteorology, Hamburg, and Sorbonne University, Paris manually labelled the images. This was extended and hosted on https://www.zooniverse.org/ for public to do the same. Both the labelling were corroborated and the final dataset was made available to the public for further research and analysis[2].

3.2.2 Model and Architecture Selection

For creating the deep learning model, segmentation model architecture has been used. Two models were created and evaluated on the training data and various classification metrics were used for their performance evaluation. The first model consists of a UNet model with ResNet34 as backbone. The second model used UNet-Xception with ImageNet as backbone. Both models are discussed in detail later in this section.

3.2.3. Data Pre-processing

Image data needs to be processed before being fed to CNNs. This requires a series of steps explained below.

3.2.3.1. Masking

Each satellite image consists of multiple regions of clouds belonging to a different cloud category. As such one particular image can belong to multiple classes. This can be seen in Fig 3.10.

[2] https://www.zooniverse.org/projects/raspstephan/sugar-flower-fish-or-gravel/classify

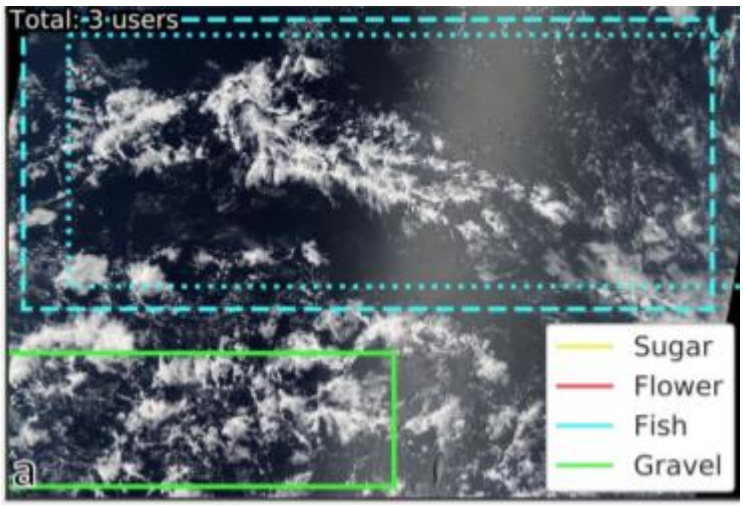


Fig 3.10: Regions belonging to different classes in a single image (Rasp et al., 2019).

To solve this challenge, a technique called masking is used. Masking refers to an image processing method in which a small area of an image is selected, and the pixels associated with that area is analysed. In case of multiple objects in an image, the pixels associated with each object are identified. This process is called image segmentation (Masking and Image Segmentation, 2020). This output of this process is an image matrix, where each cloud category and its associated pixels are recorded. This process is performed during training, testing and prediction. Before masking, the image is compressed using a technique called Run-Length Encoding or RLE. The use of the masking process and further explanation is in the training section. The pseudocode for masking (along with RLE) is given below.

```
# RLE function
Function rle2mask(input image):
  pixels = flatten image
  pixels = concatenate pixels
  return pixels

#Masking function
Function build_mask(input rles, input_shape, reshape):
  depth = length of rles
  if reshape is none:
    masks = numpy array of zeros with dimensions of input_shape and depth
  else:
    masks = numpy array of zeros with dimensions of reshape and depth
```

```
loop(all values in rles):
  if type of rle is string:
   if reshape is none:
      masks = rle2mask(rle, input_shape)
   else:
      masks = rle2mask(rle, input_shape)
      reshaped_mask = resize with dimensions of mask and reshape
      masks = reshaped_mask
```

Table 3.1: Pseudocode for masking and RLE

3.2.3.2. Data Generation

The image data that is to be fed to the neural network needs to be in a particular format, generally in the form of tensors. Also, as image data is generally large, loading all the data at once can cause resource problems. So instead of loading the entire dataset at once, the data is loaded in batches. The entire dataset is divided into a specified size. The data in each batch goes through a number of changes such as shuffling, resizing, vertical and horizontal flipping, and separate grayscale and RGB loading. These transformations help the network to learn features better leading to a better fit (Amidi, 2020).

The data generator used here is derived from the Python class DataGenerator and is made to inherit the properties of keras.utils.Sequence so as to use the functionalities of Sequential class. Albumentations library is used to perform image transformations. An example of image transformations is shown in Fig 3.11. The pseudocode for DataGenerator is given in table 3.2.

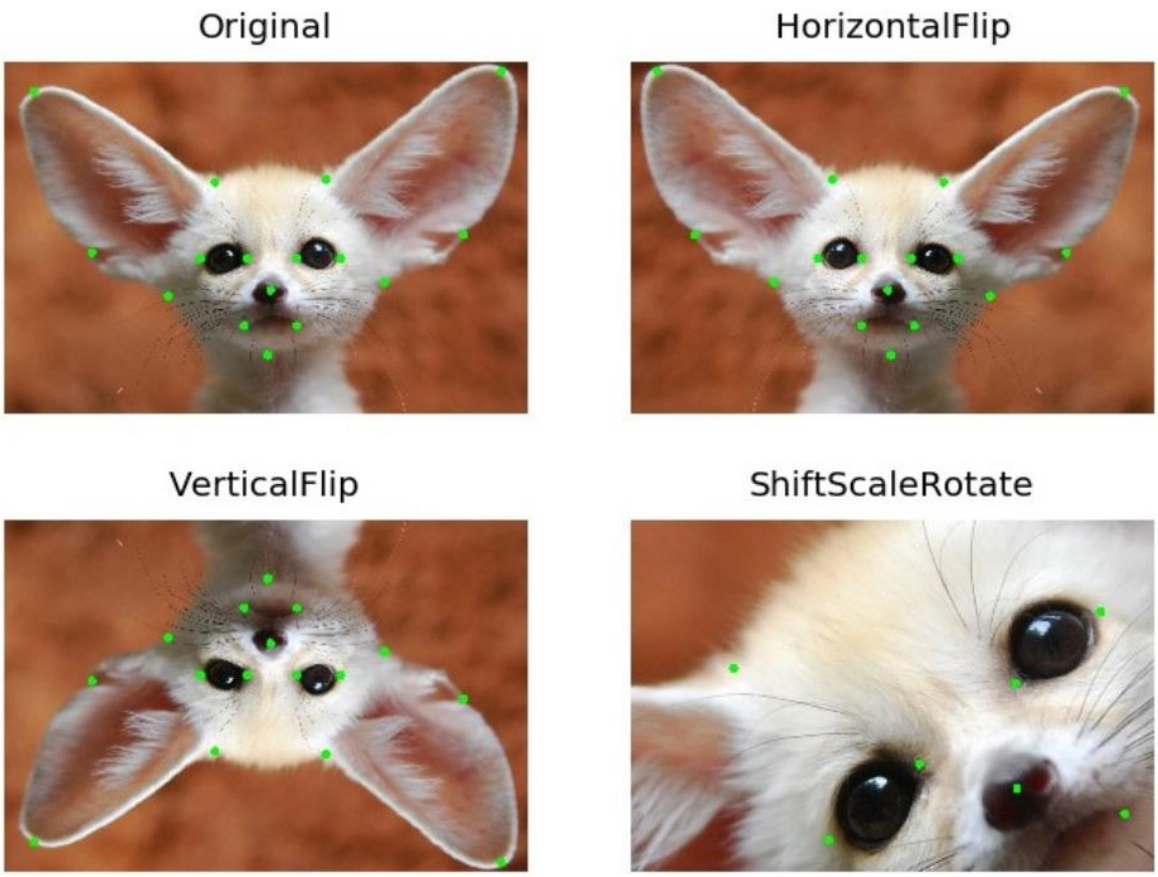


Fig 3.11: Image Transformations (Albumentations, 2020)

 # DataGenerator Class

Define DataGenerator class and inherit keras.util.Sequence

Specify batch size, input dimensions, number of channels, number of classes and random state

Generate one batch of data

Update indexes after each epoch

Generate data containing batch size samples

#Image transformations

Perform grayscale transformation

Perform RGB transformation

Perform horizontal flip

Perform vertical flip

Perform shift scale rotate

#Create finalised batch

Loop:

Concatenate transformed image data

Return final data in a batch

Table 3.2: Pseudocode for Data Generator

3.2.4. Defining Utility Functions

For creating the model, several utility functions are needed. These include optimiser, loss function and evaluation metrics. They are discussed below.

3.2.4.1. Optimiser

Optimisers use a mathematical programming technique called optimisation to change the model parameters and weights by updating the model by evaluating response to the output of the loss function. In other words, the model is moulded to generate more accurate results (Introduction to Optimizers, 2020). The optimiser used here is RAdam.

RAdam or Rectified Adaptive Moment Estimation is a variant of Adam optimiser. RAdam uses a rectification function to reduce variance and allow adaptive momentum to achieve convergence. RAdam is available for Keras in form of a third party library (Liu, 2020).

3.2.4.2. Loss Function

Loss function is used to minimise the error by mapping decisions to their associated costs. The loss function used here is a combination of dice loss and binary cross entropy.

Dice loss is a popular loss function used for segmentation problems. It is a measure of overlap between two samples, in this case areas belonging to two different classes. It

ranges from 0 to 1, with 1 denoting complete overlap and 0 denoting no overlap. It is generally used for binary data and is given as:

$$Dice = \frac{2\,|A \cap B|}{|A| + |B|}$$

Where,

Dice is the measure of overlap and A and B representing the classes.

When evaluating predictions of segmentation masks, $|A \cap B|$ represents the approximation of element-wise multiplication between the prediction and target mask. The resultant matrix is the sum.

$$|A \cap B| = \underbrace{\begin{bmatrix} 0.01 & 0.03 & 0.02 & 0.02 \\ 0.05 & 0.12 & 0.09 & 0.07 \\ 0.89 & 0.85 & 0.88 & 0.91 \\ 0.99 & 0.97 & 0.95 & 0.97 \end{bmatrix}}_{\text{prediction}} * \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}}_{\text{target}} \xrightarrow{\text{element-wise multiply}} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.89 & 0.85 & 0.88 & 0.91 \\ 0.99 & 0.97 & 0.95 & 0.97 \end{bmatrix} \xrightarrow{\text{sum}} 7.41$$

The zeros represent the pixels which did not get activated (Jordan, 2020).

Binary cross entropy is a loss function used for binary classification task. Here we are classifying the probability of a mask belonging to a particular class or not. It provides the sum of logarithmic probabilities of whether a target belongs to a particular class or not. Mathematically it is given as:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Where,

y is the target being classified and *log(p(y))* is the logarithmic probability (Godoy, 2020). This loss function is present in keras.losses.BinaryCrossentropy.

The pseudocode for the combined loss function is given below.

```
#Define Dice Loss
Function dice_loss(inputs actual target values, predicted target values):
  y_true_f = flatten actual target values
```

```
  y_pred_f = flatten predicted target values
  intersection = sum(y_true_f* y_pred_f)
  return 1 -  (2*intersection+1)/(sum(y_true_f)+sum(y_pred_f)+1)

#Define Combined Loss Function
Function bce_dice_loss(inputs actual target values, predicted target values):
  return binary_crossentropy(actual target values, predicted target values) + dice_loss(actual target values, predicted target values)
```

Table 3.3: Pseudocode for loss function.

3.2.4.3. Evaluation Metrics:

Evaluation metrics are used to evaluate model performance during training on which the model is optimised. The metric used here is dice coefficient.

Dice Coefficient is based on Sørensen–Dice coefficient. It computes the similarity between two samples. It is also known as F1 score. Mathematically it is defined as:

$$DSC = \frac{2TP}{2TP + FP + FN}.$$

where TP is true positive, FP is false positive, and FN is false negative. The pseudocode is given below.

```
#Define Dice Coefficient
Function dice_coefficient(inputs actual target values, predicted target values):
  y_true_f = flatten actual target values
  y_pred_f = flatten predicted target values
  intersection = sum(y_true_f* y_pred_f)
  return (2*intersection+1)/(sum(y_true_f)+sum(y_pred_f)+1)
```

Table 3.4: Pseudocode for dice coefficient.

3.2.5. Model Creation

Two models were created as explained earlier. The first model is based on UNet with ResNet34 as backbone and the second a hybrid UNet-Xception model with ImageNet as backbone. A few layers have been changed in both the models in order for them to work with the dataset. The pseudocode for model creation is given below.

```
#UNet-ResNet34 definition (all layers belong to Keras layers API)
```

2D Convolution layer(8, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D Convolution layer(8, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D MaxPooling Layer. Padding = ‘same’.

2D Convolution layer(16, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D Convolution layer(16, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D MaxPooling Layer. Padding = ‘same’.

2D Convolution layer(32, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D Convolution layer(32, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D MaxPooling Layer. Padding = ‘same’.

2D Convolution layer(64, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D Convolution layer(64, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D MaxPooling Layer. Padding = ‘same’.

2D Convolution layer(128, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D Convolution layer(128, (3,3)). Activation = ‘elu’, Padding = ‘same’.

2D Transposed Convolution Layer(64, (2, 2). Strides = (2,2), Padding = ‘same’.
2D Convolution layer(64, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D Convolution layer(64, (3,3)). Activation = ‘elu’, Padding = ‘same’.

2D Transposed Convolution Layer(32, (2, 2). Strides = (2,2), Padding = ‘same’.
2D Convolution layer(32, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D Convolution layer(32, (3,3)). Activation = ‘elu’, Padding = ‘same’.

2D Transposed Convolution Layer(32, (2, 2). Strides = (2,2), Padding = ‘same’.
2D Convolution layer(32, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D Convolution layer(32, (3,3)). Activation = ‘elu’, Padding = ‘same’.

2D Transposed Convolution Layer(16, (2, 2). Strides = (2,2), Padding = ‘same’.
2D Convolution layer(16, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D Convolution layer(16, (3,3)). Activation = ‘elu’, Padding = ‘same’.

2D Transposed Convolution Layer(8, (2, 2). Strides = (2,2), Padding = ‘same’.
2D Convolution layer(8, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D Convolution layer(8, (3,3)). Activation = ‘elu’, Padding = ‘same’.

2D Convolution layer(4, (1,1)). Activation = ‘sigmoid’.

Add all the layers to model object.

Table 3.5: Pseudocode for UNet-ResNet34 model definition.

#Model Compilation – UNet-ResNet34

Initialise model to Unet from segmentation model library specifying ‘resnet34’ as backbone, number of classifying classes, input shape and activation function as ‘sigmoid’.

Compile the model specifying optimiser, loss function and metrics.

Print model summary.

Table 3.6: Pseudocode for UNet-ResNet34 compilation.

#UNet-Xception definition (all layers belong to Keras layers API)
Set Xception as backbone and set weights from ImageNet
Initialise input from backbone
Specify start neurons = 16

2D Convolution layer(8, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D Convolution layer(8, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D MaxPooling Layer. Padding = ‘same’.

2D Convolution layer(16, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D Convolution layer(16, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D MaxPooling Layer. Padding = ‘same’.

2D Convolution layer(32, (3,3)). Activation = ‘elu’, Padding = ‘same’.

2D Convolution layer(32, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D MaxPooling Layer. Padding = ‘same’.

2D Convolution layer(64, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D Convolution layer(64, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D MaxPooling Layer. Padding = ‘same’.

2D Convolution layer(128, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D Convolution layer(128, (3,3)). Activation = ‘elu’, Padding = ‘same’.

2D Transposed Convolution Layer(64, (2, 2). Strides = (2,2), Padding = ‘same’.
2D Convolution layer(64, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D Convolution layer(64, (3,3)). Activation = ‘elu’, Padding = ‘same’.

2D Transposed Convolution Layer(32, (2, 2). Strides = (2,2), Padding = ‘same’.
2D Convolution layer(32, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D Convolution layer(32, (3,3)). Activation = ‘elu’, Padding = ‘same’.

2D Transposed Convolution Layer(32, (2, 2). Strides = (2,2), Padding = ‘same’.
2D Convolution layer(32, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D Convolution layer(32, (3,3)). Activation = ‘elu’, Padding = ‘same’.

2D Transposed Convolution Layer(16, (2, 2). Strides = (2,2), Padding = ‘same’.
2D Convolution layer(16, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D Convolution layer(16, (3,3)). Activation = ‘elu’, Padding = ‘same’.

2D Transposed Convolution Layer(8, (2, 2). Strides = (2,2), Padding = ‘same’.
2D Convolution layer(8, (3,3)). Activation = ‘elu’, Padding = ‘same’.
2D Convolution layer(8, (3,3)). Activation = ‘elu’, Padding = ‘same’.

2D Convolution layer(4, (1,1)). Activation = ‘sigmoid’.

Add all the layers to model object.

Table 3.7: Pseudocode for UNet-Xception model definition.

#Model Compilation – UNet-Xception
Initialise model to Unet from segmentation model library specifying ‘inceptionv3 as backbone, number of classifying classes, input shape and activation function as ‘sigmoid’. Compile the model specifying optimiser, loss function and metrics. Print model summary.

Table 3.8: Pseudocode for UNet-Xception model compilation.

3.2.6. Training

The model needs to be trained with the training data in order to optimise it. During the training process, the weights are modified according to the optimiser and the model is evaluated as per the evaluation metrics and the loss is calculated as per the loss function. This process goes back and forth and is known as back propagation. The pseudocode for training is given below.

 # Create training and validation data
Create object of DataGenerator class with required specifications for training and validation Create checkpoint for saving progress Call fit generator and pass values for training and validation data and specify epochs and callback. Save and plot training performance history.

Table 3.9: Pseudocode for training.

3.2.7. Testing

After the model has been trained, we need to test it. For this, a certain portion of data has been sliced off from the training data and the model is tested on this. The predictions are then matched with the actual labels and various metrics such as confusion matrix, precision, recall, fi score and receiver operating characteristics are used to gauge the classifier performance. The pseudocode for testing predictions is given below.

#Predictions for testing

Create dataframe for storing predictions (test_df).

Loop:

Create a list of batch ids.

Create test_generator object of DataGenerator with required specifications with mode set to predict.

Create predictions using fit_generator function of the model.

Loop:

Get image_id from filename.

Get masks of the image.

Convert masks into pixels.

Append all the data to test_df

Table 3.10: Pseudocode for testing.

In order to understand the predictions as well as model performance better, both the test dataset and the predictions have been transformed. Instead of directly comparing the predicted masks for each class, the data is transformed to show the presence of each type of cloud. To understand this transformation better, let us see some examples.

	Image_Label	EncodedPixels
0	0011165.jpg_Fish	264918 937 266318 937 267718 937 269118 937 27...
1	0011165.jpg_Flower	1355565 1002 1356965 1002 1358365 1002 1359765...
2	0011165.jpg_Gravel	NaN
3	0011165.jpg_Sugar	NaN
4	002be4f.jpg_Fish	233813 878 235213 878 236613 878 238010 881 23...
...	...	...
22179	ffd6680.jpg_Sugar	NaN
22180	ffea4f4.jpg_Fish	NaN
22181	ffea4f4.jpg_Flower	1194860 675 1196260 675 1197660 675 1199060 67...
22182	ffea4f4.jpg_Gravel	NaN
22183	ffea4f4.jpg_Sugar	NaN

Fig 3.12: Data before formatting.

	Image	Class	Fish	Flower	Sugar	Gravel
0	ab92630.jpg	{Flower, Fish, Sugar}	1	1	1	0
1	ab98b1b.jpg	{Flower, Fish}	1	1	0	0
2	abaca5e.jpg	{Fish, Sugar, Gravel}	1	0	1	1
3	abf0a99.jpg	{Fish, Gravel}	1	0	0	1
4	ac0020d.jpg	{Flower, Sugar}	0	1	1	0
...	...	...	...	...	...	...
1791	ffcedf2.jpg	{Fish}	1	0	0	0
1792	ffd11b6.jpg	{Flower, Sugar}	0	1	1	0
1793	ffd3dfb.jpg	{Sugar}	0	0	1	0
1794	ffd6680.jpg	{Flower, Gravel}	0	1	0	1
1795	ffea4f4.jpg	{Flower}	0	1	0	0

Fig 3.13: Data after formatting.

The pseudocode for the transformation is given below.

#Data transformation
Split the image label based on '_' character and store the first part in Image column and second part in Class column. Both new columns to be created in the test_df. Loop: If a Class has pixels, create a new column with the same name and assign 1 value to it. Else assign 0 value to it.

Table 3.11: Pseudocode for data transformation.

3.2.8. Model Performance Evaluation

The performance of the classifier is then evaluated using metrics such as precision, recall, f1 score and support. The function used for generating these metrics is classification_report from sklearn.metrics. In addition to these, accuracy is also computed using accuracy_score. The metrics for both the models are given below.

```
Classification accuracy for class Fish: 60.7799%
Classification Report for class Fish :
              precision    recall  f1-score   support

           0       0.33      0.75      0.46       405
           1       0.89      0.57      0.69      1390

    accuracy                           0.61      1795
   macro avg       0.61      0.66      0.58      1795
weighted avg       0.76      0.61      0.64      1795

Classification accuracy for class Flower: 67.7437%
Classification Report for class Flower :
              precision    recall  f1-score   support

           0       0.48      0.92      0.63       541
           1       0.94      0.57      0.71      1254

    accuracy                           0.68      1795
   macro avg       0.71      0.75      0.67      1795
weighted avg       0.81      0.68      0.69      1795
```

```
Classification accuracy for class Gravel: 57.9387%
Classification Report for class Gravel :
              precision    recall  f1-score   support

           0       0.14      0.83      0.24       144
           1       0.97      0.56      0.71      1651

    accuracy                           0.58      1795
   macro avg       0.56      0.69      0.47      1795
weighted avg       0.91      0.58      0.67      1795

Classification accuracy for class Sugar: 75.2646%
Classification Report for class Sugar :
              precision    recall  f1-score   support

           0       0.39      0.72      0.51       315
           1       0.93      0.76      0.83      1480

    accuracy                           0.75      1795
   macro avg       0.66      0.74      0.67      1795
weighted avg       0.83      0.75      0.78      1795
```

Table 3.12: Classification report for UNet-Xception.

```
Classification accuracy for class Fish: 63.8595%
Classification Report for class Fish :
              precision    recall  f1-score   support

           0       0.37      0.81      0.51       415
           1       0.91      0.59      0.71      1378

    accuracy                           0.64      1793
   macro avg       0.64      0.70      0.61      1793
weighted avg       0.79      0.64      0.67      1793

Classification accuracy for class Flower: 82.2644%
Classification Report for class Flower :
              precision    recall  f1-score   support

           0       0.92      0.80      0.86      1194
           1       0.69      0.87      0.77       599

    accuracy                           0.82      1793
   macro avg       0.80      0.83      0.81      1793
weighted avg       0.84      0.82      0.83      1793
```

```
Classification accuracy for class Gravel: 64.6960%
Classification Report for class Gravel :
              precision    recall  f1-score   support

           0       0.34      0.80      0.48       360
           1       0.92      0.61      0.73      1433

    accuracy                           0.65      1793
   macro avg       0.63      0.70      0.60      1793
weighted avg       0.81      0.65      0.68      1793

Classification accuracy for class Sugar: 75.1813%
Classification Report for class Sugar :
              precision    recall  f1-score   support

           0       0.39      0.72      0.51       318
           1       0.93      0.76      0.83      1475

    accuracy                           0.75      1793
   macro avg       0.66      0.74      0.67      1793
weighted avg       0.83      0.75      0.78      1793
```

Table 3.13: Classification report for UNet-ResNet.

In addition to this, charts such as Precision-Recall curve and Receiver Operating Characteristics have also been used to gauge classifier performance. All these are discussed in more detail in the next chapter.

From tables 3.12 and 3.13, it is clear that UNet-ResNet model outperforms the UNet-Xception model. Hence, we will use UNet-ResNet model for the next two parts.

3.3. Part 2: Studying the Relationship between Stratocumulus Cloud Categories and Tropical Cyclones.

The objective for this part is to study whether any relationship exists between any of the four cloud categories and tropical cyclones. For these three cyclones Irma, Dorian and Maria have been selected. All the cyclones occurred in the tropics between 35° north and the Equator in the Atlantic Ocean. A brief description of all the cyclones is given below.

Irma: Irma was a category 5 hurricane that formed and caused widespread destruction in the USA in September 2017. It formed near the western coast of Africa and reached Florida on September 9 and finally dissipating on September 14. It caused around $77 billion loss and claimed 134 lives (Hurricane Irma, 2020).

Dorian: Dorian was a powerful Atlantic Hurricane which struck Caribbean countries and the south-eastern United States between August 24 to September 10, 2019. It is considered to be the worst tropical cyclone to hit the Bahamas claiming 74 lives and rendering over 70,000 people homeless (Hurricane Dorian, 2020).

Maria: Hurricane Maria was a category 5 Atlantic cyclone that struck Dominica and Puerto Rico in September 2017. It is the most devastating cyclone to hit these islands causing more than $90 billion in damages and claiming 3059 lives. It formed on 16 September and dissipated on 2 October (Hurricane Maria, 2020).

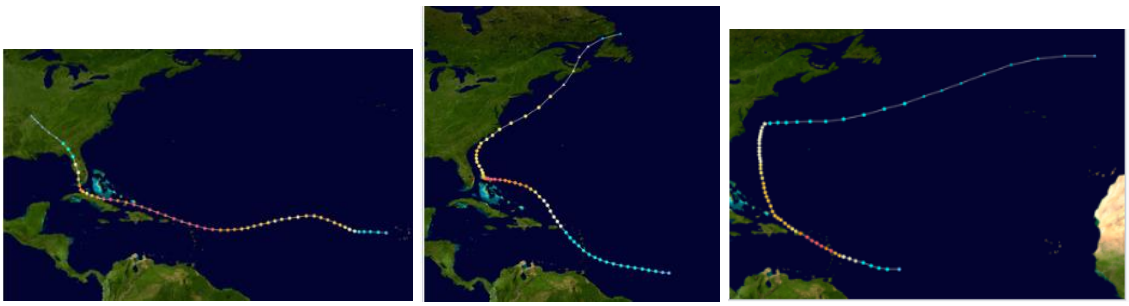


Fig 3.14: Routes of Irma (left), Dorian (centre) and Maria (right).

3.3.1. Data Collection

The dataset consists of a total of 18 satellite images manually taken from the NASA worldview website. The images were taken at 500m resolution at 1600x1600 pixels resolution. Each cyclone has six images, three taken just before the cyclone and three just after the cyclone has passed. The images are generally of an area about 14° north. The csv file used for the segmentation is also created manually.

3.3.2. Classification

For segmentation, the UNet-ResNet classifier trained during part 1 of the project has been used. All utility functions, metrics and model creation remain exactly the same as discussed in section 3.2. For classification, the process and code are same as detailed in section 3.2.7 with necessary changes such as image dimensions and base path. The segmentation results obtained after predictions are analysed and discussed in the next chapter.

3.4. Part 3: Studying the effect of the rise in Global Temperature and the Formation and Meso-Scale Organisation of Stratocumulus Clouds.

The objective of this section of the project is to analyse and study the effect of global warming on the formation of stratocumulus clouds. The intuition is to perform a time series analysis of the various cloud categories from images in the last ten years and forecast the data for the next ten years.

3.4.1. Background

The purpose of this part is to test a prediction made by scientists at the California Institute of Technology. The scientists created a simulation of stratocumulus clouds based on eddy currents and found out that with increase in global temperature and carbon dioxide, the ability of stratocumulus clouds to form would significantly go down. As stratocumulus clouds reflect a lot of sunlight, this would increase the ambient temperature further reducing the formation of these cloud types. They calculated that carbon dioxide concentrations of 1,200 ppm and a global temperature rise of 8° from current levels would initiate the breakup of these clouds (Schneider, Kaul and Pressel, 2019). These findings were published in the Nature Geoscience magazine.

However, this paper was criticised by other scientists (Voosen, 2020), who pointed out that the model based on eddy currents was too simple and cloud formation is a complex geo-chemical process where many factors come into play. As such the model, solely based on eddy currents is too simplistic.

3.4.2. The Basic Theory

The intuition is to perform a time series analysis of segmented cloud data collected from satellite images. A basic overview of time series analysis is given below.

3.4.2.1. Time-Series Analysis

Time series data is a set of data points indexed in order of time, such as minutes, hours , days or years. Time series analysis consists of methods to extract meaningful insights from time series data. This includes exploratory data analysis, where visual charts such as line charts are used to give a visual insight into the data and forecasting, where future data points are predicted (Time series, 2020). In this project, we will employ exploratory data analysis to see if a correlation exists between global temperature rise and stratocumulus cloud formation as well as forecasting to predict the nature of these cloud formations in the future.

For exploratory data analysis, we will use multiple overlapping line charts and for forecasting, ARIMA model will be used.

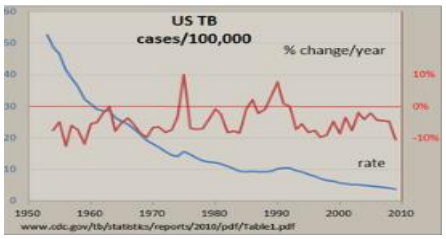


Fig 3.15: Line chart (Time series, 2020).

3.4.2.2. ARIMA

Auto Regressive Integrated Moving Average or ARIMA is a class of regression models that forecasts a time series data based on lags. This model can be used for non-seasonal time series data that exhibit patterns. ARIMA consists of three components:

- An auto-regressive term p, which describes a time varying random process.
- A moving average term q, which describes the output in terms of linearly dependent past values.
- A differencing term d, which is the number of differencing required to make the series stationary.

A pure auto-regressive model is defined as a model in which the output Yt is a function of time and is dependent only on its lags. Mathematically,

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + .. + \beta_p Y_{t-p} + \epsilon_1$$

where,

Y_{t-1} is the lag of the series, β is the coefficient of lag and α is the intercept.

A pure moving average model is defined as a model in which Yt depends on the errors of forecast lag. Mathematically,

$$Y_t = \alpha + \epsilon_t + \phi_1\epsilon_{t-1} + \phi_2\epsilon_{t-2} + \ldots + \phi_q\epsilon_{t-q}$$

where,

ϵ_t and ϵ_{t-1} are error terms, ϕ is the error coefficient and α is the intercept.

Combining both the equations, we get,

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \ldots + \beta_p Y_{t-p}\epsilon_t + \phi_1\epsilon_{t-1} + \phi_2\epsilon_{t-2} + \ldots + \phi_q\epsilon_{t-q}$$

As stated earlier, ARIMA only works on stationary data. Stationary data is one in which the probability distribution and hence mean and variance do not change with time. In order to make the data stationary, differencing is used. Differencing requires subtracting the past value of a data point and the term ‘d’ refers to the number of times this process is repeated.

The correct number for ‘d’ is estimated when after differencing, the auto-correlation function (ACF) plot approaches zero. ACF shows how the correlation between any two values changes as their separation changes. This can be gauged by differencing the series and then plotting the ACF plot and visually evaluating it. This can be explained from the figure below (Fig 3.16). We can see that after the second differencing, the ACF plot is almost near zero, barring a few spikes. As such the optimum value for ‘d’ is 2(Prabhakaran, 2020).

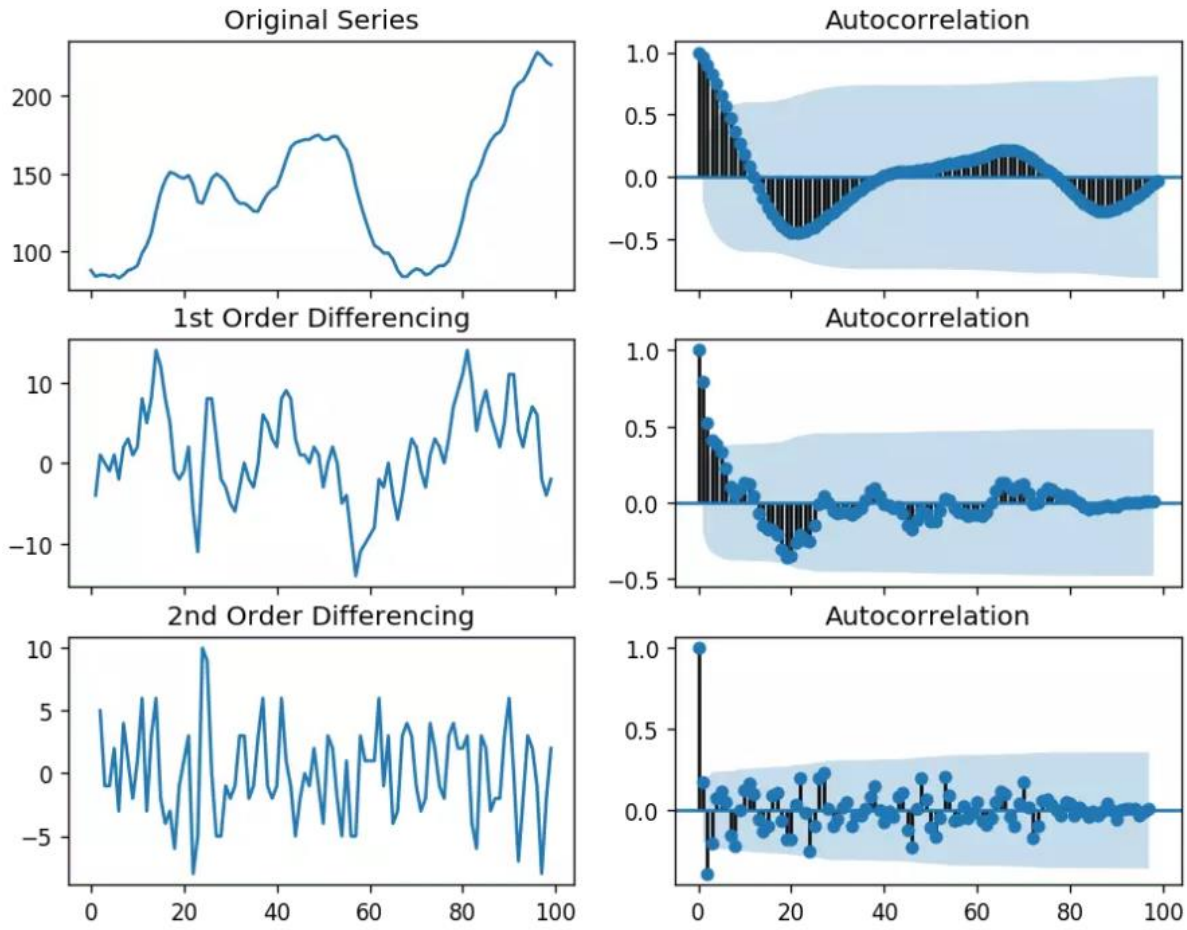


Fig 3.16: Differencing and Autocorrelation(Prabhakaran, 2020).

3.4.3. Data Collection

For the time-series analysis, satellite images of stratocumulus clouds were manually downloaded from the NASA worldview website as done earlier. In total 120 images were downloaded. A 500m resolution image of an area about 14° north in the Atlantic near the east coast of Barbados was taken. One image was taken for the first day of each month starting from January 2010 to December 2019.

The global temperature data has been downloaded from the NASA Goddard Institute for Space Studies website[3].

3.4.4. Image Segmentation and Cloud Category Prediction

The cloud category classification and predictions are done in the same way as detailed in section 3.2. The postprocessing results in a dataset similar to the one shown in fig 3.13. This dataset would be further modified and augmented to create a time-series dataset.

[3] https://data.giss.nasa.gov/gistemp/

3.4.5. Time-Series Dataset

The dataset generated after classification is further modified. A column containing the sum of each cloud class column is added. Another row containing global temperature data is also added. All these are mapped to the image name which is the same as the date on which it is taken. Two further columns containing year and month are also added. The final dataset is shown in fig 3.17.

	Image	Class	fish	sugar	gravel	flower	Global_Temperature	Year	Month	Total_Cloud_Occurence
0	10-Jan	{'fish', 'sugar', 'gravel'}	1	1	1	0	0.76	2010	Jan	3
1	10-Feb	{'fish', 'gravel'}	1	0	1	0	0.84	2010	Feb	2
2	10-Mar	{'fish', 'gravel'}	1	0	1	0	0.93	2010	Mar	2
3	10-Apr	{'gravel'}	0	0	1	0	0.85	2010	Apr	1
4	10-May	NaN	0	0	0	0	0.75	2010	May	0
...	...	...	...	...	...	...	...	...	...	...
115	19-Aug	{'gravel'}	0	0	1	0	0.95	2019	Aug	1
116	19-Sep	{'fish', 'gravel'}	1	0	1	0	0.93	2019	Sep	2
117	19-Oct	{'fish', 'sugar', 'gravel'}	1	1	1	0	1.02	2019	Oct	3
118	19-Nov	{'fish', 'gravel'}	1	0	1	0	1.00	2019	Nov	2
119	19-Dec	{'fish', 'sugar', 'gravel'}	1	1	1	0	1.11	2019	Dec	3

Fig 3.17: Time series dataset.

Note: Total_Cloud_Occurence is the sum of fish, sugar, flower and gravel.

3.4.6. Exploratory Time-Series Analysis

The exploratory time-series analysis has been done using line charts to find the relationship between each of the cloud categories, total cloud occurrence and global temperature data. The library used for this is matplotlib. The charts and related outcome is discussed in the next chapter.

3.4.7. Time-Series Forecasting using ARIMA

The model used for forecasting is ARIMA. For creating the model, the values of auto-regression (p), moving average (q) and differencing (d) needs to be calculated. The first step is to make the series stationary. The library used for plotting the autocorrelation function (acf) and partial autocorrelation (pacf) is tsaplots from statsmodel package.

The pseudocode for getting these values and differencing is given below.

#Plotting original series

Plot acf for Total_Cloud_Occurence and Global_Temperature.

Perform first differencing and plot the acf.

Perform second differencing and plot acf.

Table 3.14: Pseudocode for getting autocorrelation.

The output is shown in fig 3.18. From this, we can see that the series becomes stationary after first differencing. We will validate this with the model data.

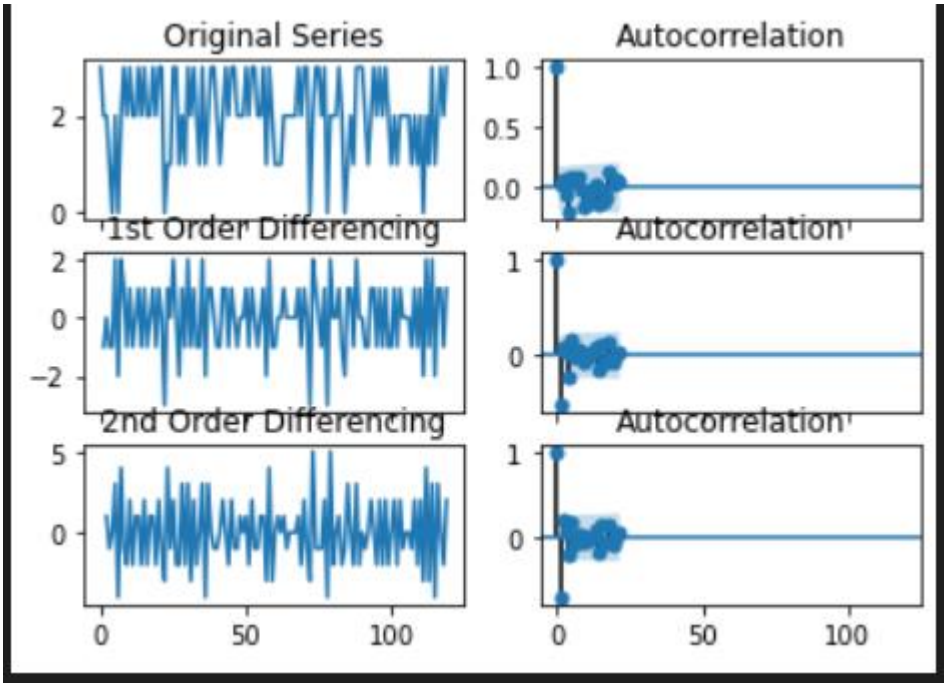


Fig 3.18: ACF plot.

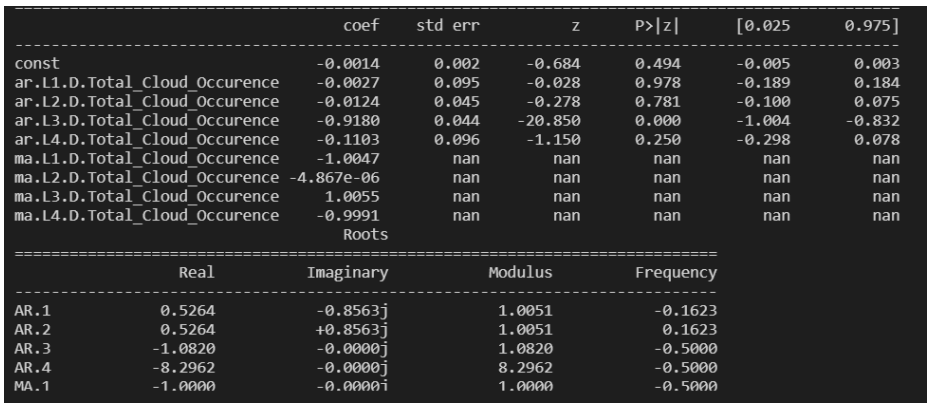
3.4.7.1. Building the ARIMA model

The ARIMA model is created using a function from library statsmodel.tsa.arima_model.ARIMA. The model takes only one value (Total_Cloud_Occurence) and hence is univariate ARIMA. The pseudocode is given below.

#Building 4,1,2 ARIMA from statsmodels.tsa.arima_model import ARIMA Create ARIMA model for Total_Cloud_Occurence with order = 4,1,2. #The order values are generated after a few hit and trail shots. Fit the model . Print model summary.

Table 3.15: Pseudocode for ARIMA.

The model summary is given below.



	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.0014	0.002	-0.684	0.494	-0.005	0.003
ar.L1.D.Total_Cloud_Occurence	-0.0027	0.095	-0.028	0.978	-0.189	0.184
ar.L2.D.Total_Cloud_Occurence	-0.0124	0.045	-0.278	0.781	-0.100	0.075
ar.L3.D.Total_Cloud_Occurence	-0.9180	0.044	-20.850	0.000	-1.004	-0.832
ar.L4.D.Total_Cloud_Occurence	-0.1103	0.096	-1.150	0.250	-0.298	0.078
ma.L1.D.Total_Cloud_Occurence	-1.0047	nan	nan	nan	nan	nan
ma.L2.D.Total_Cloud_Occurence	-4.867e-06	nan	nan	nan	nan	nan
ma.L3.D.Total_Cloud_Occurence	1.0055	nan	nan	nan	nan	nan
ma.L4.D.Total_Cloud_Occurence	-0.9991	nan	nan	nan	nan	nan

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	0.5264	-0.8563j	1.0051	-0.1623
AR.2	0.5264	+0.8563j	1.0051	0.1623
AR.3	-1.0820	-0.0000j	1.0820	-0.5000
AR.4	-8.2962	-0.0000j	8.2962	-0.5000
MA.1	-1.0000	-0.0000j	1.0000	-0.5000

Fig 3.19: ARIMA model summary.

From the model summary, we can see that the moving average becomes ‘nan’ after the second differencing. So, we will create the model without a second differencing. The model summary of that model is given below.

	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.0014	0.002	-0.609	0.543	-0.006	0.003
ar.L1.D.Total_Cloud_Occurence	0.0276	0.093	0.298	0.766	-0.154	0.209
ma.L1.D.Total_Cloud_Occurence	-1.0000	0.025	-39.659	0.000	-1.049	-0.951

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	36.2087	+0.0000j	36.2087	0.0000
MA.1	1.0000	+0.0000j	1.0000	0.0000

Fig 3.20: ARIMA model summary with only one differencing.

We can see that the P>|z| value becomes zero (which is less than 0.05) for L1 moving average. This proves that our assumption regarding the series becoming stationary after first differencing is correct. Further, we will check the residual error of the model.

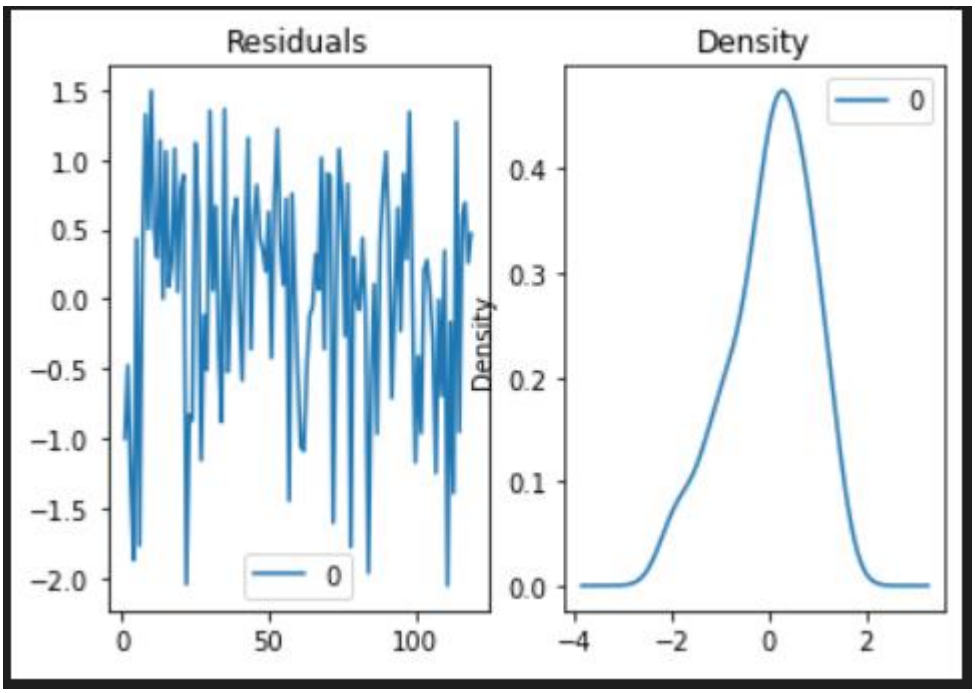


Fig 3.21: Residual error plot.

We can see that the residual mean is near 0 and the density curve is almost normally distributed. So, we will proceed with predictions.

3.4.7.2. Training and Testing the Model.

The model needs to be trained and tested before forecasting is done. The training dataframe consists of the first 100 rows and the test dataframe the last 20 rows. The column Total_Cloud_Occurence is the only feature to be included in the training, testing and forecasting process. The pseudocode is given below.

#Training and testing
Split the data into test and train in 100:20 ratio.
Create model with order 4,1,1 and training data.
Fit the model.
Forecast for 20 rows with confidence interval set to 95%.
Plot the training, testing and forecasted data with the x-axis representing month.

Table 3.16: Pseudocode for training and testing.

The resultant plot is shown below.

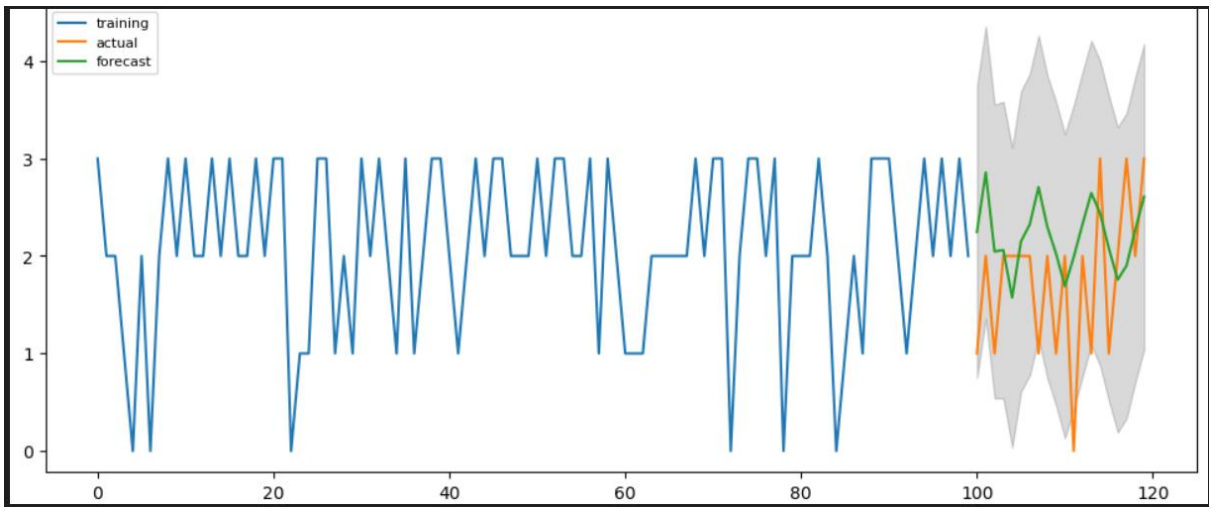


Fig 3.22: Training, actual and forecasted data plot.

3.4.7.3. Forecasting

The model is used for forecasting up to 120 months i.e. 10 years. The pseudocode is given below.

#Forecasting
Forecast using the fitted model for 120 rows with confidence interval set to 95%.
Get the forecasted values and confidence interval.
Get the index values from the forecasted data and convert the values to pandas series for better plotting.
Plot the actual and forecasted values along with the confidence interval region.

Table 3.17: Pseudocode for forecasting.

Chapter 4. Research Findings and Analysis

The entire project is divided into three parts, each fulfilling a research objective. As such the findings of each section will be discussed separately.

4.1. Findings and Analysis of Part 1: Creating a Deep Learning Classifier

In the first part, two classifiers were created, and their performance evaluated. In this section, we will analyse the training and testing performance and discuss the factors which led to finalising the better performing classifier.

4.1.1. Analysing the Training Performance Metrics

The classification models created were UNet-ResNet and UNet-Xception. The UNet segmentation architecture was the same for both the models with the difference being the neural network.

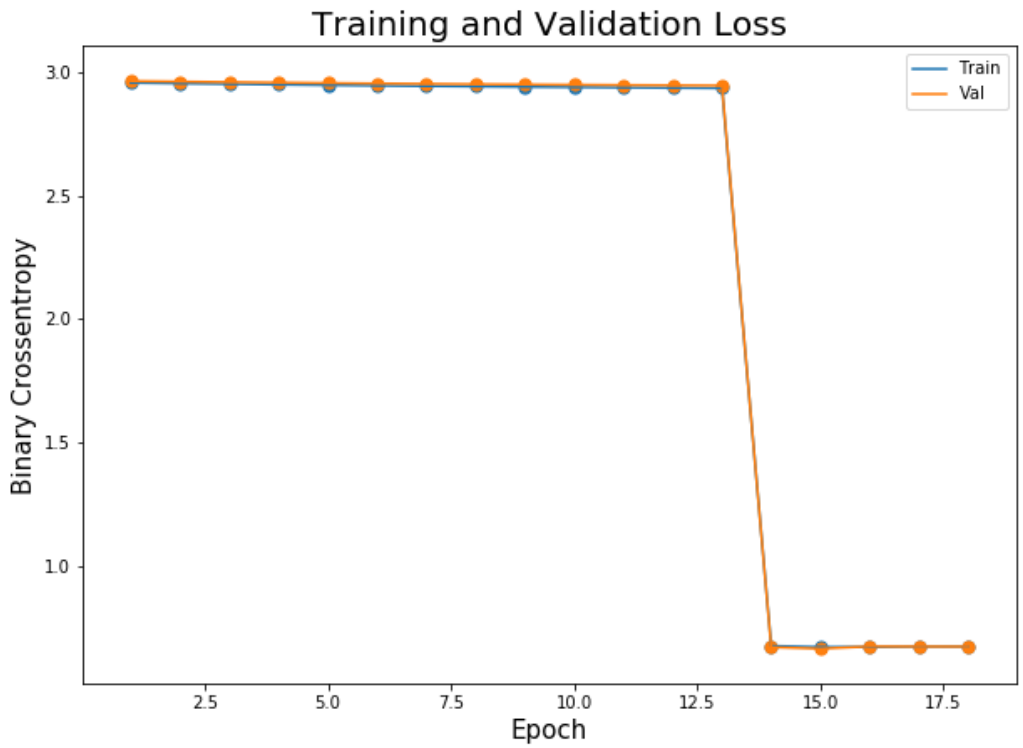


Fig 4.1: Training and validation loss of UNet-ResNet.

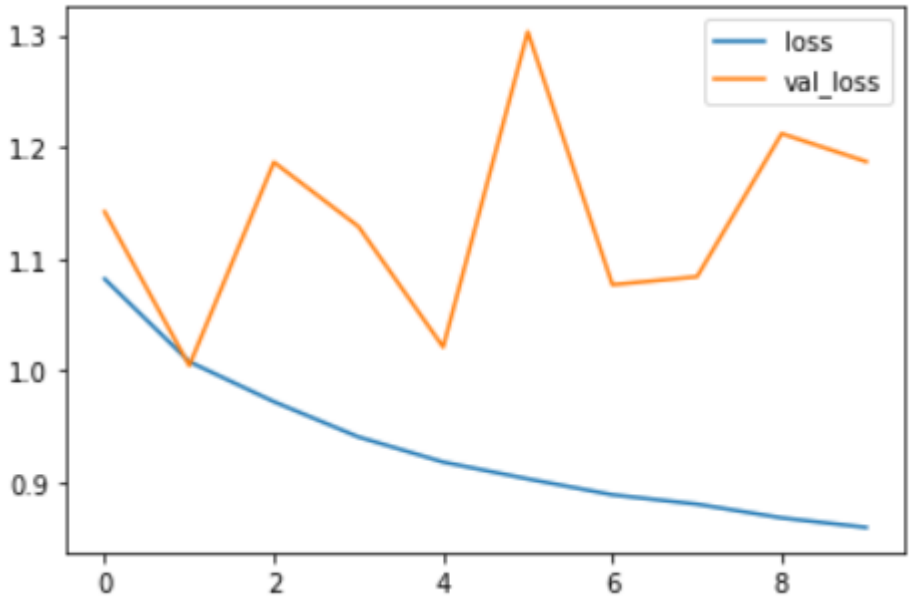


Fig 4.2: Training and validation loss for UNet-Xception

Fig 4.1 and 4.2 show the training and validation loss for UNet-ResNet and UNet-Xception models respectively. As can be seen from Fig 4.1, the training and validation loss is almost the same. However, in Fig 4.2, i.e. the UNet-Xception model, there seems to be a very high divergence between the training and validation losses. Although the training loss goes down with the number of epochs as expected, the validation loss actually goes up. This means the model is overfitted.

Overfitting means that the model is fitted too closely to the training data and as such, it cannot fit additional data or make proper predictions (Overfitting, 2020). This can be seen from the model evaluation as shown in table 3.12 and 3.13.

4.1.2. Analysing Classification Metrics

From these tables, it can be seen that the UNet-ResNet model outperforms the UNet-Xception model. The average classification accuracy for all classes is 71.56% for the former and 65.44% for the latter. Other metrics such as precision, recall, f1 score, and support also shows similar results.

We shall now see the PR and ROC charts of both the models for further analysing their performance.

4.1.2.1. Receiver Operating Characteristic Curves

As stated in the earlier chapter, Receiver Operating Characteristics or ROC curve shows the relationship between the true positive rate and false positive rate. The better classifier will have a ROC curve more left leaning and hence have a higher ROC area. The figures below show the ROC curves for all the classes for both the classifiers. For a better visual representation, a straight diagonal dashed line is drawn.

Class Fish

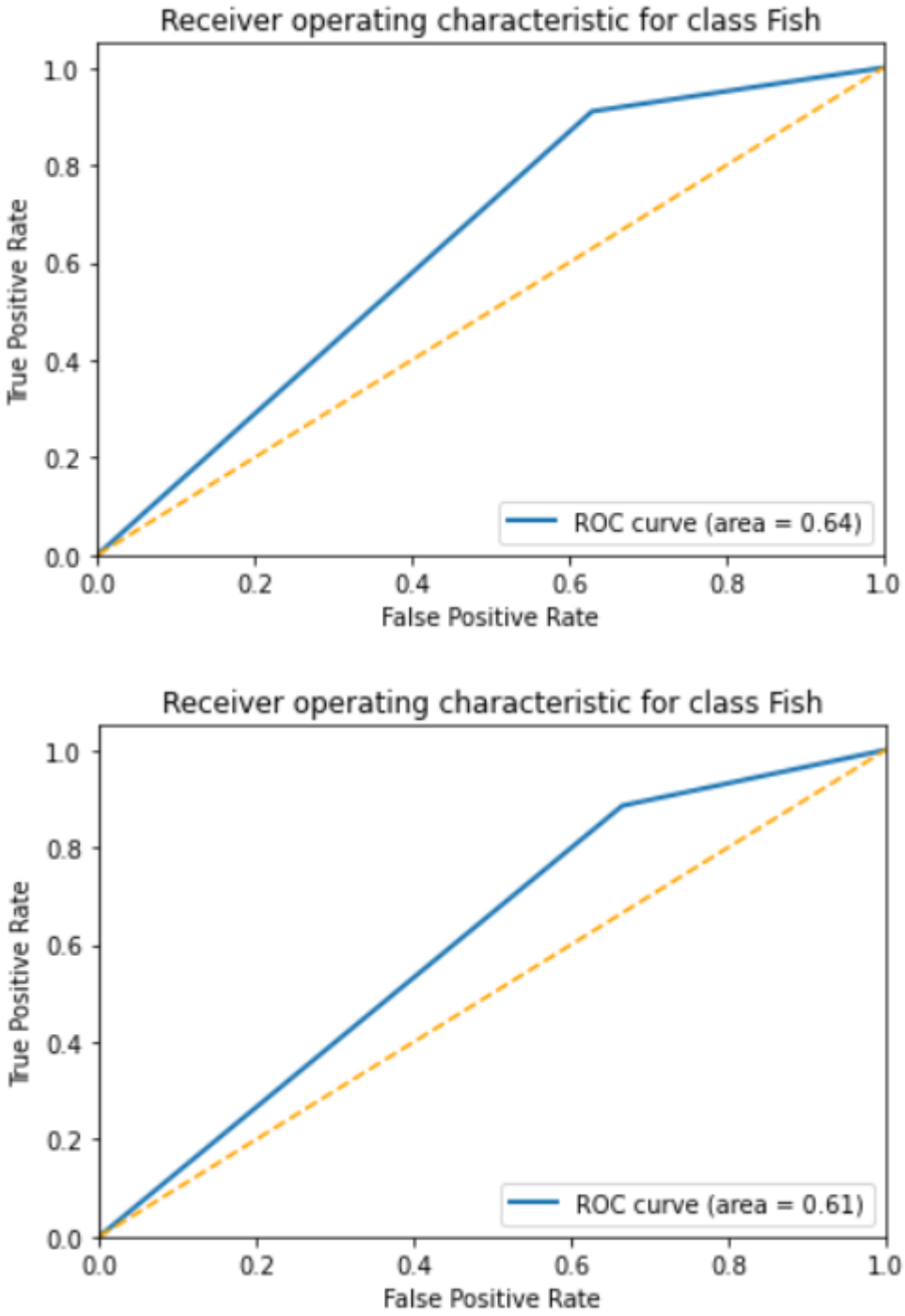
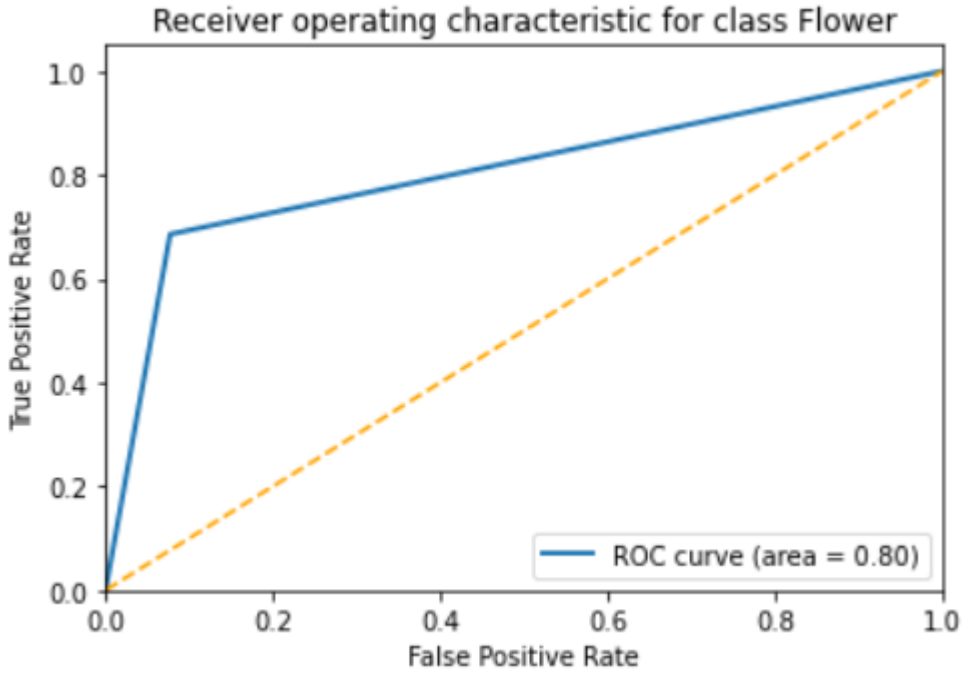


Fig 4.3: ROC for class 'fish'. UNet-ResNet (above), UNet-Xception (below).

Class Flower



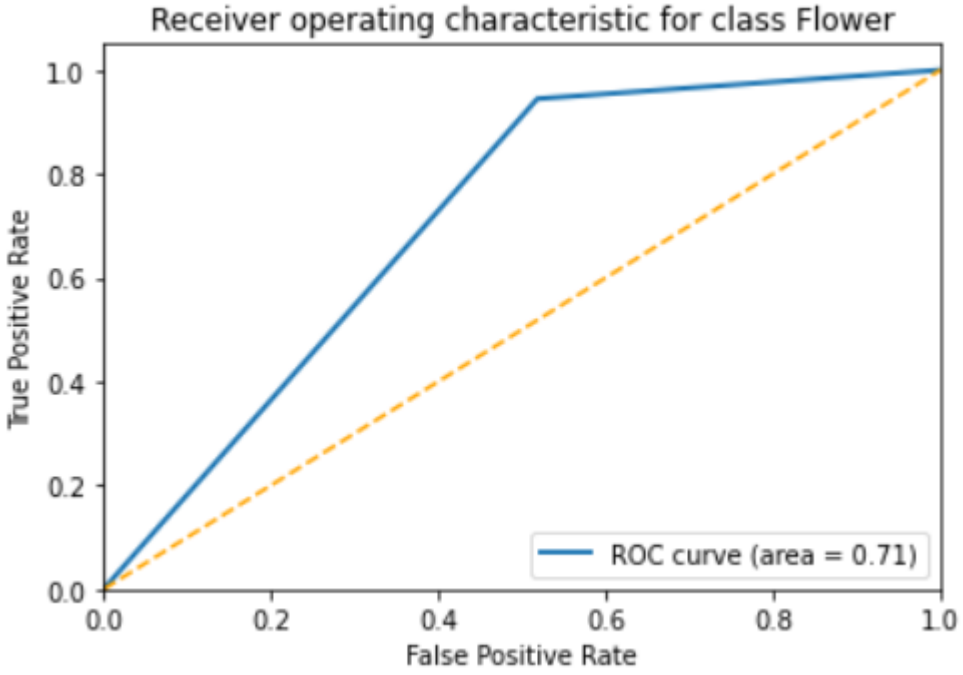


Fig 4.4: ROC for class 'flower'. UNet-ResNet (above), UNet-Xception (below).

Class Gravel

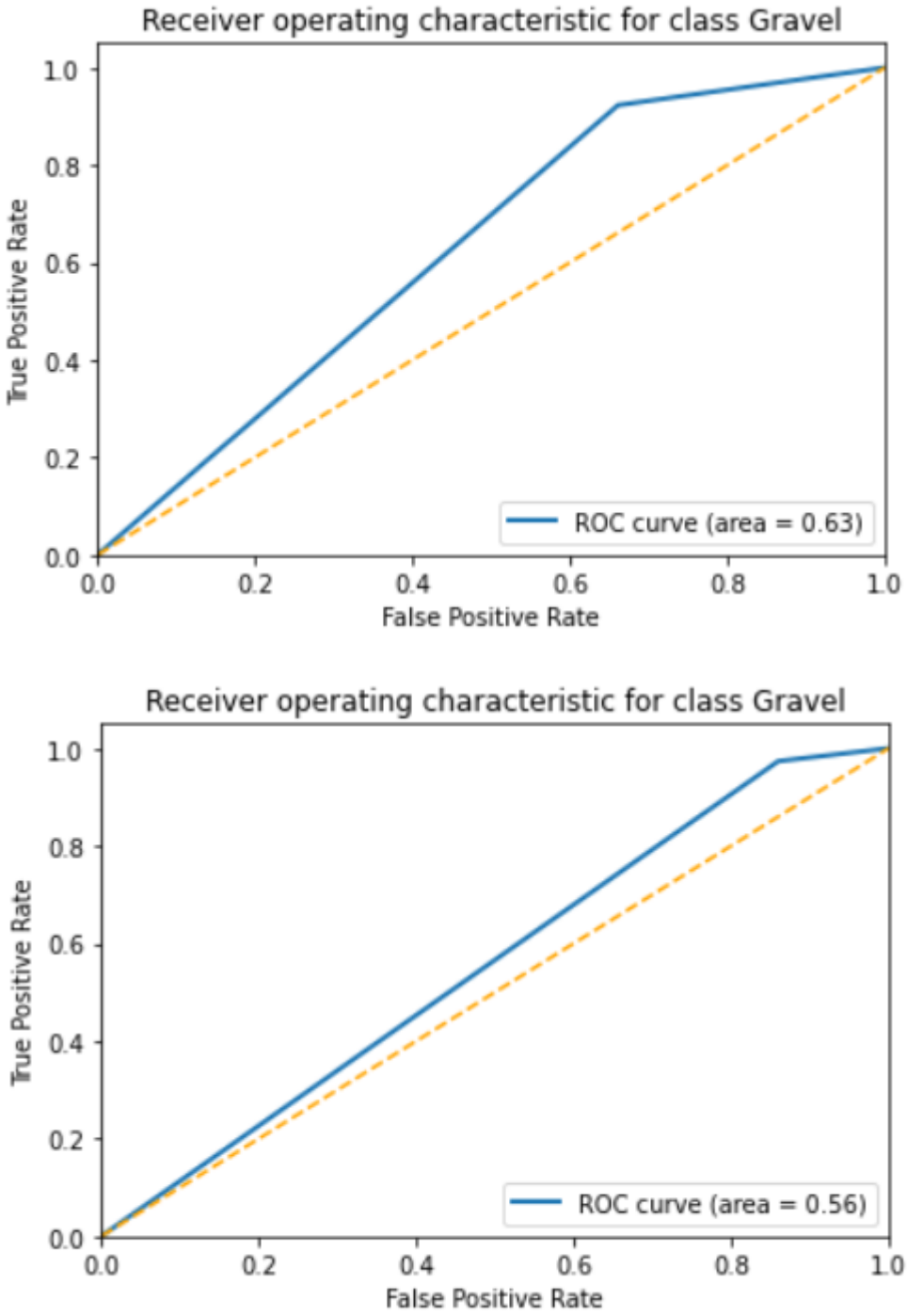


Fig 4.5: ROC for class 'gravel'. UNet-ResNet (above), UNet-Xception (below).

Class Sugar

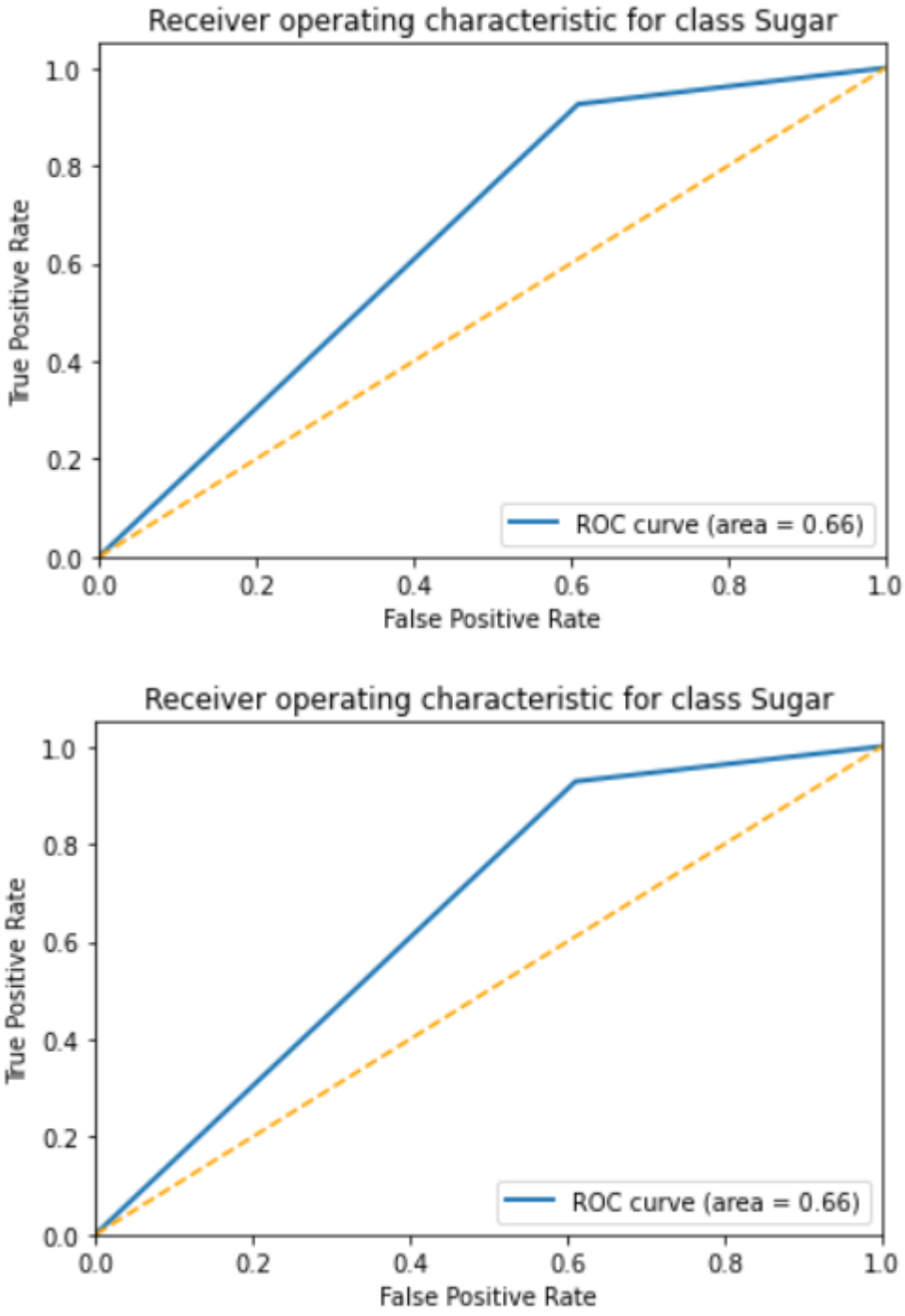


Fig 4.6: ROC for class 'sugar'. UNet-ResNet (above), UNet-Xception (below).

From the above charts, we can see that the UNet-ResNet model has a more left leaning curve as well as a higher ROC curve area, other than class 'sugar', where both are the

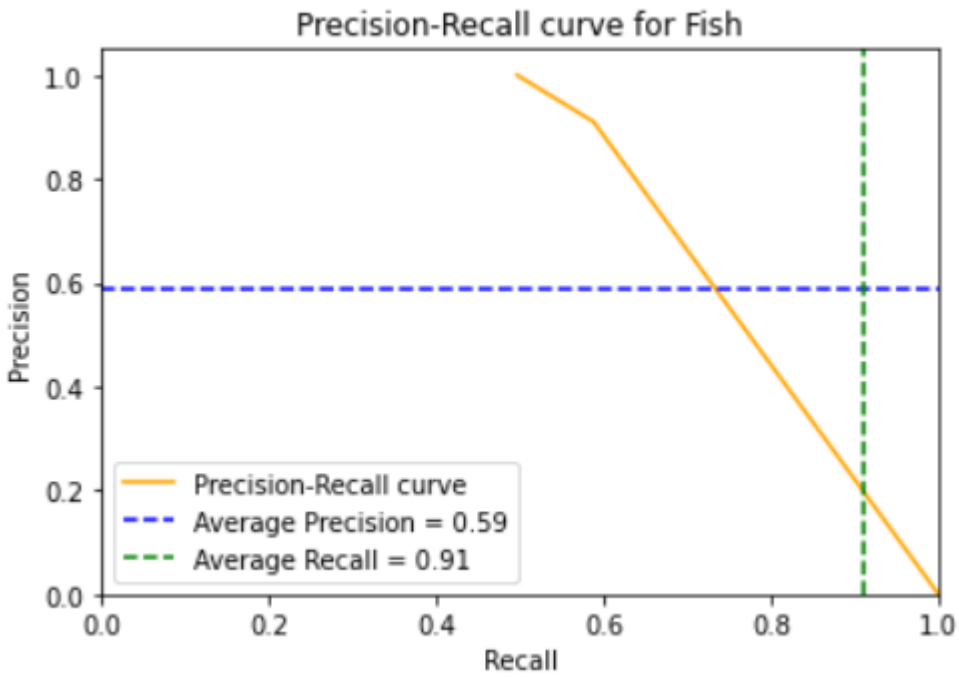
same. This observation is in line with the previous observations from classification metrics that the ResNet model performs better.

Next, we will take a look at the Precision-Recall curves.

4.1.2.2. Precision-Recall Curves

Precision-Recall curve or PR curve is a plot of precision versus recall. This plot basically shows the skill of the classifier in classifying correctly. It has a scale of 0 to 1 on both the axes. The point of (1,1) represents the best skill. A good classifier will have a curve that bows near that point. In other words, the point where the curve begins to slope downwards is nearer to the (1,1) point. The figures below show the PR curve for all the classes for both the classifiers. In addition, the average precision and recall is also plotted. The nearer the point of intersection of these lines to the point of best skill (1,1), better the classifier.

Class Fish



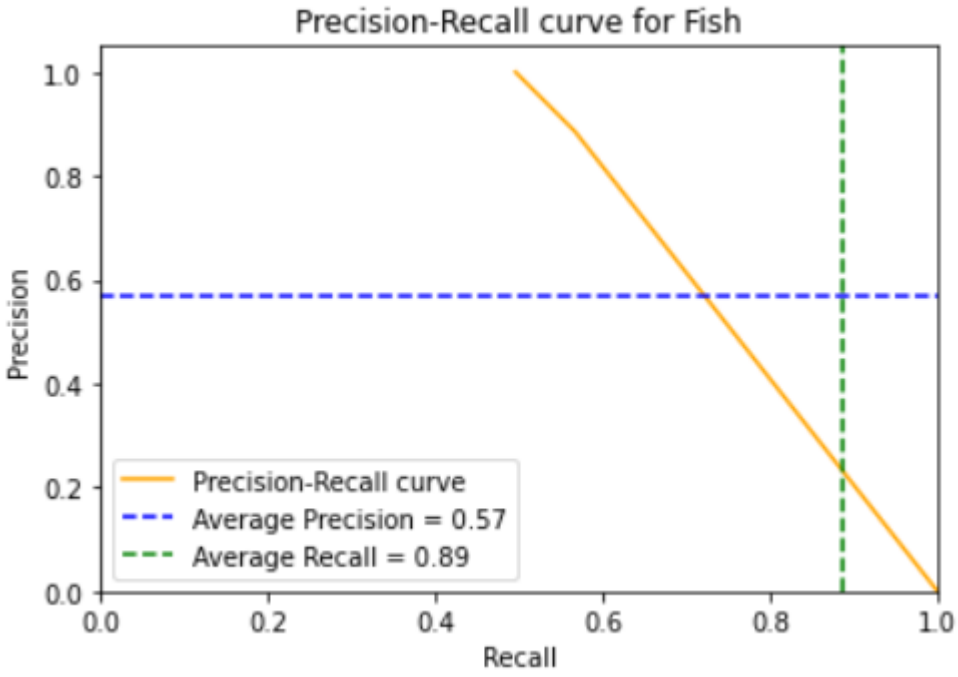
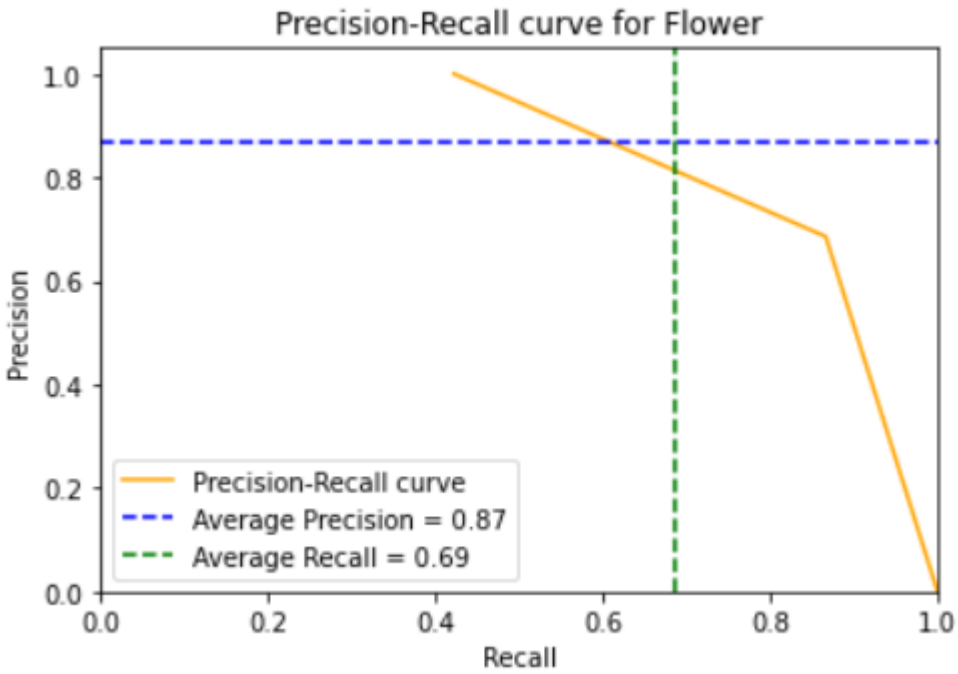


Fig 4.7: PR curve for class 'fish. UNet-ResNet (above), UNet-Xception (below).

Class Flower



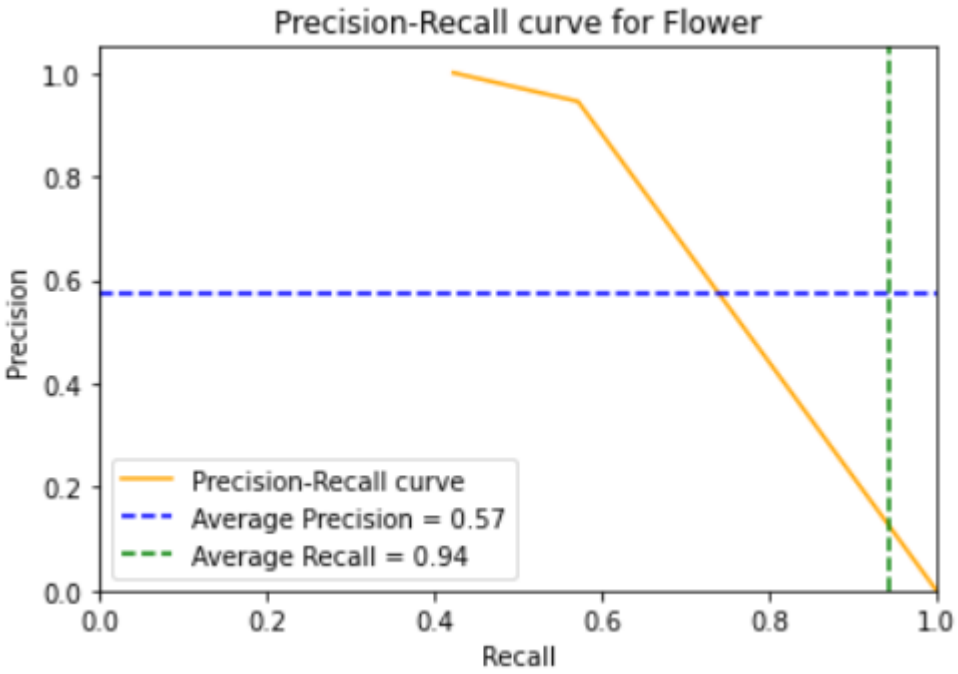
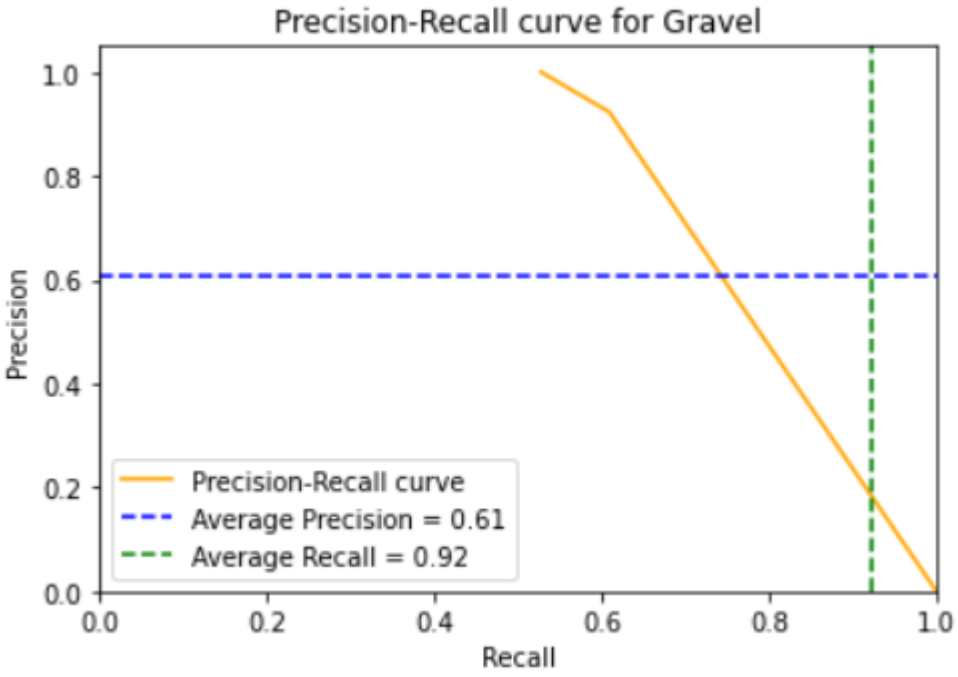


Fig 4.8: PR curve for class 'flower'. UNet-ResNet (above), UNet-Xception (below).

Class Gravel



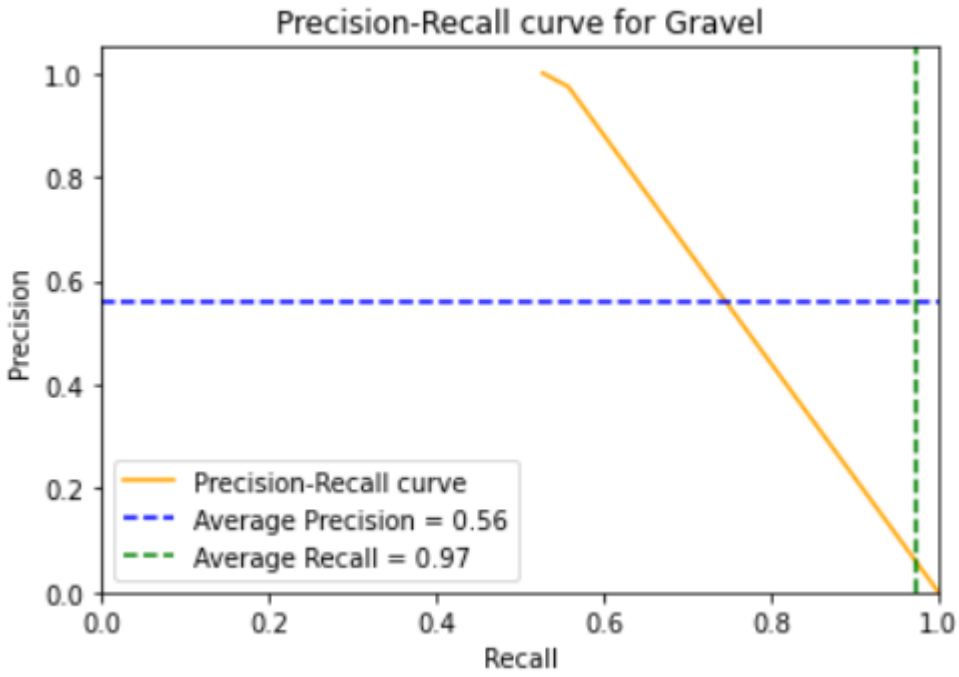


Fig 4.9: PR curve for class 'gravel'. UNet-ResNet (above), UNet-Xception (below).

Class Sugar

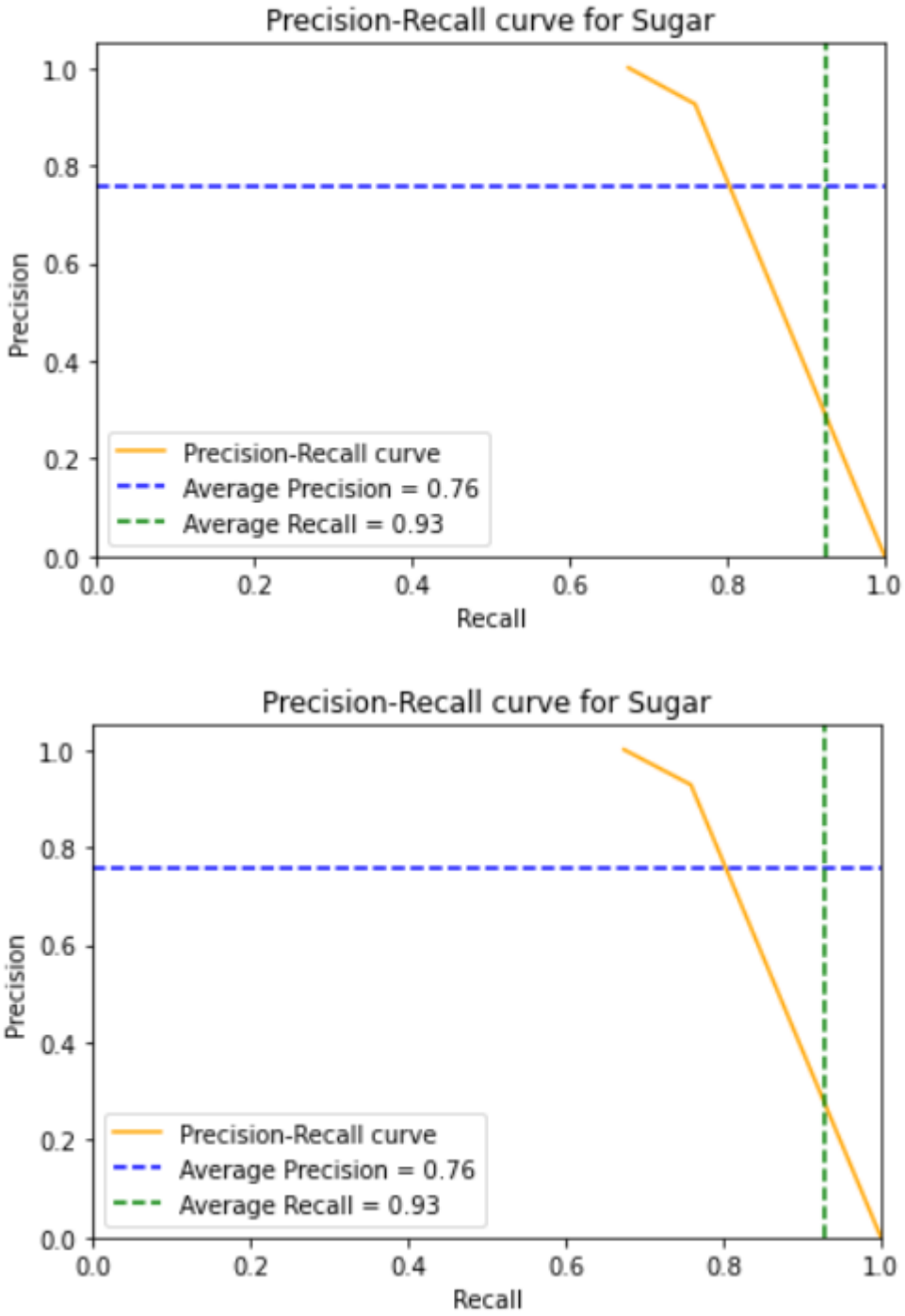


Fig 4.10: PR curve for class 'sugar'. UNet-ResNet (above), UNet-Xception (below).

From the observations above, it is clear that the ResNet model has a better PR curve than the Xception model. It further reinforces the earlier findings that the ResNet classifier is more skilful. As such, this model has been used for classification in the later parts of the project.

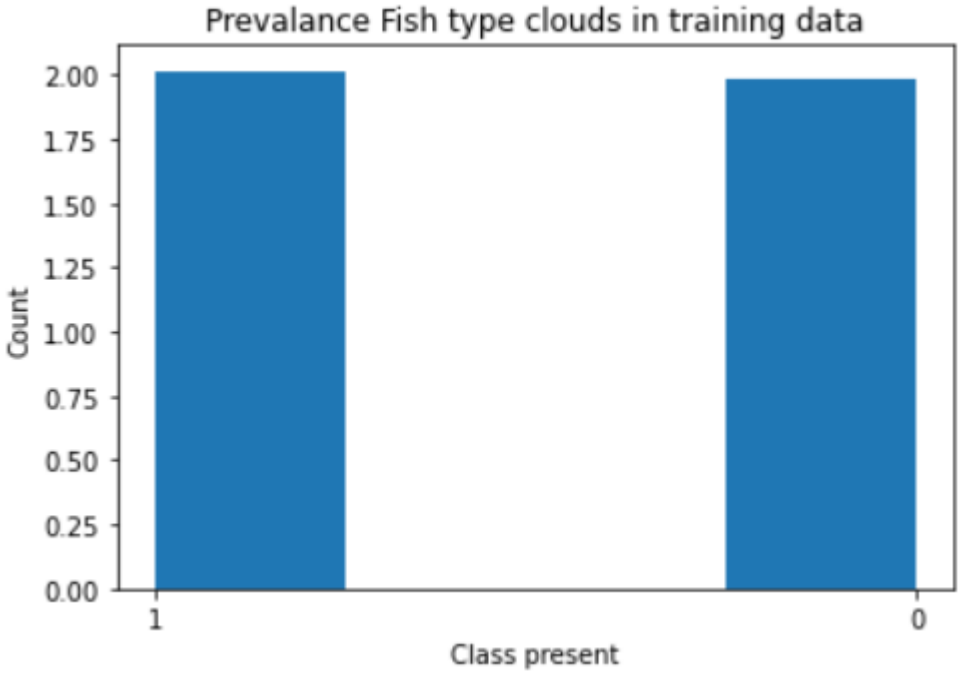
4.2. Findings and Analysis of Part 2: Studying the Relationship between Cloud Categories and Tropical Cyclones

The second part of the project deals with studying the relationship between stratocumulus clouds and tropical cyclones and answering the research question: Is there a relationship between the cloud categories and tropical cyclones?

Here, we will study the presence of each cloud category's presence in the training data and the cyclone dataset. The observations are scaled to make a better comparative analysis as the training dataset is much bigger than the cyclone dataset.

4.2.1. Exploratory Data Analysis of Cloud Categories

Class Fish:



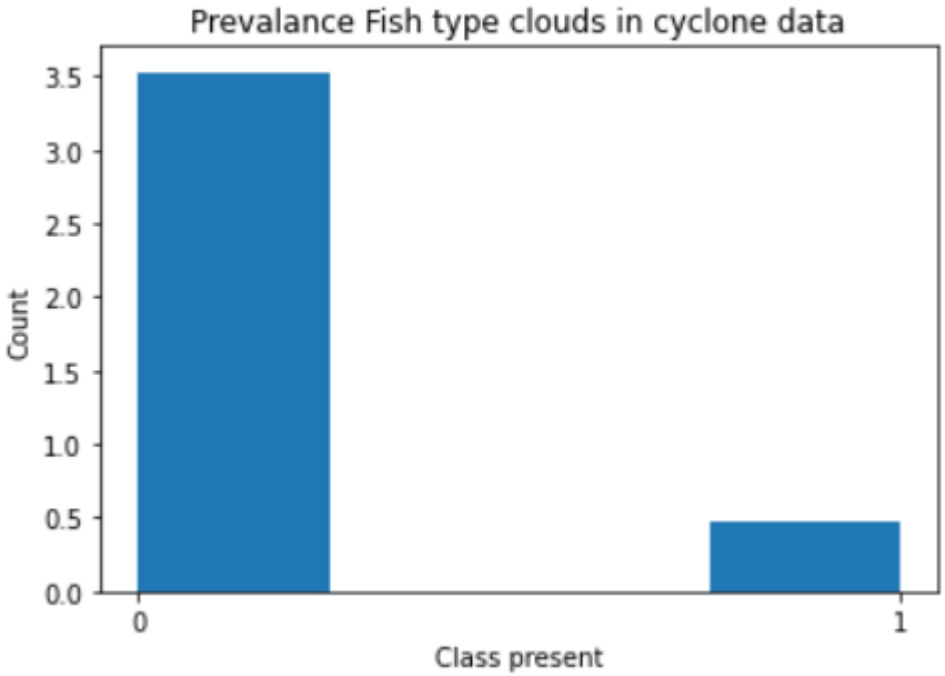
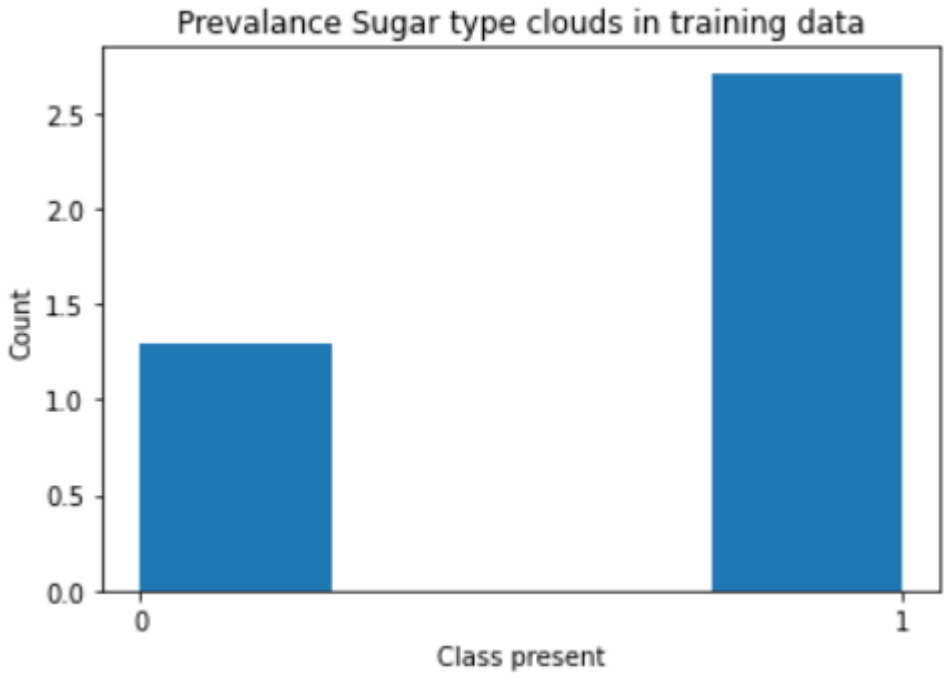


Fig 4.11: Presence of 'fish' clouds in training and cyclone data.

We can see that the clouds of type 'fish' are present equally in the training data, whereas their presence diminishes significantly during cyclones.

Class Sugar:



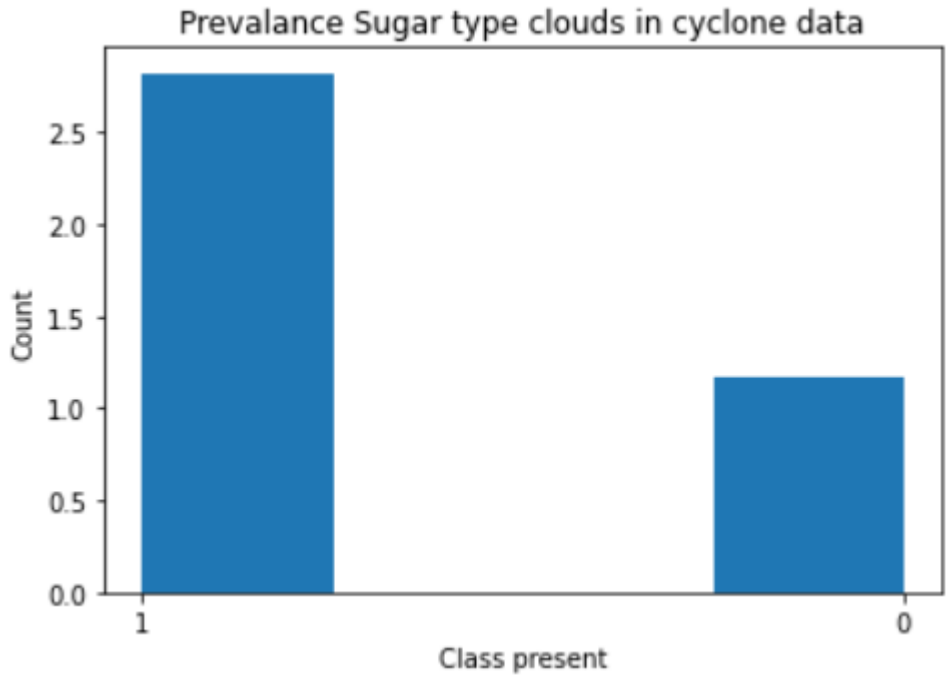
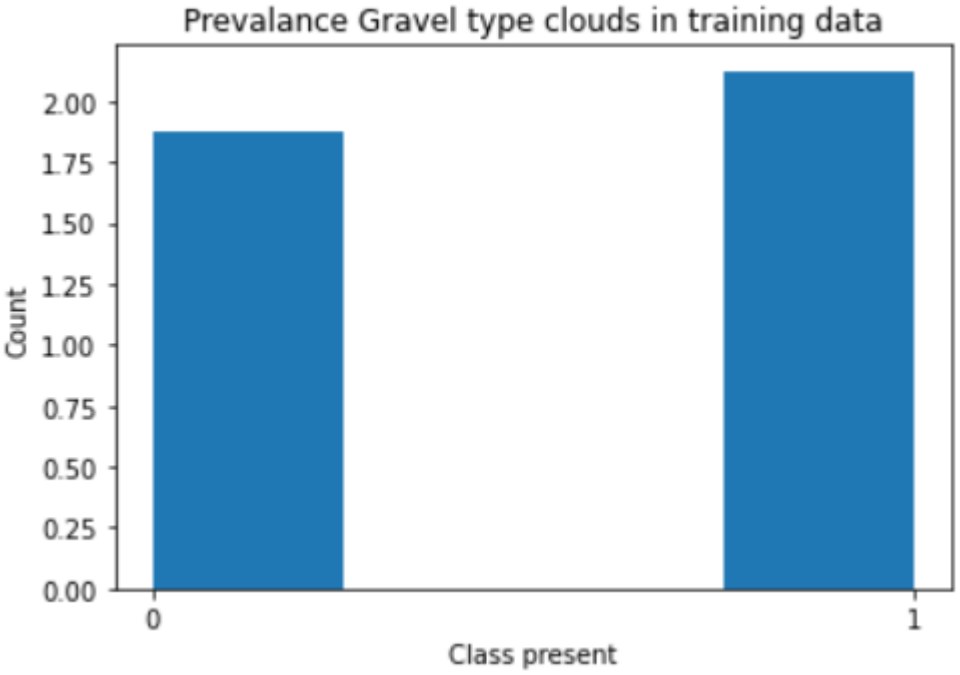


Fig 4.12: Presence of 'sugar' clouds in training and cyclone data.

The clouds of class 'sugar' have a higher presence in the training dataset while its presence significantly diminishes during cyclones.

Class Gravel



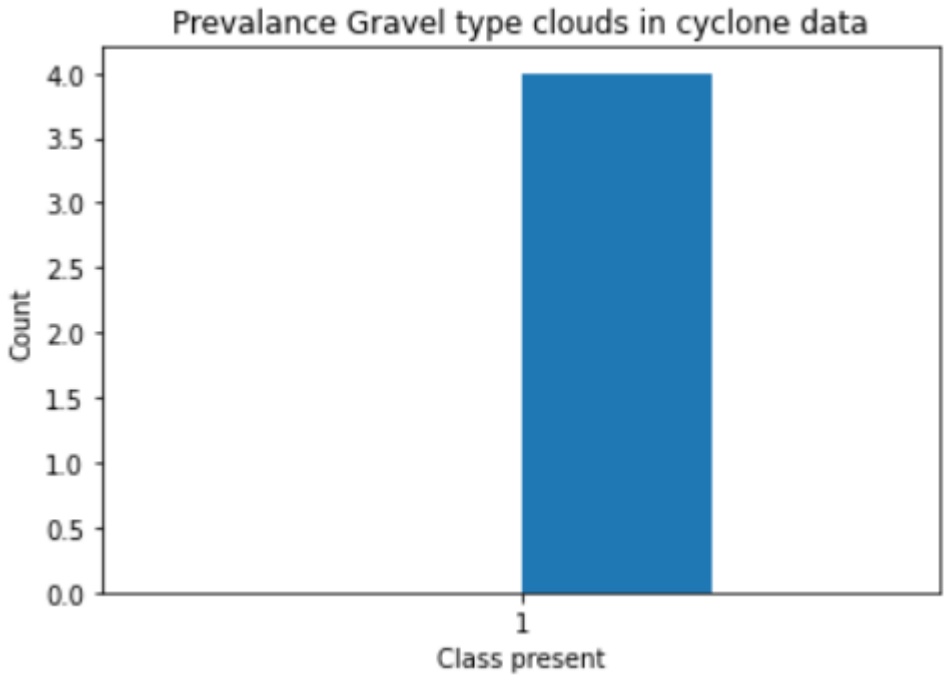


Fig 4.13: Presence of 'gravel' clouds in training and cyclone data.

We make an interesting observation here. The clouds of type 'gravel' have a little more than 50% occurrence in the training data whereas, the cyclone dataset has clouds of type 'gravel' in every observation.

Class Flower:

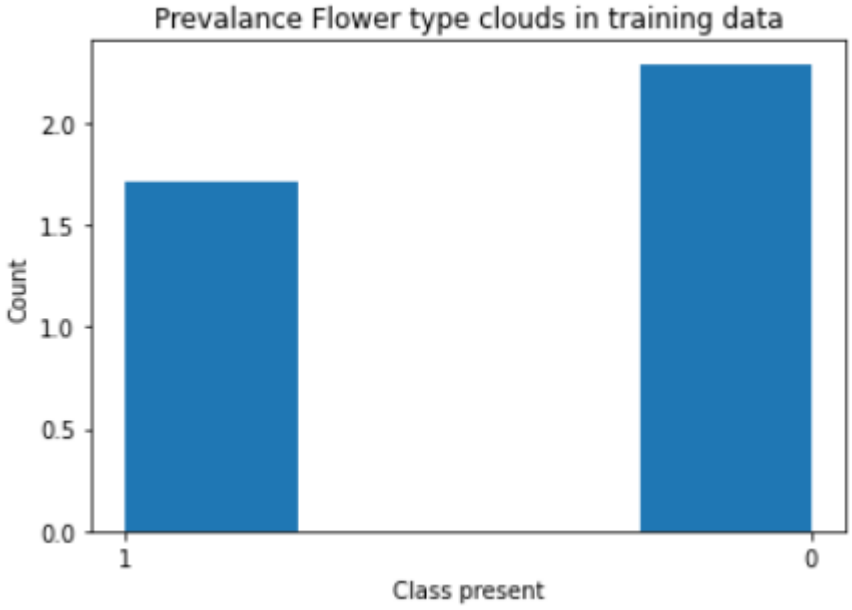


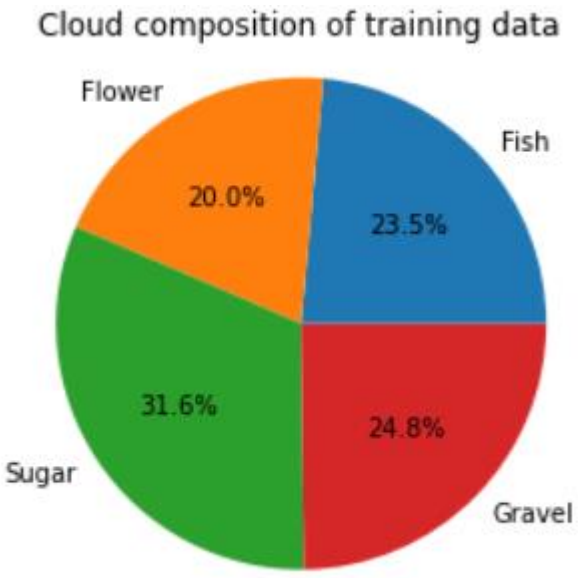
Fig 4.14: Presence of 'flower clouds in training data.

We see an interesting thing here. Although the clouds of type ‘flower have a fairly good presence in the training dataset, the classifier has not detected the presence of these clouds in the cyclone data. This makes plotting a histogram impossible as the particular type is absent from the dataset. This can be seen from fig 4.15, where the column for ‘flower’ simply does not exist.

Date	Image	Class	gravel	sugar	fish
09-09-19	dorian090919	{sugar, gravel}	1	1	0
10-09-19	dorian100919	{sugar, gravel}	1	1	0
11-09-19	dorian110919	{sugar, gravel}	1	1	0
21-08-19	dorian210819	{sugar, gravel}	1	1	0
22-08-19	dorian220819	{sugar, gravel}	1	1	0
23-08-19	dorian230819	{fish, gravel}	1	0	1

Fig 4.15: Column for ‘flower’ is absent.

This becomes more evident from the pie charts in fig 4.16. The pie charts show the composition of clouds in the training and cyclone datasets. The percentage occurrence is calculated as the sum of occurrence of each cloud type divided by the total possible occurrence (number of cloud categories present*number of rows, 4 categories for training and 3 for cyclone data) and expressed as a percentage.



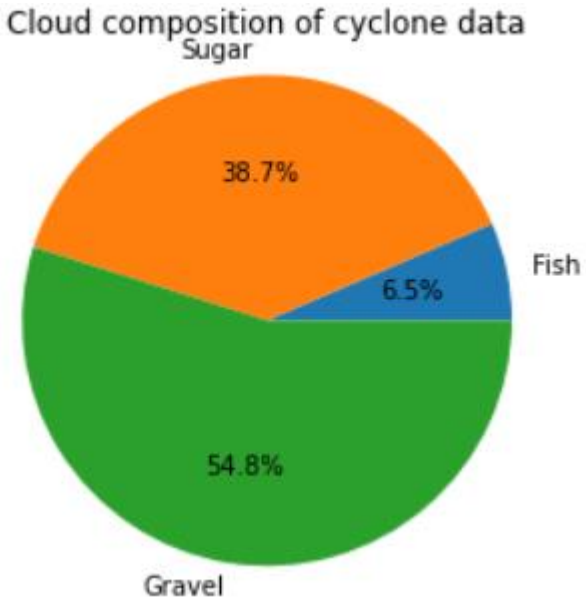


Fig 4.16: Pie chart showing cloud composition in training and cyclone data.

We see that the presence of ‘fish’ type of clouds decreases from 23.5% in the training data to 6.5% in cyclone data. The clouds of type ‘sugar’ and ‘gravel’ increase from 31.6% and 24.8% to 38.7% and 54.8% in the cyclone data respectively. The presence of type ‘flower’ decreases from 20% to zero.

From these observations, we can infer that:

- ‘Sugar’ and ‘gravel’ type of clouds increase during a cyclone with ‘gravel’ type of clouds present in almost every image taken during a cyclone.
- ‘Fish’ type cloud formation decreases during a cyclone.
- ‘Flower’ type clouds are not formed at all during cyclones.

The drastic changes in ‘gravel’ and ‘flower’ type cloud occurrence can be due to two reasons. Either the classifier is misidentifying the image segments or cyclonic activity increases the formation of ‘gravel’ type clouds and restricting the formation of ‘flower’ type clouds.

If we see the classification metrics of ‘gravel’ and ‘flower’ classes of the UNet-ResNet model, we see a classification accuracy of 64.69% for class ‘gravel’ and 82.26% for class ‘flower’.

If we further refer to table 3.13, for class gravel, we see a precision of 0.92 for predicting and 0.34 for not predicting this type of clouds. The recall values of predicting and not

predicting are 0.61 and 0.80 respectively. This means the classifier is performing well on the highly randomised training data and its performance is up to the mark. Further a f1 score of 0.73 and 0.48 for predicting and not predicting shows that there is not much overlap between the image segments. We also see that all the values are at a considerable distance from the 50-50 region which means the classifier is not predicting on chance. Also, there is not a single image in three different cyclones where this category of clouds has not been predicted.

For the class ‘flower’, the precision values for predicting and not predicting the particular cloud type are 0.69 and 0.92 respectively. Similarly, the recall values are 0.87 and 0.8 for predicting and not predicting respectively. The f1 scores for predicting and not predicting 0.77 and 0.86 respectively. These scores are similar (slightly better) to the scores observed for class ‘gravel’. This further strengthens our assumption that the classifier performance is up to the mark. Observations made in section 4.1.2.1 also points in this direction.

Thus, we can safely say that the classifier is not misidentifying image segments at a rate that is not acceptable. This means that we can proceed with the observations made from the exploratory data analysis done in section 4.2.1. The conclusion from the EDA is:

- Tropical cyclones do affect the formation and meso-scale organisation of stratocumulus clouds.
- Cyclonic activity increases the formation of ‘gravel’ and ‘sugar’ type clouds and inhibits the formation of ‘flower’ and ‘fish’ type clouds.

Next, we will see the presence of different cloud categories before and after for each of the three cyclones.

The charts in fig 4.17 show the cloud composition in the form of a stacked bar chart. There are 6 bars for cyclones Dorian and Irma and 5 for Maria. The image clipped on the first date of Maria did not have any stratocumulus cloud and hence was removed after data transformation. The first 3 dates (2 for Maria) are just before the cyclone reached the region and the last three are after it has dissipated.

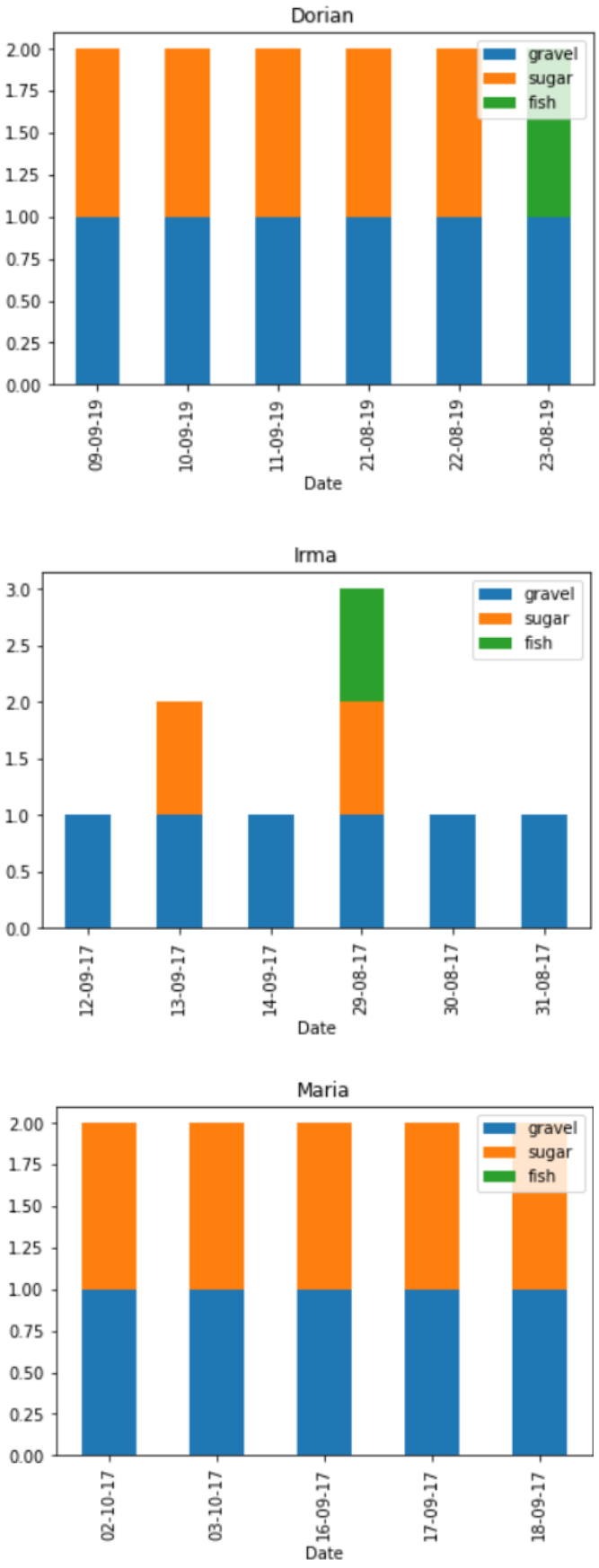


Fig 4.17: Cloud composition before and after cyclones.

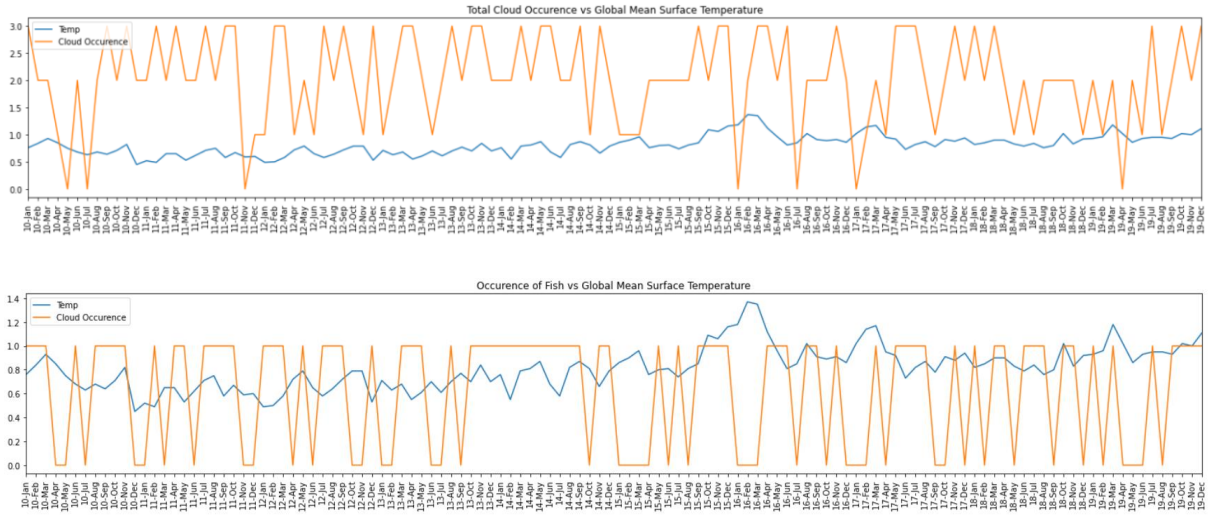
From fig 4.17, we can infer that ‘gravel’ type clouds remain constant during the cyclone, while ‘fish’ type clouds generally form at the end of the cyclone. Nothing conclusive can be said about ‘sugar’ type clouds.

4.3. Findings and Analysis of Part 3: Studying the effect of the rise in Global Temperature and the Formation and Meso-Scale Organisation of Stratocumulus Clouds.

This section involved analysing cloud images of the past 10 years in order to test a prediction made by scientists at the California Institute of Technology. The prediction claimed that the stratocumulus clouds will disappear due to a rise in global temperatures. As global warming is already undergoing currently, such an effect would be seen in the past data. As such, the cloud formations have been plotted along with corresponding global mean surface temperature to perform visual analysis. Further, a time-series forecasting using ARIMA has also been performed to see the future trends of cloud formation.

4.3.1. Exploratory Data Analysis

In this section, we will use line charts to visually compare the trends of total cloud occurrence and each type of cloud formation along with global mean surface temperature over the last 10 years. We will also see if a correlation exists between cloud occurrence and global mean surface temperature.



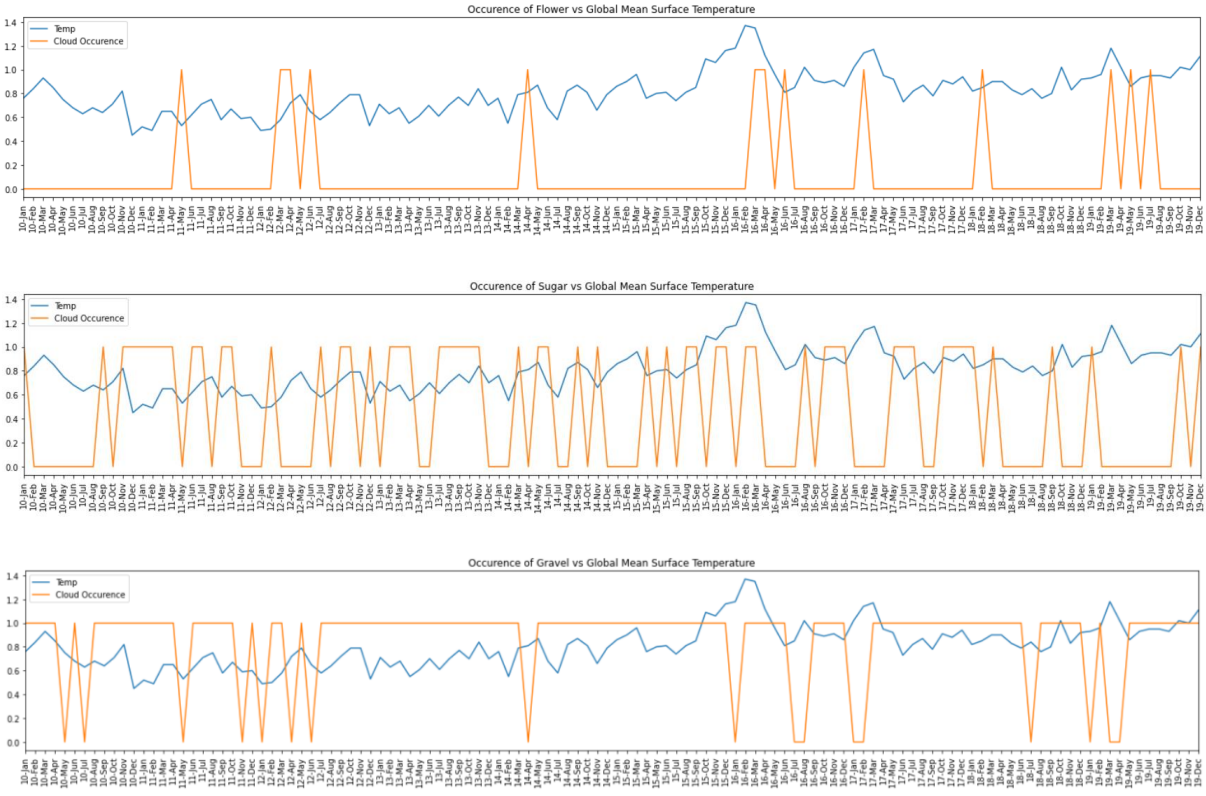


Fig 4.18: Cloud formation occurrences and global temperature over time.

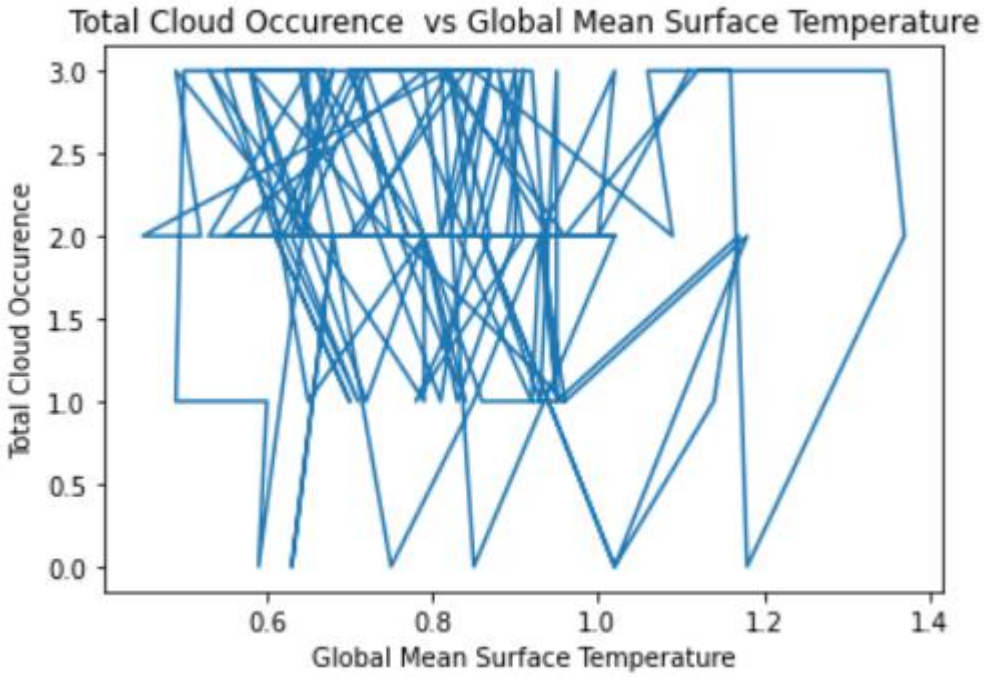


Fig 4.19: Total cloud occurrence vs temperature.

From fig 4.18, we can see that there is no clear pattern in either the total cloud occurrence or individual cloud type formation over time. In the case of temperature, we see a slightly increasing gradient over time, which is in line with the current global warming scenario. Also, no clear relationship can be found between temperature rise and cloud formation.

This can be further seen in fig 4.19, where no proper correlation can be seen between total cloud occurrence and global mean surface temperature.

From the observations made from fig 4.18 and fig 4.19, we can conclude that rising temperatures do not affect the formation and meso-scale organisation of stratocumulus clouds.

4.3.2. Time-Series Forecasting using ARIMA

The aim of the last part of the project is to forecast the cloud formations for the next 10 years. From fig 3.22, we see that the forecasted data points do not follow the actual data points. This may be because of the discrete nature of the cloud data when plotted against time. This type of data is difficult to be fitted in an ARIMA or other time series models (Time series, 2020). However, we get a confidence interval of 95% (shown as the shaded grey region). This means 95% of all the data points will fall in this region (Confidence interval, 2020). The forecast for the next 10 years is shown in fig 4.20.

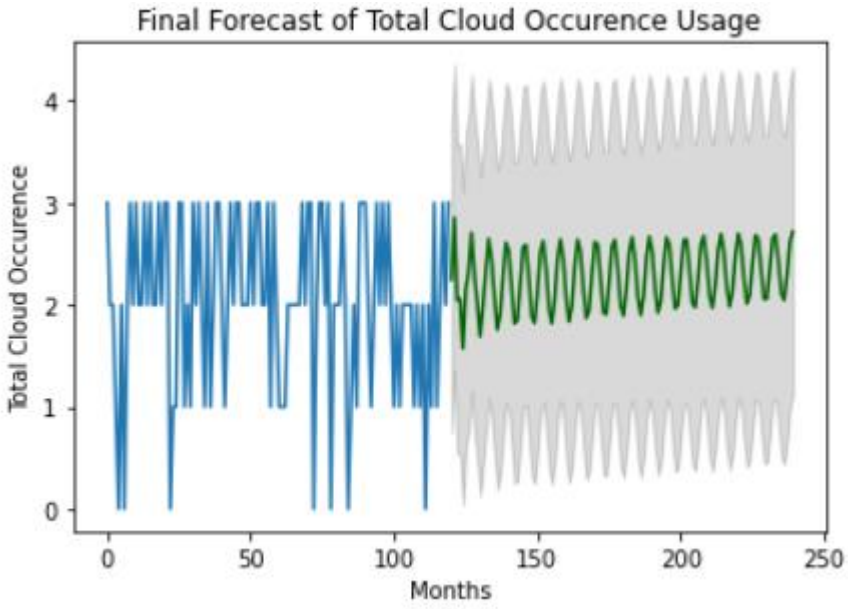


Fig 4.20: Total cloud occurrence forecast.

However, we get a confidence interval range which is similar to the range of actual data points. As such, if we were to consider this mode, then it can be agreed upon that there would not be much variance in the cloud formations in the future.

However, as we have already concluded from section 4.3.1, that temperature does not affect the formation of stratocumulus clouds, we would not need this model and can be sure that the cloud formations would not deviate much from the current levels in the future.

And so, we can conclude that the predictions made by the scientists at the California Institute of Technology (Schneider, Kaul and Pressel, 2019) are indeed too simplistic and do not predict the future formation and meso-scale organisation of stratocumulus clouds in a proper way.

Chapter 5. Conclusions, Limitations and Future Work

In this section, we will conclude the project, discuss the limitations of the work, and give recommendations for future work. As this project has three parts, we shall look into the conclusions of each part, and present a final summarised conclusion. Then we shall look at the limitations and problems face in the project and how they were dealt and what limitations still remain. Finally, future work that can be carried out would be discussed.

5.1. Conclusions

The conclusions to each of the three parts and answers to the research questions are given below.

5.1.1. Conclusion of Part 1: Creating the Classifier

Conclusion to research objective 1: Creating a classifier that can identify and classify stratocumulus clouds from satellite images and answer to research question 1: Is it possible to create a deep learning model that can identify and classify stratocumulus clouds from satellite images?

The first part of this project dealt with creating a neural network based classify that performed multi-class classification of clouds from satellite images. To achieve this objective, we created two segmentation models – UNet-ResNet and UNet-Xception, among which, the UNet-ResNet model performed better. The model used segmentation model architecture, UNet with ResNet forming the backbone. The model performed considerably well both on the training as well as the validation data achieving an average classification accuracy of 71.56% for all classes on the validation dataset. Thus, the research objective 1 stands fulfilled.

The answer to research question 1 is yes, it is possible to create a deep learning model that can identify and classify stratocumulus clouds from satellite images.

5.1.2. Conclusion of Part 2: Studying the relationship between Stratocumulus Cloud Categories and Tropical Cyclones

Conclusion to research objective 2: Studying the relationship between stratocumulus cloud categories and tropical cyclones and answer to research question 2: Do tropical cyclones affect the formation and meso-scale organisation of stratocumulus clouds?

The observations from the results of this part has already been concluded in section 4.2.1. The conclusions are summarised below. The first point answers research question 2.

- Tropical cyclones do affect the formation and meso-scale organisation of stratocumulus clouds.
- Cyclonic activity increases the formation of 'gravel' and 'sugar' type clouds and inhibits the formation of 'flower' and 'fish' type clouds.

5.1.3. Conclusion of Part 3: Studying the effect of the rise in Global Temperature and the Formation and Meso-Scale Organisation of Stratocumulus Clouds.

Conclusion to research objective 3: Studying the effect of the rise in global temperature and the formation of stratocumulus clouds and research question 3: Does the rise in global temperature affect the formation and meso-scale organisation of stratocumulus clouds?

The las part of the project consists of further two parts. The first part is to perform an exploratory data analysis on past 10 years data to determine whether the rise in global temperature affects the formation and meso-scale organisation of stratocumulus clouds. The second part dealt with creating a time-series model to forecast the data for the next 10 years and perform the same study.

For the first part, we concluded that the rise in global temperature does not affect the formation and meso-scale organisation of stratocumulus clouds in any way. For the second part, a good enough ARIMA model could not be created due to the discrete nature of the data. But, as we saw in the first part, that the rise in global temperature has not affected the formation of these clouds in the past and as such, we can safely assume that it would not do so in the future.

So, the answer to research question 3 is no, the rise in global temperature does not affect the formation and meso-scale organisation of stratocumulus clouds.

5.2. Limitations

This project has quite a few limitations and as done earlier, these limitations are described part-wise.

5.2.1. Limitations of Part 1

The creation and training of deep neural network model was severely limited by the availability of computing power and resources. Deep neural networks require a lot of computing power for training. The available computing resources for this project was quite low. The project was executed on a laptop computer with an AMD Ryzen 5 CPU, Nvidia GTX-1650 GPU and 8 GB of RAM. Normally workstation computers having server grade CPUs such as Intel Xenon, multiple dedicated GPUs like Nvidia Tesla and RAM capacity more than 64GB are used for deep learning projects. Unpaid versions of cloud computing services such as Google Colab also offer limited resources and have a time limit. This made it unsuitable for this project. As such training the model even with a very small batch size took a couple of days at least. This meant that reconfiguring the models, especially the Xception model to remove overfitting, unviable due to time constraints.

5.2.2. Limitations of Part 2

The analysis of cloud images was done for only 3 cyclones. Analysing a larger number of cyclones would have provided a better understanding and a much larger dataset to derive insights from. However, the time constraints and scale of the project restricted the number of cyclones to just 3.

5.2.3. Limitations of Part 3

The time-series model did not work as expected due to the discrete nature of the data. Due to which a proper forecasting could not be done. Various hyperparameter optimisations were also done on the model. However, the result could not be improved. Even Auto ARIMA was used (not shown in the methodology section), but still, the outcome remained the same.

5.3. Future Work

Each section can be worked upon further to improve the performances of the models and better analysis can be performed. These recommendations are laid out section-wise below.

5.3.1. Future Work and Recommendations for Part 1

Access to better resources and more time could lead to more manual optimisations of the model parameters. This could lead to a more robust and better performing model which in turn would lead to better predictions and improve the analysis done in the later parts of

the project. Some of the optimisations may include making the network deeper or shallower, using a different backbone or even using a different segmentation architecture.

5.3.2. Future Work and Recommendations for Part 2

The time constraints allowed only three cyclones to be evaluated. The number of images evaluated was also limited to six images per cyclone. This part can be scaled up and more cyclones can be evaluated. This can lead to selection of cyclones from a more varied geographic area and across a much larger time scale. This would lead to the generation of a much bigger dataset and would allow a far better analysis.

5.3.3. Future Work and Recommendations for Part 3

The time-series model failed to work as expected which did not allow a proper forecast. This in turn did not allow for exploratory data analysis to be performed on predicted data. As concluded earlier, the problem was with the discrete nature of the data. If the data points could be plotted in a way more suitable for time-series analysis, this would enable us to perform better forecasting and hence exploratory data analysis of the forecasted data.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J. and Monga, R., 2016. TensorFlow: A System for Large-Scale Machine Learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16). [online] Savannah, GA: USENIX. Available at: <https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf> [Accessed 3 June 2020].

Albumentations, A., 2020. Albumentations-Team/Albumentations. [online] GitHub. Available at: <https://github.com/albumentations-team/albumentations> [Accessed 19 July 2020].

Algorithmia Blog. 2020. Introduction To Optimizers. [online] Available at: <https://algorithmia.com/blog/introduction-to-optimizers> [Accessed 19 July 2020].

Altair-viz.github.io. 2020. Overview — Altair 4.1.0 Documentation. [online] Available at: <https://altair-viz.github.io/getting_started/overview.html> [Accessed 4 August 2020].

Amidi, S., 2020. A Detailed Example Of Data Generators With Keras. [online] Stanford.edu. Available at: <https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly> [Accessed 19 July 2020].

Arakawa, A., and W. H. Schubert, 1974: Interaction of a Cumulus Cloud Ensemble with the Large-Scale Environment, Part I. Journal of the Atmospheric Sciences, 31 (3), 674{701, doi: 10.1175/1520-0469(1974)031h0674:IOACCEi2.0.CO;2.

Bai, K., 2020. A Comprehensive Introduction To Different Types Of Convolutions In Deep Learning. [online] Available at: <https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215> [Accessed 4 July 2020].

Bony, S., Schulz, H., Vial, J. and Stevens, B., 2020. Sugar, Gravel, Fish, and Flowers: Dependence of Mesoscale Patterns of Trade-Wind Clouds on Environmental Conditions. Geophysical Research Letters, 47(7).

Bretherton, C., 2015. Insights into low-latitude cloud feedbacks from high-resolution models. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 373(2054), p.20140415.

Brownlee, J., 2020. How To Use ROC Curves And Precision-Recall Curves For Classification In Python. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/> [Accessed 26 July 2020].

Brownlee, J., 2020. Transfer Learning In Keras With Computer Vision Models. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/> [Accessed 4 August 2020].

Chollet, F., 2017. Xception: Deep Learning with Depthwise Separable Convolutions. [online] arXiv:1610.02357v3. Available at: <https://arxiv.org/pdf/1610.02357.pdf> [Accessed 1 June 2020].

Ciresan, D.C., Gambardella, L.M., Giusti, A., Schmidhuber, J.: Deep neural networks segment neuronal membranes in electron microscopy images. In: NIPS. pp. 2852–2860 (2012)

Das, S. (2019). CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more. [online] Medium. Available at: https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5 [Accessed 4 June 2020].

En.wikipedia.org. 2020. Confidence Interval. [online] Available at: <https://en.wikipedia.org/wiki/Confidence_interval> [Accessed 29 July 2020].

En.wikipedia.org. 2020. F1 Score. [online] Available at: <https://en.wikipedia.org/wiki/F1_score> [Accessed 26 July 2020].

En.wikipedia.org. 2020. Hurricane Dorian. [online] Available at: <https://en.wikipedia.org/wiki/Hurricane_Dorian> [Accessed 23 July 2020].

En.wikipedia.org. 2020. Hurricane Irma. [online] Available at: <https://en.wikipedia.org/wiki/Hurricane_Irma> [Accessed 23 July 2020].

En.wikipedia.org. 2020. Hurricane Maria. [online] Available at: <https://en.wikipedia.org/wiki/Hurricane_Maria> [Accessed 24 July 2020].

En.wikipedia.org. 2020. Matplotlib. [online] Available at: <https://en.wikipedia.org/wiki/Matplotlib> [Accessed 1 August 2020].

En.wikipedia.org. 2020. Overfitting. [online] Available at: <https://en.wikipedia.org/wiki/Overfitting> [Accessed 27 July 2020].

En.wikipedia.org. 2020. Time Series. [online] Available at: <https://en.wikipedia.org/wiki/Time_series> [Accessed 24 July 2020].

Fung, V., 2020. An Overview Of Resnet And Its Variants. [online] Medium. Available at: <https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035> [Accessed 31 July 2020].

Girshick, R., Donahue, J., Darrell, T. and Malik, J., 2016. Region-Based Convolutional Networks for Accurate Object Detection and Segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 38(1), pp.142-158.

Godoy, D., 2020. Understanding Binary Cross-Entropy / Log Loss: A Visual Explanation. [online] Medium. Available at: <https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a> [Accessed 20 July 2020].

Goodfellow, I, Bengio, Y and Courville, A. (2016) Deep Learning MIT Press

GroundAI. 2020. Combining Crowd-Sourcing And Deep Learning To Understand Meso-Scale Organization Of Shallow Convection. [online] Available at: <https://www.groundai.com/project/combining-crowd-sourcing-and-deep-learning-to-understand-meso-scale-organization-of-shallow-convection/1> [Accessed 30 May 2020].

International Cloud Atlas. (2019). Definitions of clouds | International Cloud Atlas. [online] Available at: https://cloudatlas.wmo.int/clouds-definitions.html [Accessed 22 Dec. 2019].

International Cloud Atlas. 2020. Stratocumulus (Sc) | International Cloud Atlas. [online] Available at: <https://cloudatlas.wmo.int/en/stratocumulus-sc.html> [Accessed 4 June 2020].

Jordan, J., 2020. An Overview Of Semantic Image Segmentation.. [online] Jeremy Jordan. Available at: <https://www.jeremyjordan.me/semantic-segmentation/> [Accessed 20 July 2020].

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In Proc. IEEE Conf. Comp. Vis. Patt. Recogn., 2016.

Keras.io. 2020. Keras Documentation: About Keras. [online] Available at: <https://keras.io/about/> [Accessed 3 June 2020].

Keras.io. 2020. Keras Documentation: Keras API Reference. [online] Available at: <https://keras.io/api/> [Accessed 3 June 2020].

Keras.io. 2020. Keras Documentation: Keras Applications. [online] Available at: <https://keras.io/api/applications/> [Accessed 30 May 2020].

Kohli, S., 2020. Understanding A Classification Report For Your Machine Learning Model. [online] Medium. Available at: <https://medium.com/@kohlishivam5522/understanding-a-classification-report-for-your-machine-learning-model-88815e2ce397> [Accessed 26 July 2020].

Krizhevsky, A., Sutskever, I. and Hinton, G., 2017. ImageNet classification with deep convolutional neural networks. Communications of the ACM, 60(6), pp.84-90.

Lecun, Y., Bottou, L., Bengio, Y. and Haffner, P., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), pp.2278-2324.

Liu, L., 2020. Liyuanlucasliu/Radam. [online] GitHub. Available at: <https://github.com/LiyuanLucasLiu/RAdam> [Accessed 19 July 2020].

Mathworld.wolfram.com. 2020. Convolution -- From Wolfram Mathworld. [online] Available at: <https://mathworld.wolfram.com/Convolution.html> [Accessed 4 July 2020].

Medium. 2020. An Overview Of Resnet And Its Variants. [online] Available at: <https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035> [Accessed 31 May 2020].

O'Brien, T. (2019). Met Éireann defends timings of warnings given over Storm Elsa. [online] The Irish Times. Available at: https://www.irishtimes.com/news/ireland/irish-news/met-%C3%A9ireann-defends-timings-of-warnings-given-over-storm-elsa-1.4120156 [Accessed 22 Dec. 2019].

Paperswithcode.com. 2020. Papers With Code - Imagenet Leaderboard. [online] Available at: <https://paperswithcode.com/sota/image-classification-on-imagenet> [Accessed 4 June 2020].

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Lin, Z., Chintala, S. and Chilamkurthy, S., 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. Conference on Neural Information Processing Systems, [online] 9015. Available at: <https://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf> [Accessed 3 June 2020].

Prabhakaran, S., 2020. ARIMA Model - Complete Guide To Time Series Forecasting In Python | ML+. [online] Machine Learning Plus. Available at: <https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/> [Accessed 24 July 2020].

Rasp, Stephan & Schulz, Hauke & Bony, Sandrine & Stevens, Bjorn. (2019). Combining crowd-sourcing and deep learning to understand meso-scale organization of shallow convection.

Robogrok.com. 2020. Masking And Image Segmentation. [online] Available at: <http://www.robogrok.com/2-2-1_Masking_and_Image_Segmentation.php#:~:text=Masking%20is%20an%20image%20processing,motion%20detection%2C%20and%20noise%20reduction.> [Accessed 18 July 2020].

Ronneberger, O., Fischer, P. and Brox, T., 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. Computer Science Department and BIOSS Centre for Biological Signalling Studies, University of Freiburg, Germany, [online] arXiv:1505.04597v1 [cs.CV]. Available at: <https://arxiv.org/pdf/1505.04597v1.pdf> [Accessed 1 June 2020].

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision, 115(3), pp.211-252.

Saha, S., 2020. A Comprehensive Guide To Convolutional Neural Networks—The ELI5 Way. [online] Medium. Available at: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53> [Accessed 4 July 2020].

Schneider, T., Kaul, C. and Pressel, K., 2019. Possible climate transitions from breakup of stratocumulus decks under greenhouse warming. Nature Geoscience, 12(3), pp.163-167.

Stevens, B., Bony, S., Brogniez, H., Hentgen, L., Hohenegger, C., Kiemle, C., L'Ecuyer, T., Naumann, A., Schulz, H., Siebesma, P., Vial, J., Winker, D. and Zuidema, P. (2019). Sugar, gravel, fish and flowers: Mesoscale cloud patterns in the trade winds. Quarterly Journal of the Royal Meteorological Society.

Tsang, S., 2020. Review: Inception-V3 — 1St Runner Up (Image Classification) In ILSVRC 2015. [online] Medium. Available at: <https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c> [Accessed 4 August 2020].

Voosen, P., 2020. A World Without Clouds? Hardly Clear, Climate Scientists Say. [online] Science | AAAS. Available at:

<https://www.sciencemag.org/news/2019/02/world-without-clouds-hardly-clear-climate-scientists-say> [Accessed 24 July 2020].

Wang SC. (2003) Artificial Neural Network. In: Interdisciplinary Computing in Java Programming. The Springer International Series in Engineering and Computer Science, vol 743. Springer, Boston, MA

Wei, Y., Xia, W., Lin, M., Huang, J., Ni, B., Dong, J., Zhao, Y. and Yan, S., 2016. HCP: A Flexible CNN Framework for Multi-Label Image Classification. IEEE Transactions on Pattern Analysis and Machine Intelligence, 38(9), pp.1901-1907.

Zooniverse.org. 2020. Sugar, Flower, Fish Or Gravel. [online] Available at: <https://www.zooniverse.org/projects/raspstephan/sugar-flower-fish-or-gravel> [Accessed 30 May 2020].

Appendix – A

As the code required for this project is too large, only the code directly related to machine learning has been provide in this section. The complete code work can be found in the form of annotated Jupyter notebooks can be found at: https://github.com/aibhishek/MSC_ThesisProject.

Python code for creating the classifiers

Library import and dataset loading

```
#Library Import
import json
import os
import albumentations as albu
import cv2
import keras
import tensorflow as tf
from keras import backend as K
from keras.models import Model
from keras.layers import Input
from keras.layers.convolutional import Conv2D, Conv2DTranspose
from keras.layers.pooling import MaxPooling2D
from keras.layers.merge import concatenate
from keras.losses import binary_crossentropy
from keras.optimizers import Adam, Nadam
from keras.callbacks import Callback, ModelCheckpoint
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from tqdm import tqdm
from sklearn.model_selection import train_test_split
import segmentation_models as sm

#Configuring to run session on GPU
config = tf.compat.v1.ConfigProto(device_count = {'GPU': 0})
tf.compat.v1.Session(config=config)

os.getcwd()
os.chdir('D:\\DS\\Datasets\\understanding_cloud_organization')

train_df = pd.read_csv('train.csv')
train_df['ImageId'] = train_df['Image_Label'].apply(lambda x: x.split('_')[0])
train_df['ClassId'] = train_df['Image_Label'].apply(lambda x: x.split('_')[1])
train_df['hasMask'] = ~ train_df['EncodedPixels'].isna()
print(train_df.shape)

mask_count_df = train_df.groupby('ImageId').agg(np.sum).reset_index()
mask_count_df.sort_values('hasMask', ascending=False, inplace=True)
print(mask_count_df.shape)
mask_count_df.head()

sub_df = pd.read_csv('sample_submission.csv')
sub_df['ImageId'] = sub_df['Image_Label'].apply(lambda x: x.split('_')[0])
test_imgs = pd.DataFrame(sub_df['ImageId'].unique(), columns=['ImageId'])
```

Utility functions

References: Mask Conversion: **https://www.kaggle.com/paulorzp/rle-functions-run-lenght-encode-decode**

- RAdam Optimiser: Adam - A Method for Stochastic Optimization](https://arxiv.org/abs/1412.6980v8)
- On the Convergence of Adam and Beyond (https://openreview.net/forum?id=ryQu7f-RZ)
- On The Variance Of The Adaptive Learning Rate And Beyond (https://arxiv.org/pdf/1908.03265v1.pdf)

```python
#Defining Utility Functions
#Resize function
#Reshape a numpy array, which is input_shape=(height, width), as opposed to input_shape=(width, height) for cv2
def np_resize(img, input_shape):
    """
    Reshape a numpy array, which is input_shape=(height, width),
    as opposed to input_shape=(width, height) for cv2
    """
    height, width = input_shape
    return cv2.resize(img, (width, height))

#Mask Conversion 1 - mask, 0 - background Returns run length as string formated
#Reference: https://www.kaggle.com/paulorzp/rle-functions-run-lenght-encode-decode
def mask2rle(img):
    '''
    img: numpy array, 1 - mask, 0 - background
    Returns run length as string formated
    '''
    pixels= img.T.flatten()
    pixels = np.concatenate([[0], pixels, [0]])
    runs = np.where(pixels[1:] != pixels[:-1])[0] + 1
    runs[1::2] -= runs[::2]
    return ' '.join(str(x) for x in runs)

def rle2mask(rle, input_shape):
    width, height = input_shape[:2]

    mask= np.zeros( width*height ).astype(np.uint8)

    array = np.asarray([int(x) for x in rle.split()])
    starts = array[0::2]
    lengths = array[1::2]

    current_position = 0
    for index, start in enumerate(starts):
        mask[int(start):int(start+lengths[index])] = 1
        current_position += lengths[index]

    return mask.reshape(height, width).T

def build_masks(rles, input_shape, reshape=None):
    depth = len(rles)
    if reshape is None:
        masks = np.zeros((*input_shape, depth))
    else:
        masks = np.zeros((*reshape, depth))
```

```
            else:
                mask = rle2mask(rle, input_shape)
                reshaped_mask = np_resize(mask, reshape)
                masks[:, :, i] = reshaped_mask

    return masks

def build_rles(masks, reshape=None):
    width, height, depth = masks.shape

    rles = []

    for i in range(depth):
        mask = masks[:, :, i]

        if reshape:
            mask = mask.astype(np.float32)
            mask = np_resize(mask, reshape).astype(np.int64)

        rle = mask2rle(mask)
        rles.append(rle)

    return rles

#RAdam
class RAdam(keras.optimizers.Optimizer):
    """RAdam optimizer.
    # Arguments
        lr: float >= 0. Learning rate.
        beta_1: float, 0 < beta < 1. Generally close to 1.
        beta_2: float, 0 < beta < 1. Generally close to 1.
        epsilon: float >= 0. Fuzz factor. If `None`, defaults to `K.epsilon()`.
        decay: float >= 0. Learning rate decay over each update.
        weight_decay: float >= 0. Weight decay for each param.
        amsgrad: boolean. Whether to apply the AMSGrad variant of this
            algorithm from the paper "On the Convergence of Adam and
            Beyond".
        total_steps: int >= 0. Total number of training steps. Enable warmup by setting a positive value.
        warmup_proportion: 0 < warmup_proportion < 1. The proportion of increasing steps.
        min_lr: float >= 0. Minimum learning rate after warmup.
    # References
        - [Adam - A Method for Stochastic Optimization](https://arxiv.org/abs/1412.6980v8)
        - [On the Convergence of Adam and Beyond](https://openreview.net/forum?id=ryQu7f-RZ)
        - [On The Variance Of The Adaptive Learning Rate And Beyond](https://arxiv.org/pdf/1908.03265v1.pdf)
    """
```

```
    def __init__(self, lr=0.001, beta_1=0.9, beta_2=0.999,
                 epsilon=None, decay=0., weight_decay=0., amsgrad=False,
                 total_steps=0, warmup_proportion=0.1, min_lr=0., **kwargs):
        super(RAdam, self).__init__(**kwargs)
        with K.name_scope(self.__class__.__name__):
            self.iterations = K.variable(0, dtype='int64', name='iterations')
            self.lr = K.variable(lr, name='lr')
            self.beta_1 = K.variable(beta_1, name='beta_1')
            self.beta_2 = K.variable(beta_2, name='beta_2')
            self.decay = K.variable(decay, name='decay')
            self.weight_decay = K.variable(weight_decay, name='weight_decay')
            self.total_steps = K.variable(total_steps, name='total_steps')
            self.warmup_proportion = K.variable(warmup_proportion, name='warmup_proportion')
            self.min_lr = K.variable(min_lr, name='min_lr')
        if epsilon is None:
            epsilon = K.epsilon()
        self.epsilon = epsilon
        self.initial_decay = decay
        self.initial_weight_decay = weight_decay
        self.initial_total_steps = total_steps
        self.amsgrad = amsgrad

    def get_updates(self, loss, params):
        grads = self.get_gradients(loss, params)
        self.updates = [K.update_add(self.iterations, 1)]

        lr = self.lr

        if self.initial_decay > 0:
            lr = lr * (1. / (1. + self.decay * K.cast(self.iterations, K.dtype(self.decay))))

        t = K.cast(self.iterations, K.floatx()) + 1

        if self.initial_total_steps > 0:
            warmup_steps = self.total_steps * self.warmup_proportion
            lr = K.switch(
                t <= warmup_steps,
                lr * (t / warmup_steps),
                self.min_lr + (lr - self.min_lr) * (1.0 - K.minimum(t, self.total_steps) / self.total_steps),
            )

        ms = [K.zeros(K.int_shape(p), dtype=K.dtype(p), name='m_' + str(i)) for (i, p) in enumerate(params)]
        vs = [K.zeros(K.int_shape(p), dtype=K.dtype(p), name='v_' + str(i)) for (i, p) in enumerate(params)]
```

```
        if self.amsgrad:
            vhats = [K.zeros(K.int_shape(p), dtype=K.dtype(p), name='vhat_' + str(i)) for (i, p) in enumerate(params)]
        else:
            vhats = [K.zeros(1, name='vhat_' + str(i)) for i in range(len(params))]

        self.weights = [self.iterations] + ms + vs + vhats

        beta_1_t = K.pow(self.beta_1, t)
        beta_2_t = K.pow(self.beta_2, t)

        sma_inf = 2.0 / (1.0 - self.beta_2) - 1.0
        sma_t = sma_inf - 2.0 * t * beta_2_t / (1.0 - beta_2_t)

        for p, g, m, v, vhat in zip(params, grads, ms, vs, vhats):
            m_t = (self.beta_1 * m) + (1. - self.beta_1) * g
            v_t = (self.beta_2 * v) + (1. - self.beta_2) * K.square(g)

            m_corr_t = m_t / (1.0 - beta_1_t)
            if self.amsgrad:
                vhat_t = K.maximum(vhat, v_t)
                v_corr_t = K.sqrt(vhat_t / (1.0 - beta_2_t) + self.epsilon)
                self.updates.append(K.update(vhat, vhat_t))
            else:
                v_corr_t = K.sqrt(v_t / (1.0 - beta_2_t) + self.epsilon)

            r_t = K.sqrt((sma_t - 4.0) / (sma_inf - 4.0) *
                         (sma_t - 2.0) / (sma_inf - 2.0) *
                         sma_inf / sma_t)

            p_t = K.switch(sma_t >= 5, r_t * m_corr_t / v_corr_t, m_corr_t)

            if self.initial_weight_decay > 0:
                p_t += self.weight_decay * p

            p_t = p - lr * p_t

            self.updates.append(K.update(m, m_t))
            self.updates.append(K.update(v, v_t))
            new_p = p_t

            # Apply constraints.
            if getattr(p, 'constraint', None) is not None:
                new_p = p.constraint(new_p)
```

```
            # Apply constraints.
            if getattr(p, 'constraint', None) is not None:
                new_p = p.constraint(new_p)

            self.updates.append(K.update(p, new_p))
        return self.updates

    def get_config(self):
        config = {
            'lr': float(K.get_value(self.lr)),
            'beta_1': float(K.get_value(self.beta_1)),
            'beta_2': float(K.get_value(self.beta_2)),
            'decay': float(K.get_value(self.decay)),
            'weight_decay': float(K.get_value(self.weight_decay)),
            'epsilon': self.epsilon,
            'amsgrad': self.amsgrad,
            'total_steps': float(K.get_value(self.total_steps)),
            'warmup_proportion': float(K.get_value(self.warmup_proportion)),
            'min_lr': float(K.get_value(self.min_lr)),
        }
        base_config = super(RAdam, self).get_config()
        return dict(list(base_config.items()) + list(config.items()))

#Loss Function
def dice_coef(y_true, y_pred, smooth=1):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (2. * intersection + smooth) / (K.sum(y_true_f) + K.sum(y_pred_f) + smooth)

def dice_loss(y_true, y_pred):
    smooth = 1.
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = y_true_f * y_pred_f
    score = (2. * K.sum(intersection) + smooth) / (K.sum(y_true_f) + K.sum(y_pred_f) + smooth)
    return 1. - score

def bce_dice_loss(y_true, y_pred):
    return binary_crossentropy(y_true, y_pred) + dice_loss(y_true, y_pred)
```

Data Generator

Reference: https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly

```python
#Data Generator
class DataGenerator(keras.utils.Sequence):
    'Generates data for Keras'
    def __init__(self, list_IDs, df, target_df=None, mode='fit',
                 base_path='train_images',
                 batch_size=32, dim=(1400, 2100), n_channels=3, reshape=None,
                 augment=False, n_classes=4, random_state=2019, shuffle=True):
        self.dim = dim
        self.batch_size = batch_size
        self.df = df
        self.mode = mode
        self.base_path = base_path
        self.target_df = target_df
        self.list_IDs = list_IDs
        self.reshape = reshape
        self.n_channels = n_channels
        self.augment = augment
        self.n_classes = n_classes
        self.shuffle = shuffle
        self.random_state = random_state

        self.on_epoch_end()
        np.random.seed(self.random_state)

    def __len__(self):
        'Denotes the number of batches per epoch'
        return int(np.floor(len(self.list_IDs) / self.batch_size))

    def __getitem__(self, index):
        'Generate one batch of data'
        # Generate indexes of the batch
        indexes = self.indexes[index*self.batch_size:(index+1)*self.batch_size]

        # Find list of IDs
        list_IDs_batch = [self.list_IDs[k] for k in indexes]

        X = self.__generate_X(list_IDs_batch)

        if self.mode == 'fit':
            y = self.__generate_y(list_IDs_batch)

            if self.augment:
                X, y = self.__augment_batch(X, y)

            return X, y
```

```python
        elif self.mode == 'predict':
            return X

        else:
            raise AttributeError('The mode parameter should be set to "fit" or "predict".')

    def on_epoch_end(self):
        'Updates indexes after each epoch'
        self.indexes = np.arange(len(self.list_IDs))
        if self.shuffle == True:
            np.random.seed(self.random_state)
            np.random.shuffle(self.indexes)

    def __generate_X(self, list_IDs_batch):
        'Generates data containing batch_size samples'
        # Initialization
        if self.reshape is None:
            X = np.empty((self.batch_size, *self.dim, self.n_channels))
        else:
            X = np.empty((self.batch_size, *self.reshape, self.n_channels))

        # Generate data
        for i, ID in enumerate(list_IDs_batch):
            im_name = self.df['ImageId'].iloc[ID]
            img_path = f"{self.base_path}/{im_name}"
            img = self.__load_rgb(img_path)

            if self.reshape is not None:
                img = np_resize(img, self.reshape)

            # Store samples
            X[i,] = img

        return X

    def __generate_y(self, list_IDs_batch):
        if self.reshape is None:
            y = np.empty((self.batch_size, *self.dim, self.n_classes), dtype=int)
        else:
            y = np.empty((self.batch_size, *self.reshape, self.n_classes), dtype=int)

        for i, ID in enumerate(list_IDs_batch):
            im_name = self.df['ImageId'].iloc[ID]
            image_df = self.target_df[self.target_df['ImageId'] == im_name]
```

```
            rles = image_df['EncodedPixels'].values

            if self.reshape is not None:
                masks = build_masks(rles, input_shape=self.dim, reshape=self.reshape)
            else:
                masks = build_masks(rles, input_shape=self.dim)

            y[i, ] = masks

        return y

    def __load_grayscale(self, img_path):
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        img = img.astype(np.float32) / 255.
        img = np.expand_dims(img, axis=-1)

        return img

    def __load_rgb(self, img_path):
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = img.astype(np.float32) / 255.

        return img

    def __random_transform(self, img, masks):
        composition = albu.Compose([
            albu.HorizontalFlip(),
            albu.VerticalFlip(),
            albu.ShiftScaleRotate(rotate_limit=45, shift_limit=0.15, scale_limit=0.15)
        ])

        composed = composition(image=img, mask=masks)
        aug_img = composed['image']
        aug_masks = composed['mask']

        return aug_img, aug_masks

    def __augment_batch(self, img_batch, masks_batch):
        for i in range(img_batch.shape[0]):
            img_batch[i, ], masks_batch[i, ] = self.__random_transform(
                img_batch[i, ], masks_batch[i, ])

        return img_batch, masks_batch
```

Model creation – UNet-ResNet

```
#Model Architecture
def vanilla_unet(input_shape):

    inputs = Input(input_shape)

    c1 = Conv2D(8, (3, 3), activation='elu', padding='same') (inputs)
    c1 = Conv2D(8, (3, 3), activation='elu', padding='same') (c1)
    p1 = MaxPooling2D((2, 2), padding='same') (c1)

    c2 = Conv2D(16, (3, 3), activation='elu', padding='same') (p1)
    c2 = Conv2D(16, (3, 3), activation='elu', padding='same') (c2)
    p2 = MaxPooling2D((2, 2), padding='same') (c2)

    c3 = Conv2D(32, (3, 3), activation='elu', padding='same') (p2)
    c3 = Conv2D(32, (3, 3), activation='elu', padding='same') (c3)
    p3 = MaxPooling2D((2, 2), padding='same') (c3)

    c4 = Conv2D(64, (3, 3), activation='elu', padding='same') (p3)
    c4 = Conv2D(64, (3, 3), activation='elu', padding='same') (c4)
    p4 = MaxPooling2D((2, 2), padding='same') (c4)

    c5 = Conv2D(64, (3, 3), activation='elu', padding='same') (p4)
    c5 = Conv2D(64, (3, 3), activation='elu', padding='same') (c5)
    p5 = MaxPooling2D((2, 2), padding='same') (c5)

    c55 = Conv2D(128, (3, 3), activation='elu', padding='same') (p5)
    c55 = Conv2D(128, (3, 3), activation='elu', padding='same') (c55)

    u6 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same') (c55)
    u6 = concatenate([u6, c5])
    c6 = Conv2D(64, (3, 3), activation='elu', padding='same') (u6)
    c6 = Conv2D(64, (3, 3), activation='elu', padding='same') (c6)

    u71 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same') (c6)
    u71 = concatenate([u71, c4])
    c71 = Conv2D(32, (3, 3), activation='elu', padding='same') (u71)
    c61 = Conv2D(32, (3, 3), activation='elu', padding='same') (c71)

    u7 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same') (c61)
    u7 = concatenate([u7, c3])
    c7 = Conv2D(32, (3, 3), activation='elu', padding='same') (u7)
    c7 = Conv2D(32, (3, 3), activation='elu', padding='same') (c7)
```

```
u8 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same') (c7)
u8 = concatenate([u8, c2])
c8 = Conv2D(16, (3, 3), activation='elu', padding='same') (u8)
c8 = Conv2D(16, (3, 3), activation='elu', padding='same') (c8)

u9 = Conv2DTranspose(8, (2, 2), strides=(2, 2), padding='same') (c8)
u9 = concatenate([u9, c1], axis=3)
c9 = Conv2D(8, (3, 3), activation='elu', padding='same') (u9)
c9 = Conv2D(8, (3, 3), activation='elu', padding='same') (c9)

outputs = Conv2D(4, (1, 1), activation='sigmoid') (c9)

model = Model(inputs=[inputs], outputs=[outputs])

return model
```

```
model = sm.Unet(
    'resnet34',
    classes=4,
    input_shape=(320, 480, 3),
    activation='sigmoid'
)
model.compile(optimizer=Nadam(lr=0.0002), loss=bce_dice_loss, metrics=[dice_coef])
model.summary()
```

Model creation – UNet-Xception

```
def xception_unet(input_shape):

    backbone = Xception(input_shape=input_shape,weights='imagenet',include_top=False)
    input = backbone.input
    start_neurons = 16

    c1 = Conv2D(8, (3, 3), activation='elu', padding='same') (inputs)
    c1 = Conv2D(8, (3, 3), activation='elu', padding='same') (c1)
    p1 = MaxPooling2D((2, 2), padding='same') (c1)

    c2 = Conv2D(16, (3, 3), activation='elu', padding='same') (p1)
    c2 = Conv2D(16, (3, 3), activation='elu', padding='same') (c2)
    p2 = MaxPooling2D((2, 2), padding='same') (c2)

    c3 = Conv2D(32, (3, 3), activation='elu', padding='same') (p2)
    c3 = Conv2D(32, (3, 3), activation='elu', padding='same') (c3)
    p3 = MaxPooling2D((2, 2), padding='same') (c3)

    c4 = Conv2D(64, (3, 3), activation='elu', padding='same') (p3)
    c4 = Conv2D(64, (3, 3), activation='elu', padding='same') (c4)
    p4 = MaxPooling2D((2, 2), padding='same') (c4)

    c5 = Conv2D(64, (3, 3), activation='elu', padding='same') (p4)
    c5 = Conv2D(64, (3, 3), activation='elu', padding='same') (c5)
    p5 = MaxPooling2D((2, 2), padding='same') (c5)

    c55 = Conv2D(128, (3, 3), activation='elu', padding='same') (p5)
    c55 = Conv2D(128, (3, 3), activation='elu', padding='same') (c55)
```

```
u6 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same') (c55)
u6 = concatenate([u6, c5])
c6 = Conv2D(64, (3, 3), activation='elu', padding='same') (u6)
c6 = Conv2D(64, (3, 3), activation='elu', padding='same') (c6)

u71 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same') (c6)
u71 = concatenate([u71, c4])
c71 = Conv2D(32, (3, 3), activation='elu', padding='same') (u71)
c61 = Conv2D(32, (3, 3), activation='elu', padding='same') (c71)

u7 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same') (c61)
u7 = concatenate([u7, c3])
c7 = Conv2D(32, (3, 3), activation='elu', padding='same') (u7)
c7 = Conv2D(32, (3, 3), activation='elu', padding='same') (c7)

u8 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same') (c7)
u8 = concatenate([u8, c2])
c8 = Conv2D(16, (3, 3), activation='elu', padding='same') (u8)
c8 = Conv2D(16, (3, 3), activation='elu', padding='same') (c8)

u9 = Conv2DTranspose(8, (2, 2), strides=(2, 2), padding='same') (c8)
u9 = concatenate([u9, c1], axis=3)
c9 = Conv2D(8, (3, 3), activation='elu', padding='same') (u9)
c9 = Conv2D(8, (3, 3), activation='elu', padding='same') (c9)

outputs = Conv2D(4, (1, 1), activation='sigmoid') (c9)

model = Model(inputs=[inputs], outputs=[outputs])

return model
```

```
model = sm.Unet(
    'inceptionv3',
    classes=4,
    input_shape=(320, 480, 3),
    activation='sigmoid'
)
model.compile(optimizer=Nadam(lr=0.0002), loss=bce_dice_loss, metrics=[dice_coef])
model.summary()
```

Training

```
#Training
BATCH_SIZE = 2

train_idx, val_idx = train_test_split(
    mask_count_df.index, random_state=2019, test_size=0.2
)

train_generator = DataGenerator(
    train_idx,
    df=mask_count_df,
    target_df=train_df,
    batch_size=BATCH_SIZE,
    reshape=(320, 480),
    augment=True,
    n_channels=3,
    n_classes=4
)
```

```
val_generator = DataGenerator(
    val_idx,
    df=mask_count_df,
    target_df=train_df,
    batch_size=BATCH_SIZE,
    reshape=(320, 480),
    augment=False,
    n_channels=3,
    n_classes=4
)

#Creating checkpoint for saving weights
checkpoint = ModelCheckpoint('model_1.h5', save_best_only=True)

history = model.fit_generator(
    train_generator,
    validation_data=val_generator,
    callbacks=[checkpoint],
    epochs=10
)
```

Model evaluation

```
#Evaluation
with open('history.json', 'w') as f:
    json.dump(history.history, f)

history_df = pd.DataFrame(history.history)
history_df[['loss', 'val_loss']].plot()
history_df[['dice_coef', 'val_dice_coef']].plot()

model.load_weights('model.h5')
test_df = []

for i in range(0, test_imgs.shape[0], 500):
    batch_idx = list(
        range(i, min(test_imgs.shape[0], i + 500))
    )

    test_generator = DataGenerator(
        batch_idx,
        df=test_imgs,
        shuffle=False,
        mode='predict',
        dim=(350, 525),
        reshape=(320, 480),
        n_channels=3,
        base_path='test_images',
        target_df=sub_df,
        batch_size=1,
        n_classes=4
    )

    batch_pred_masks = model.predict_generator(
        test_generator,
        workers=1,
        verbose=1
    )

    for j, b in enumerate(batch_idx):
        filename = test_imgs['ImageId'].iloc[b]
        image_df = sub_df[sub_df['ImageId'] == filename].copy()

        pred_masks = batch_pred_masks[j, ].round().astype(int)
        pred_rles = build_rles(pred_masks, reshape=(350, 525))

        image_df['EncodedPixels'] = pred_rles
        test_df.append(image_df)
```

```
test_df = test_df[~test_df['EncodedPixels'].isnull()]
test_df['Image'] = test_df['Image_Label'].map(lambda x: x.split('_')[0])
test_df['Class'] = test_df['Image_Label'].map(lambda x: x.split('_')[1])
classes = test_df['Class'].unique()
test_df = test_df.groupby('Image')['Class'].agg(set).reset_index()
for class_name in classes:
    test_df[class_name] = test_df['Class'].map(lambda x: 1 if class_name in x else 0)
```

```
test_df_1 = test_df_1[~test_df_1['EncodedPixels'].isnull()]
test_df_1['Image'] = test_df_1['Image_Label'].map(lambda x: x.split('_')[0])
test_df_1['Class'] = test_df_1['Image_Label'].map(lambda x: x.split('_')[1])
classes = test_df_1['Class'].unique()
test_df_1 = test_df_1.groupby('Image')['Class'].agg(set).reset_index()
for class_name in classes:
    test_df_1[class_name] = test_df_1['Class'].map(lambda x: 1 if class_name in x else 0)
```

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report


print("Classification accuracy for class Fish: {:.4%}".format(accuracy_score(test_df_1['Fish'], test_df['Fish'])))
print('Classification Report for class Fish : ')
print(classification_report(test_df_1['Fish'], test_df['Fish']))

print("Classification accuracy for class Flower: {:.4%}".format(accuracy_score(test_df_1['Flower'], test_df['Flower'])))
print('Classification Report for class Flower : ')
print(classification_report(test_df_1['Flower'], test_df['Flower']))

print("Classification accuracy for class Gravel: {:.4%}".format(accuracy_score(test_df_1['Gravel'], test_df['Gravel'])))
print('Classification Report for class Gravel : ')
print(classification_report(test_df_1['Gravel'], test_df['Gravel']))

print("Classification accuracy for class Sugar: {:.4%}".format(accuracy_score(test_df_1['Sugar'], test_df['Sugar'])))
print('Classification Report for class Sugar : ')
print(classification_report(test_df_1['Sugar'], test_df['Sugar']))
```

Part 2 – Python code for classification

Importing model and classification

```
import keras.losses
keras.losses.custom_loss = bce_dice_loss
model.load_weights(os.path.join('model.h5'))
```

```
test_df_1 = []

for i in range(0, test_imgs.shape[0], 500):
    batch_idx = list(
        range(i, min(test_imgs.shape[0], i + 500))
    )

    test_generator = DataGenerator(
        batch_idx,
        df=test_imgs,
        shuffle=False,
        mode='predict',
        dim=(1600,1600),
        reshape=(320, 480),
        n_channels=3,
        base_path='irma',
        target_df=test_df,
        batch_size=1,
        n_classes=4
    )

    batch_pred_masks = model.predict_generator(
        test_generator,
        workers=1,
        verbose=1
    )

    for j, b in enumerate(batch_idx):
        filename = test_imgs['ImageId'].iloc[b]
        image_df = test_df[test_df['ImageId'] == filename].copy()

        pred_masks = batch_pred_masks[j, ].round().astype(int)
        pred_rles = build_rles(pred_masks, reshape=(350, 525))

        image_df['EncodedPixels'] = pred_rles
        test_df_1.append(image_df)
```

Part 3 – Python code for time-series analysis

Getting autocorrelation and differencing

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
# Original Series
fig, axes = plt.subplots(3, 2, sharex=True)
axes[0, 0].plot(time_series_df.Total_Cloud_Occurence); axes[0, 0].set_title('Original Series')
plot_acf(time_series_df.Total_Cloud_Occurence, ax=axes[0, 1])

# 1st Differencing
axes[1, 0].plot(time_series_df.Total_Cloud_Occurence.diff()); axes[1, 0].set_title('1st Order Differencing')
plot_acf(time_series_df.Total_Cloud_Occurence.diff().dropna(), ax=axes[1, 1])

# 2nd Differencing
axes[2, 0].plot(time_series_df.Total_Cloud_Occurence.diff().diff()); axes[2, 0].set_title('2nd Order Differencing')
plot_acf(time_series_df.Total_Cloud_Occurence.diff().diff().dropna(), ax=axes[2, 1])
```

Building the ARIMA model

```
from statsmodels.tsa.arima_model import ARIMA

# 1,1,2 ARIMA Model
model = ARIMA(time_series_df.Total_Cloud_Occurence, order=(4,1,4))
model_fit = model.fit(disp=0)
print(model_fit.summary())
```

```
# 4,1,4 ARIMA Model
model = ARIMA(time_series_df.Total_Cloud_Occurence, order=(4,1,4))
model_fit = model.fit(disp=0)
print(model_fit.summary())
```

Forecasting

```
# Build Model
#model = ARIMA(train, order=(3,2,1))
model = ARIMA(train, order=(4, 1, 4))
fitted = model.fit(disp=-1)

# Forecast
fc, se, conf = fitted.forecast(20, alpha=0.05)  # 95% conf

# Make as pandas series
fc_series = pd.Series(fc, index=test.index)
lower_series = pd.Series(conf[:, 0], index=test.index)
upper_series = pd.Series(conf[:, 1], index=test.index)

# Plot
plt.figure(figsize=(12,5), dpi=100)
plt.plot(train, label='training')
plt.plot(test, label='actual')
plt.plot(fc_series, label='forecast')
plt.fill_between(lower_series.index, lower_series, upper_series,
                 color='k', alpha=.15)
plt.title('Forecast vs Actuals')
plt.legend(loc='upper left', fontsize=8)
plt.show()
```

```
# Forecast
n_periods = 120
fc, se, confint = fitted.forecast(120, alpha=0.05)  # 95% conf
index_of_fc = np.arange(len(time_series_df.Total_Cloud_Occurence), len(time_series_df.Total_Cloud_Occurence)+n_periods)

# make series for plotting purpose
fc_series = pd.Series(fc, index=index_of_fc)
lower_series = pd.Series(confint[:, 0], index=index_of_fc)
upper_series = pd.Series(confint[:, 1], index=index_of_fc)

# Plot
plt.plot(time_series_df.Total_Cloud_Occurence)
plt.plot(fc_series, color='darkgreen')
plt.fill_between(lower_series.index,
                 lower_series,
                 upper_series,
                 color='k', alpha=.15)

plt.title("Final Forecast of Total Cloud Occurence Usage")

plt.xlabel('Months')
plt.ylabel('Total Cloud Occurence')
plt.show()
```

Appendix B

Turnitin similarity report

Document Viewer

Turnitin Originality Report

Processed on: 11-Aug-2020 00:26 IST
ID: 1366214927
Word Count: 18880
Submitted: 3

Thesis_Draft By Abhishek Mandal

Similarity Index	Similarity by Source	
22%	Internet Sources:	11%
	Publications:	8%
	Student Papers:	17%