Saurav Mainali

Machine Learning II, Fall 2020

Individual Project Report

Scene Classification

Introduction

Computer Vision, a field in Computer Science, which aims to replicate the complex human vision system can today be found surpassing the humans. Thanks to the incessant advancement in Deep Learning and Neural Networks. This field has created wide range of applications, one of which is - ability to classify Scenes around the world.

We, members of Group 6 worked on the problem of classifying Natural Scenes around the globe, whose application could range from organization of photos in Smartphones; assist in growth of country's economy through Tourism Planning and so on.

My Individual Role on this project was to experiment with different pre trained networks and see how its results differed from CNN model which was trained by me from the scratch.

Description of Individual Work

Firstly, I started with preprocessing of data images which were obtained from Kaggle. The images were read using cv2 library, and then convert into a standard 150 by 150 pixels. The target label for each image was also recorded and all of them were saved. Code snippet for the data preparation is provided below:

```
for scene,label in order:
    for img in os.listdir("seg_train/seg_train/" + scene +"/"):
        x_train.append(cv2.resize(cv2.imread("seg_train/seg_train/" + scene + "/" + img), (150, 150)))
        y_train.append(label)
x_train = np.array(x_train)
y_train = np.array(y_train)
```

Then, I created a custom CNN Network using the Keras Framework, so that its performance could be compared with pre-trained models. The code for structure of the model can be seen below:

```
model = Sequential([
    Conv2D(16, (3,3),  activation="relu"),
    BatchNormalization(),
    MaxPooling2D((2,2)),
    Conv2D(32, (3,3), activation="relu"),
    BatchNormalization(),
    AveragePooling2D((2,2)),
    Conv2D(64, (3,3), activation="relu"),
    BatchNormalization(),
    AveragePooling2D((2,2)),
    Flatten(),
    Dense(400, activation="tanh"),
    Dropout(drop),
    BatchNormalization(),
    Dense(6, activation="softmax")
])
```

Three Conv2D layers with 16, 32 and 64 filters respectively, all of which had kernel size of 3, 3 and stride of 1 were connected end by end. Each had Relu activation unit. After the first layer Batch Normalization was performed followed by Max Pooling with kernel of 2,2. For the remaining two layers Batch Normalization was followed with Average Pooling of the same kernel size. The output was then flattened and passed into Dense layer with 400 filters and Tanh activation units. Finally, with the dropout rate of 0.5, they were all Normalized again and passed into SoftMax Unit having filters equal to number of class labels. Learning Rate was set to 0.001 and with the batch size of 512 the model was trained for 40 epochs with Callback monitoring loss of validation set.

Finally, I made use of five different pretrained CNN networks namely -VGG16, VGG19, ResNet152, InceptionV3, InceptionResV2, which were trained using the ImageNet dataset. These models were available inside keras.applications library. Performance for each of these models were compared with each other and with that of custom trained CNN model too. A code snippet of pretrained model can be seen below:

```
inc = applications.InceptionV3(input_shape=(150,150,3), weights='imagenet', include_top=False)
inc.trainable = False
print('inception pre trained model is loaded ....')

model = Sequential([inc,
                    Flatten(),
                    Dense(400,activation='tanh'),
                    Dropout(0.5),
                    BatchNormalization(),
                    Dense(6,activation='softmax')
                    ])
```

The weights of pre-trained networks were fixed. The output from pretrained networks were flattened and passed to Dense layer with 400 filters and Tanh activation unit. Then with dropout rate of 0.5 the outputs were normalized and passed into a final Dense layer with 6 filters and SoftMax as activation unit.

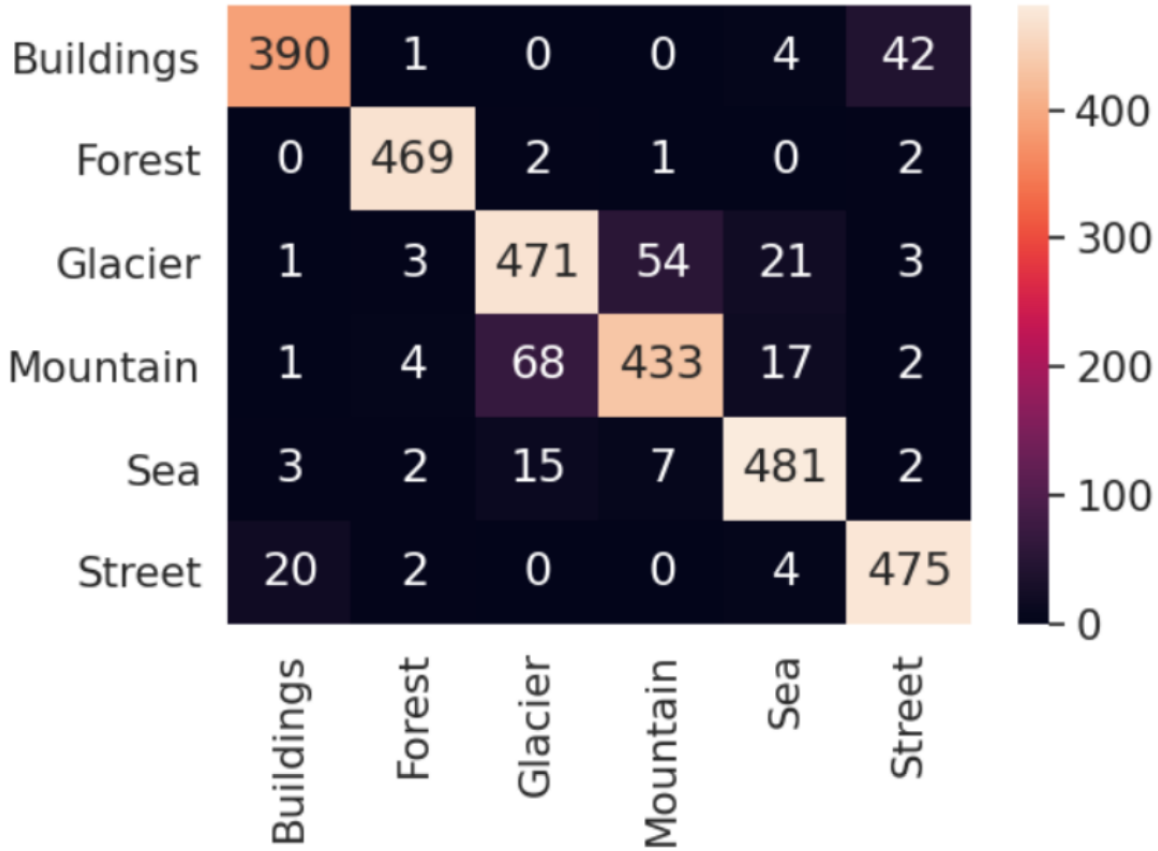
Results

To measure the performance of model Accuracy, Cohen Kappa Score and Macro F1 Score were used. Since our dataset is balanced, Accuracy Score can be considered as the prime performance indicator. It gives us idea regarding how many predictions were correct among all the predictions made. Similarly, Macro F1 Score hints us on performance of model by giving equal importance to each of the class label. Result obtained is shown below:

Model	Accuracy	Cohen Kappa Score	Macro F1 Score
VGG16	89.70 %	0.87	0.89
VGG19	88.49 %	0.86	0.88
ResNet152V2	90.63 %	0.88	0.90
InceptionV3	87.23 %	0.84	0.87
InceptionResV2	88.13 %	0.85	0.88
Self-Trained Model	83.33 %	0.79	0.83

ResNet152V2 performed the best with Accuracy of 90.63%, Cohen Kappa Score of 0.88 and Macro F1 Score of 0.90. Performance of remaining pre-trained models were close to the best model. The Self-trained model performance was worst with accuracy of 83.33%.

Below is Confusion Matrix of the best performing model, which gives deeper insight on predictions made and the training images:



The matrix indicates there is confusion between ‘Glacier & Mountain’ and ‘Buildings & Street’. So, on investing the dataset further, I found this problem was due to the nature of the training images.



Images which were labelled as Glacier had Mountains in the background making the model prone to misclassification. Similarly, error between Buildings and Street was also due to the similar reason. Images that were classified as Street had Buildings in it and vice versa. The images can be seen below:



Summary and Conclusions

Thus, the pretrained models performed much better than the self-trained CNN model. This could be because pretrained models were trained using huge dataset with very large number of trainable parameters. This project taught me that by using deep networks we can create extremely powerful models, given we have large number of quality training data with proper labels.

To make further improvement on this project I would recommend approaching this problem as Multi-Labelled Classification problem by making necessary amendments in the class labels.

NOTE: About 25% of codes were copied from the internet.

References

1. https://github.com/amir-jafari/Deep-Learning
2. https://github.com/krishnaik06/Transfer-Learning