ITNPAI1 PROJECT REPORT – FOREST/CITY CLASSIFICATION

https://github.com/surajjeoor/ITNPAI_Project_Forestcityclassifier

Suraj Jeoor	Samuel Sarpong
3033368	3089568
Pune, India	Accra, Ghana
sjeoor@gmail.com	sarpongs@ymail.com

INTRODUCTION

This project was to build a computer vision model that classifies a set of images into the appropriate classes (forests and streets). We collected data, divided it into training and test sets, trained our data with various models and parameters, and then tested the models on the test data.

The data for the model was collected from three cities: Accra (Ghana), Pune (India), and Stirling (Scotland).
For training data, 200 images were taken from Accra and Pune together (100 of forests and 100 of streets), and for test data, 200 images of the same content were taken from Stirling.

DESCRIPTION OF PROPOSED SOLUTIONS

We started by building a VGG neural network model using PyTorch [1]. We completed the data loader and applied standard preprocessing techniques. We performed some augmentation techniques, such as resizing, normalising, random flipping, etc., on the training and test images.
We implemented the model so that each layer consisted of VGG blocks, and each block contained lazy convolutional layers in 2D, under which we provided a number of channels. We set our kernel size to 3 for the convolutional layers and appended the layers with Relu. We built it by following the tutorial [2]. After following through with the model, we were not satisfied with the outcome because training accuracy was within the normal range but testing accuracy was relatively low, overfitting the model (notebook: ITNPAI1_VGGNet_.ipynb).

Furthermore, we used a pre-trained model, Yolo (You Only Look Once) [3]. We just had to load our data into train, validate, and test, then set the parameters, after which we left the model to run since Yolo trains, validates, and tests simultaneously. This enabled us to take advantage of the deep learning features of Yolo to analyse and process our data efficiently (notebook: ITNPAI1_YOLO.ipynb) [4].

We proceeded with building our model from the ground up, following a tutorial [5], and exploring the TensorFlow and Keras libraries. We began by loading the training data, followed by the test data. We then performed some augmentation techniques, such as resizing and normalising the training images, and then we set our classes (forests and streets). We then converted the images to be represented in NumPy arrays and viewed them as graphs, labelling each image with its appropriate class (notebook: ITNPAI1: Personal Model.ipynb).

The next step in building the model was to introduce convolutional layers with parameters such as the size of the images and activation type. We set the model's optimizer and loss function to determine the accuracy. We then split the data for the model to train, validate, and test. We set the model to run for 10 epochs. The overall accuracy of the model improved after the run of each epoch [6].

RESULTS

The results for each model differ based on parameters, procedures, and the complexity of the models involved. Here are some visuals of the results. More information is provided in the description of the results.

Yolo

For the Yolo model, the figure below represents the training of the data, where images have been loaded and labelled to allow the machine to recognise streets from forests.

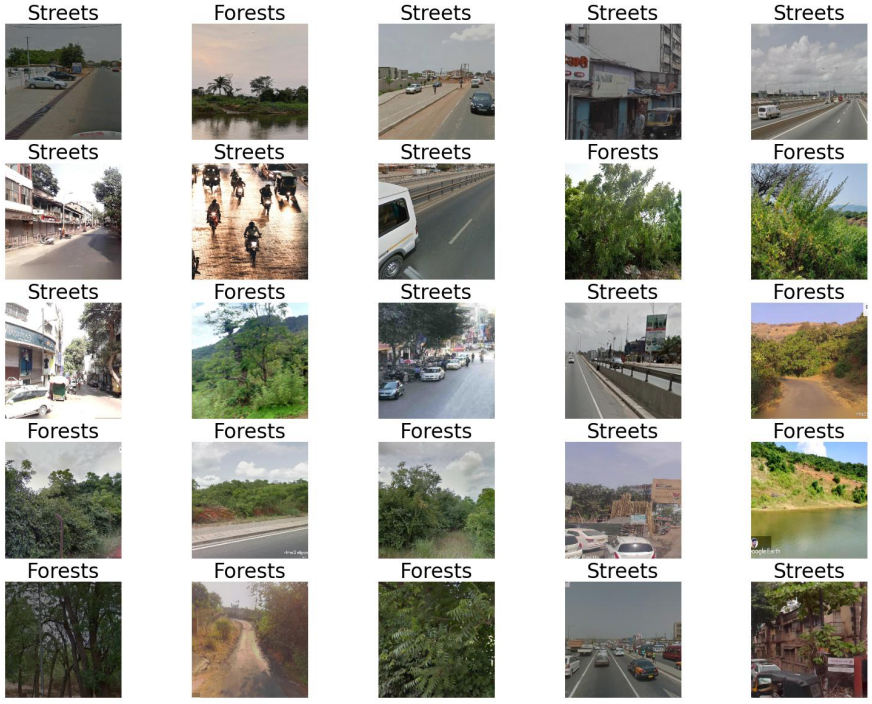

Figure 1: Training model to recognise and classify given images

This picture demonstrates the predictive accuracy of the model. In these situations, we compare the prediction to the actual outcome. It is accurate to say that the model foretells the presence of a forest or street in an image. 24 of the 25 randomly generated images that the machine predicted were accurate.

Figure 2: Predictive accuracy of the model

Here are images representing the training and testing losses of the Yolo model as well as their accuracy. You can see that the loss decreases after each epoch, which is positive.
Similarly, you can tell that the training accuracy remains perfect while the test accuracy increases with each epoch.

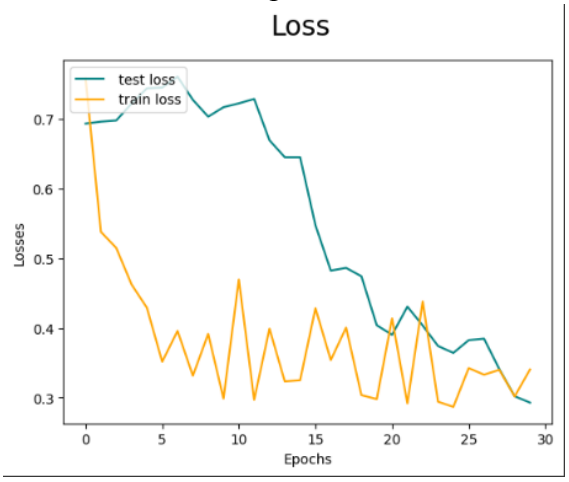

Figure 3: Training and Testing Loss

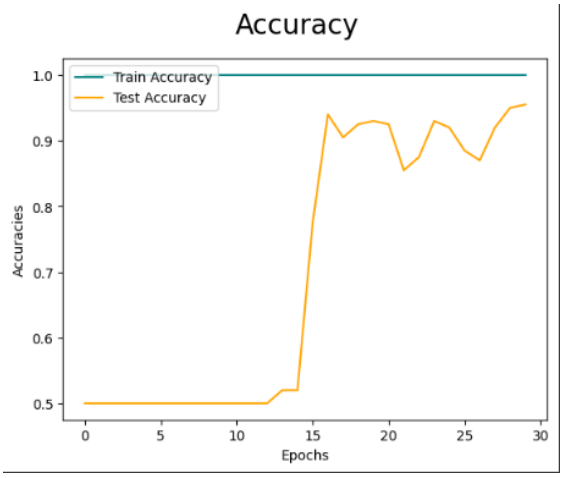

Figure 4: Training and Testing Accuracy

Personal Model

These images represent the training and validation accuracy as well as the training and validation loss. We run the model for 10 epochs. Both the training and validation accuracy increased after each epoch. The losses also decreased after each epoch.

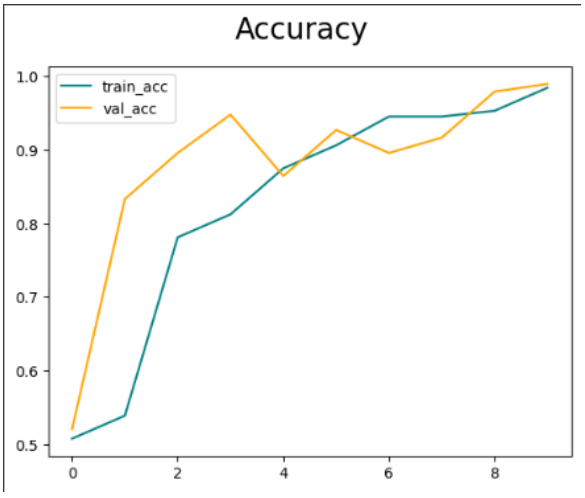

Figure 5: Training and Validation Losses

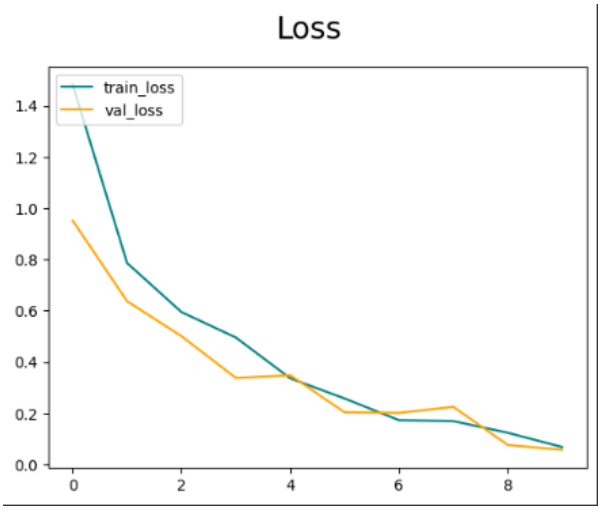

Figure 6: Training and Validation Accuracy

DESCRIPTION OF RESULTS

VGG Model

after designing the VGG model architecture with Pytorch, choosing appropriate hyperparameters, and preprocessing the data. Despite our best efforts, we ran into a significant problem: the model's accuracy was consistently low, falling short of our expectations and requirements. The optimal train and test values were only between 1% and 5%, with a huge amount of training loss.

```
Epoch:0 Train Loss :tensor(0.0283) Train Accuracy:0.5 Test Accuracy:0.015451511306454531
Epoch:1 Train Loss :tensor(0.0286) Train Accuracy:0.485 Test Accuracy:0.02045226129712586
Epoch:2 Train Loss :tensor(0.0283) Train Accuracy:0.435 Test Accuracy:0.035426632547126566
Epoch:3 Train Loss :tensor(0.0284) Train Accuracy:0.5 Test Accuracy:0.025452135047110466
Epoch:4 Train Loss :tensor(0.0284) Train Accuracy:0.5 Test Accuracy:0.015377386306469764
Epoch:5 Train Loss :tensor(0.0284) Train Accuracy:0.515 Test Accuracy:0.020476635040860936
Epoch:6 Train Loss :tensor(0.0283) Train Accuracy:0.49 Test Accuracy:0.020502383790829453
Epoch:7 Train Loss :tensor(0.0282) Train Accuracy:0.5 Test Accuracy:0.0203275094439075
Epoch:8 Train Loss :tensor(0.0278) Train Accuracy:0.5 Test Accuracy:0.030426760665891877
Epoch:9 Train Loss :tensor(0.0276) Train Accuracy:0.5 Test Accuracy:0.030351883806516877
Epoch:10 Train Loss :tensor(0.0278) Train Accuracy:0.5 Test Accuracy:0.0004022606752979687
Epoch:11 Train Loss :tensor(0.0283) Train Accuracy:0.5 Test Accuracy:0.0255270087877357
Epoch:12 Train Loss :tensor(0.0297) Train Accuracy:0.5 Test Accuracy:0.030451758190813753
Epoch:13 Train Loss :tensor(0.0282) Train Accuracy:0.505 Test Accuracy:0.020476882547173125
Epoch:14 Train Loss :tensor(0.0283) Train Accuracy:0.485 Test Accuracy:0.01537713380648578
Epoch:15 Train Loss :tensor(0.0279) Train Accuracy:0.49 Test Accuracy:0.030501886925172892
Epoch:16 Train Loss :tensor(0.0283) Train Accuracy:0.5 Test Accuracy:0.015402511906516796
Epoch:17 Train Loss :tensor(0.0279) Train Accuracy:0.5 Test Accuracy:0.025452260040829455
Epoch:18 Train Loss :tensor(0.0283) Train Accuracy:0.5 Test Accuracy:0.025302010681532344
Epoch:19 Train Loss :tensor(0.0283) Train Accuracy:0.5 Test Accuracy:0.015302133184735391
```

Figure 7: Run Summary

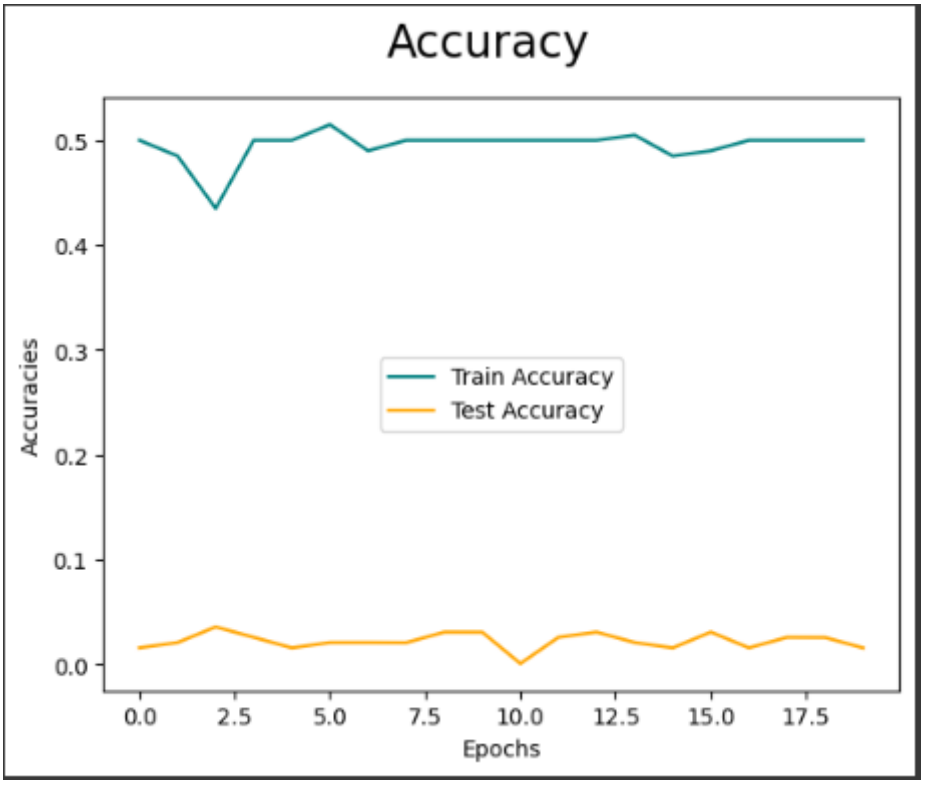

Figure 8: Graph of Training and Testing Accuracy

Yolo

Throughout the training, validation, and testing processes of Yolo, we gained valuable insights into how Yolo perceives data, and we observed that each model has its own unique way of interpreting and understanding the input data. We observed that Yolo has its own learned representations of objects, scenes, and features as it was trained using our dataset. These representations were based on the training data. Yolo could detect and label objects with remarkable accuracy and speed, and it could accurately classify objects in real time. The values for training, testing, and validation ranged between 94% and 98% with very little loss.

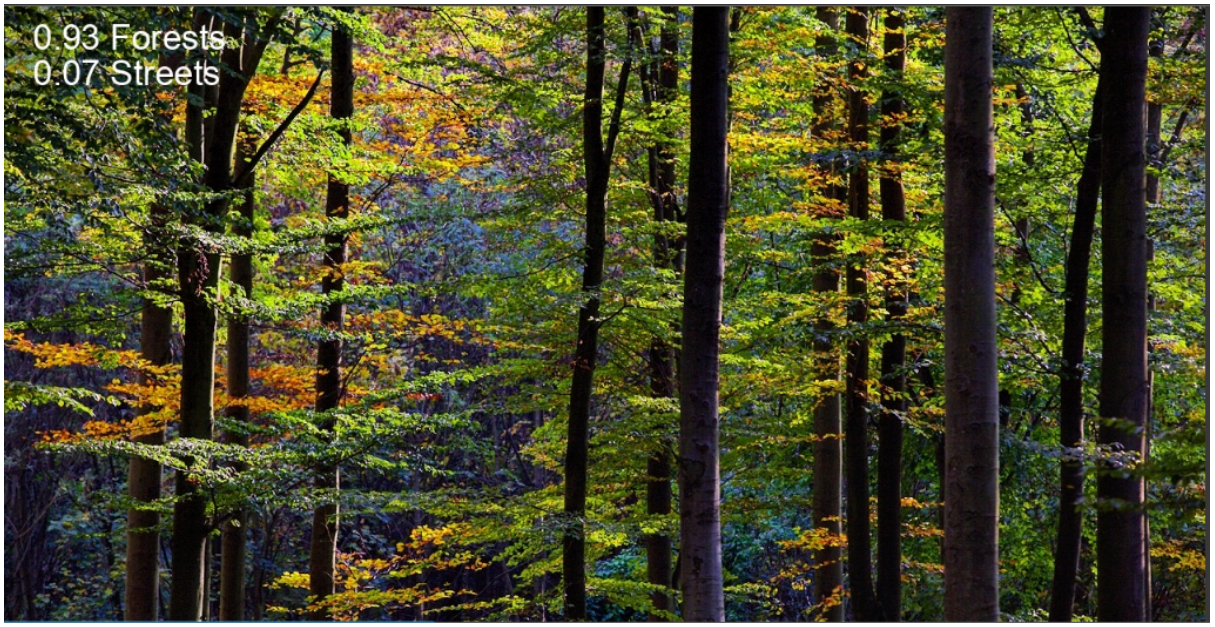

Figure 9: Prediction with Additional Image

Personal Model

Just like the Yolo model, our model did very well in terms of training accuracy, validation, and test accuracy. The model was able to efficiently classify city or street images from forest images. After several runs, the training accuracy and the testing accuracy were between 92% and 98%, and the training loss was very minimal.

```
Loss =  0.06468462198972702  Accuracy =  0.9800000190734863
```

Figure 10: Testing Accuracy and Loss

```
Precision: 1.0, Recall: 0.9599999785423279, Test Accuracy: 98.00000190734863%
```

Figure 11: Precision and Recall

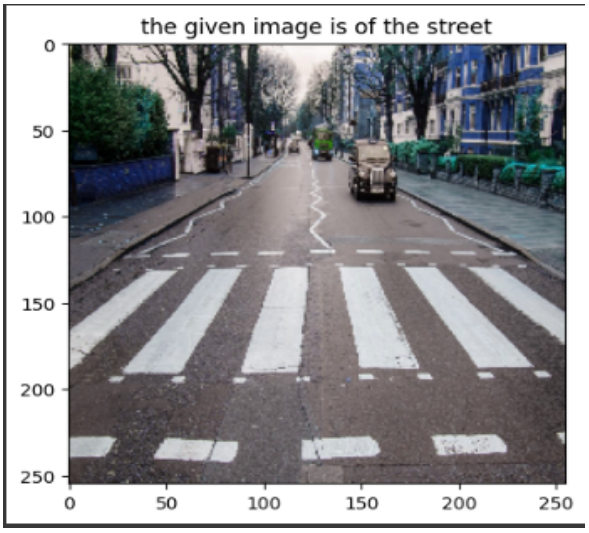

Figure 12: Prediction with Additional Image

CONCLUSION

We have been able to experiment with and explore different computer vision models to determine the best ones that can classify given data. In conclusion, we successfully created our unique model from scratch using the TensorFlow and Keras libraries. Our model now accurately recognises and distinguishes between these two categories after being trained with 200 images. The personal model is quite robust and efficient, as its accuracy depicts.

REFERENCES

[1]. Networks Using Blocks (VGG)—Dive into Deep Learning 1.0.0 beta0 Documentation," *d2l.ai*. [Online]. Available: https://d2l.ai/chapter_convolutional-modern/vgg.html [Accessed: Apr. 10, 2023]

[2]. Gaurav67890, "gaurav67890/Pytorch_Tutorials," *GitHub*, April 6, 2023 [online]. Available: https://github.com/gaurav67890/Pytorch_Tutorials/blob/master/cnn-scratch-training.ipynb [Accessed: Apr. 10, 2023]

[3]. O. IQ, "YOLO v5 model architecture [explained]," *OpenGenus IQ: Computing Expertise and Legacy*, October 28, 2022. [Online]. Available: https://iq.opengenus.org/yolov5/#:~:text=YOLOv5%20uses%20the%20same%20head. [Accessed: Apr. 11, 2023]

[4]. P. Guerrie, T. L. Aug. 19, and Read 2022: 8 M., "How to Train YOLOv5-Classification on a Custom Dataset," *Roboflow Blog*, Aug. 19, 2022. [Online]. Available: https://blog.roboflow.com/train-yolov5-classification-custom-data/

[5]. N. Renotte, "Build a Deep CNN Image Classifier with Any Images," *www.youtube.com*, March 25, 2022. [Online]. Available: https://www.youtube.com/watch?v=jztwpsIzEGc&t=3180s&ab_channel=NicholasRenotte [Accessed: April 10, 2023]

[6]. N. Renotte, "nicknochnack/ImageClassification," *GitHub*, Aug. 29, 2022. [Online]. Available: https://github.com/nicknochnack/ImageClassification [Accessed: Apr. 10, 2023]