Deep Learning Project 3 - Classifying Images of Outdoor Scenes

Daniel Carter & Jack Schwanewede

Abstract

This paper explores how Deep Learning techniques and models can classify an image from the following six classes, which consist of buildings, mountains, glaciers, seas, streets, and forests. We train two different models with different architectures. First, we test a handbuilt and customized Convolutional Neural Network using the Machine Learning library pyTorch, and second we tune a pre-trained model ResNet-18 to see how well the models can classify images. We use an image dataset from Intel, clean and prepare the data, and assess our models using Confusion Matrices and F1 Scores/Accuracy percentages. The goal of our models is to be able to accurately classify each image into one of the six categories. We find that the ResNet-18 model performs better than our pyTorch model, with F1 scores of 0.91 and 0.84, respectively. We that Convolutional Neural Networks can be effective in classifying different outdoor scenes, yet there can be further improvements on our results that can be implemented in future instances of similar models for classification tasks.

Introduction

Convolutional Neural Networks (CNNs) have become one of the most powerful tools for solving complex image classification problems in modern Machine Learning and Deep Learning settings. Image classification plays a critical role in fields ranging from facial-recognition software to self-driving vehicles, and the ability to accurately recognize and categorize images is increasingly important as data grows in scale and complexity (Voulodimos et al., 2018). In this project, we implement a CNN-based model to classify images into six different categories: buildings, forest, glacier, mountain, sea, and street, using the provided dataset intended for a multi-class classification problem.

To achieve this, we train and evaluate both our hand-built model and our pretrained transfer learning model. For our hand-built model, we used the code and techniques from the Machine Learning Library pyTorch, building upon our knowledge from class activities and previous projects. For our transfer learning model, we selected a pretrained model called ResNet-18, which is known for its relatively small size, fast training speed, and strong performance on image classification tasks. We test our model's performance using a Cross-Entropy loss function and compute an F1 score and accuracy percentage for each model. Our findings provide insights into the effectiveness of multiclass classification models in discriminating between different types of outdoor scenes.

Background

Data Exploration

To begin our data exploration, we first examined the provided images and their source. These images come from Intel, but beyond that the original sources are unknown and there is no additional information that is provided. The dataset that we were provided with was already split into a training and testing set, where we had $14,034$ examples in the training set and $3,000$ examples in the testing set. Both sets had six different sub-folders for each class, labeled as the following:

- Buildings
- Forest
- Glacier
- Mountain
- Sea
- Street

There were no corrupted image files or missing data in the provided dataset, and the distribution of images among the six different classes is roughly uniform, as we can see in Figure 1, which shows the distribution of the images within our training set.

Pre-Processing

After exploring the dataset, we took additional measures to pre-process the data and prepare it for our models. Our first step in pre-processing the data was checking for missing or bad images. Since no images were corrupted or missing, we created a validation set by splitting the provided training set. We split 20% of the training set to create our validation set, in order to evaluate the performance of our models throughout the process by testing our models on unseen data. After ensuring we had sufficient sizes in our test, validation and training dataset, we continued our

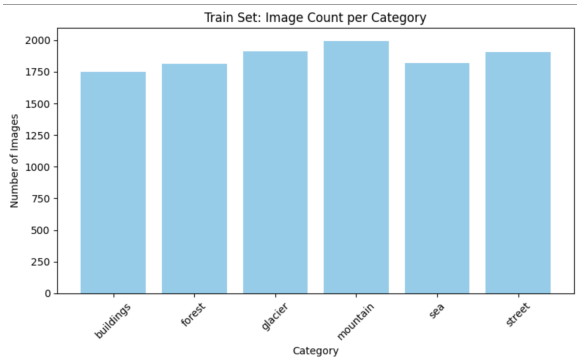

Figure 1: Distribution of Images in Training Set

pre-processing by checking the dimensions of the input images. We wrote a short Python script that analyzed the pixel dimensions of each image, and we found that $16,979$ (99.7%) of the examples in our dataset were 150 by 150 pixels. The remaining 55 values had various different smaller dimensions. We decided to remove these images from the data, as it would be easier to just remove them from our dataset rather than try to resize them. 55 images only account for 0.3% of the total dataset and should not impact our results, especially since their distribution was relatively balanced across classes. The final splits for our datasets are $11,185$ examples in our training set, $2,993$ examples in our testing set, and $2,801$ examples in our validation set. Lastly, another factor we were considering in our pre-processing was whether to convert the images to grayscale or keep them in their standard Red-Green-Blue (RGB) color scale. We opted to keep them in color to assist model performance, where color could help classification, and because our pre-trained model uses color.

Models

In this section, we will give an overview on the methods we used to create and train our models. Our first model was a handbuilt neural network utilizing Convolutional Neural Networks with the pyTorch library. Our second model was built upon a pre-trained model, ResNet-18, which

Handbuilt pyTorch Model

For our handbuilt model, we used the pyTorch library for machine learning, which allows us to create instances of Convolutional Neural Networks (CNNs) and customize and tune it to our task, classifying images from natural scenes. Once again, our goal is to train a model that can accurately take in an image and predict whether it is a building, forest, glacier, mountain, sea, or street.

To start, we loaded in the dataset and created the dataloaders for our test, train and validation sets, with a batch size of 64.

Our model has three 2D convolutional layers, each using a 3 by 3 filter and kernel size and a padding of 1 to maintain the dimensions of the images. Our first layer takes in the three channels from the RGB input of the image and has an output of 32 channels. We then apply a ReLu activation function, and then use 2x2 Max Pooling to reduce the dimensionality and complexity. In this Max Pooling, the image dimensions would be reduced in half by taking the maximum value of each 2 by 2 kernel. Our second layer follows a similar pattern, taking the 32 channels and increasing to 64, before applying ReLu and 2x2 Max Pooling.

Our last convolutional layer increases channels from 64 to 128, applies the ReLu activation function, but uses a different kind of pooling. We use Adaptive Avgerage Pooling to reduce the image dimensionality down to 2 by 2. This pooling specifies a final output and averages values in each region. In this case, our image dimensionality decreases from 150x150 to 75x75 after our first Max Pooling, then to 37x37 in our second Max Pooling, and down to 2x2 from Adaptive Average Pooling. Lastly, after each convolutional layer we applied batch normalization in order to scale and standardize our outputs, limiting the impact of outliers and speeding up the model.

After our three convolutional layers, we flattened the tensor in order to input our data into two hidden layers. Our first layer took in the 128 channels from the third convolutional layer, multiplied by the 2x2 dimensions of each image, and applied a linear transformation. This layer was also normalized using 1D Batch Normalization, and had ReLu applied. Our last hidden layer reduced the 128 channels down to 6, for our 6 natural scene classes. We did not need to apply a final Softmax layer, since we defined our loss function as Categorical Cross Entropy, which handles loss calculation and the Softmax probabilities. For this model, we also used the Adam optimizer using Stochastic Gradient Descent with a learning rate of 0.001.

$$CE = -\sum_{i=1}^{i=N} y\_true_i \cdot log(y\_pred_i)$$

$$CE = -\sum_{i=1}^{i=N} y_i \cdot log(\widehat{y}_i)$$

Figure 2: Equation for Cross Entropy Loss Calculation

ResNet-18 Transfer Learning Model

For our transfer learning model we selected a pretrained ResNet-18 model, which is known for its relatively small size, fast training speed, and strong performance on image classification tasks. ResNet-18 uses residual connections to improve training efficiency and was originally trained on the large-scale ImageNet dataset, making it ideal for adapting to our 6-class problem. It has excellent performance in recognizing image features from millions of training examples, with an error rate of just 3.57% (He et al., 2016).

We will be using the pre-trained version of ResNet-18, so that it will retain the original weights it had from its train-

ing on the ImageNet data. This will assist in reducing the time it takes to train and ideally improve performance due to exposure to previous training.

ResNet-18 Hyperparameters

When working with the ResNet-18 CNN model, we realized it takes in a 3-channel RGB image that is 224x224 pixels. As mentioned before, our cleaned dataset consists of images that are only 150x150 pixels. In order to use this model, we had to stretch the images to 224x224 pixels using the transform package from torch. This may slightly distort the images, but since 150x150 and 224x224 have the same 1:1 square ratio, we believe it is safe to assume that the image quality should not be negatively impacted. We also used the mean and standard deviation from the ImageNet dataset to normalize our three color channels, as the ResNet-18 model was already pre-trained on this dataset. The ResNet-18 model consists of 18 layers, mainly composed of four residual block groups. The model is relatively complex for a neural network, as it uses batch normalization, average pooling, and ReLU activation functions in between many of the layers. This background is important to understanding the functionality of the model, but since we are only using it for pretrained transfer learning, we will not go further in depth about the architecture and the finer details.

The table below depicts the tunable hyperparameters for the ResNet-18 model. The first parameter, learning rate, controls the threshold and speed in which the optimizer adjusts the weights in the model. We used the values of 0.001 and 0.005 for this hyperparameter. The next hyperparameter we have is batch size, which we fixed at a value of 32 for this model. This is because we wanted to update the weights fairly often in relation to the size of our training set, in hopes of creating an accurate multi-class image classification model. The next hyperparameter we have is the number of epochs, which we also plan to keep at a constant value of 20. We have also implemented an early stopping condition, where the model would stop early if the training loss does not improve over one epoch. This would ideally prevent overfitting and reduce unnecessary training times. ResNet-18 uses the Adam optimizer, a common optimizer used in our previous class models and in our pyTorch model as well.

HyperParameter	Default Value(s)
Learning Rate	0.001, 0.005
Batch Size	32
Number of Epochs	20
Optimizer	Adam
Data Augmentation	Stretched images to 224x224 pixels
Frozen Layers	All except for output

Table 1: ResNet-18 Hyperparameters

Lastly, we froze all of the layers in this neural network except for the output layer to adapt it to our six-class problem. We did this to keep the pre-trained weights, transferring the learning from the ImageNet dataset to our task of classifying outdoor scenes, using the powerful feature extraction capabilities of ResNet-18.

Results

To test the performance of each model, we used a combination of accuracy percentage, F1-score, and a graph of a Confusion Matrix, which depicts a distribution of class predictions versus the true classes. Accurate predictions are seen along the diagonal, and each grid is color coded based on the frequency of the prediction. Since this is a multi-class classification problem, we will mainly focus on the Macro F1 score, which treats the performance of each class equally.

Handbuilt pyTorch Model

First, our pyTorch model had decent results, with a macro F1 score of 0.84 and an accuracy of 84.2% with our best training examples. The figure below depicts our metrics with different hyperparameters, such as the amount of convolutional layers we applied, the batch size, and the epochs trained for.

CNN Layers	Batch Size	Accuracy	F1 Score	# Epochs
2	32	78%	0.78	5
3	64	84.2%	0.8406	15

Table 2: pyTorch Model Performance Metrics for Different Hyperparameters

The architecture described in the methods section above was our better performing model. Adding another convolutional layer, as well as altering the batch size and pooling type (implementing Adaptive Average Pooling for the final layer), improved our performance, and this effect improved as we let it run for more epochs. Ideally, we would have had more epochs than 15, but due to the computationally complex nature of the classification task and the architecture, running even 15 epochs on our systems took as long as 1.5-2 hours.

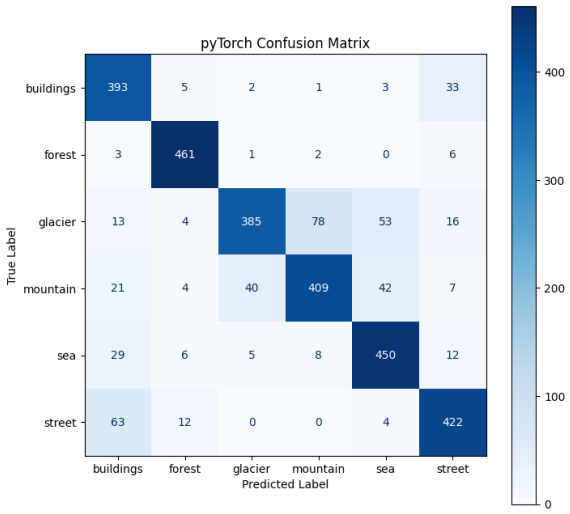

Figure 3: Confusion Matrix for the pyTorch Model

In the Confusion Matrix, we see that the model was mostly accurate, especially in predicting forest images. For forests, the pyTorch model had an F1 score of .96 and was extremely accurate. This could have to do with the color, as the forest class has a distinctively greener input than the other classes in comparison. This could also explain why the model had a harder time distinguishing between mountains and glaciers, as some of the mountain images had snow or ice that could have been confused with the color or features of glaciers. This is also seen in their class F1 scores, with glaciers having a score of 0.78, and mountains having a score of 0.80. There were also some incorrect predictions between streets and buildings, as some of the street images may have had buildings in the background and vice versa.

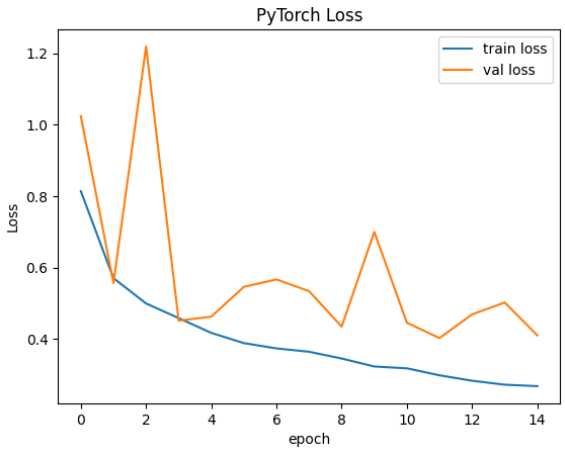

Figure 4: Loss Curve over Epoch for Validation/Train in pyTorch Model

Finally, above we have our loss curve that depicts the change in loss (calculated using Categorical Cross Entropy). This curve is limited by the number of epochs, but we can generally see that the train loss decreases towards zero, and the validation loss has a similar trend. Since the batches for validation may have been inconsistent, it is more volatile and increases in some epochs, but gernerally is trending downwards.

ResNet-18 Transfer Learning Model

For the ResNet-18 transfer learning model, we measured the performance of across two different learning rates, 0.001 and 0.005. These results are depicted below in Table 3.

Learning Rate	Accuracy	F1 Score	# Epochs
0.001	91.2%	0.914	12
0.005	90.1%	0.902	9

Table 3: ResNet-18 Model Performance Metrics for Different Learning Rates

These results indicate that the ResNet-18 model performed well on the test data set with both learning rates, as the 0.001 learning rate model had an accuracy of 91.2% and an average macro F1-score of 0.914, finishing in 12 epochs. The model that was trained with a 0.005 learning rate achieved an accuracy of 90.1% and an average macro F1-score of 0.902, and finished in 9 epochs. The model may have performed slightly better with the learning rate of 0.001 because a smaller learning rate allows the final layer to adapt slowly, which can help the model match the features extracted from the rest of the frozen layers.

The confusion matrix below visualizes how our model classified each class of images and where the most common mistakes were. Overall, our model performed well on all classes, but it took a slight hit in performance with distinguishing between mountains and glaciers as well as streets and buildings. Again, this could be because of the similarites between mountains and glaciers have similar features or values on the RGB scale, even though color may not the issue here. We have the same issue as the pyTorch model, as some of the images in the buildings class likely contain streets in them, and vice versa. This may be confusing our model as there are two classes of images within a single image, even though we only have a single class output, and therefore it could end up being classified into the wrong class.

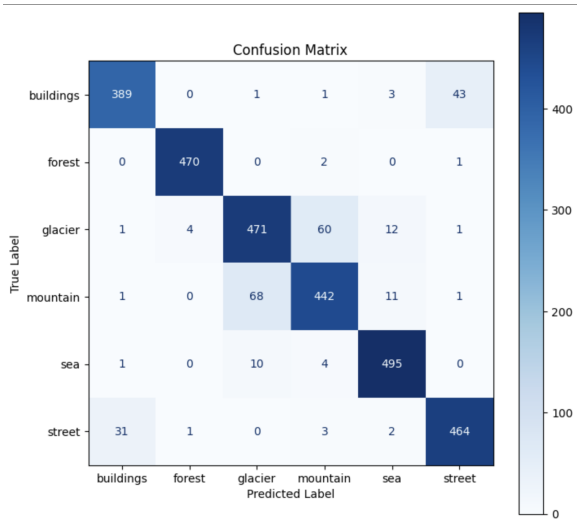

Figure 5: Confusion Matrix from ResNet-18 model trained with 0.001 learning rate

Ethical Considerations/Limitations

Multi-class image classification models have been discussed heavily in recent years due to their applications in several fields, and they can be helpful in improving the state of many people's lives. Real world applications can range from facial recognition software to diagnosing certain medical conditions, such as classifying between benign and malignant tumors (Street et al., 1993), and this technology has been developing even more rapidly lately. However, these advantages do not come without any drawbacks. Image classification models that have been trained on a dataset where certain classes are underrepresented can

produce discriminatory results applicable to that problem (Peng et al., 2021). As a result, there can be gender or racial discrimination in facial recognition software, or medical models can result in misdiagnosing a critical medical condition.

This also raises the question of when these models should be deployed in the real world if they are not equally accurate for all groups. We would recommend against deploying a biased model, and it is critical that we address these shortcomings before these models could lead to systematic harm.

However, in terms of our project and the dataset we used, we believe that there are little to no ethical concerns. There may be issues with privacy and confidentiality of the locations used in the Intel dataset, and whether there was consent in using pictures of private buildings or streets. Intel's lack of transparency about where the data is sourced from may be a concern, as there is no way to validate whether they obtained proper consent to use certain images. Also, as a limitation to the dataset, there is a possibility that images are geographically biased, where some regions are more prominently featured than others. This mean that certain classes may be unrepresentative, such as buildings, as there may be differences in the way buildings look in different regions. By definition, some of our classes may be geographically biased, such as in the glaciers class, but it would be ideal to have known more information on the original source for the images.

Conclusion

In this paper, we created two instances of CNNs: one handbuilt model using pyTorch, and a pretrained model using ResNet-18. Our goal was to create models that could classify between six different classes of outdoor scenes: buildings, forests, glaciers, mountains, sea, and streets. We found that the ResNet-18 model outperformed our pyTorch model in accuracy (91.2% vs 84.2%), and macro F1 score (0.91 vs 0.84). Both models excelled in predicting forests, but had trouble discriminating between mountains and glaciers, and between buildings and streets. Still, our results were strong overall, showing how CNNs can be effective in classifying images.

We also conclude that researchers creating multi-class image classification models should think ethically about how their models might be implemented in the real world. Datasets should be as representative as possible to be as generalizable to other populations, regions, etc., and also to mitigate harm.

Acknowledgments

ChatGPT was used as a tool to assist in graph making and high-level error code research in compliance with the class policies agreed upon at the onset of this project. Comment lines in our code indicate where and what code was ChatGPT generated/assisted and to what extent. We would like to thank Dr. Kuchera for her assistance in CSC 374, as what she has taught us has helped us create and run projects like this. We would also like to thank Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun for their paper on Deep Residual Learning for Image Recoginition, which included the ResNet-18 model- this paper was a huge help with getting started in the process.

References

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

Peng, C., Liu, Y., Zhang, X., Kang, Z., Chen, Y., Chen, C., & Cheng, Q. (2021). Learning discriminative representation for image classification. Knowledge-based systems, 233, 107517.

Street, W. N., Wolberg, W. H., & Mangasarian, O. L. (1993, July). Nuclear feature extraction for breast tumor diagnosis. In Biomedical image processing and biomedical visualization (Vol. 1905, pp. 861-870). SPIE.

Voulodimos, A., Doulamis, N., Doulamis, A., & Protopapadakis, E. (2018). Deep learning for computer vision: A brief review. Computational intelligence and neuroscience, 2018(1), 7068349.