DS 4400: Machine Learning and Data Mining I

Spring 2021

Project Report

Project Title: Multi-Class Scene Classification

TA: Prabal Malviya

Team Members: Hari Muralikrishnan and Aneesha Sreerama

Access

You can access the Google Colab notebook our entire project code is on through this link directly:
https://colab.research.google.com/drive/16LZqCM7Dr_lgdjbvdcMEpJJAN5mDrRLL?usp=sharing

The presentation for our project can be found here:
https://docs.google.com/presentation/d/1urWCBcWR5xYlgu62jsRmc6fHhkSadAwTxGsx8secHUY/edit?usp=sharing

Problem Description

In this project, our team is trying to classify image data of scenes we might find in the world around us today. Our goal is to use the power of Computer Vision to see if we can develop models to be able to predict images we receive into one of six categories: Building, Forest, Glacier, Mountain, Sea, and Street. As such, this is a clear classification task. This project is important because it will allow our team to acquire real-world experience in creating Deep Learning models using the latest data science tools. Gaining this valuable knowledge will equip us to add value to the image classification domain in the future, which is highly utilized for so many critical tasks. Popular applications of this field include self-driving cars, medical imagery, and face recognition. The most complex solutions start with the simplest building blocks, the latter being the aim for us to master in this project by building upon the exposure given to us in class.

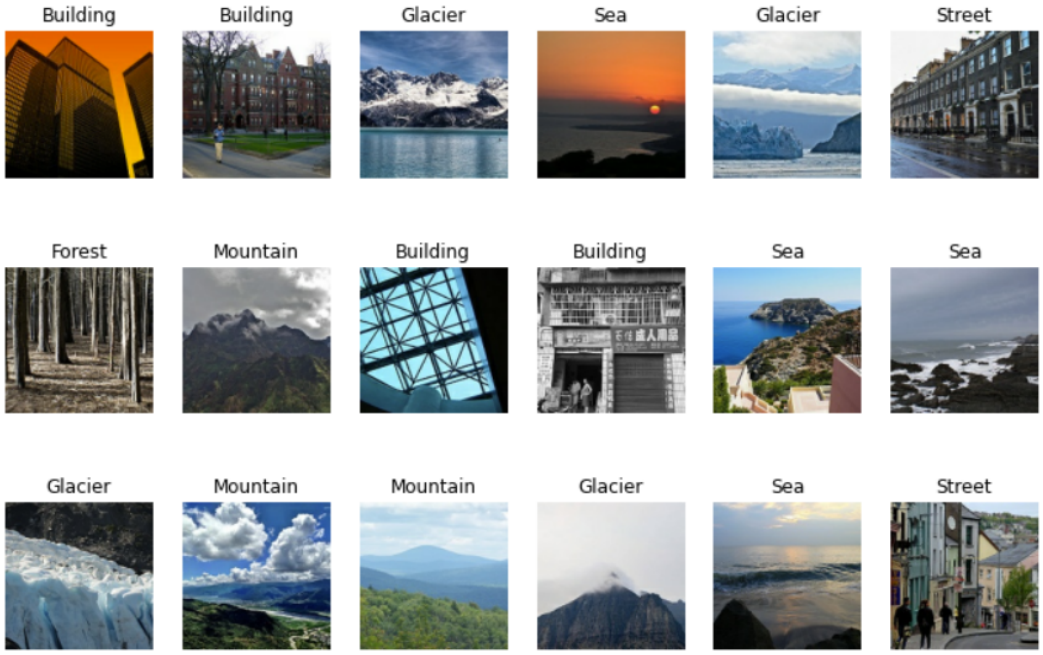
We utilized documentation on a variety of libraries in order to gain a better understanding of Deep Learning, Image Classification, and Image Augmentation. This was done by looking primarily into three important libraries for these tasks: Keras, TensorFlow, and OpenCV. These sources allowed us to see how to use the latest technology to solve our problems, and our project was greatly inspired by the tutorials and samples available on their sites. Likewise, learning about the challenges with image classification such as noise, angle of objects, image quality, and scale of objects gave us more insight into how this may affect the images that we will be testing our models with. Using research papers that involved image classification in order to solve problems, such as identifying plant species with specific diseases, was a great reference for learning how to structure our own analysis and communicate with the audience. Moreover, using blogs in order to peek into the types of problems and approaches taken when structuring an analysis involving pixel data have been reflected in our project.

References

https://docs.opencv.org/master/pages.html
https://www.tensorflow.org/tutorials/images/classification
https://keras.io/examples/vision/image_classification_from_scratch/

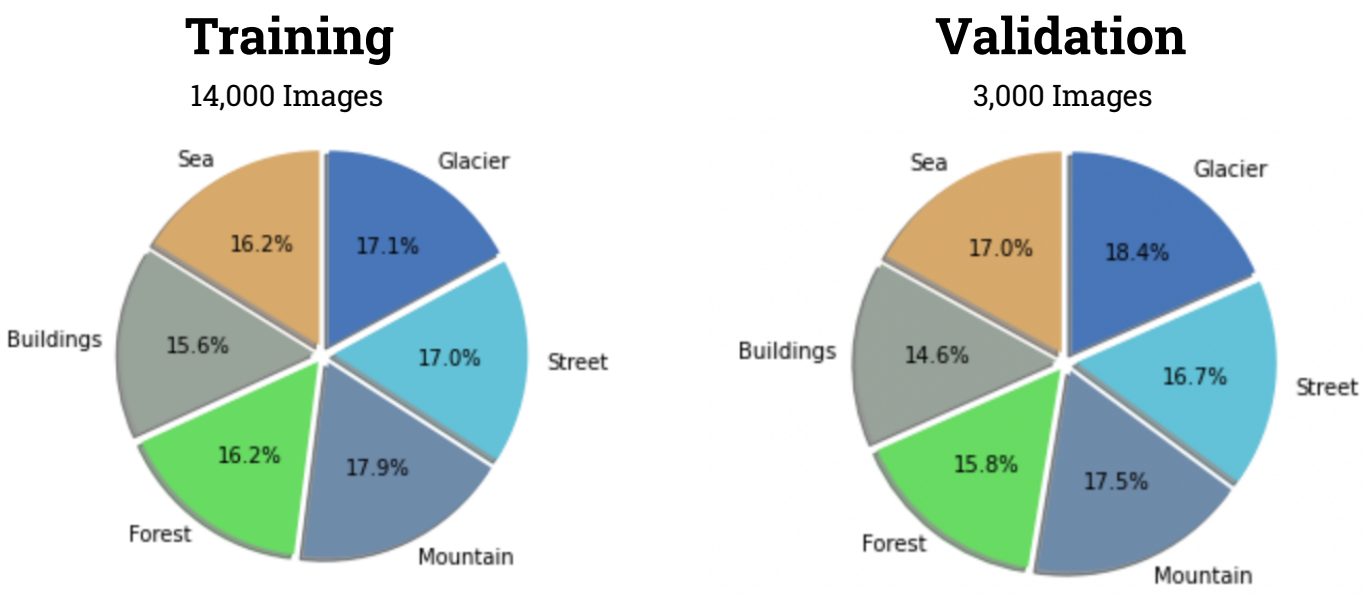
https://towardsdatascience.com/main-challenges-in-image-classification-ba24dc78b558
https://www.hindawi.com/journals/cin/2016/3289801/
https://medium.com/alumnaiacademy/introduction-to-computer-vision-4fc2a2ba9dc#:~:text=Computer%20Vision%20is%20an%20interdisciplinary,way%20a%20human%20mind%20does.&text=And%2C%20Convolutional%20Neural%20Network%20

Dataset

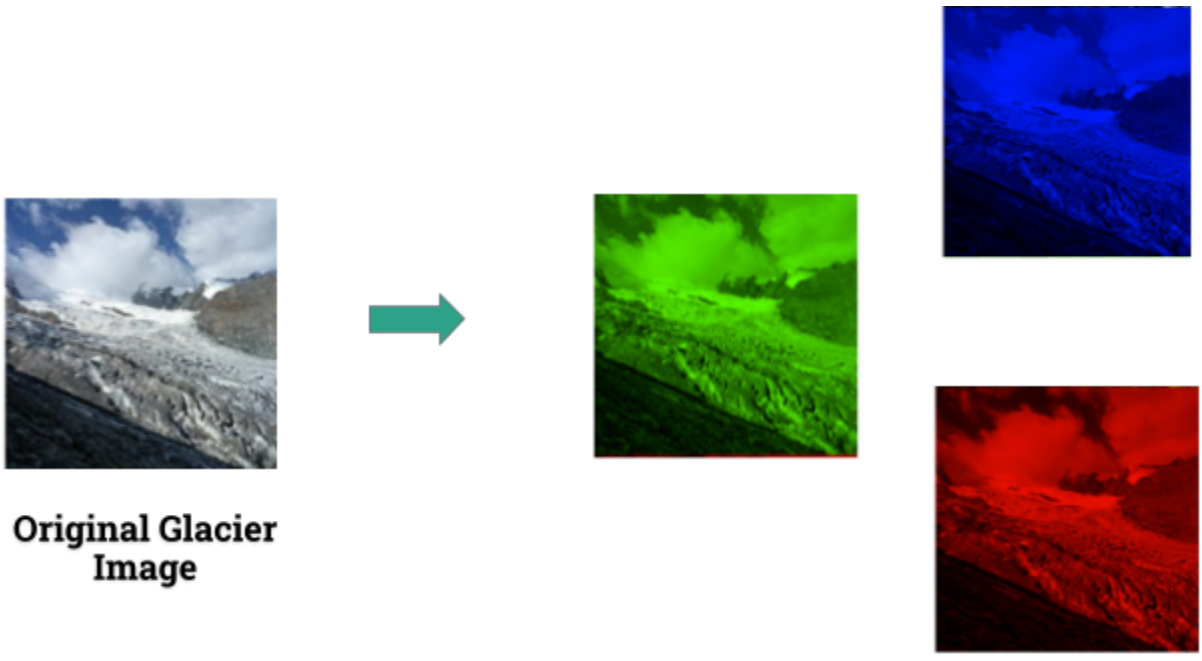


This dataset was initially published by Intel to host a Image Classification Challenge on Analytics Vidhya, and then later posted on Kaggle for public access, where our team found it at https://www.kaggle.com/puneet6060/intel-image-classification. The dataset has about 24,000 images split up into predefined training, validation, and testing sets. The training dataset has about 14,000 images, the validation dataset has about 3,000 images, and the testing dataset has 7,000 images. The classes are Building, Forest, Glacier, Mountain, Sea, and Street, as you can see in the image above. The number of features aren't defined for this problem, as the images have a dimension of 150x150 pixels. We used these image pixels as all the features, so one feature would represent one pixel in the image The feature types are therefore numerical in nature, between a range of 0-255 for each RGB (red, blue, green) value. There isn't missing data since every image has pixels we extracted out to use.

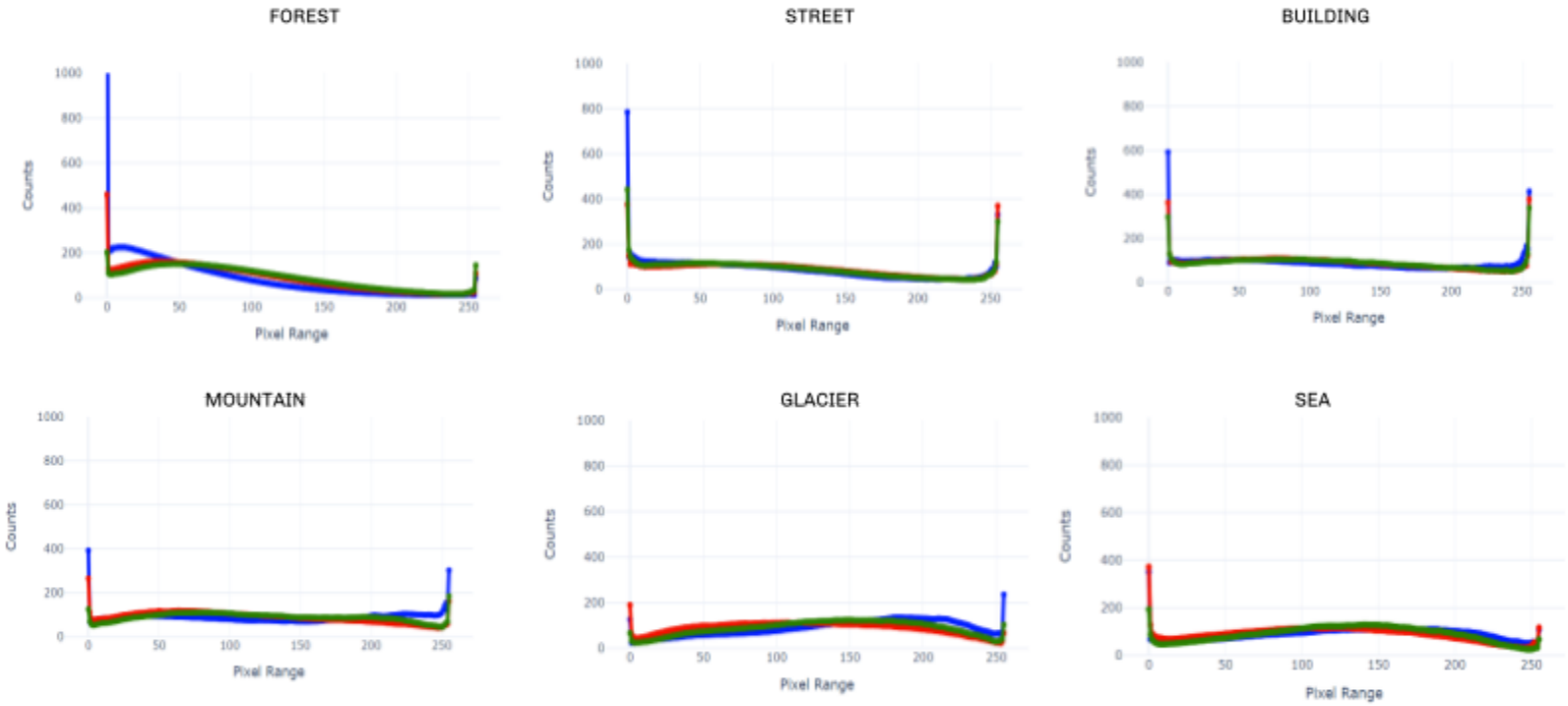
Exploratory Data Analysis



The first analysis we did was looking at the distribution of classes inside our dataset, and it is obvious upon quick review that both the training and validation sets had equal distribution of our 6 class labels.

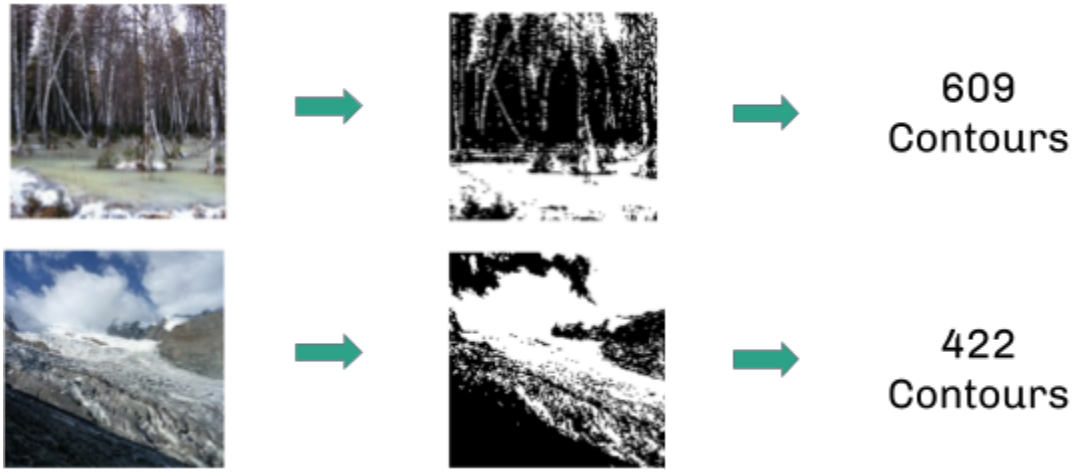


Next, we wanted to look into the color distributions of the different classes in our dataset, specifically looking into RGB pixels - Reds, Greens, and Blues. As a visual representation of what this means, we show above how a glacier image can be broken down into sub-images with the different color filters. Below, we show the results of a similar color analysis on our 6 different classes:

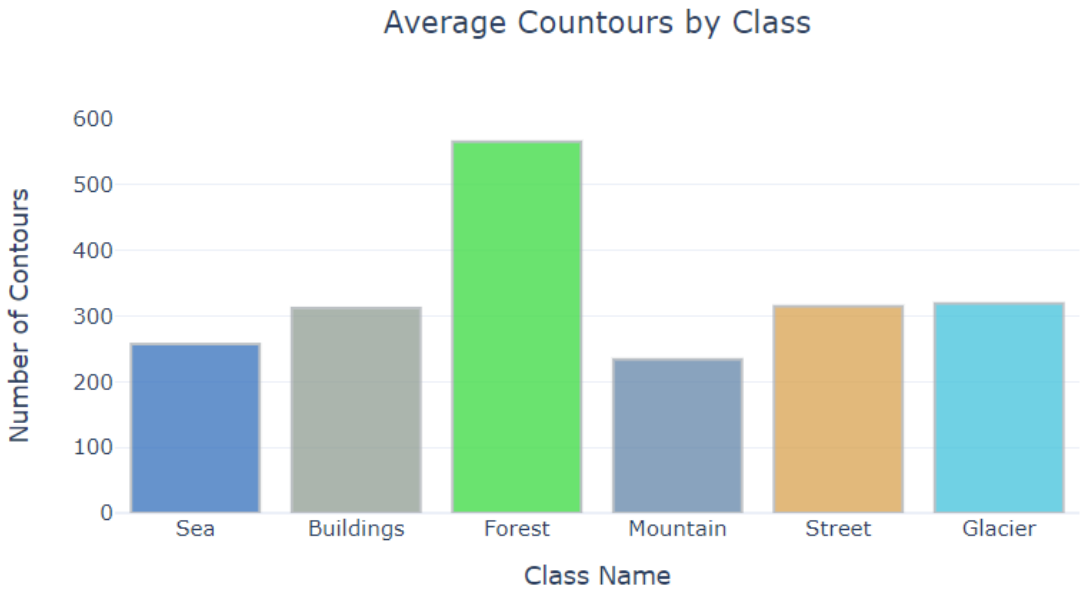


These graphs represent the average distribution of RGB values across all pixels in the images for each class. The x-values of the given visuals represent the different concentration of colors (none to max) for each of RGB. The y-values represent the count of how many pixels have that concentration of color. Moreover, the different colored lines represent the different 3 colors we are focusing on (red, green, blue).

Since each image is 150x150 pixels, there are a total of 22,500 pixels in each image. The graph for the Forest class indicates that on average, about 1000 of the pixels have 0 intensity for the Blue color. This makes a lot of sense, since forests are generally more green and brown in nature. A similar trend occurs when looking across the rest of the top row of graphs, specifically in Streets and Buildings, all of which have a significant amount of 0 intensity Blue pixels.



The last EDA approach we used was surveying the number of contours across the image classes. A contour represents a boundary of an object, so an image with more contours means that it contains more unique objects with edges. We used an OpenCV technique to take an image, make it black and white, and finally count contours in it. Applying this process across all classes resulted in this aggregation:



This shows that the Forest class far surpasses other classes in terms of average contours, indicating more objects in its images. Logically, this checks out since forests usually have branches, tree trunks, and leaves within the different parts of their images. Deep Learning models do well with detecting these object changes and can use that to better approximate the likelihood of an image being of a certain class label, improving its success.

Approach and Methodology

Our approach to solving this problem has evolved since we first began this project. Originally, we were planning on taking a traditional approach to machine learning - we started preparing our data in order to train it using models such kNN, Support Vector Machines, and Decision trees. However, as we began to work with nearly a million features, we realized we had issues with memory, speed, and efficiency when trying to construct one large pandas dataframe with all the data. We also realized that traditional machine learning wasn't well suited for image classification.

Our actual approach ended up relying heavily on the effective use of neural networks and batching in order to process our data. This was a game-changer, and allowed for the rest of the project to become much more manageable. Our team familiarized ourselves with the operations present in TensorFlow and Keras to go through the image classification project pipeline. First, we loaded the data in batches. Then, we constructed neural networks with different architectures and fit them. Finally, we wrote and called functions to calculate metrics on the performances of these different models. The methodology of our project was to go from the most rudimentary model to the most robust model for classifying images of scenes into the 6 classes we had. This approach allowed us to make incremental improvements to our architectures until we reached the best results, which let us learn which techniques within the field of Deep Learning were best suited for Image Classification, and our dataset in particular.

The high-level roadmap we journeyed through using:

1. Feed-Forward Neural Network (FFNN)
2. Convolutional Neural Network (CNN)
3. CNN with Dropout Layers
4. CNN with Image Augmentation
5. Transfer Learning through ImageNet

The next section will break down the results we got as we worked through this approach.

Discussion and Result Interpretation

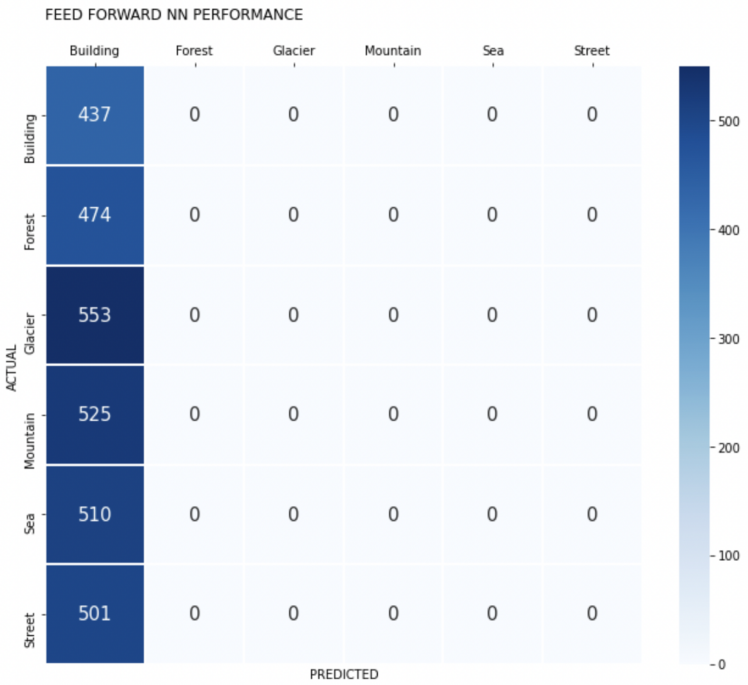
Feed Forward Neural Network (FFNN)

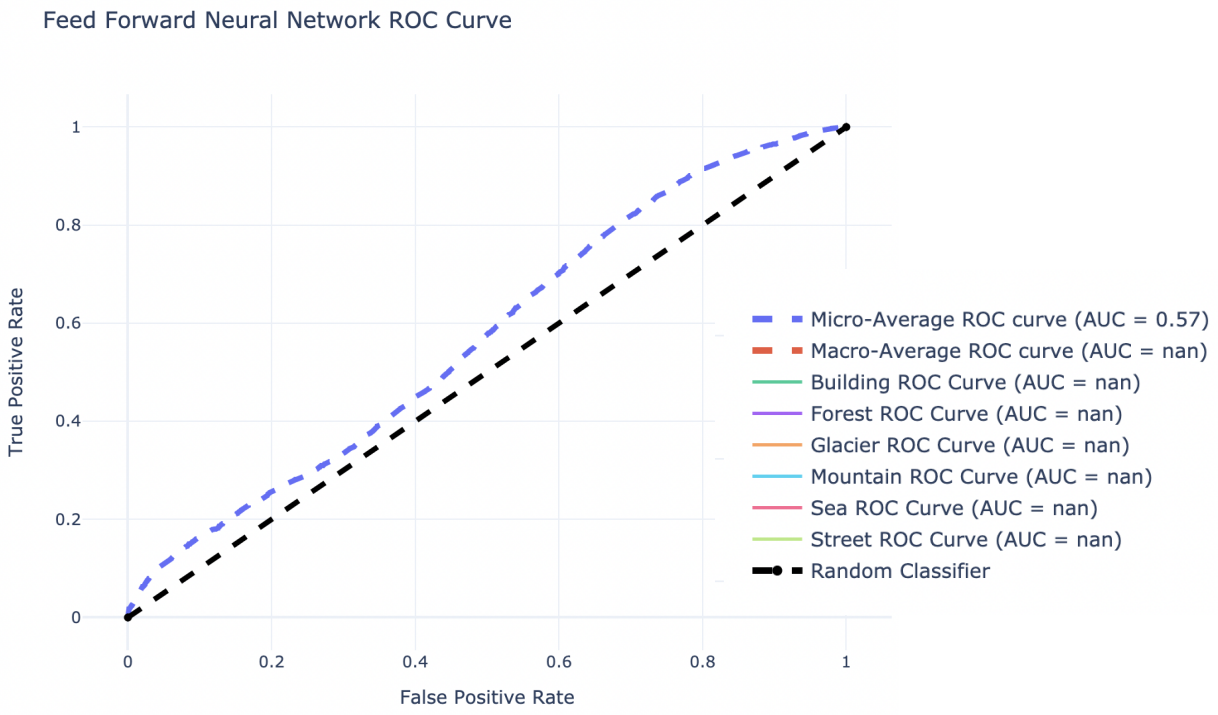
The first model we used was a standard FFNN. It had an architecture with 2 fully connected layers, and it utilized the ReLU activation. Here is the model summary:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_1 (Rescaling)      (None, 150, 150, 3)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 67500)             0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                675010
_________________________________________________________________
activation_2 (Activation)    (None, 10)                0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 11
=================================================================
Total params: 675,021
Trainable params: 675,021
Non-trainable params: 0
_________________________________________________________________
```

The results were quite horrendous for this model, as it predicted everything to be a part of the Building class 100% of the time. This made the accuracy about 15%, which is not much better than a random classifier. These results are corroborated by the metrics shown below in the classification report, confusion matrix, and ROC curve.

	precision	recall	f1-score	support
Building	0.15	1.00	0.25	437
Forest	0.00	0.00	0.00	474
Glacier	0.00	0.00	0.00	553
Mountain	0.00	0.00	0.00	525
Sea	0.00	0.00	0.00	510
Street	0.00	0.00	0.00	501
accuracy			0.15	3000
macro avg	0.02	0.17	0.04	3000
weighted avg	0.02	0.15	0.04	3000





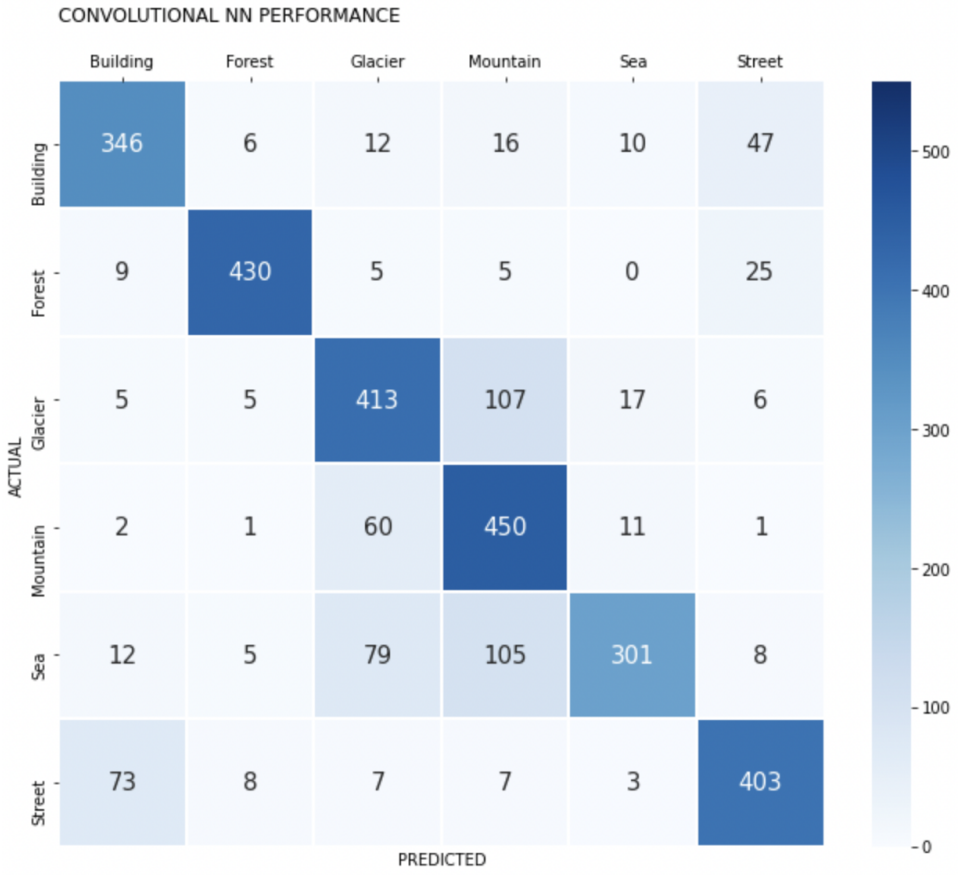
The ROC curve doesn't even plot the curves for each class since this model is so poor. The interpretation of these results is that the FFNN is not meant to be applied for image classification. We learned in class that this is because it does not preserve locality when making predictions, since the architecture takes in a vector only rather than a matrix, meaning every image has to be flattened in order to be processed. Applying this operation makes the input to the neural network much less meaningful, and it has a hard time making even a moderately accurate prediction.

Convolutional Neural Network (CNN)

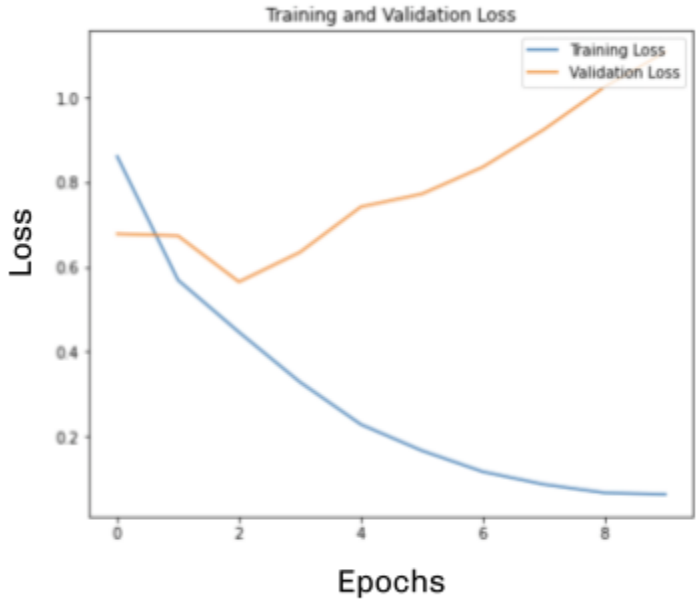
Next, we went to a more appropriate neural network type for our problem - a CNN. Our architecture had 3 convolutional layers, 3 max pooling layers, and then finished off with 2 fully connected layers. Here is the model summary:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling (Rescaling)        (None, 150, 150, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 150, 150, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 75, 75, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 75, 75, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 37, 37, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 37, 37, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 18, 18, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 20736)             0
_________________________________________________________________
dense (Dense)                (None, 128)               2654336
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 774
=================================================================
Total params: 2,678,694
Trainable params: 2,678,694
Non-trainable params: 0
_________________________________________________________________
```

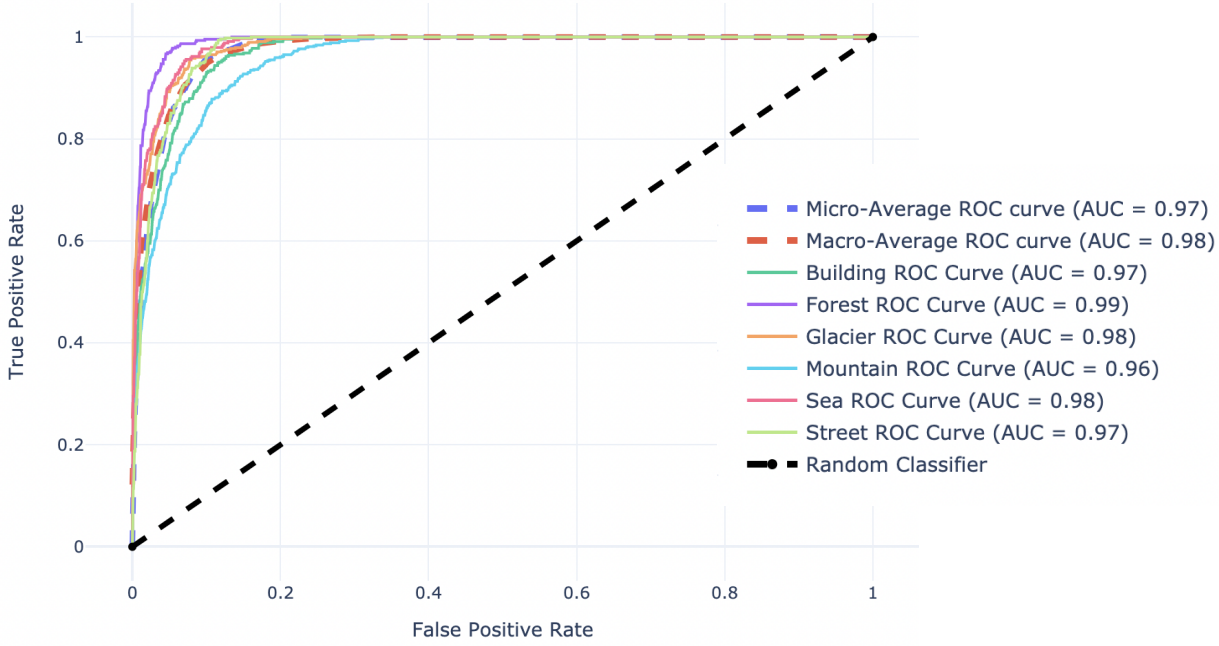
The results for this model had a huge jump in performance. We went from a 15% accuracy in our first architecture to a 78% accuracy here. The confusion matrix shows a much better set of true positive predictions along the diagonal with darker blue squares. Interpreting this confusion matrix gives us an insight into the mode's predictions: it shows that images belonging to the Glaciers and Mountains classes are often mislabelled between each other's classes. Further inspection reveals that even Sea images are wrongly predicted as Glaciers and Mountains sometimes too.



	precision	recall	f1-score	support
Building	0.77	0.79	0.78	437
Forest	0.95	0.91	0.93	474
Glacier	0.72	0.75	0.73	553
Mountain	0.65	0.86	0.74	525
Sea	0.88	0.59	0.71	510
Street	0.82	0.80	0.81	501
accuracy			0.78	3000
macro avg	0.80	0.78	0.78	3000
weighted avg	0.80	0.78	0.78	3000



Convolutional NN ROC Curve



Continuing to observe the other metrics reveals another issue with this setup: overfitting. The loss graph clearly indicates that our model has a high variance because as the number of epochs increase, the training loss goes down

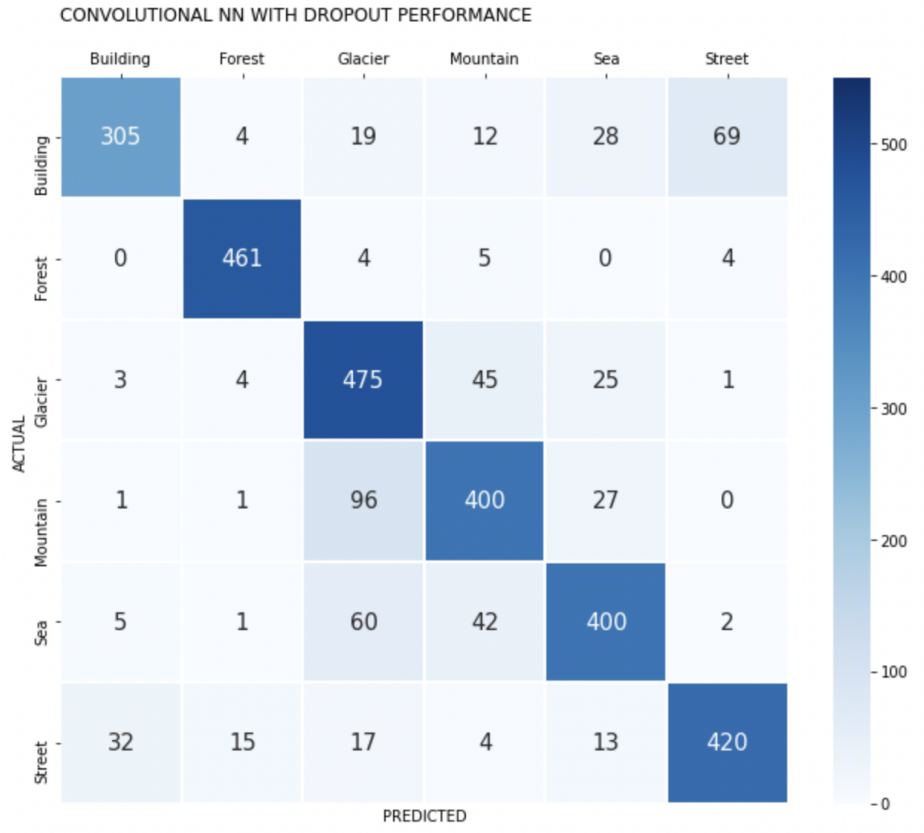
significantly, but the validation loss also skyrockets. Finally, the ROC curve indicates that our model does a much better job than for the FFNN to predict each class when comparing the True Positive Rate to the False Positive Rate.

CNN with Dropout Layers

To solve the overfitting problem, research indicated that adding some dropout layers would be a strong option. Our team decided to take our model from our CNN section and simply add in 4 of these layers with most of them having a 0.25 coverage, meaning that 75% of the neurons in the layers would be utilized for training, with random selection in every epoch. The resulting architecture had this model summary:

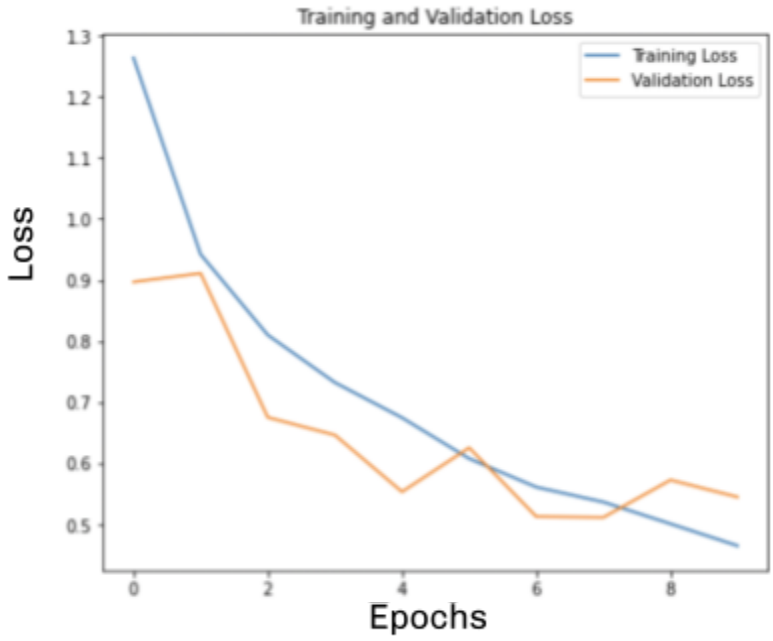
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_1 (Rescaling)      (None, 150, 150, 3)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 150, 150, 16)      448
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 75, 75, 16)        0
_________________________________________________________________
dropout (Dropout)            (None, 75, 75, 16)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 75, 75, 32)        4640
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 37, 37, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 37, 37, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 37, 37, 64)        18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 18, 18, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 18, 18, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 20736)             0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               2654336
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 774
=================================================================
Total params: 2,678,694
Trainable params: 2,678,694
Non-trainable params: 0
_________________________________________________________________
```

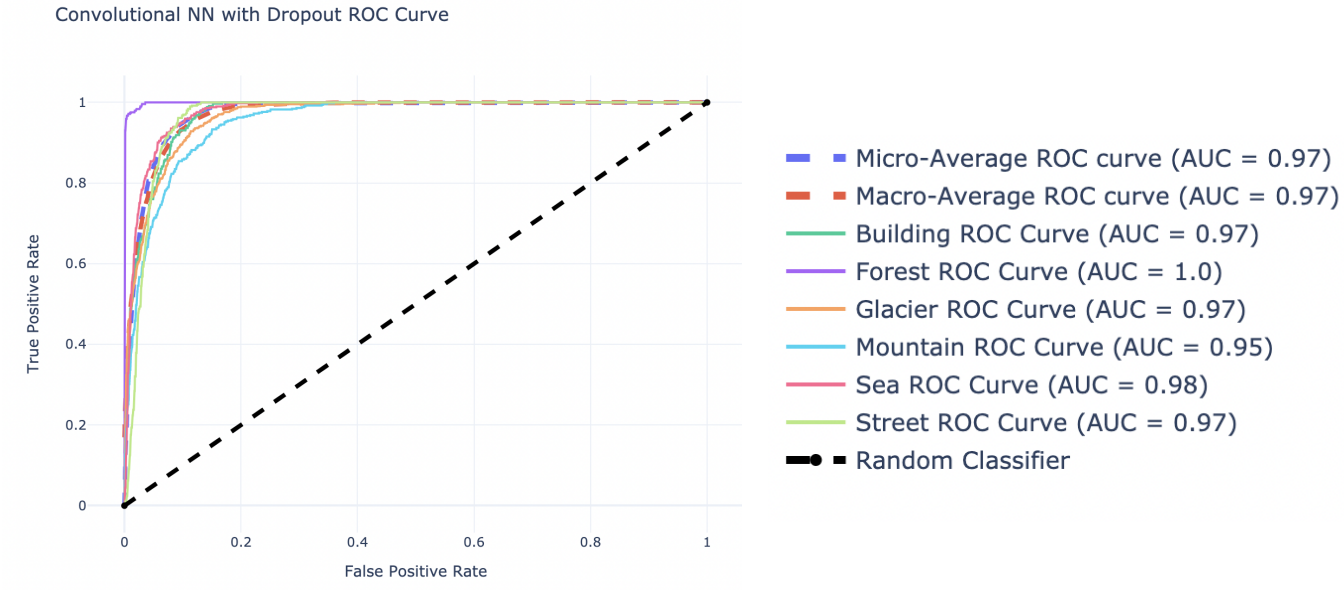
After fitting this updated model, the results were very promising. In terms of accuracy, we only had marginal improvement from 78% from before to 82% now. The confusion matrix continued to show the same model weaknesses as before at



	precision	recall	f1-score	support
Building	0.88	0.70	0.78	437
Forest	0.95	0.97	0.96	474
Glacier	0.71	0.86	0.78	553
Mountain	0.79	0.76	0.77	525
Sea	0.81	0.78	0.80	510
Street	0.85	0.84	0.84	501
accuracy			0.82	3000
macro avg	0.83	0.82	0.82	3000
weighted avg	0.83	0.82	0.82	3000

a high-level, but with much fewer incorrect predictions between Glaciers, Mountains, and Seas.



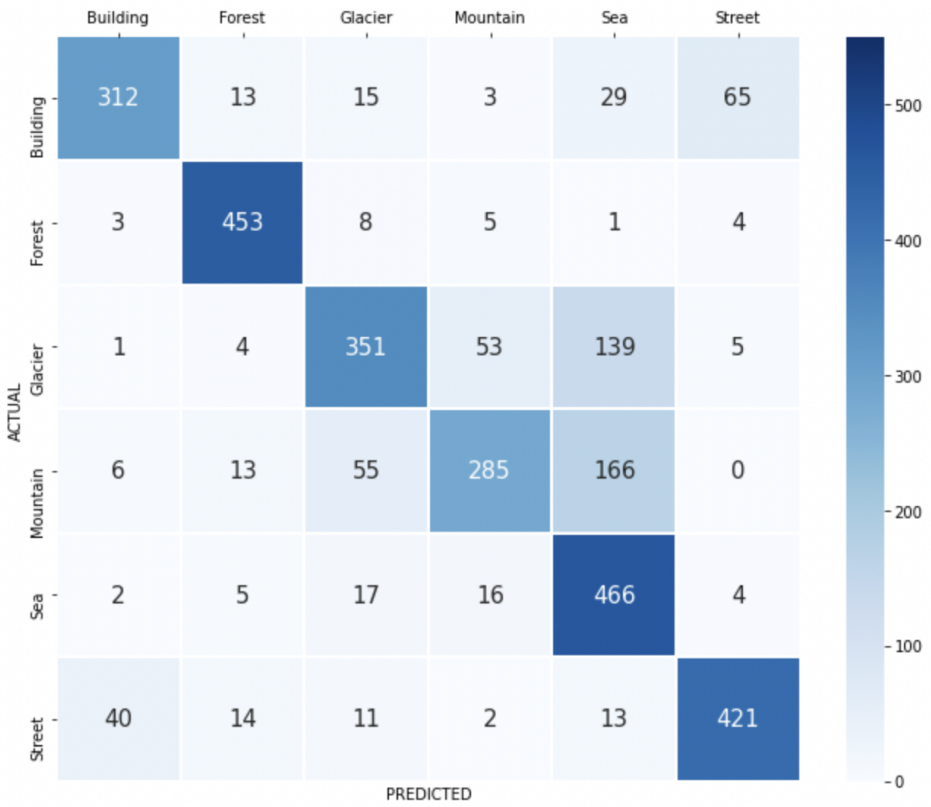


The most important aspect of these results can be seen in the loss graph, as the overfitting problem was solved! As the epochs increased, both our training and validation losses came down together, indicating that the model with dropout layers had a much better tradeoff between bias and variance than before. And this was done while retaining the accuracy and validity of our model, which our ROC curve shows.

CNN with Image Augmentation

Now that our model was starting to get stronger, we wanted to see how well it could perform on images that weren't clear and almost too "perfect". This is because we want our model to be able to make valuable predictions even when getting more noisy or messy inputs, as real-life image classification tools have to do. In order to test if our model was more flexible, we found that the best method to do so would be to augment the images in our dataset. Using OpenCV, we researched different operations that could be applied to augment images, and finally we decided that an appropriate one to apply on our own project dataset would be the bilateral filter. This operation blurs the entire image except for the edges, which are often used as a "feature" by neural networks to classify images.

The above image tries to illustrate how the bilateral filter reduces noise within the image and "smooths" it out. So our approach was to take each image in our training dataset, augment it using the bilateral filter, and replace it in its location. We then took our CNN model with Dropout Layers and fitted it on the augmented dataset. When applied to predict against the validation dataset, which was untouched (no bilateral filter), here are the results:



	precision	recall	f1-score	support
Building	0.79	0.85	0.82	437
Forest	0.94	0.98	0.96	474
Glacier	0.85	0.71	0.77	553
Mountain	0.75	0.81	0.78	525
Sea	0.80	0.88	0.84	510
Street	0.90	0.80	0.85	501
accuracy			0.83	3000
macro avg	0.84	0.84	0.84	3000
weighted avg	0.84	0.83	0.83	3000

The accuracy only increased negligibly from our unaugmented dataset model, but the more significant change from the bilateral filter operation being applied can be seen in the confusion matrix. Before, glaciers and mountains used to heavily misclassify between each other's classes. Now, the problem has shifted so that both those classes misclassify as the Sea class very frequently, as indicated by the

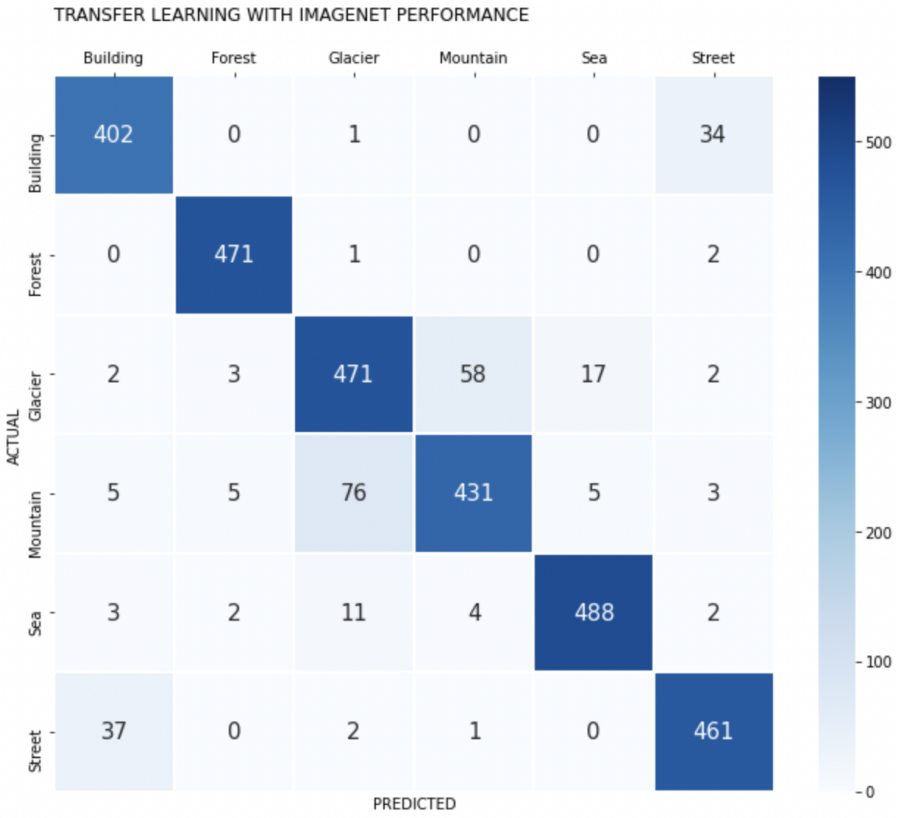
darker blue squares in the Sea column. Overall, the augmentation retained accuracy and solved one classification problem only to create another one.

Transfer Learning through ImageNet

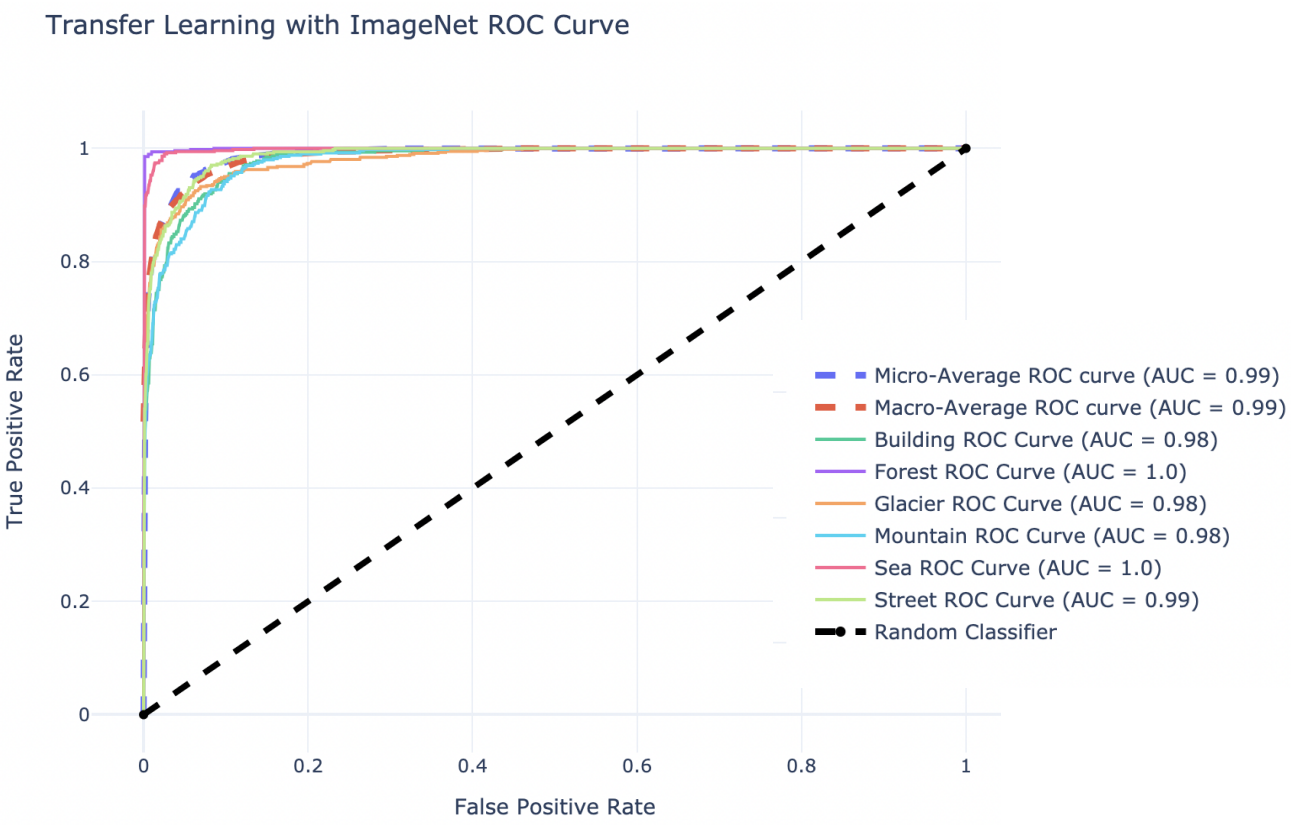
Finally, our team wanted to apply transfer learning to our project. The reasoning behind this was that there are much stronger models out there pre-trained for image classification problems, so it would be fascinating to see if those models could be plugged into our project here and tested for success. We chose to go with the Xception model trained on the ImageNet dataset, which is famous for having billions of images in it. To incorporate it for use in our project, we took all the pre-trained layers and associated weights and kept them as they were. All we changed was to remove their last fully connected layer and replace it with our own, so that the model could predict for our 6 classes. Here is the model summary:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 150, 150, 3)]     0
_________________________________________________________________
normalization (Normalization (None, 150, 150, 3)       7
_________________________________________________________________
xception (Functional)        (None, 5, 5, 2048)        20861480
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0
_________________________________________________________________
dropout_4 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_6 (Dense)              (None, 6)                 12294
=================================================================
Total params: 20,873,781
Trainable params: 12,294
Non-trainable params: 20,861,487
_________________________________________________________________
```

After fitting the model using a learning rate of 0.001, the results were extremely impressive, as accuracy leaped from 83% from before to 91% here, as seen in the classification report. The confusion matrix shows that almost all the misclassification problems were resolved, although the Glacier-Mountain confusion remains in the background very slightly as indicated by the light aqua blue squares. Here are those metrics:



	precision	recall	f1-score	support
Building	0.91	0.92	0.91	437
Forest	0.99	0.99	0.99	474
Glacier	0.89	0.79	0.84	553
Mountain	0.83	0.88	0.85	525
Sea	0.94	0.98	0.96	510
Street	0.91	0.92	0.91	501
accuracy			0.91	3000
macro avg	0.91	0.91	0.91	3000
weighted avg	0.91	0.91	0.91	3000



The ROC curve also shows that the model is essentially a perfect classifier, even while comparing across all the different labels. This is proven by the AUC scores, all of which range between 98% to 100% success. It's quite astounding how powerful this model trained using ImageNet was, and how robust it was to be applied to our own project problem without trouble. This illustrates the strength of transfer learning with neural networks!

(Note: we also trained this transfer learning model using a learning rate of 0.0001 as per the TA's recommendation, but the results had negligible differences with this one)

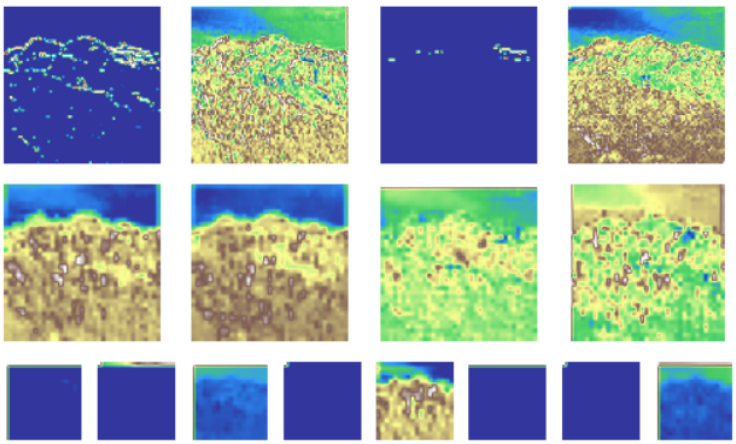
Conclusion

Our team was extremely pleased with this project. From start to finish, we witnessed the evolution of success in our models all the way from having an initial 15% accuracy with our FFNN, all the way to a 91% accuracy using a model trained on ImageNet. This journey did not come without its challenges however. We learnt early on that it was best to use the capabilities offered by Deep Learning packages like Keras, TensorFlow, and OpenCV rather than use the traditional methods offered by Numpy, Pandas, and Sci-kit Learn for processing large datasets and classifying images. As we made more models more and more robust, we discovered that it was imperative to make sure that we were training them using GPU rather than CPU, because the former cuts down times by significant amounts.

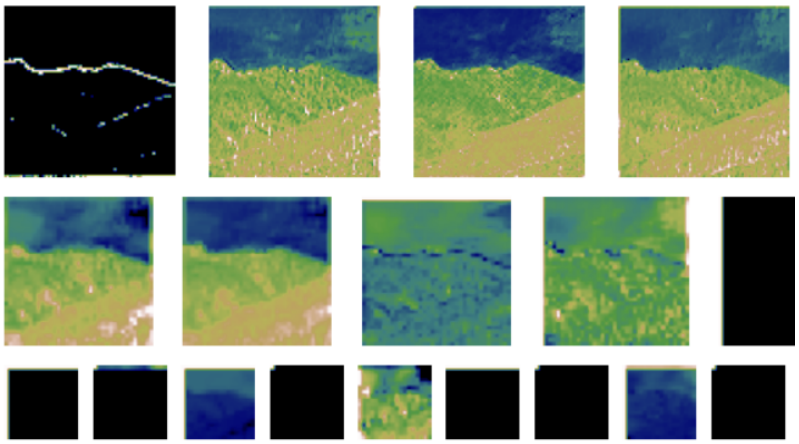
The last important challenge that we faced but did not solve with 100% success was making our models intelligent enough to effectively discriminate between pictures of the Glacier and Mountain types. Upon further investigation, our team understood why this misclassification was occurring across all our models. In each layer of our CNN's, the models had filters that were used to breakdown a 150x150 image into smaller and smaller regions until it could compress the data enough to be classified after funnelling through a fully connected network. Visualizing what these filters observed give great insight into the problem. Consider these 2 images:

Glacier

Mountain



Glacier Filters



Mountain Filters

Looking at the original images, it is hard enough to discriminate between a Glacier and Mountain with the naked human eye. Then, observing the filters, where each row in the filter picture represents what each layer in the CNN captures, it is quite evident that our model will have clear trouble because the data is further reduced for comparison. Therefore, the structure of the dataset and the images contained within it are the cause of this issue, rather than a weakness in any particular CNN.

Ultimately, our team learnt a lot through this project, and we are proud of our results. The experience we have gained in taking a rudimentary neural network architecture and evolving it to improve in success was invaluable. As to the bigger picture, we can take these insights and use it to solve much more interesting problems than classifying images into 6 different types of scenes.

Team Member Contribution

Our team worked very closely in a pair-programming fashion to complete most of the work on this project. However, each of us focused on different aspects of the project to become an “expert” in, and so the high-level split of this work is:

Aneesha - EDA and Metric visualizations, Image Augmentation, Presentation
Hari - Modeling and Architecture selection, Transfer Learning, Project Report