Project Report COMP 551: Planet, Understanding the Amazon from Space

Faizan Safdar Ali, Palwisha Akhtar, Waris Gill

[a]*{fali18, pakhtar19, wgill18}@ku.edu.tr*

Abstract: Kaggle has provided the satellite data to track the human footprints in the Amazon rain forest. Each picture in the dataset has multiple labels demonstrating the type of picture. The primary goal is to predict the labels on the new image that is presented to the learned model. This model can then be used to predict the images with different timestamps to observe the changes in the areas. These changes can help predict the human footprints (which is out of the scope of this project). Different models and optimizations were used to get the best F2 score. Also, the comparisons of these techniques are also presented.

1. Introduction

The Amazon rain-forest is the biggest forest of the world containing billions of trees and thousands of different species covering 5,500,000 square kilometers of rainforest. This big land is facing issues like habitat loss, reduced biodiversity and climate change due to deforestation. However, this problem can be solved if governments can locate affected areas by just using satellite images. For the purpose, Kaggle's Planet: Understanding the Amazon from Space Challenge has provided the dataset that contains the pictures of earth taken from the satellite. Idea is to use satellite data to track changes in the environment and understand the human footprint in the Amazon rainforest. A deep neural network for Multilabel Classification will label satellite image chips with atmospheric conditions and various classes of land cover/land use. The classifier will be evaluated based on the mean F2 score which measures accuracy using the precision p and recall r. Recall is the ratio of true positives to all actual positives and precision is the ratio of true positives to all predicted positives. The F2 score is shown in Figure 1.

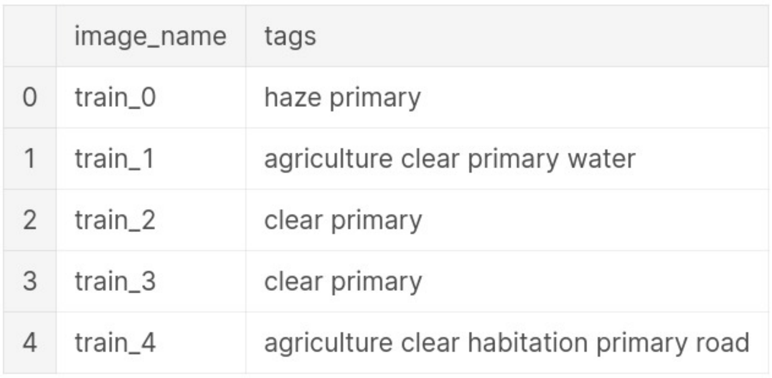
$$(1+\beta^2)\frac{pr}{\beta^2 p + r} \quad \text{where } p = \frac{tp}{tp+fp}, \quad r = \frac{tp}{tp+fn}, \ \beta = 2.$$

Figure 1: F2 Score formula.

Resulting neural network will help the global community better understand where, how, and why deforestation happens all over the world - and ultimately how to respond. Another aspect of the project was to try different methods for optimizations and see how they work in the given scenario. This is very important as now a days, knowing how the AI/Machine learning works is a scarce knowledge. So through experimentation, we tried to learn more about learning models. For that, we prepossessed the image data through normalization and augmentations. Then we choose the right optimizer and loss function. On top of that, we implemented different models. We also fine-tuned the probability threshold for predictions to better fit the evaluation matrix.

Our Aim:

1. Develop a deep neural network module for Multilabel Image Classification.
2. Achieve the highest f2 mean score attained in Kaggle competition.
3. Apply optimization techniques learned during the course.
4. Try different models and compare their performances.



	image_name	tags
0	train_0	haze primary
1	train_1	agriculture clear primary water
2	train_2	clear primary
3	train_3	clear primary
4	train_4	agriculture clear habitation primary road

Figure 2: Tags of some images in csv file.

2. Related work

The sophisticated methods have been implemented that classify the image data and can also perform the image captioning. For example, VGGNet model [1], ResNet model [2], and DenseNet model [3]. The VGGNet model consists of several combinations of filters and pooling layers. ResNet is a very deep model that uses the residual connections. As the vanishing gradient problem does not exist in this model, that allows fine tuning and learning of the model. The DenseNet model is a relatively new model. Its architecture is made up of blocks that are connected to every other layers in the model. It alleviates the problem of vanishing gradients and reuses features in the model. GoogLeNet [4] is also a model that can be used for the image captioning. This model contains inception modules that are connected to make a deeper architecture.

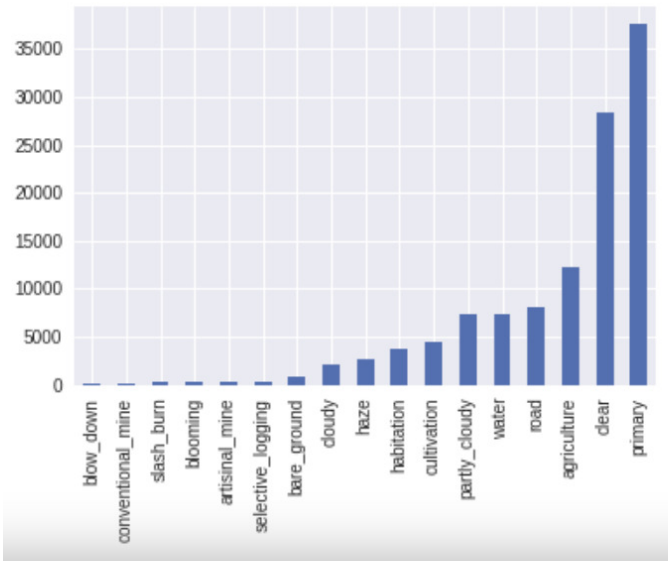


Figure 3: Frequencies of the images tags.

3. Dataset

For this project, the dataset was taken from the kaggle website. According to description from the Kaggle website, the images come from Planet's Flock 2 satellites in both sun-synchronous orbit (SSO) and International Space Station (ISS) orbit. The images were taken from January 1st, 2016 to February 1st, 2017, and all of them come from Amazon basin.

Each image is 256 x 256 in size and each containing RGB channels. The images are provided in the JPEG format and can be downloaded from the Kaggle website. GeoTiff format is also available that contain infrared channel in addition to the RGB channels. The tags of the images are given in the csv file.

3.1. Data Visualization

Dataset consisting of TIFF, JPEG and accompanying CSV was downloaded from the Kaggle website [5]. CSV file consists of filenames and their true labels (Figure 2). We visualized the data for better understanding and modeling of the deep learning network as shown in Figure 4. There are a total of 40478 images and 17 labels. Figure 3 shows the frequencies of the labels. Primary and clear are the most common amongst land and weather labels respectively.

As this is a Multilabel Classification problem i.e each image can be tagged with multiple tags, we look at the co-occurrence matrices of the labels (Figure 5a - 5d). Following deductions can be made from the co-occurrence matrices:

1. Land labels may overlap.
2. Each image should have exactly one weather label.
3. Rarer labels have very little overlap.

3.2. Data Preprocessing

After understanding the data, we preprocess to prepare it for our models in the following steps:

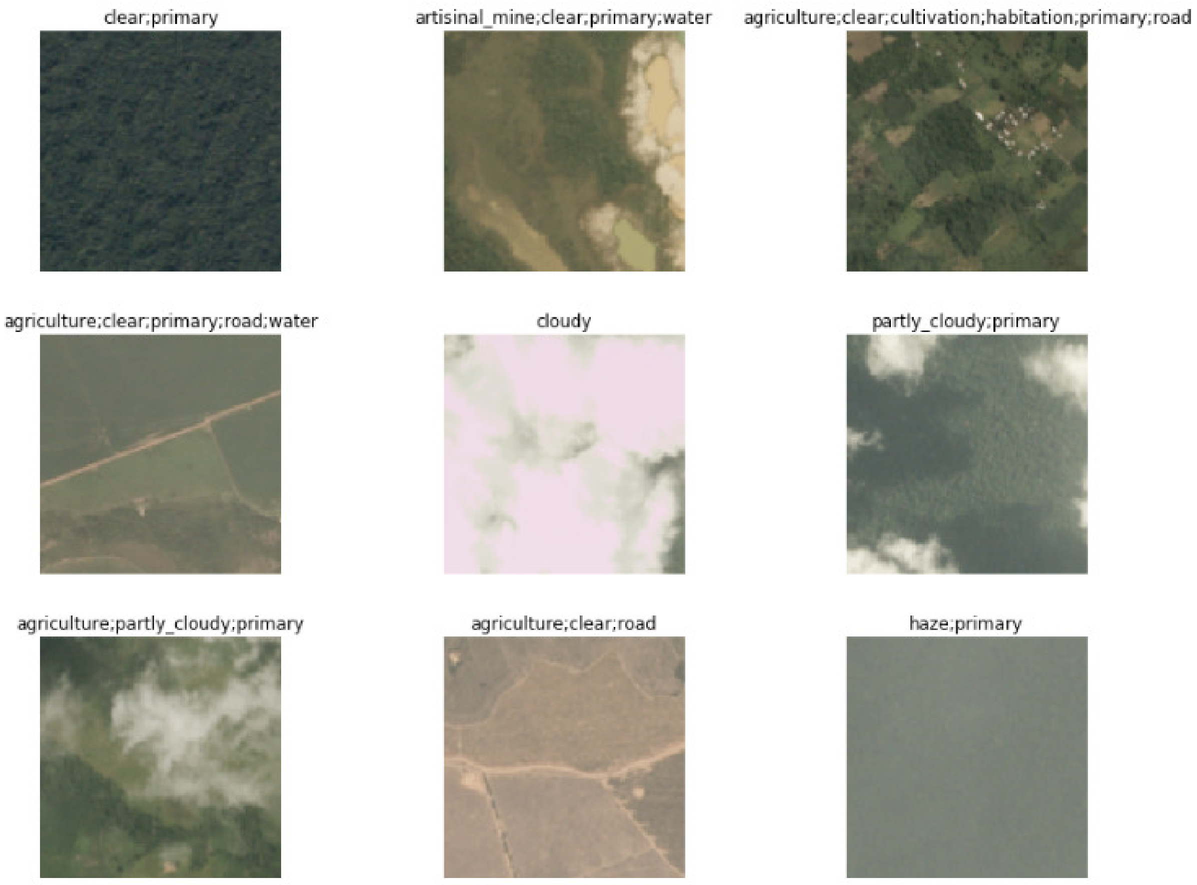

clear;primary
artisinal_mine;clear;primary;water
agriculture;clear;cultivation;habitation;primary;road
agriculture;clear;primary;road;water
cloudy
partly_cloudy;primary
agriculture;partly_cloudy;primary
agriculture;clear;road
haze;primary

Figure 4: Tags of some images in csv file.

1. **Data Splitting:** Dataset consists of 40478 images in total. We split data into Train, Validation and Test sets by 90:5:5 ratio which is 36438, 2024 and 2023, respectively. This was done according to the best practices defined in the slides of the lectures. In appendix, listing 3 shows the code for data splitting.
2. **Centering and Scaling:** For normalization of the data, we wrote an algorithm for calculation of standard deviation and mean of the training data.

3. **Transformations:** We have applied different transformations which are as follows:
   (a) Horizontal flip: Flips the image on the horizontal axis.
   (b) Vertical flip: Flips the image on the vertical axis.
   (c) Resize: Reduce/increase the dimentionality of the image.
   (d) Rotation: Rotates the image with the given degree.

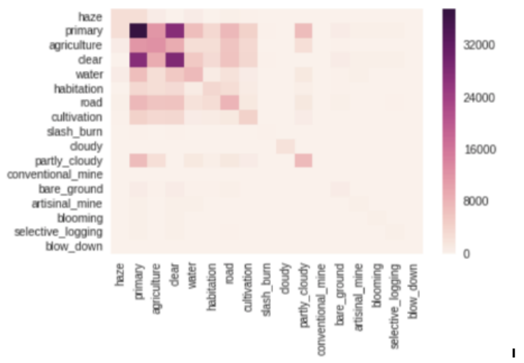


Fig. 5a: Co-occurrence matrix for all labels

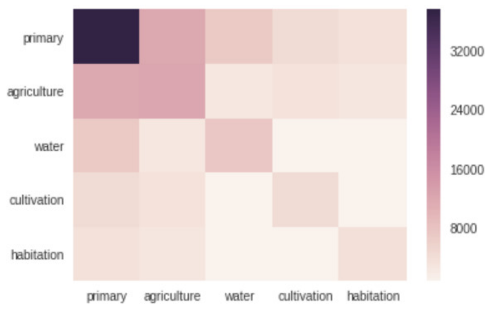


Fig. 5b: Co-occurrence matrix land labels

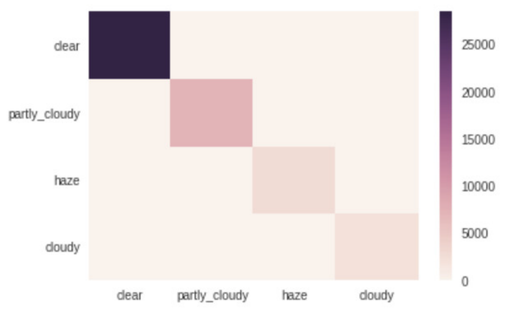


Fig. 5c: Co-occurrence matrix for atmospheric conditions

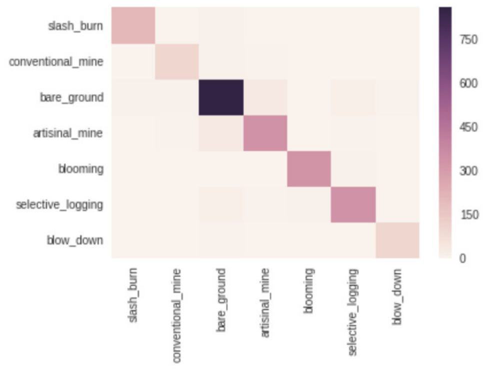


Fig. 5d: Co-occurrence matrix for rare labels

Figure 5: Data visualization using the hot graphs.

4. Training Preliminaries

Before the training starts, there are some preliminaries to be considered that are explained in this section.

4.1. Loss Function & Optimizer

For the training of the models, we first have to decide on the Loss Function and the Optimizer. The loss function determines how the model is performing and the optimizer is used to update the weights on the layers during the training iterations.

Looking at the data, we realized that an image can have multiple labels associated to that. For that reason, we used the MultiLabelSoftMarginLoss function available in the PyTorch. For the optimizer, we tried different optimizers and comparing the results, we used our own optimized implementation. That implementation is SGD with Nesterov Momentum with periodic weight decay. The comparison of different optimizers in figure 6c. The interesting question over can be asked over here are:

1. Does the choice of optimizer depends on the data?
2. Can we change the settings of the optimizers to make them perform better?

4.2. Learning Rate

The learning rate determines how much the weights will be updated in each iteration. This is the important hyperparameter to be determined as it affects the learning procedure. To determine the best learning rate, firstly, we started with the small learning rate and increase it in each iteration (i.e., for each mini batch). After this step, plot the learning rate vs loss as shown in the figure 6a. Lasltly, we picked the learning rate before it diverages (i.e., ater the dip in the graph the loss explode). We took the learning rate 1/10th before the dip.

We did two different tests, one to determine the learning rates for each of the optimizers. The second test was to determine the learning rate for each model we implemented. From the results, we can see that the learning rates are different for the different implementations. The graphs for the learning rate for each model and optimizer are shown in the figure 6a and 6b, respectively.

4.3. Threshold Prediction

As we are predicting multiple labels, the results we get are the probabilities of 17 classes. Now we have to find the threshold above which the label will be 1 for the picture or zero. In short, we want to convert the probability values into binary values.

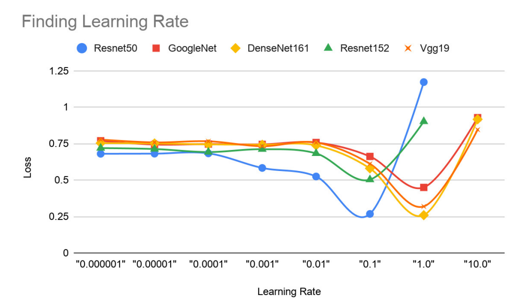


Fig. 6a: Learning Rate varying models.

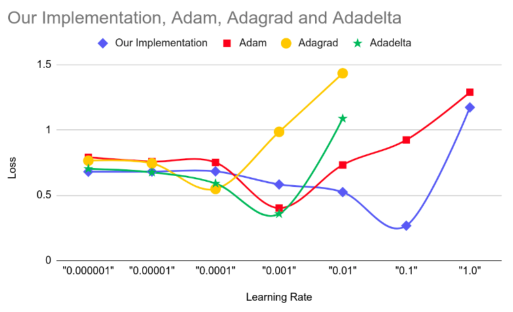


Fig. 6b: Learning Rate varying the optimizers.

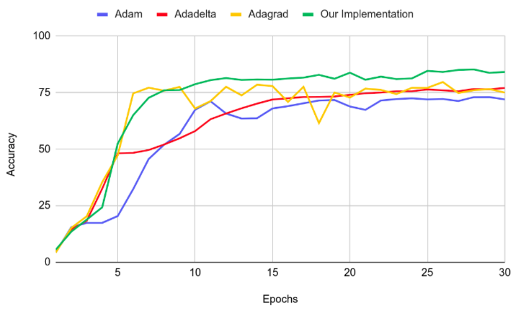


Fig. 6c: Accuracy with different optimizers.

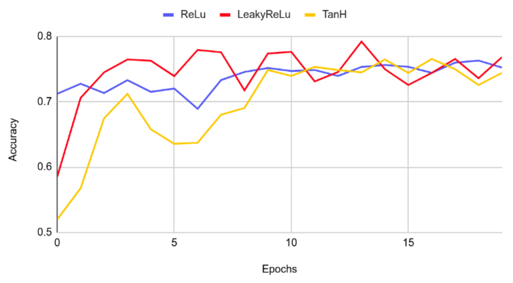


Fig. 6d: F2 Score with varying activation functions.

Figure 6: Training preliminaries.

For finding this threshold, we used the learning regression technique, where we used all the possible values and then try to maximize the F2 Score. There is a very good tool in PyTorch "basin hopping" that we used for this purpose.

4.4. Activation Function

Activation is a very important layer in the deep models. This introduces the regularization and also helps in identifying the important neurons. Initially, all the models available in PyTorch use the ReLu function. But for our testing, we also used LeakyReLu and Tanh function. From the results, we do not see much difference between the implementation. One thing noticeable is that ReLu is stable than the other functions on this dataset. Figure 6d shows the f2 score with different activation functions.

4.5. Learning Rate Scheduler

Learning rate schedule is to change the value of the learning rate for the optimizer and different methods are present for this purpose. Some methods reduce the learning rate after some iterations while other increase that periodically. We used the cyclic learning rate scheduler in which, we set the minimum and the maximum learning rate value and the learning rate oscillated between the values. We put maximum value to 1e-1 and minimum to 1e-2 as most of our models take this learning rate values (see table 1).

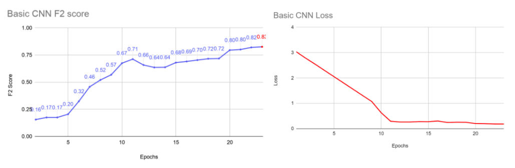


Fig. 7a: Basicnet results.

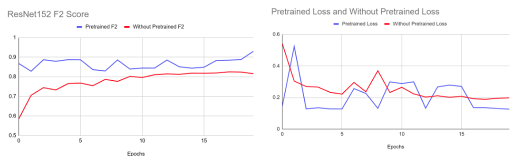


Fig. 7b: ResNet152 results.

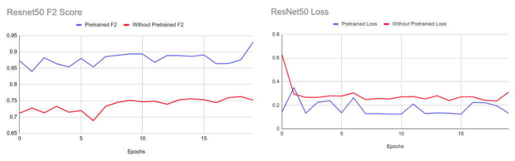


Fig. 7c: ResNet50 results.

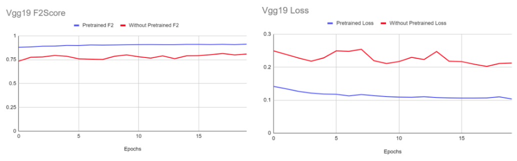


Fig. 7d: Vgg19 results.

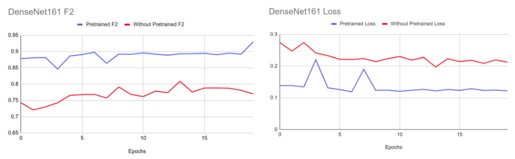


Fig. 7e: DenseNet161 results.

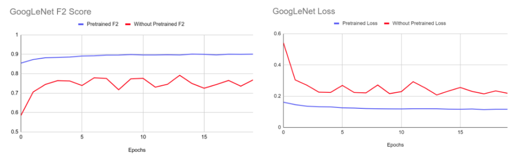


Fig. 7f: GoogLeNet results.

Figure 7: Training results.

5. Implemented Models

In this section, we will present the results of the models that we train. We have plotted the graphs based on both the loss and the F2 score after each epochs. Each epoch contains 1800 iterations and each iteration process 16 images. Following models are implemented:

1. Basicnet (Results in Figure 7a)
2. ResNet152 (Results in Figure 7b)
3. ResNet50 (Results in Figure 7c)
4. VGG19 (Results in Figure 7d)
5. DenseNet161 (Results in Figure 7e)
6. GoogLeNet (Results in Figure 7f)
7. ResNet18
8. ResNet34
9. ResNet101

Due to limited space & time, the we did extensive study on the first 5 models. However, we have presented the F2 Score for all of the models in the final table (Table 1). In figure 8a, the losses of different models are shown. This graph shows that within few iterations the loss of each model quickly goes down and each model converges within a few iterations.

5.1. Effect of Transfer Learning on Less Data

The effect of transfer learning cannot be seen in huge datasets. So, to see how much data is required for such task, we reduce the data size to 5,000 images and afterward we increase it to 20,000 images with an increment of 5,000. We use ResNet50 for this task. From the figure 8b we can see that we can achieve 92.49 F2 score (that we were achieving when we trained with 36,438 images). So we can conclude that, using transfer learning, we can get the roughly same performance with less data.

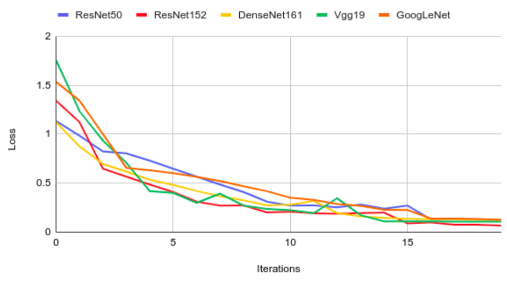


Fig. 8a: Per iteration loss for the models

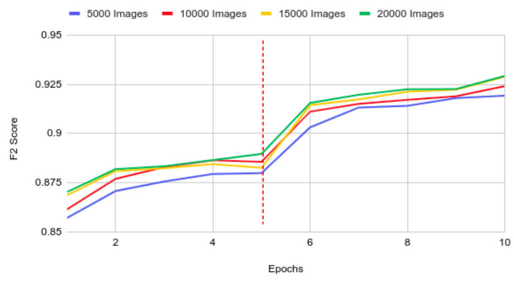


Fig. 8b: Transfer learning results. Before dotted red line, the layers are freezed and after that, they are unfreezed.

5.2. Model Ensemble

The main problem in ensembling is that we cannot train multiple CNN architectures at the same because if the network is deeper it will result in memory error [6]. So, we use the average ensembling [7] to address this issue.

We ensemble ResNet50, ResNet152 and DenseNet161. First, we have trained the above architectures one by one and save them. Then we load each model and evaluated the test data and in the end, we took the average scores and computed the F2 score as described in section one. The ensemble code is given in the appendix. As we are using the average ensembling, our results were not as expected and improvements were not observed.

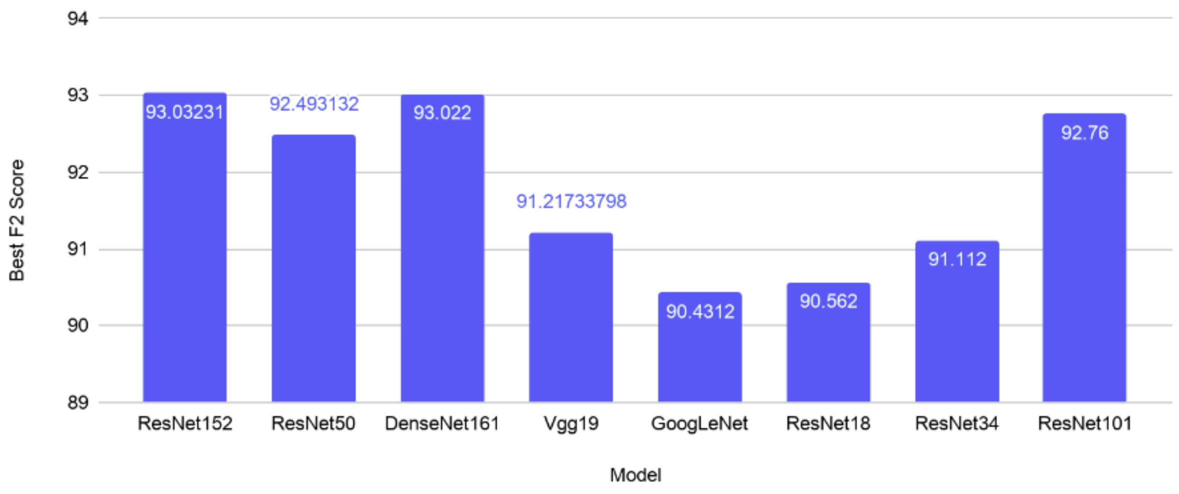


Figure 9: F2 Score comparison.

6. Implementation Explanation

This section explains the code base of the project.

1. **Data loading:**We wrote a class for loading and storing the data in the meaningful way. Then we used that class for the data spliting and training. This is present in dataloader.py file. Some code extracted from [8].
2. **Threshold finder:** A linear regression model using basin hoping is written for determining the threshold. The code is present in threshold.py file. Some code extracted from [9].
3. **Models:** A one call function is implemented that requires the name of the model required. The code is present in models.py file.
4. **Helper Functions:** The function for the data visualization and the mean/std calculation are present in helper.py.
5. **Training, validation and prediction:** A generic code is implemented. The functions require the model as input and perform the respective functionality. The code is in trainAndtest.py

6. **Main and Jupyter Notebook:** A jupyter notebook is implemented for the usage of the code. In case notebook is not accessible, a python code is written to run the training from the command prompt. The codes are in main-notebook.ipynb and test.py respectively.

Table 1: Performance comparison

Model	**Learning Rate**	**Changed Layers**	**Loss Function**	**Optimizer**	**LR Scheduler**	**F2 Score**
ResNet152	1e-2	Fully Connected	MultiLabel SoftMargin-Loss	SGD + Nesterov	Cyclic	93.03
ResNet50	1e-2	Fully Connected	MultiLabel SoftMargin-Loss	SGD + Nesterov	Cyclic	92.49
DenseNet161	1e-1	Fully Connected	MultiLabel SoftMargin-Loss	SGD + Nesterov	Cyclic	93.02
VGG19	1e-1	Fully Connected	MultiLabel SoftMargin-Loss	SGD + Nesterov	Cyclic	91.21
GoogLeNet	1e-1	Fully Connected	MultiLabel SoftMargin-Loss	SGD + Nesterov	Cyclic	90.43
ResNet18	1e-2	Fully Connected	MultiLabel SoftMargin-Loss	SGD + Nesterov	Cyclic	90.562
ResNet34	1e-2	Fully Connected	MultiLabel SoftMargin-Loss	SGD + Nesterov	Cyclic	91.112
ResNet101	1e-2	Fully Connected	MultiLabel SoftMargin-Loss	SGD + Nesterov	Cyclic	92.76

7. Conclusion

The best performance of 93.03 is achieved from the training of the ResNet150. Different variations were tried during the project and the results were presented in this report. The best performing model uses the SGD with Nesterov momentum with weight decay of 1e-4 and the learning rate of 1e-2.

For the future work, take a pretrained model on Amazone dataset (transfer learning), and then train this model on satellite images of glaciers to detect glacier calving margins or similar images [10].

8. Acknowledgements

We want to thank Professor Yücel Yemez and all the teaching assistants for the course who helped us with the understanding of the concepts of deep learning that were required for the successful implementation of the project.

References

[1] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556.

[2] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[3] G. Huang, Z. Liu, L. Van Der Maaten, K. Q. Weinberger, Densely connected convolutional networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 4700–4708.

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 1–9.

[5] Planet: Understanding the amazon from space.
URL `https://www.kaggle.com/c/planet-understanding-the-amazon-from-space/overview`

[6] J. Kim, Memory problem on ensembling the several pretrained models (May 2018).
URL `https://discuss.pytorch.org/t/memory-problem-on-ensembling-the-several-pretrained-models/17896`

[7] A. Singh, A comprehensive guide to ensemble learning (with python codes) (Dec 2019).
URL `https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/`

[8] Zhangjb, pytorch baseline model -0.916 (Jul 2017).
URL `https://www.kaggle.com/zhangjb/pytorch-baseline-model-0-916`

[9] liuyd2018, planet: multi-label image classification (Mar 2019).
URL `https://www.kaggle.com/liuyd2018/planet-multi-label-image-classification`

[10] Y. Mohajerani, M. Wood, I. Velicogna, E. Rignot, Detection of glacier calving margins with convolutional neural networks: A case study, Remote Sensing 11 (1) (2019) 74.

Appendices

```
# test is the file name.
# mt tag gives the tyoe of the model
# -p tag includes prediction during the run
# -v tag includes validation in the run
# -t tag includes the training in the run and "train_test" is
# where it will be saved

python test.py -mt resnet50 -p -v -t train_test
```

Listing 1: Example for running the code

```

def f_neg(threshold, predictions, true_labels):
    return - fbeta_score(true_labels, predictions > threshold, beta=2, average='
    samples')

def best_f2_score(true_labels, predictions):
    temp_threshold = [0.20] * 17
    minimizer_kwargs = {"method": "L-BFGS-B", "bounds":[(0.,1.)] * 17, "options"
    :{"eps": 0.05}}
    fg = partial(f_neg, true_labels = true_labels, predictions = predictions)

    opt_output = basinhopping(fg, temp_threshold,
                              stepsize = 0.1,
                              minimizer_kwargs=minimizer_kwargs,
                              niter=10,
                              accept_test=bounds)

    return opt_output.x
```

Listing 2: Threshold calculation

```
    ##########################################
    #
    # Load the data from Kaggle and split into
    # train, validation and test sets
    #
    ##########################################

    X_train = KaggleAmazonDataset(TRAIN_DATA,IMG_PATH,IMG_EXT,transform)
    X_val = KaggleAmazonDataset(TRAIN_DATA,IMG_PATH,IMG_EXT,transform_val)
    X_test = KaggleAmazonDataset(TRAIN_DATA,IMG_PATH,IMG_EXT,transform_val)

    x = int(math.floor((2*per) * len(X_train)))

    X_train.splits(0, len(X_train) - x)
    X_val.splits(len(X_val) - x, len(X_val) - (x/2))
    X_test.splits(len(X_test) - (x/2), len(X_test))
```
Listing 3: Data Spliting

```
def modelFreeze(model):
    for layer in model.parameters():
        layer.requires_grad = False

    model.fc.requires_grad = True

    return model

def modelUnFreeze(model):
    for layer in model.parameters():
        layer.requires_grad = True

    return model
```
Listing 4: Freeze/unfreezing layers for the learning

```
def ensembleResnetModelsPredictions(test_loader, model, threshold, X_test,
    run_name, models):
    main_predictions = None
    flag = True

    print("====== Starting Prediction Model Enesembling ======")

    for mt in models:
        mname = "{}.pth".format(mt)
        model = get_from_models(mt, 17, not(mname==None), mname)
        model.cuda()
        model.eval()
        predictions = []
        true_labels = []

        dir_path = './out'
        mlb = X_test.getLabelEncoder()


        with torch.no_grad():
            for id_batch, (data, target) in enumerate(test_loader):
                data = data.cuda(async=True)
                data = Variable(data)

                pred = model(data)

                predictions.append(pred.data.cpu().numpy())
                true_labels.append(target.data.cpu().numpy())

                if id_batch % 10 == 0:
                    print("Done {}%".format(100 * id_batch * len(data)/float(len(
    test_loader.dataset))))

        predictions = np.vstack(predictions)

        if flag == False:
            main_predictions = main_predictions + predctions
        else:
            flag = False
            main_predictions = predictions

    predictions = np.sigmoid(main_predictions/len(models))
    true_labels = np.vstack(true_labels)

    predictions = predictions > threshold
    pred_path = os.path.join(dir_path, run_name + '-raw-pred-1'+'.csv')
    np.savetxt(pred_path, predictions, delimiter=";")

    result = pd.DataFrame({
        'image_name': X_test.name(),
        'tags': mlb.inverse_transform(predictions)
    })

    result['tags'] = result['tags'].apply(lambda tags: " ".join(tags))

    result_path = os.path.join(dir_path, run_name + '-final-pred-'+'.csv')
    result.to_csv(result_path, index=False)

    print("Final predictions saved to {}".format(result_path))

    ff2 = fbeta(np.array(true_labels)[:len(predictions)], np.array(predictions)>
    threshold)

    print("Final fbeta score is " + str(ff2*100))
```
Listing 5: Ensemble model testing code