Image Classification: CNN vs Random Forest

Shree Murthy

2024-02-19

Table of Contents

1. Problem Statement

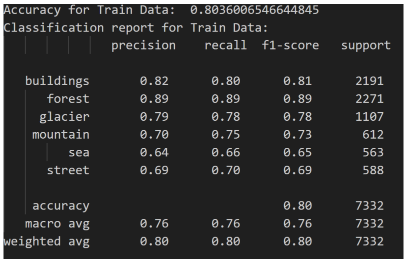
The Intel Image Classification dataset contains thousands of images of natural scenery around the world. The images are divided into 6 categories: buildings, forest, glacier, mountain, sea, and street. The goal of this project is to build a model that can accurately classify these images into their respective categories.

Image classification is an ideal way to learn about CNN-based models and understand how it can outperform traditional machine learning models like Random Forest. To explain, why we need to use a CNN model I will describe how the Random Forest Model performed and why it was not ideal for this task. The rest of this report will delve into how the Random Forest was built and how the data was preprocessed.

The Random Forest model performed poorly on the dataset. The model was unable to learn the complex relationships between the images and the classes.

1.1 Random Forest Results and Analysis

The Random Forest model was trained and tested on the seg_train and seg_test data. The model's performance was as follows:

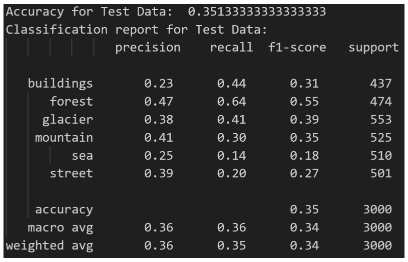


```
Accuracy for Train Data:  0.8036006546644845
Classification report for Train Data:
              precision    recall  f1-score   support

   buildings       0.82      0.80      0.81      2191
      forest       0.89      0.89      0.89      2271
     glacier       0.79      0.78      0.78      1107
    mountain       0.70      0.75      0.73       612
         sea       0.64      0.66      0.65       563
      street       0.69      0.70      0.69       588

    accuracy                           0.80      7332
   macro avg       0.76      0.76      0.76      7332
weighted avg       0.80      0.80      0.80      7332
```

Figure 1: Random Forest - Training Results



```
Accuracy for Test Data:  0.3513333333333333
Classification report for Test Data:
              precision    recall  f1-score   support

   buildings       0.23      0.44      0.31       437
      forest       0.47      0.64      0.55       474
     glacier       0.38      0.41      0.39       553
    mountain       0.41      0.30      0.35       525
         sea       0.25      0.14      0.18       510
      street       0.39      0.20      0.27       501

    accuracy                           0.35      3000
   macro avg       0.36      0.36      0.34      3000
weighted avg       0.36      0.35      0.34      3000
```

Figure 2: Random Forest - Testing Results

Based on these results, the model became extremely overfit. The model's training accuracy was nearly double the testing accuracy (80% and 35% respectively). Furthermore, the model was overfit because the F1-score, precision, and recall scores were all balanced and similar to each other. However, when the testing data was used, those scores were all over the place and heavily biased in favor of the forest class. This imbalance just underscores that the model was trained and was overfit. Running MAE, MSE, R2 on this model doesn't make sense because this is a classification task.

Also, the model not being able to discern when the sea was present is very concerning. The other classes may have confused the model and made it think that light, or dark, blue referred sky and not the sea. Below are some example outcomes of the test predictions:

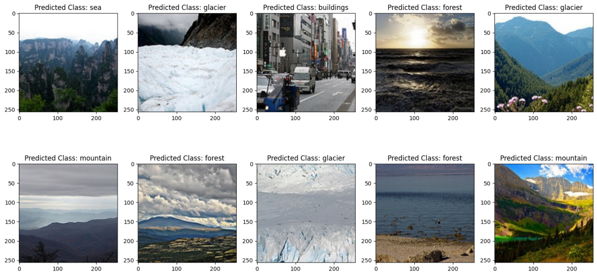


Figure 3: Random Forest - Test Predictions

This picture ties the numerical outcomes together. So many pictures are misclassified as forest. This idea is inline with the previous analysis that the model was biased towards classifying predictions as the forest class. The sea class definitely took the most impact from this bias. For a dataset that is supposed to be natural scenery and have a variety of pictures of different angles and areas, the model should be able to discern what class represents what class. The Random Forest was just not able to do that.

1.2 Overall

Thus, the inability of the Random Forest model to learn relationships and not process the information properly to classify images correctly is why I will be using a CNN model to classify the images. The model should perform and learn the relationships better than the Random Forest model because the pixels aren't flattened and the model reads the images in its totality.

2. Exploratory Data Analysis

The dataset contained roughly 24,000 images of natural scenery. However, I only used ~12,000 images for the scope of this project. I develop code on a virtual server used by other students. When copying over the files, I was unable to copy all the images. To copy the images I used the `scp` command. Nevertheless, ~12,000 images made for a solid sample set. The specific breakdowns of the train, test, and prediction sets are as follows:

- Train: 7335 images
- Test: 3000 images
- Prediction: 1785 images

After copying the data, I ran an `eda.py` file to visualize the classes and the images.

The first step was to visualize the train and test class distributions.

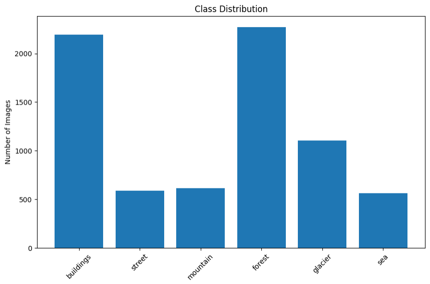


Figure 4: Train Class Distribution

This train class was imbalanced. The street, mountain, and sea classes had significantly fewer images than the other classes. The buildings and forest classes had the most images. This imbalance could lead to biased models. However, I chose not to address the imbalance issues because models need to be trained on a variety of sources. This imbalance could occur in the real world, albeit for a different situation, and the model must handle it and learn relationships.

The test class distribution is as follows:

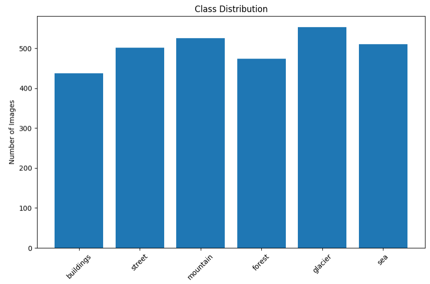


Figure 5: Test Class Distribution

The test class was not as imbalanced as the train class. The classes were more evenly distributed. While some classes had more than others, the difference was not large enough such that the model wouldn't have sufficient data to predict unseen data of a wide variety.

Next, I visualized the images of the train and test sets.

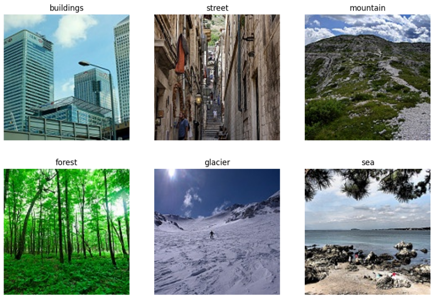


Figure 6: Train Images

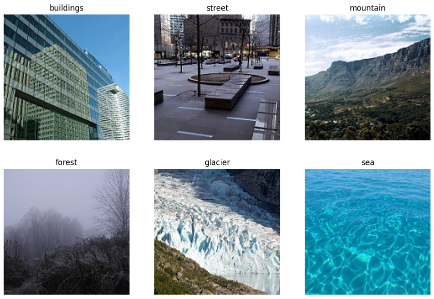


Figure 7: Test Images

The images were of high quality and had similar sizes. Based on these sample images, the model should be able to understand where everything is located and not worry about areas being darker and unrecognizable.

The final step I took was to analyze the image sizes.

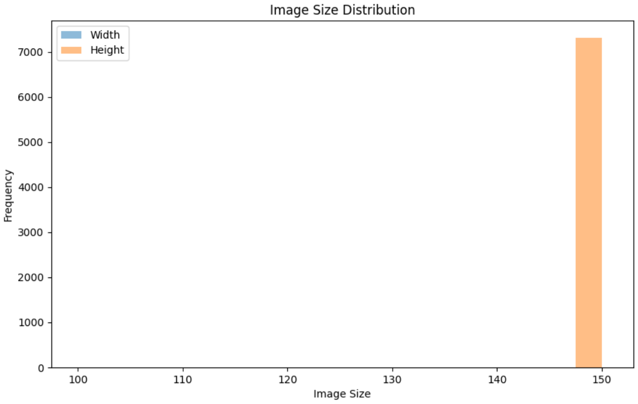


Figure 8: Train Image Sizes

Both, the train and test had the same image size distribution; thus, I am only including the Train distribution. All the images were 150x150. While this is a good size to train with, I augmented the images to a 256x256 size. When scaling an image to a larger size, the pictures can become more unclear and tougher to analyze. This is done purposefully to ensure that the model is trained on grainer data and can handle unclear images.

Finally, for the prediction data, the data is completely unlabeled and there are no class distributions. The images are of the same size as the train and test images. From this analysis of the prediction data, I will only use this to predict the model and not for other analysis. The goal of this dataset will be to see how the CNN model and Random Forest model perform on unseen, unlabeled data.

Overall, the images are high quality and are a good size. While, there are imbalances within the train data, I will be keeping this imbalance to see how the models will handle it. The dataset is perfect for the scope of this project. If I didn't have issues with the copying, I envision some of the imbalances may have been slightly allieviated. Nevertheless, I'm content with with the dataset. After all the exploratory data analaysis, the next step is to preprocess the data and proceed with the model building.

3. Methods

This section will be split it into two subsections. The first will delve into data preprocessing and the second will delve into model building. This section aims to detail my steps to ensure I'm creating the best model possible and to be consistent between the two models.

3.1 Data Preprocessing

On a high level, both models will resize the image to 256x256; however, there are slight differences within each one due to different operations available within the models.

3.1.1 Random Forest Model

The Random Forest model's data preprocessing will be as follows:

1. Resize the image to 256x256
2. Flatten the image to a single row
3. Store the class label and flattened image into an array
4. Convert into np.array and split into Train and Test sets
5. Pickle the data to reduce randomness, leakage, and better protect the data

The Random Forest model is a simple model that cannot take in multi-dimensional data. Thus, the image must be flattened into a single row. This results in every pixel being treated as an independent feature. While, this may not be ideal for image classification, it is a great way to provide a baseline that can be compared to the CNN model and inform us if the CNN model is worth using for your task.

During the preprocessing the label and image are stored into separate arrays that are then converted to an np.array. This is step enables the model to read the data and understand the class labels. Pickling the data is done to reduce the number of times the images are preprocessed; since we aren't using batches of images to be preprocessed, the preprocessing might take a long time. Pickling the data will ensure that the data is preprocessed once and can be used multiple times.

3.1.2 CNN Model

The CNN model's data preprocessing will be as follows:

1. Use ImageDataGenerator to augment the images (rotation and flip were applied to the train and validation sets)
2. Resize the image to 256x256
3. Batch size is set to 64
4. Images are preprocessed using the preprocess_input function from the VGG16 model

While the above 4 steps showcase the general steps there are more tweaks done within the preprocessing steps. The way the data is provided there are three directories (seg_train, seg_test, and seg_pred). seg_train and seg_test are meant to be used for model training. The seg_pred data is meant to be unlabeled data that is used to test how well the model predicts unseen data (i.e there is no class label; therefore, the we cannot derive specific metrics). I want to ensure that I can gather the model's metrics. Thus, I took the seg_train data and ran a train test split to create a new validation set. Thus, the model will be trained on this new validation set and the remaining seg_train data. This leaves the seg_test set, which is labeled and unseen by the model, to assess the model's performance and gather metrics.

For the CNN model the information isn't pickled because the ImageDataGenerator enables us to batch the images and preprocess them faster.

3.2 Model Building

3.2.1 Random Forest Model

The Random Forest model was built using the scikit library. Random Forests are a simple model that can be used for classification tasks and depend on multiple decision trees to determine the best class given the input data. The model was built using 10 estimators (i.e. 10 decision trees). The model was fitted with the seg_train data and model performance was assessed with the seg_test data. I also used the model to predict the seg_pred data to see how well it will do on those picturesand produce some outputs that will be analyzed in the results section.

3.2.2 CNN Model

The CNN model was built using the VGG16 model. VGG16 is a pre-trained model that was trained on the ImageNet dataset. The model was built using the Keras library. The layers of the model were frozen and I added a few layers to the model. The layers were as follows:

```
# Add custom layers
x = Flatten()(base_model.output)
x = Dense(1024, activation="relu")(x)
x = Dropout(0.5)(x)
x = BatchNormalization()(x)
x = Dense(512, activation="relu")(x)
x = BatchNormalization()(x)
x = Dense(256, activation="relu")(x)
x = BatchNormalization()(x)
output = Dense(num_classes, activation="softmax")(x)
```

The dense layers are activated using the ReLu function and the output layer is activated using the softmax function. Relu is used to ensure that the model can learn complex relationships and the softmax function is used to ensure that the model can output a probability distribution. The softmax activation is ideal for multi-class classification tasks because the probability distribution can be used to determine the best class (highest probability = best class).

There are 3 dense layers to ensure the model can learn about the complex relationships in the data. The dropout layer is added to prevent overfitting and ensure the model doesn't learn weird features that are specific to the training set. The batch normalization layers are added to ensure the outputs are normalized and smoothed out. This better improves the relationships learned and reduce overfitting.

The model was compiled using the Adam optimizer and the categorical crossentropy loss function. The model was trained using the new train and validation data that was derived from the seg_test dataset. The model was trained for 10 epochs and the batch size was set to 64. Early Stopping was also used to prevent overfitting. The patience was set to 3 (i.e if the model's validation loss did not improve for 3 epochs, the model would stop training). The model's performance was assessed using the seg_test data. The model was also used to predict the seg_pred data to see how well it will do on those pictures and produce some outputs that will be analyzed in the results section.

Once the model is done training, the model is saved to be used for future predictions.

4. Results

5. Discussion

6. Works Cited