Intel Image Classification Using Neural Network

Duc, N., Lan, N., Nam, L. & Tuan, V.[1]
[1]College of Engineering and Computer Science and College of Health Science

973 Words Student ID:V202000210
Keywords: Image classification, Neural network 18th June, 2021

Abstract

In the 21^{st} century where the application of Image Classification can be seen everywhere from the face recognition feature on iPhone, object detection in autonomous cars, to medical diagnosis, it is essential to optimize this process. Image Classification is an application of Machine Learning in solving Computer Vision problems. In this project, we aim to solve an Intel Image Classification challenge on Kaggle by optimizing an already existing Convolutional Neural Network model to classify images into different sceneries. In the end, we can classify approximately 25,000 into 6 labels using *resnet50* with the accuracy of 92.6%.

1 Introduction

We use a manual process to classify images every day to navigate our lives in this world. However, in industrialized sectors, manual classification is not efficient enough to satisfy the need of its industry. For instance, in medical diagnosis, it would be revolutionary to have a machine that reads X-ray images and predict the interpretation of that X-ray. Image Classification in Computer Vision generates to do such tasks. By definition, Image Classification is the process of assigning labels to images according to their types. With this feature of Machine Learning, we can classify a massive number of images of *n* potential categories conveniently and effectively. The process includes defining a set of objects to identify in images, train a model to recognize them using labeled example photos, and test the model for the best accuracy.

Before doing such tasks for real-world application, we start by building a model for a simple problem of classifying sceneries. We use the datasets published by Intel for an Image Classification challenge with approximately 25,000 images and attempt to group it in 6 different categories: *buildings*, *forest*, *glacier*, *mountain*, *sea*, and *street*.

2 Method

The broad idea of our method is based on the concepts of *Convolutional Neural Network* (CNN) which is the most popular deep learning algorithm for image data [1]. The structure of the model is bulit based on the PyTorch library with the support from online PyTorch tutorial [2]

2.1 Loading and Processing Data

In our first attempt of engineering a solution, our team faced inconsistent and long run times while designing an algorithm to read through all the available test images. With the assistance from the TAs, we were then informed to use the Torch Library from Python to load and process data, avoiding the computationally complex nested loops ($\approx N^2$) of our initial approach. Additionally, the Torch library also provided us with useful functions, such as resizing our existing training set to 150x150 pixels so that all the training pictures are consistent in size.

2.2 Designing a Model

Instead of designing a convolutional neural network, which requires the technical background that our team doesn't possess, we chose to adopt and implement an existing built-in architecture called *resnet50* from "Deep residual Learning for Image Recognition" [3]. *Resnet50* has already formulated 50 layers with a detail setup for the kernel size, padding and activation functions. We chose this over other models, like the *resnet18*, due to *resnet50*'s sheer ease of implementation and accuracy. Additionally, we used the pretrained model from ImageNet to ensure the quality performance overall.

The only task in designing a model is to adjust the hyper-parameters, for example: learning rates, weight decay, and optimizer. For optimizer, we choose *stochastic gradient descent* (SGD) as it is the most used optimization method as well as we are most familiar with that. We set the learning rate to be 0.1 and 0.001 respectively, and the performance of the model in 0.001 tended to be much better. For regularization, at first, we did not set the weight decay and the model demonstrate a bad performance on the validation set. We recognized that this problem was overfitting, and has choose the value for weight decay to be *6e-4*. With all of these setup, the result of the model in predicting unseen images is promising.

2.3 Implementation details

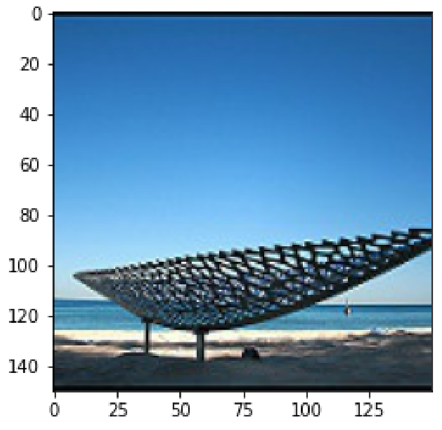
All the processing code has been deposited on the link: *http://bitly.ws/e4Fr*

3 Result

We successfully implemented *resnet50* in the effort to classify images into the categories defined in our code [Figure 2]. After a run time of around 129 minutes, we have achieved the accuracy of about 92.6 % on the test set [Figure 3], outperforming other methods in terms of the accuracy metric. Furthermore, we also calculated the accuracy and loss of each prediction computation so as to see where our model can be improved upon [Figure 4 and 5].

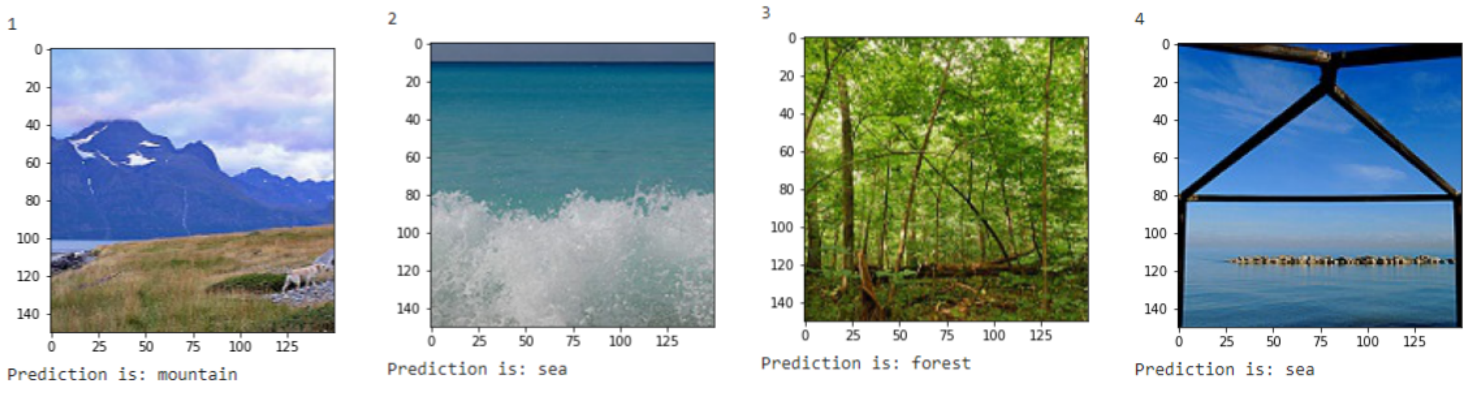
Although our model is relatively accurate, we did however find some areas of improvements. For example, in Figure 1, the image shows a building in the foreground, but the sea in the background. Since our network only puts sort images into one category, our model predicted the image to be of a building. In the future, when we gain enough expertise to develop our own convolutional neural network, we hope to be able to extend our prediction to different objects in an image by identifying different features of the image.

```
5
Label:  sea (4)
```



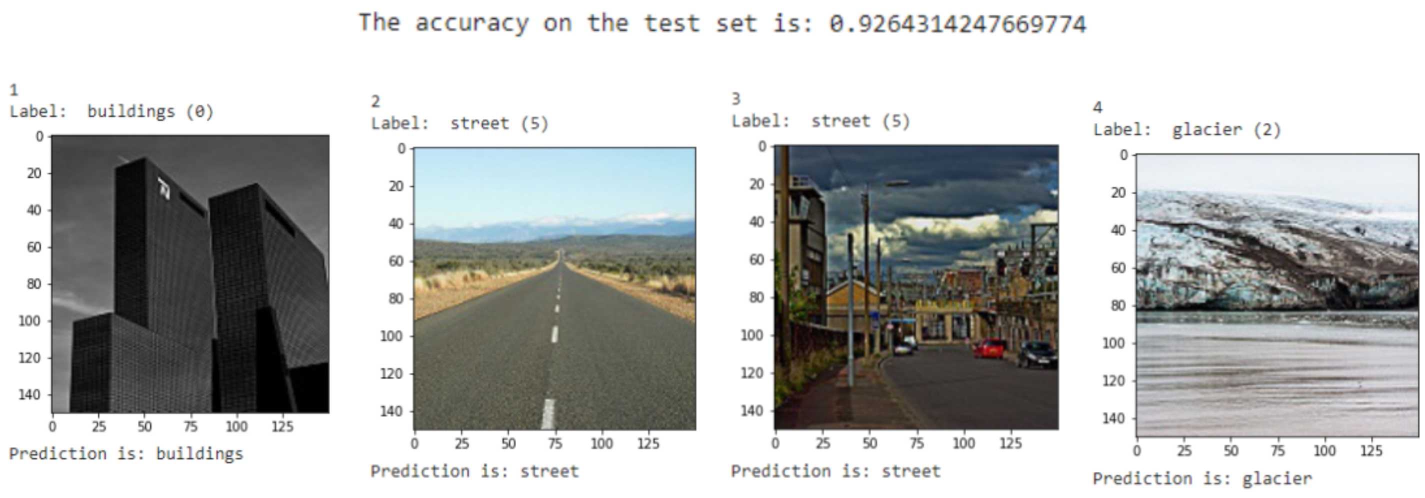
```
Prediction is: buildings
```

Fig. 1. Limited Classification



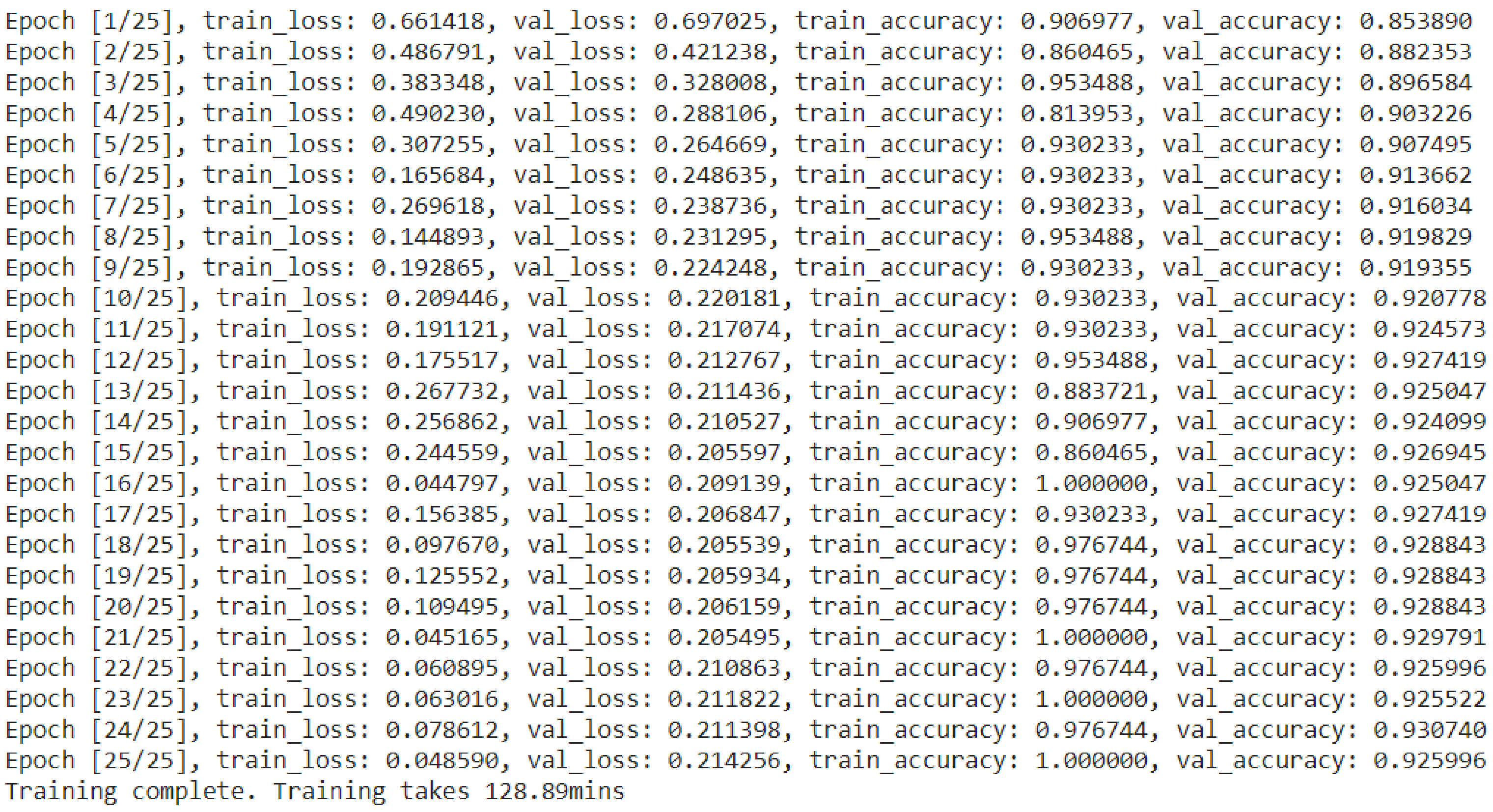
```
1
Prediction is: mountain

2
Prediction is: sea

3
Prediction is: forest

4
Prediction is: sea
```

Fig. 2. The result on unseen images



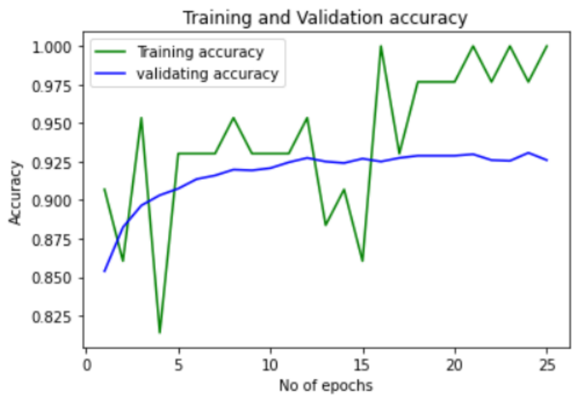
```
The accuracy on the test set is: 0.9264314247669774

1
Label:  buildings (0)
Prediction is: buildings

2
Label:  street (5)
Prediction is: street

3
Label:  street (5)
Prediction is: street

4
Label:  glacier (2)
Prediction is: glacier
```

Fig. 3. Checking the accuracy on test set



```
Epoch [1/25], train_loss: 0.661418, val_loss: 0.697025, train_accuracy: 0.906977, val_accuracy: 0.853890
Epoch [2/25], train_loss: 0.486791, val_loss: 0.421238, train_accuracy: 0.860465, val_accuracy: 0.882353
Epoch [3/25], train_loss: 0.383348, val_loss: 0.328008, train_accuracy: 0.953488, val_accuracy: 0.896584
Epoch [4/25], train_loss: 0.490230, val_loss: 0.288106, train_accuracy: 0.813953, val_accuracy: 0.903226
Epoch [5/25], train_loss: 0.307255, val_loss: 0.264669, train_accuracy: 0.930233, val_accuracy: 0.907495
Epoch [6/25], train_loss: 0.165684, val_loss: 0.248635, train_accuracy: 0.930233, val_accuracy: 0.913662
Epoch [7/25], train_loss: 0.269618, val_loss: 0.238736, train_accuracy: 0.930233, val_accuracy: 0.916034
Epoch [8/25], train_loss: 0.144893, val_loss: 0.231295, train_accuracy: 0.953488, val_accuracy: 0.919829
Epoch [9/25], train_loss: 0.192865, val_loss: 0.224248, train_accuracy: 0.930233, val_accuracy: 0.919355
Epoch [10/25], train_loss: 0.209446, val_loss: 0.220181, train_accuracy: 0.930233, val_accuracy: 0.920778
Epoch [11/25], train_loss: 0.191121, val_loss: 0.217074, train_accuracy: 0.930233, val_accuracy: 0.924573
Epoch [12/25], train_loss: 0.175517, val_loss: 0.212767, train_accuracy: 0.953488, val_accuracy: 0.927419
Epoch [13/25], train_loss: 0.267732, val_loss: 0.211436, train_accuracy: 0.883721, val_accuracy: 0.925047
Epoch [14/25], train_loss: 0.256862, val_loss: 0.210527, train_accuracy: 0.906977, val_accuracy: 0.924099
Epoch [15/25], train_loss: 0.244559, val_loss: 0.205597, train_accuracy: 0.860465, val_accuracy: 0.926945
Epoch [16/25], train_loss: 0.044797, val_loss: 0.209139, train_accuracy: 1.000000, val_accuracy: 0.925047
Epoch [17/25], train_loss: 0.156385, val_loss: 0.206847, train_accuracy: 0.930233, val_accuracy: 0.927419
Epoch [18/25], train_loss: 0.097670, val_loss: 0.205539, train_accuracy: 0.976744, val_accuracy: 0.928843
Epoch [19/25], train_loss: 0.125552, val_loss: 0.205934, train_accuracy: 0.976744, val_accuracy: 0.928843
Epoch [20/25], train_loss: 0.109495, val_loss: 0.206159, train_accuracy: 0.976744, val_accuracy: 0.928843
Epoch [21/25], train_loss: 0.045165, val_loss: 0.205495, train_accuracy: 1.000000, val_accuracy: 0.929791
Epoch [22/25], train_loss: 0.060895, val_loss: 0.210863, train_accuracy: 0.976744, val_accuracy: 0.925996
Epoch [23/25], train_loss: 0.063016, val_loss: 0.211822, train_accuracy: 1.000000, val_accuracy: 0.925522
Epoch [24/25], train_loss: 0.078612, val_loss: 0.211398, train_accuracy: 0.976744, val_accuracy: 0.930740
Epoch [25/25], train_loss: 0.048590, val_loss: 0.214256, train_accuracy: 1.000000, val_accuracy: 0.925996
Training complete. Training takes 128.89mins
```

Fig. 4. Learning process



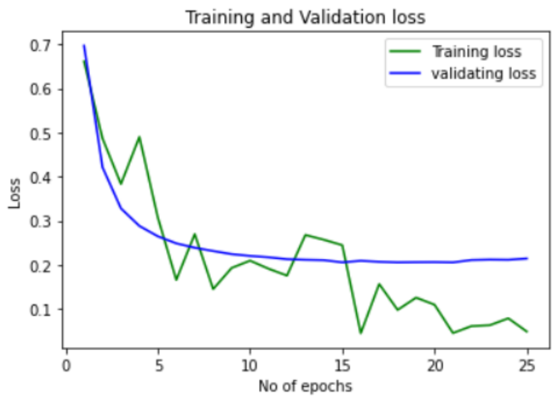


Fig. 5. Loss and Accuracy plot

4 Discussion

The results presented on the test set, including the randomly selected training set and validation set, and the prediction set indicate the right approach to simple image classification problems. The results also display the comparison between the accuracy when using the *resnet50* architecture with using other methods on Kaggle such as implementing the library Keras. However, we have addressed several areas of improvement to optimize our algorithm. Firstly, it is beneficial to either modify or enrich our data by using more comprehensive datasets from different data points or data augmentation. Moreover, we have proposed regularization and the creation of an original neural network to enhance the accuracy of the model.

Overall, the generic nature of the proposed method is a promising beginning for further applications. We plan to conduct further in-depth research to develop this into models for more complex image classifications (for instance, object detection in autonomous cars or X-ray image classification).

5 Conclusion

In summary, we have classified approximately 25000 Intel images of 6 labels using *resnet50* architecture - one of the most common models in convolutional neural network in specific and machine learning in general. According to the results, the suggested model significantly outperforms the existing algorithms on the main page of Kaggle. Therefore, many applications can use our model to implement and may receive better results.

6 Acknowledgement

We would like to thank Professor Minh Do, Doctor Hieu Pham and TA Hieu Hoang for the intriguing project. We hope to work and collaborate with you in many machine learning as well as artificial intelligence projects in the future.

7 References

[1] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*, 2017, pp. 1–6.

[2] *Finetuning Torchvision Models — PyTorch Tutorials 1.2.0 documentation*.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.