Computer Vision Assignment 6

Dirk Bester

October 28, 2024

1 Deep learning packages

1.1 Task Description

We have to choose a deep learning package and explain the reason behind our choice.

1.2 Method and Discussion

For this assignment I decided to use Keras. Keras is a high-level deep learning package that runs on top of Tensorflow. Tensorflow is an end-to-end platform for machine learning. Keras was initially released in 2015 by Francois Chollet and has now grown into one of the most popular libraries for building deep learning models due to its simplicity. Keras is widely used in acedemia and in the industry because it abstracts alot of the complexity. Keras also supports a large variety of neural network architectures including convolution neural networks (CNNs).

I chose Keras over PyTorch as it is much easier to use and much easier to create simple deep learning models. Keras also offer a wide range of pre-trained models and support transfer learning. Keras is also good with stuff like integrating data augmentation which helped us overcome the limitations of our small dataset.

Getting Keras to run in my environment was quite easy. We first had to create a virtual environment and then run (pip install tensorflow). After that we ran code to see if our GPU was detected and being used by Keras. We wanted to use the GPU to speed things up. After that we also had to re-organiuze our file structure as Keras expects it in a specific structure to use (flowfromdirectory). We also had to import the neccesary libraries such as tensorflow.keras, numpy, matplotlib, and seaborn.

2 Training a CNN from scratch

2.1 Task Description

We have to start by resizing our scene images and then we have to set up a CNN with three convolution stages (each consisting of a convolutional layer, batch normalisation, ReLU, and max pooling), followed by a fully-connected layer with 9 outputs, and softmax activation. After that we have to reduce over-fitting using certain remedies.

2.2 Method and Discussion

We start this task by making use of the Image Data Generator which resizes all the images for us. We do the resizing by just defining the target size in our training and testing generators.

Now before we start building the model we have to restructure our current images file structure. We do this because the flow_from_directory method expects the files to be in a certain structure. The images are organized into two main directories: `train` and `validation`. Within each of these directories, the images are further organized into subdirectories representing each class. For example:

- `images/train/Coast`
- `images/train/Mountain`
- `images/validation/Coast`
- `images/validation/Mountain`

This structure allows us to easily load the training and validation datasets using Keras.

Now that we have the correct file structure we define two data generators and we input all our images into them as previously described with the flow_from_directory method by defining the root folder. We make sure to make the class mode categorical as we are working with 9 distinct categories. The categorical mode also converts the labels into one-hot encoded vectors.

Now it is time to start building the model. To start I thought it was a good idea to first explore other models and which of them works best for what kind of datasets. I found this very nice website that made very good suggestions for small datasets. I decided to build by model based of VGG-16 which is very good with small datasets. We thus started by using a sequential type. Sequential allows us to stack layers in linear order.

We built ther model using three convolution stages, all with ReLu activation, padding and max pooling. ReLu activation introduces non-linearity into the model and ensures it is fast to compute. Padding ensures that the output dimensions after convolution. The Polling reduces the network complexity and reduces noise.

After the convolutional and pooling layers, the output is flattened into a 1D vector to prepare it for the fully connected layers. The final layer has 9 neurons because there are 9 output classes in your dataset. The softmax activation function is used to output probabilities for each class, ensuring that the sum of all outputs equals 1. The class with the highest probability is chosen as the predicted class.

Our original model overfits too much so we must come up with methods that could counteract this. We implement two tricks for this. We started by doing data augmentation. We did a horizontal flip to the training images, which effectively doubles the amount of training images the model can use. This makes the model more robust and open to different variations and basically increases the accuracy.

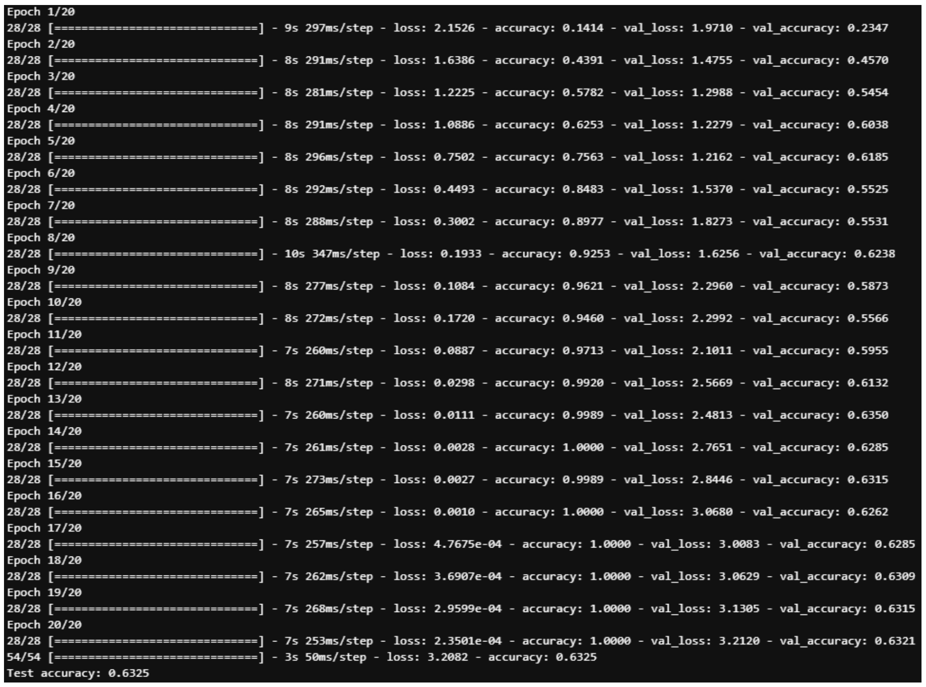
The second method we used is dropout regulation. We do this by adding a dropout layer just before the fully connected layers. It ”drops out” a fraction of the neurons during training, which means that those neurons are ignored during forward and backward propagation. This helps the model to generalize better.

We then proceed to train the model and output the test accuracy. We also then create a confusion matrix and print out the top-1 and top-3 accuracy. Lastly we plot 5 correct and 5 incorrect classifications.

2.3 Results

The accuracy of our VGG-16 inspired model looks as expected. Our average accuracy is around 60-65%. We did not expect better results as we have not done anything to try and decrease over fitting. The batch size of 64 were chosen with a learning rate of 0.0001 as this yielded the best results.

We then started work on our new and improved model with the over-fitting techniques implemented. For the data augmentation we did a horizontal flip and for the dropout layer we decided on a rate of 50% which also gave us the best results.



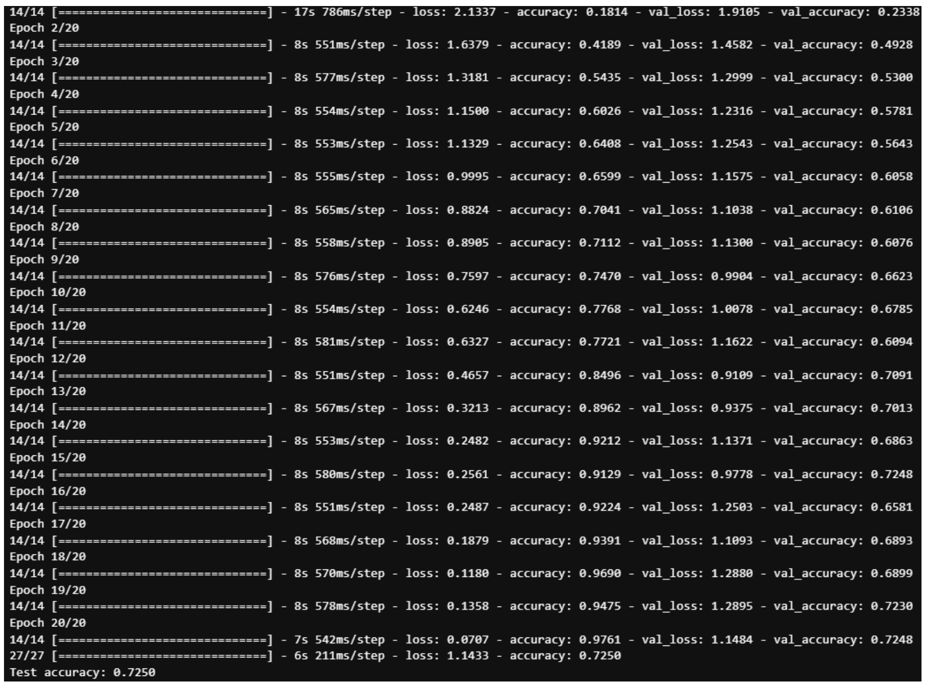
```
Epoch 1/20
28/28 [==============================] - 9s 297ms/step - loss: 2.1526 - accuracy: 0.1414 - val_loss: 1.9710 - val_accuracy: 0.2347
Epoch 2/20
28/28 [==============================] - 8s 291ms/step - loss: 1.6386 - accuracy: 0.4391 - val_loss: 1.4755 - val_accuracy: 0.4570
Epoch 3/20
28/28 [==============================] - 8s 281ms/step - loss: 1.2225 - accuracy: 0.5782 - val_loss: 1.2988 - val_accuracy: 0.5454
Epoch 4/20
28/28 [==============================] - 8s 291ms/step - loss: 1.0886 - accuracy: 0.6253 - val_loss: 1.2279 - val_accuracy: 0.6038
Epoch 5/20
28/28 [==============================] - 8s 296ms/step - loss: 0.7502 - accuracy: 0.7563 - val_loss: 1.2162 - val_accuracy: 0.6185
Epoch 6/20
28/28 [==============================] - 8s 292ms/step - loss: 0.4493 - accuracy: 0.8483 - val_loss: 1.5370 - val_accuracy: 0.5525
Epoch 7/20
28/28 [==============================] - 8s 288ms/step - loss: 0.3002 - accuracy: 0.8977 - val_loss: 1.8273 - val_accuracy: 0.5531
Epoch 8/20
28/28 [==============================] - 10s 347ms/step - loss: 0.1933 - accuracy: 0.9253 - val_loss: 1.6256 - val_accuracy: 0.6238
Epoch 9/20
28/28 [==============================] - 8s 277ms/step - loss: 0.1084 - accuracy: 0.9621 - val_loss: 2.2960 - val_accuracy: 0.5873
Epoch 10/20
28/28 [==============================] - 8s 272ms/step - loss: 0.1720 - accuracy: 0.9460 - val_loss: 2.2992 - val_accuracy: 0.5566
Epoch 11/20
28/28 [==============================] - 7s 260ms/step - loss: 0.0887 - accuracy: 0.9713 - val_loss: 2.1011 - val_accuracy: 0.5955
Epoch 12/20
28/28 [==============================] - 8s 271ms/step - loss: 0.0298 - accuracy: 0.9920 - val_loss: 2.5669 - val_accuracy: 0.6132
Epoch 13/20
28/28 [==============================] - 7s 260ms/step - loss: 0.0111 - accuracy: 0.9989 - val_loss: 2.4813 - val_accuracy: 0.6350
Epoch 14/20
28/28 [==============================] - 7s 261ms/step - loss: 0.0028 - accuracy: 1.0000 - val_loss: 2.7651 - val_accuracy: 0.6285
Epoch 15/20
28/28 [==============================] - 7s 273ms/step - loss: 0.0027 - accuracy: 0.9989 - val_loss: 2.8446 - val_accuracy: 0.6315
Epoch 16/20
28/28 [==============================] - 7s 265ms/step - loss: 0.0010 - accuracy: 1.0000 - val_loss: 3.0680 - val_accuracy: 0.6262
Epoch 17/20
28/28 [==============================] - 7s 257ms/step - loss: 4.7675e-04 - accuracy: 1.0000 - val_loss: 3.0083 - val_accuracy: 0.6285
Epoch 18/20
28/28 [==============================] - 7s 262ms/step - loss: 3.6907e-04 - accuracy: 1.0000 - val_loss: 3.0629 - val_accuracy: 0.6309
Epoch 19/20
28/28 [==============================] - 7s 268ms/step - loss: 2.9599e-04 - accuracy: 1.0000 - val_loss: 3.1305 - val_accuracy: 0.6315
Epoch 20/20
28/28 [==============================] - 7s 253ms/step - loss: 2.3501e-04 - accuracy: 1.0000 - val_loss: 3.2120 - val_accuracy: 0.6321
54/54 [==============================] - 3s 50ms/step - loss: 3.2082 - accuracy: 0.6325
Test accuracy: 0.6325
```

Figure 1: Three-layer model

Our top 1% accuracy is what we expected and then our top 3% accuracy looks very good.

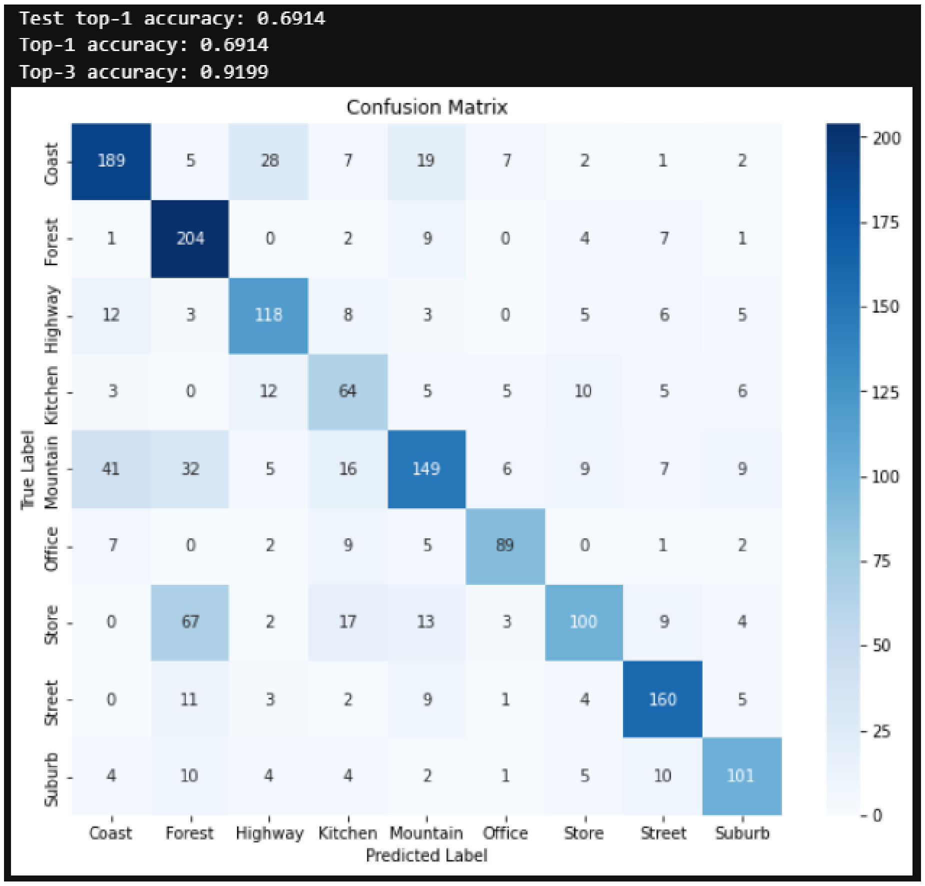
Then we created the confusion matrix. The result looks good as the diagonal is much darker than the other squares. This refers to the correct matches because the labels matches here. Some of the squares are a bit of a worry for example Mountain, Forest and Coast were wrongly classified between each other.

Then we also decided to plot 5 correctly classified scenes and also 5 incorrectly classified scenes. Highway and coast are here twice but this makes sense as highways and coast has both long lines in it and no good way to classify between the two. The 4th incorrect image is ambiguous because you can sort off see an highway but should have still been classified as the coast.



```
14/14 [==============================] - 17s 786ms/step - loss: 2.1337 - accuracy: 0.1814 - val_loss: 1.9105 - val_accuracy: 0.2338
Epoch 2/20
14/14 [==============================] - 8s 551ms/step - loss: 1.6379 - accuracy: 0.4189 - val_loss: 1.4582 - val_accuracy: 0.4928
Epoch 3/20
14/14 [==============================] - 8s 577ms/step - loss: 1.3181 - accuracy: 0.5435 - val_loss: 1.2999 - val_accuracy: 0.5300
Epoch 4/20
14/14 [==============================] - 8s 554ms/step - loss: 1.1500 - accuracy: 0.6026 - val_loss: 1.2316 - val_accuracy: 0.5781
Epoch 5/20
14/14 [==============================] - 8s 553ms/step - loss: 1.1329 - accuracy: 0.6408 - val_loss: 1.2543 - val_accuracy: 0.5643
Epoch 6/20
14/14 [==============================] - 8s 555ms/step - loss: 0.9995 - accuracy: 0.6599 - val_loss: 1.1575 - val_accuracy: 0.6058
Epoch 7/20
14/14 [==============================] - 8s 565ms/step - loss: 0.8824 - accuracy: 0.7041 - val_loss: 1.1038 - val_accuracy: 0.6106
Epoch 8/20
14/14 [==============================] - 8s 558ms/step - loss: 0.8905 - accuracy: 0.7112 - val_loss: 1.1300 - val_accuracy: 0.6076
Epoch 9/20
14/14 [==============================] - 8s 576ms/step - loss: 0.7597 - accuracy: 0.7470 - val_loss: 0.9904 - val_accuracy: 0.6623
Epoch 10/20
14/14 [==============================] - 8s 554ms/step - loss: 0.6246 - accuracy: 0.7768 - val_loss: 1.0078 - val_accuracy: 0.6785
Epoch 11/20
14/14 [==============================] - 8s 581ms/step - loss: 0.6327 - accuracy: 0.7721 - val_loss: 1.1622 - val_accuracy: 0.6094
Epoch 12/20
14/14 [==============================] - 8s 551ms/step - loss: 0.4657 - accuracy: 0.8496 - val_loss: 0.9109 - val_accuracy: 0.7091
Epoch 13/20
14/14 [==============================] - 8s 567ms/step - loss: 0.3213 - accuracy: 0.8962 - val_loss: 0.9375 - val_accuracy: 0.7013
Epoch 14/20
14/14 [==============================] - 8s 553ms/step - loss: 0.2482 - accuracy: 0.9212 - val_loss: 1.1371 - val_accuracy: 0.6863
Epoch 15/20
14/14 [==============================] - 8s 580ms/step - loss: 0.2561 - accuracy: 0.9129 - val_loss: 0.9778 - val_accuracy: 0.7248
Epoch 16/20
14/14 [==============================] - 8s 551ms/step - loss: 0.2487 - accuracy: 0.9224 - val_loss: 1.2503 - val_accuracy: 0.6581
Epoch 17/20
14/14 [==============================] - 8s 568ms/step - loss: 0.1879 - accuracy: 0.9391 - val_loss: 1.1093 - val_accuracy: 0.6893
Epoch 18/20
14/14 [==============================] - 8s 570ms/step - loss: 0.1180 - accuracy: 0.9690 - val_loss: 1.2880 - val_accuracy: 0.6899
Epoch 19/20
14/14 [==============================] - 8s 578ms/step - loss: 0.1358 - accuracy: 0.9475 - val_loss: 1.2895 - val_accuracy: 0.7230
Epoch 20/20
14/14 [==============================] - 7s 542ms/step - loss: 0.0707 - accuracy: 0.9761 - val_loss: 1.1484 - val_accuracy: 0.7248
27/27 [==============================] - 6s 211ms/step - loss: 1.1433 - accuracy: 0.7250
Test accuracy: 0.7250
```

Figure 2: Improved Three-layer model



```
Test top-1 accuracy: 0.6914
Top-1 accuracy: 0.6914
Top-3 accuracy: 0.9199
```

Confusion Matrix

True Label \ Predicted Label	Coast	Forest	Highway	Kitchen	Mountain	Office	Store	Street	Suburb
Coast	189	5	28	7	19	7	2	1	2
Forest	1	204	0	2	9	0	4	7	1
Highway	12	3	118	8	3	0	5	6	5
Kitchen	3	0	12	64	5	5	10	5	6
Mountain	41	32	5	16	149	6	9	7	9
Office	7	0	2	9	5	89	0	1	2
Store	0	67	2	17	13	3	100	9	4
Street	0	11	3	2	9	1	4	160	5
Suburb	4	10	4	4	2	1	5	10	101

Figure 3: Confusion Matrix

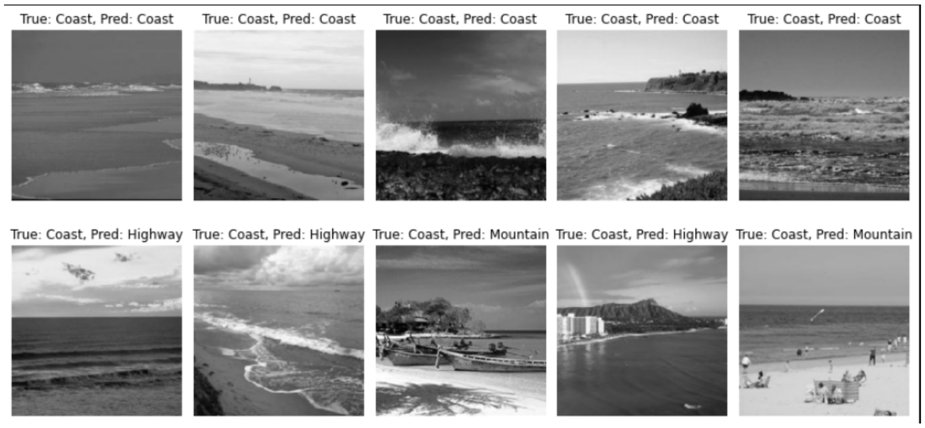

True: Coast, Pred: Coast
True: Coast, Pred: Coast
True: Coast, Pred: Coast
True: Coast, Pred: Coast
True: Coast, Pred: Coast
True: Coast, Pred: Highway
True: Coast, Pred: Highway
True: Coast, Pred: Mountain
True: Coast, Pred: Highway
True: Coast, Pred: Mountain

Figure 4: Classifications

3 Transfer learning

3.1 Task Description

We have to firstly decide on a pre-trained model. We then have to load this model and with similar parameters and no data augmentation. Then we have to visualise and repeat question 2d for this model.

3.2 Method and Discussion

We selected VGG-16 as the pre-trained model for transfer learning due to its simplicity and effectiveness in image classification tasks specifically. Furthermore, VGG16 is widely used and well-documented, making it easier to fine-tune and adapt to new tasks. It has also been said that VGG-16 performs very well on small datasets.

We started exactly the same way as we did in Question 2 by restructuring the file-structure. We then loaded that data in the same way. We kept all the other parameters the same as requested. After that we loaded in the pre-trained model with the imagenet weights.

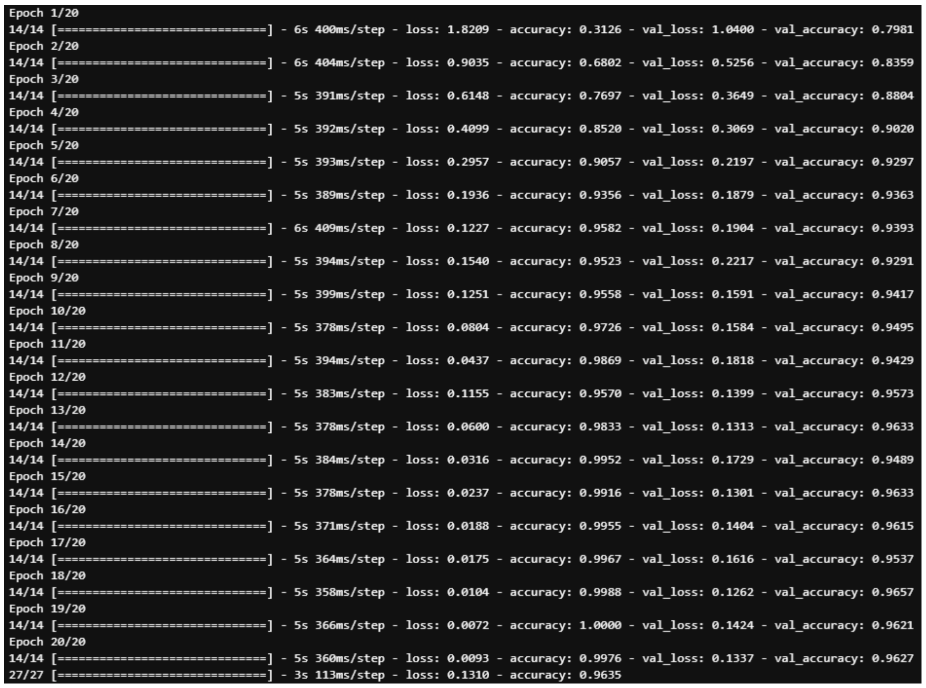
Initially we freeze all the layers of the VGG-16 based model to prevent them from being updated during training, preserving the knowledge from the pre-trained weights. Then we unfreeze the last 4 layers to allow fine-tuning.

Again we create a sequential model. We use global average pooling to reduce the dimentionality. A Dense layer with 512 units and ReLU activation is added to introduce more learnable parameters. A final Dense layer with 9 units (one for each class) and softmax activation is used to output the predicted probabilities for each of the 9 classes.

We then train the model and start visualizing with a confusion matrix and correct and incorrect classifications.

3.3 Results

The accuracy we achieve ends up in between 93% and 96% which is very good and what we expected. As mentioned we kept the parameters the same. The freezing and unfreezing made a very large change in the accuracy.

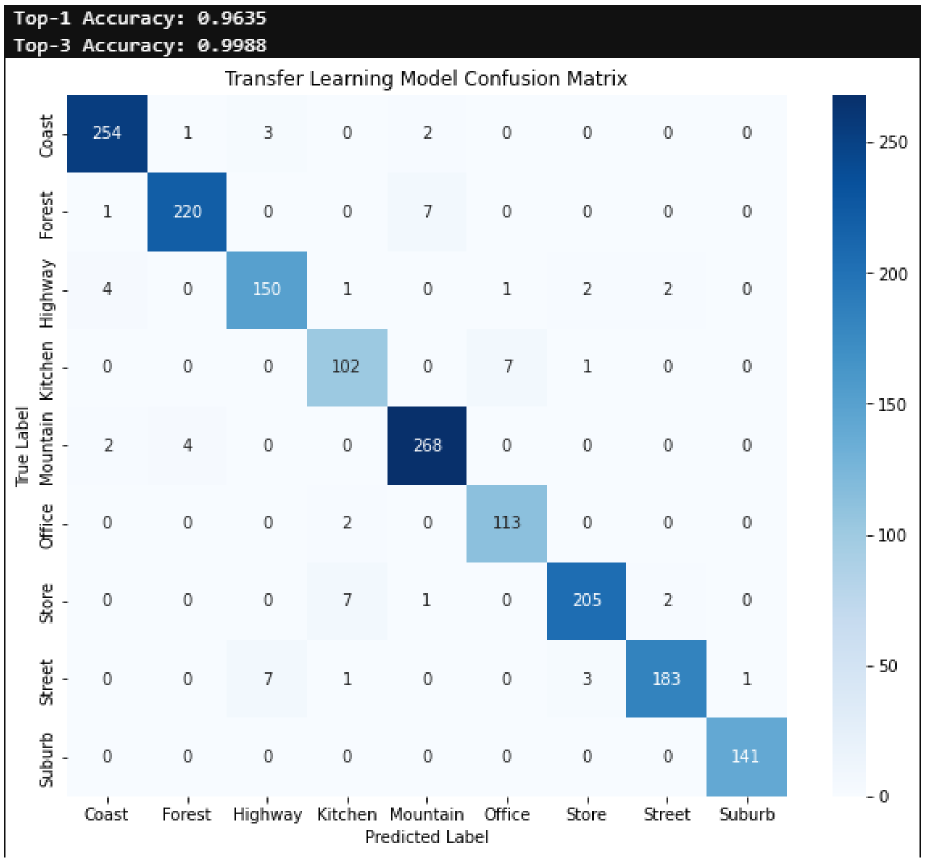


```
Epoch 1/20
14/14 [==============================] - 6s 400ms/step - loss: 1.8209 - accuracy: 0.3126 - val_loss: 1.0400 - val_accuracy: 0.7981
Epoch 2/20
14/14 [==============================] - 6s 404ms/step - loss: 0.9035 - accuracy: 0.6802 - val_loss: 0.5256 - val_accuracy: 0.8359
Epoch 3/20
14/14 [==============================] - 5s 391ms/step - loss: 0.6148 - accuracy: 0.7697 - val_loss: 0.3649 - val_accuracy: 0.8804
Epoch 4/20
14/14 [==============================] - 5s 392ms/step - loss: 0.4099 - accuracy: 0.8520 - val_loss: 0.3069 - val_accuracy: 0.9020
Epoch 5/20
14/14 [==============================] - 5s 393ms/step - loss: 0.2957 - accuracy: 0.9057 - val_loss: 0.2197 - val_accuracy: 0.9297
Epoch 6/20
14/14 [==============================] - 5s 389ms/step - loss: 0.1936 - accuracy: 0.9356 - val_loss: 0.1879 - val_accuracy: 0.9363
Epoch 7/20
14/14 [==============================] - 6s 409ms/step - loss: 0.1227 - accuracy: 0.9582 - val_loss: 0.1904 - val_accuracy: 0.9393
Epoch 8/20
14/14 [==============================] - 5s 394ms/step - loss: 0.1540 - accuracy: 0.9523 - val_loss: 0.2217 - val_accuracy: 0.9291
Epoch 9/20
14/14 [==============================] - 5s 399ms/step - loss: 0.1251 - accuracy: 0.9558 - val_loss: 0.1591 - val_accuracy: 0.9417
Epoch 10/20
14/14 [==============================] - 5s 378ms/step - loss: 0.0804 - accuracy: 0.9726 - val_loss: 0.1584 - val_accuracy: 0.9495
Epoch 11/20
14/14 [==============================] - 5s 394ms/step - loss: 0.0437 - accuracy: 0.9869 - val_loss: 0.1818 - val_accuracy: 0.9429
Epoch 12/20
14/14 [==============================] - 5s 383ms/step - loss: 0.1155 - accuracy: 0.9570 - val_loss: 0.1399 - val_accuracy: 0.9573
Epoch 13/20
14/14 [==============================] - 5s 378ms/step - loss: 0.0600 - accuracy: 0.9833 - val_loss: 0.1313 - val_accuracy: 0.9633
Epoch 14/20
14/14 [==============================] - 5s 384ms/step - loss: 0.0316 - accuracy: 0.9952 - val_loss: 0.1729 - val_accuracy: 0.9489
Epoch 15/20
14/14 [==============================] - 5s 378ms/step - loss: 0.0237 - accuracy: 0.9916 - val_loss: 0.1301 - val_accuracy: 0.9633
Epoch 16/20
14/14 [==============================] - 5s 371ms/step - loss: 0.0188 - accuracy: 0.9955 - val_loss: 0.1404 - val_accuracy: 0.9615
Epoch 17/20
14/14 [==============================] - 5s 364ms/step - loss: 0.0175 - accuracy: 0.9967 - val_loss: 0.1616 - val_accuracy: 0.9537
Epoch 18/20
14/14 [==============================] - 5s 358ms/step - loss: 0.0104 - accuracy: 0.9988 - val_loss: 0.1262 - val_accuracy: 0.9657
Epoch 19/20
14/14 [==============================] - 5s 366ms/step - loss: 0.0072 - accuracy: 1.0000 - val_loss: 0.1424 - val_accuracy: 0.9621
Epoch 20/20
14/14 [==============================] - 5s 360ms/step - loss: 0.0093 - accuracy: 0.9976 - val_loss: 0.1337 - val_accuracy: 0.9627
27/27 [==============================] - 3s 113ms/step - loss: 0.1310 - accuracy: 0.9635
```

Figure 5: Pre-trained Model

The confusion matrix looks very good with a very strong diagonal. It contains minimal wrong classifications. There was not 1 scene that was misclassified more than 7 times.

Again the classification has a few ambiguous scene for example the street and highway image can go either way, it might as well be a highway.



```
Top-1 Accuracy: 0.9635
Top-3 Accuracy: 0.9988
```

Transfer Learning Model Confusion Matrix

True Label \ Predicted Label	Coast	Forest	Highway	Kitchen	Mountain	Office	Store	Street	Suburb
Coast	254	1	3	0	2	0	0	0	0
Forest	1	220	0	0	7	0	0	0	0
Highway	4	0	150	1	0	1	2	2	0
Kitchen	0	0	0	102	0	7	1	0	0
Mountain	2	4	0	0	268	0	0	0	0
Office	0	0	0	2	0	113	0	0	0
Store	0	0	0	7	1	0	205	2	0
Street	0	0	7	1	0	0	3	183	1
Suburb	0	0	0	0	0	0	0	0	141

Figure 6: Confusion Matrix

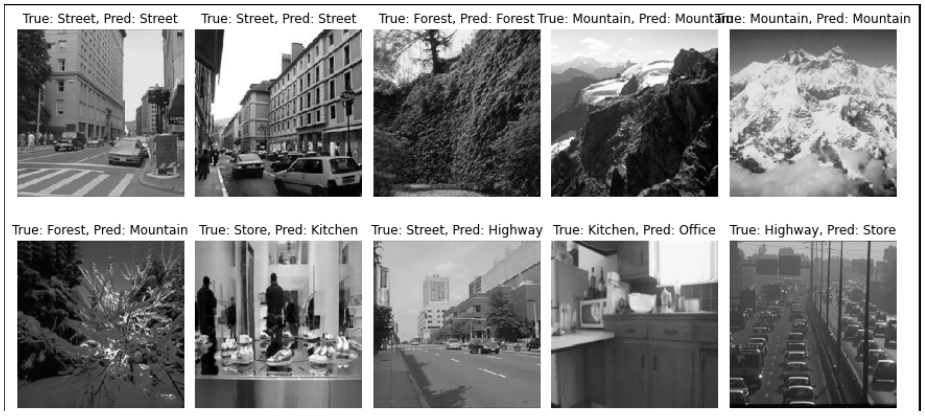
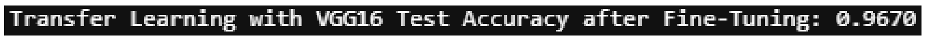

True: Street, Pred: Street
True: Street, Pred: Street
True: Forest, Pred: Forest
True: Mountain, Pred: Mountain
True: Mountain, Pred: Mountain
True: Forest, Pred: Mountain
True: Store, Pred: Kitchen
True: Street, Pred: Highway
True: Kitchen, Pred: Office
True: Highway, Pred: Store

Figure 7: Classifications

4 Further Findings

I was not satisfied with 95% accuracy so I decided to play around and see if I can achieve a even higher accuracy. I got a highest accuracy of 96.70%.



```
Transfer Learning with VGG16 Test Accuracy after Fine-Tuning: 0.9670
```

Figure 8: Best Accuracy

I firstly added an learning rate scheduler which dynamically reduces the learning rate during training when the model hits a plateau to help it fine-tune and converge better. I also added early stopping that stops training when validation accuracy stops improving. I also unfreeze more layers toward the end of training, allowing the model to adapt more to the dataset by refining weights.

5 References

- R. Hartley, A. Zisserman - *Multiple View Geometry in Computer Vision*, A comprehensive reference for projective geometry and camera models.
- ChatGPT by OpenAI for generating ideas, code assistance, and explanations.
- TensorFlow Documentation.
- Kanerika Inc - Keras vs PyTorch: Which ML Framework is the Best for You?
- Keras Documentation.
- Quora - CNN Architectures for Small Data Sets.
- GeeksforGeeks - VGG-16 CNN Model.