CUNY MSDS – DATA622

Homework 4

Introduction

For this assignment, I will construct two models to perform image classification. The first model will leverage methodologies from the first ten weeks of the course — namely, XGBoost and Principal Component Analysis (PCA). The second will leverage a methodology from the final weeks of the course — a pre-trained convolutional neural network (CNN) called ResNet-50, fine-tuned on the dataset in question.

I sourced the dataset from Kaggle [1], and it contains ~25,000 images of landscapes / city scenes, categorized into six classes:

- 'buildings' -> 0,
- 'forest' -> 1,
- 'glacier' -> 2,
- 'mountain' -> 3,
- 'sea' -> 4,
- 'street' -> 5

Code for all analysis is saved here: https://github.com/kac624/cuny/tree/main/D622.

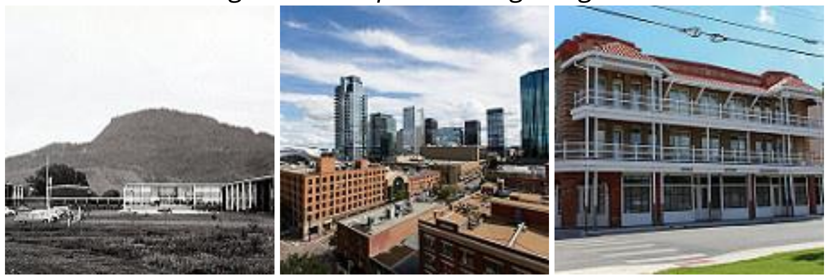
Exploratory Data Analysis

The data are split into three subsets, with ~14k images in Train, ~3k in Test and ~7k in Prediction. The dataset was initially published as part of a hackathon competition from Analytics Vidhya and Intel [2], so the Prediction subset does *not* have labels. I require labels for this assignment, so the Prediction subset was excluded. Instead, I further divided the Train data into Train and Validation subsets, with ~11k and ~3k images each.

The vast majority of images have a standard size (150 by 150 pixels), but a small number (48 or 0.34%) had irregular heights, ranging from 76 to 149 pixels. I standardized all images sizes before feeding data to the model (see Preprocessing), but these irregular heights may lead to distorted images that are difficult to classify. Given the very small volume of such images, however, I did not consider this finding significant.

Images are a mix of color and black-and-white, and a sample viewing (see Figures 1 to 6) shows them to be mostly distinct across classes. However, there does appear to be some overlap. For example, some Glacier images might be confused for Mountain images, and vice versa. Similarly, some Building images might be confused for Street images, and vice versa.

Figure 1: Sample Building Images



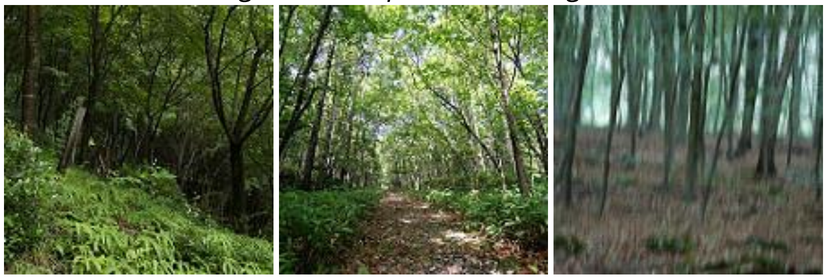


Figure 2: Sample Forest Images

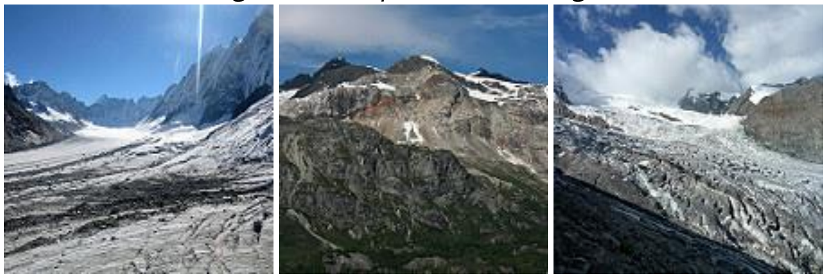


Figure 3: Sample Glacier Images

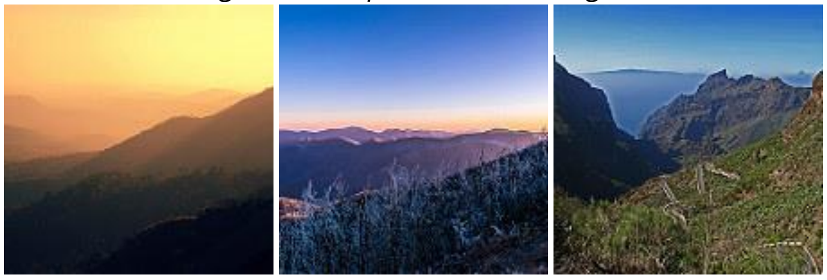


Figure 4: Sample Mountain Images

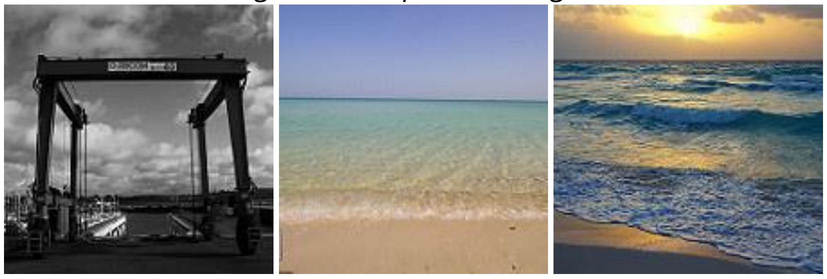


Figure 5: Sample Sea Images

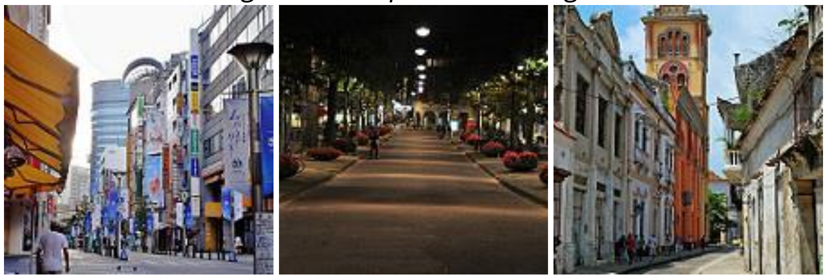
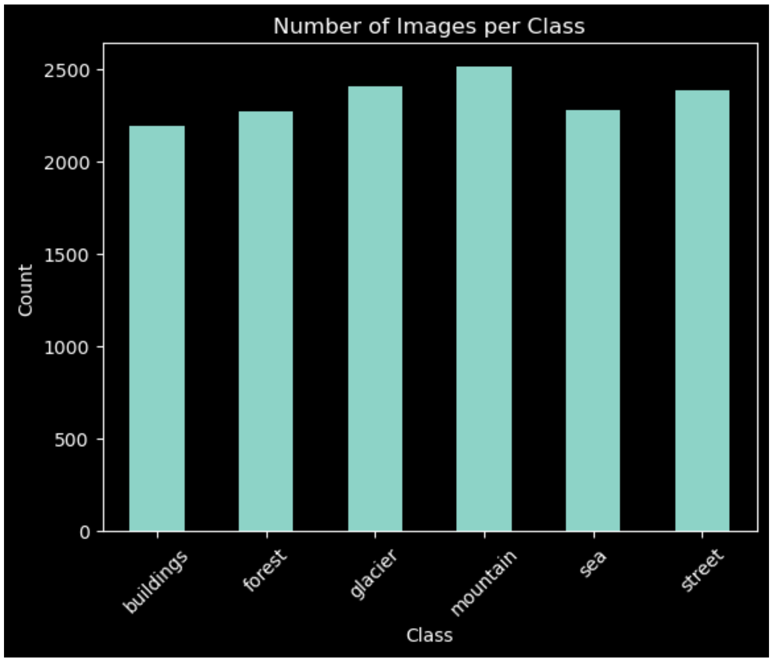


Figure 6: Sample Street Images

The images are roughly evenly distributed across classes, as shown in Figure 7. As a result, I will not employ any kind of over- or under-sampling techniques.

Figure 7: Distribution of Images Across Classes



Moreover, analysis of the distributions of pixel intensity and colors do not reveal any significant differences across classes (see Figures 8 and 9 below). Some differences emerge, such as the generally lower intensity of Forest images and generally higher intensity for Sea images. Similarly, Glacier images appear to be more blue than images in other classes, and Forest images appear to be less red. However, these differences do not appear sufficiently stark to require manipulation (beyond the standard normalization of pixel values; see Preprocessing).

Figure 8: Pixel Intensity Distribution per Class (Log Scale)

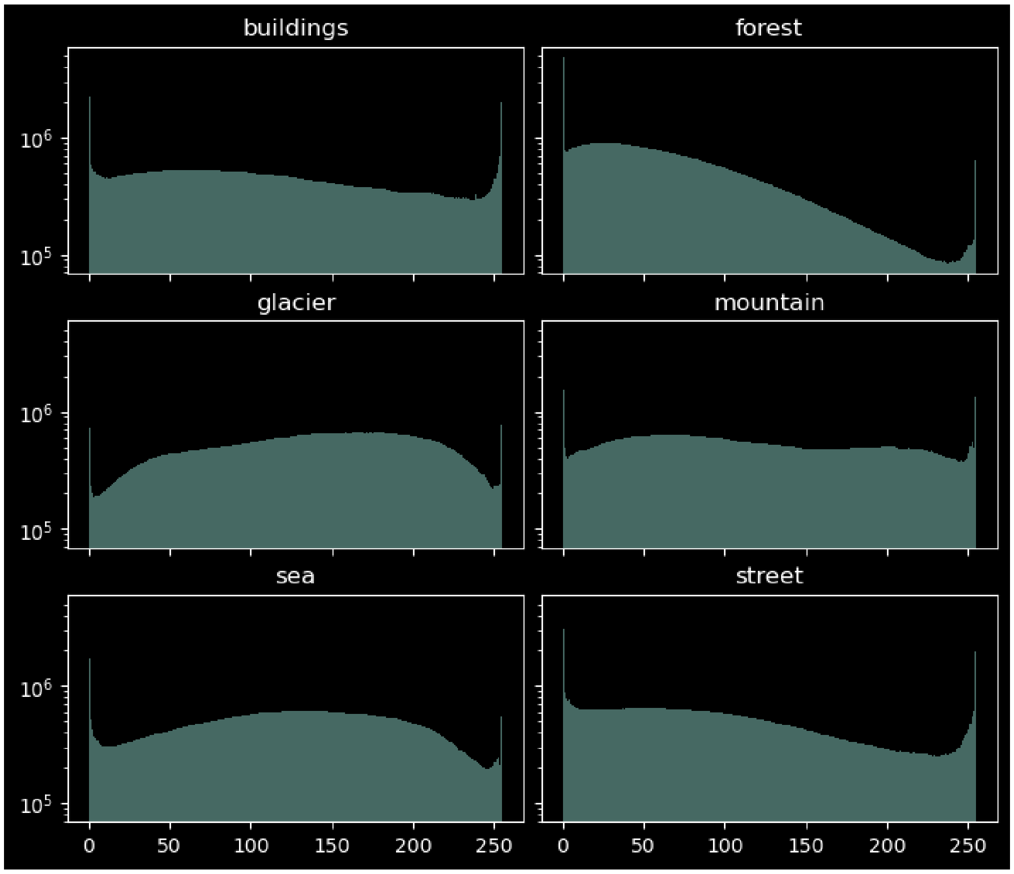
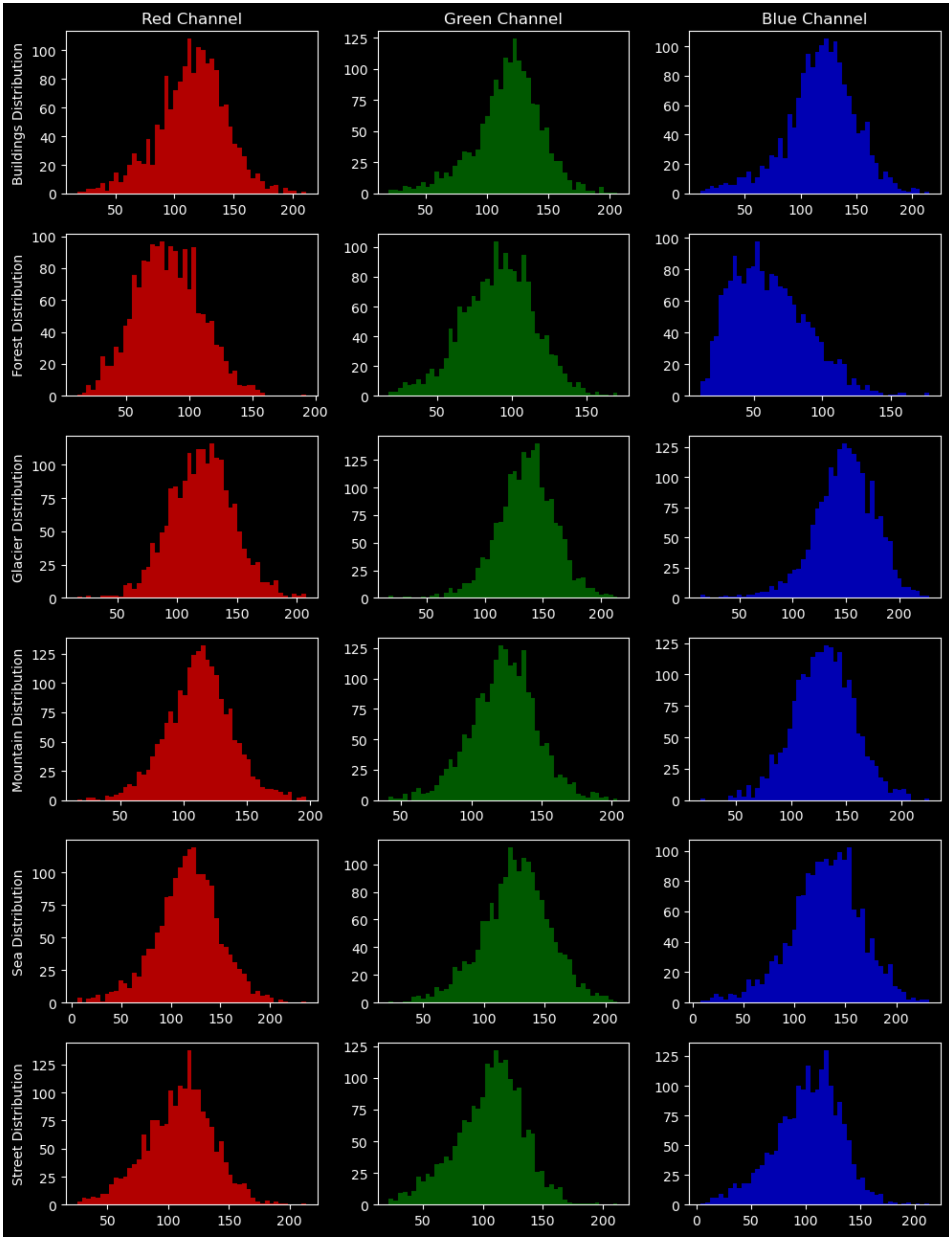


Figure 9: Distribution Across RGB Color Channels Per Class



Preprocessing

To prepare the data for ResNet-50, I had to create a pipeline with three steps:

1. Resize Images: ResNet-50 expects input images of a consistent size. The typical input size for ResNet-50 is 224 by 224 pixels, so all images were resized to this dimension.
2. Convert to Tensor: ResNet-50 requires numerical data. Moreover, it is designed to work with images having three color channels (RGB). So, all images were converted to tensors of size 224 by 224 by 3.
3. Normalization: Because ResNet-50 is a pre-trained model, performance can be enhanced by normalizing the data to match the distribution of the data used during pre-training. Images should be normalized by subtracting the mean and dividing by the standard deviation for each color channel. For ResNet-50, the typical means are [0.485, 0.456, 0.406] and standard deviations are [0.229, 0.224, 0.225] for the RGB channels, respectively.

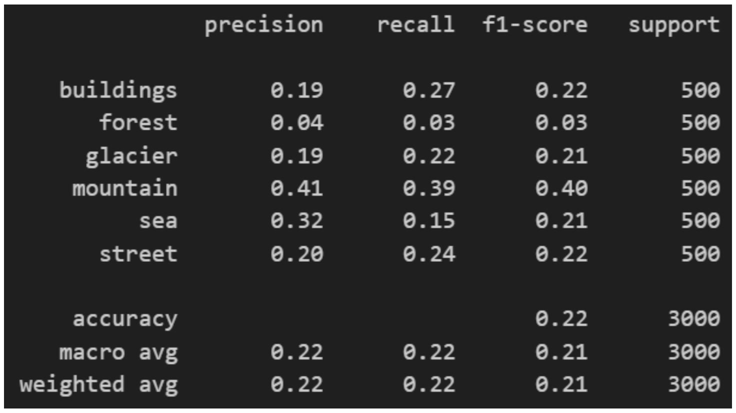
Additionally, I included some common data augmentation techniques in the pipeline. In particular, I applied random resizing and random horizontal flip. The "random" aspect of these techniques indicates that they are dynamic. In other words, they are applied randomly during each epoch in training. As a result, the model is exposed to slightly altered image data each training cycle, providing regularization and better generalization to new data.

To prepare data for the XGBoost benchmark, I leveraged the same pipeline as above to ensure consistency across models. However, XGBoost requires a tabular dataset, so the 3-dimensional tensor will not work. Moreover, normalization is unnecessary and may introduce bias. So, I reshaped each image into a flat vector and de-normalized the values. That flat vector has a dimension of over 150,000, which would prove problematic in terms of training time. So, I applied Principal Component Analysis (PCA) to the flattened image data to provide a more salient set of features.

Modeling

In line with expectations, the XGBoost model performed poorly. While this algorithm is one of the most performant in terms of tabular datasets, it typically does not handle image data very well (even when flattened into a tabular format). The figure below details performance, with an accuracy of only ~22%.

Figure 10: Benchmark XGBoost Performance on Validation Data



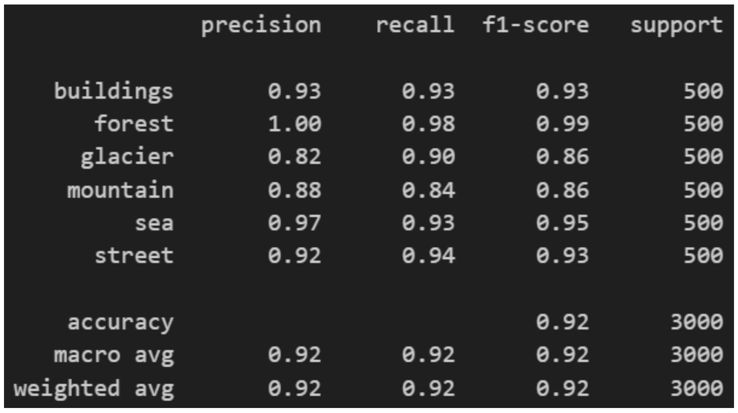
```
              precision    recall  f1-score   support

   buildings       0.19      0.27      0.22       500
      forest       0.04      0.03      0.03       500
     glacier       0.19      0.22      0.21       500
    mountain       0.41      0.39      0.40       500
         sea       0.32      0.15      0.21       500
      street       0.20      0.24      0.22       500

    accuracy                           0.22      3000
   macro avg       0.22      0.22      0.21      3000
weighted avg       0.22      0.22      0.21      3000
```

The ResNet-50 mode, on the other hand, performed very well. To prepare the model, I used pretrained weights ("IMAGENET1K_V1") and adjusted the final fully connected layer to generate six outputs (rather than the default 1000), aligning to our six classes. I also froze all layers so that only the parameters of the final layer would be updated during training. As shown in Figure 11, the model achieved 92% accuracy.

Figure 11: Baseline ResNet-50 Performance on Validation Data



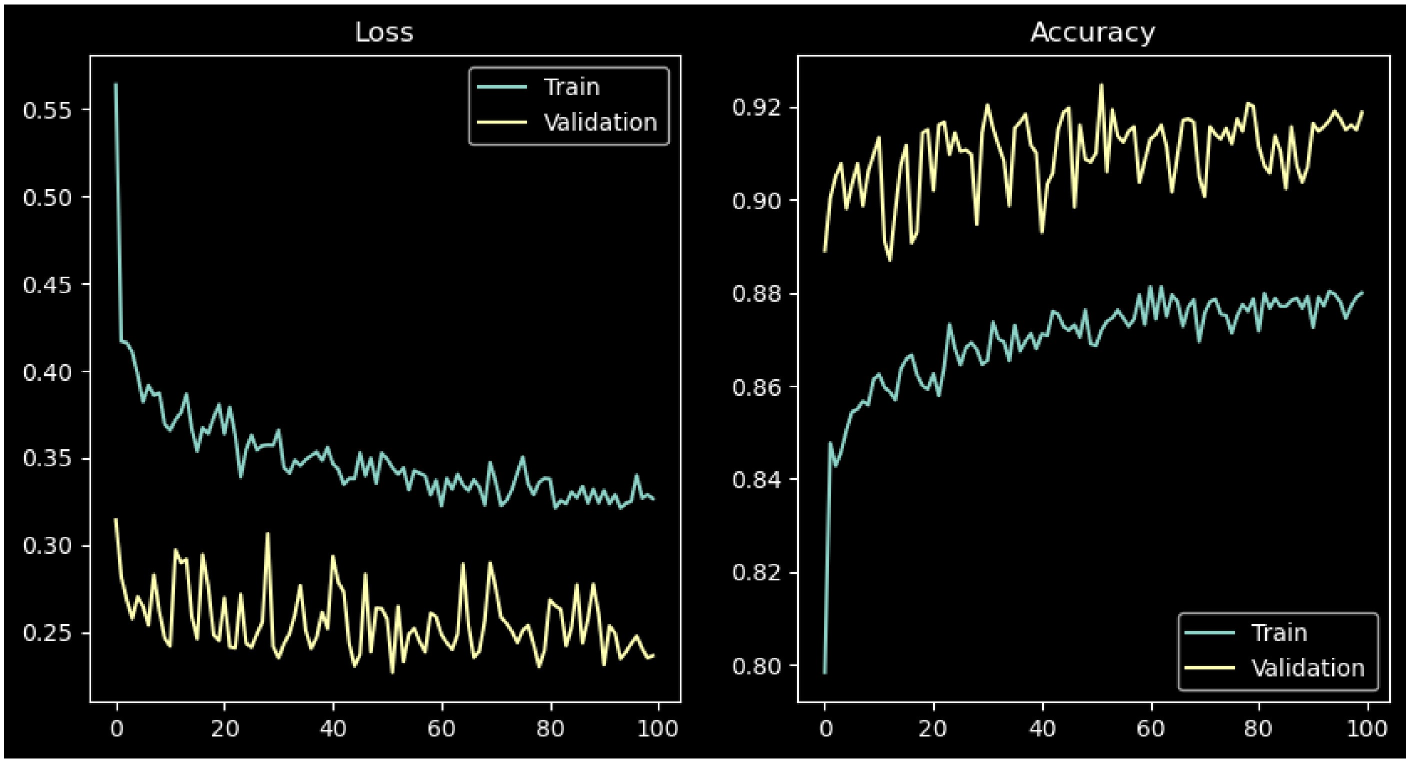
```
              precision    recall  f1-score   support

   buildings       0.93      0.93      0.93       500
      forest       1.00      0.98      0.99       500
     glacier       0.82      0.90      0.86       500
    mountain       0.88      0.84      0.86       500
         sea       0.97      0.93      0.95       500
      street       0.92      0.94      0.93       500

    accuracy                           0.92      3000
   macro avg       0.92      0.92      0.92      3000
weighted avg       0.92      0.92      0.92      3000
```

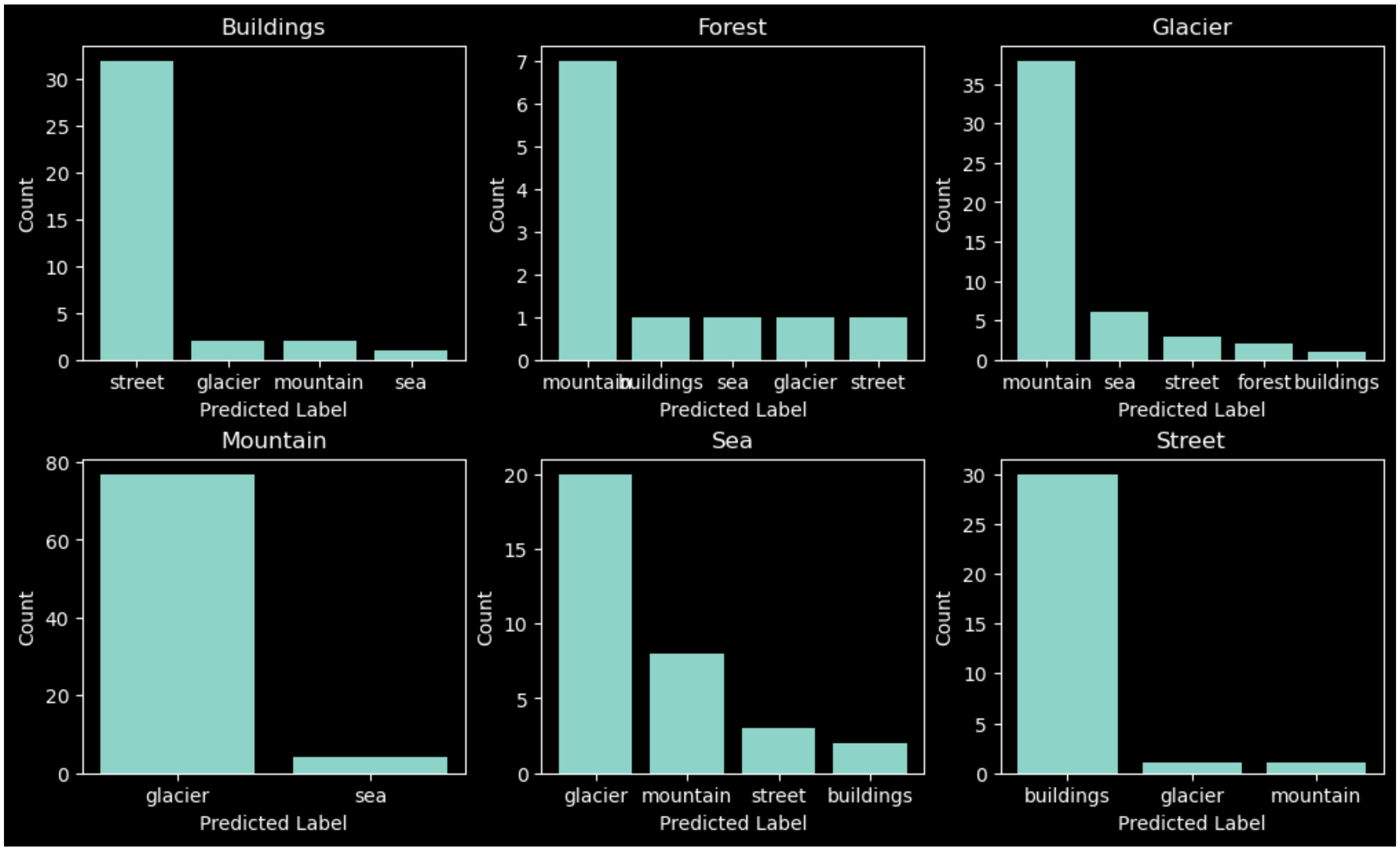
Figure 12 below shows the change in raw, aggregate losses and accuracy over the 100 training epochs. While the validation plots are especially noisy, they do show a trend over the full cycle, indicating that model performance continued to improve over time despite short-term ups and downs. However, we do see significantly better improvement in the training subset, indicating the need for stronger regularization.

Figure 12: Losses and Accuracy Over 100 Epochs



I conducted error analysis to identify potential trends on the model's incorrect predictions. Figure 13 shows the distribution of the model's predictions for all incorrect predictions, broken out by the class of the true label. The most notable takeaway is that, as predicted during EDA, the model appears to confuse Mountain and Glacier images, as well as Street and Building images. It also, however, seems to conflate Forest and Mountain images.

Figure 13: Distribution of Predictions for Model Errors by Class

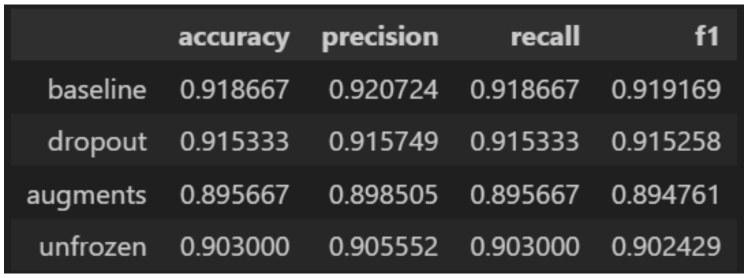


To address this overlap, I explored three model enhancements:

1. Regularization through Dropout: I adjusted the final, fully connected layer of the model to include a dropout layer. This layer will randomly ignore or "drop out" activation values from the previous layer (with a probability of 0.3), with the aim of preventing overfitting.
2. Additional Data Augmentation: In addition to the initial random resized crop and random horizontal flip, I added two more augments to the pipeline: random rotation and color jitter. As with the dropout layer, the goal here is to introduce some "noise" in the training cycle and encourage the model to better generalize to new data.
3. Fine-Tuning Additional Layers: The baseline model only focused on training the final layer of the model. To better fine-tune the model to the data, I unfroze the final two broad layers (which consist of multiple underlying layers). During training, more parameters were updated to better fit the data.

Unfortunately, none of these techniques improved performance on the validation data. In fact, they all resulted in slightly lower accuracy, as shown in Figure 14.

Figure 14: Comparison of Model Performance

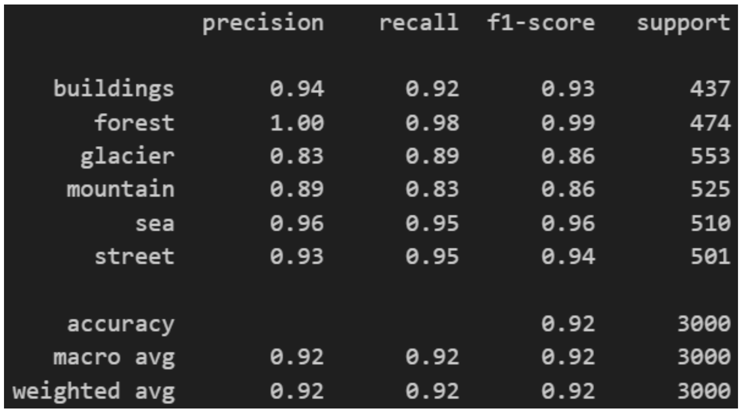


	accuracy	precision	recall	f1
baseline	0.918667	0.920724	0.918667	0.919169
dropout	0.915333	0.915749	0.915333	0.915258
augments	0.895667	0.898505	0.895667	0.894761
unfrozen	0.903000	0.905552	0.903000	0.902429

Conclusion

Ultimately, the baseline model appeared to be the best. So, I assessed that model's performance on the holdout test data, detailed in Figure 15.

Figure 15: Final Evaluation – Baseline Model on Test Data



```
              precision    recall  f1-score   support

   buildings       0.94      0.92      0.93       437
      forest       1.00      0.98      0.99       474
     glacier       0.83      0.89      0.86       553
    mountain       0.89      0.83      0.86       525
         sea       0.96      0.95      0.96       510
      street       0.93      0.95      0.94       501

    accuracy                           0.92      3000
   macro avg       0.92      0.92      0.92      3000
weighted avg       0.92      0.92      0.92      3000
```

The performance is consistent with performance on the validation set at ~92%. Overall, the exercise yielded the following takeaways.

- XGBoost appears to underperform on image classification, at least when dealing with features extracted via PCA. Other feature extraction methods might prove more effective, but those methods may require the use of pre-trained neural networks.
- CNNs appear well suited for image classification, and the pre-trained ResNet-50 model performs quite well "out of the box."
    - ResNet-50 appears responsive to fine-tuning, but progress is achieved relatively slowly, and may require a significant amount of data, coupled with many training iterations and strong regularization. Tuning of key hyperparameters (especially the learning rate) may also prove fruitful.
    - With our dataset, ResNet-50 appeared unresponsive to data augmentation, regularization, and more comprehensive fine-tuning. The inability to significantly exceed the baseline ~90% accuracy may be attributable to poorly defined classes (e.g., overlap between mountains and glaciers, as well as streets and buildings).

Sources

[1] “Intel Image Classification.” Sourced from https://www.kaggle.com/datasets/puneet6060/intel-image-classification.

[2] “Data Hack.” Refer to https://datahack.analyticsvidhya.com/.