Bereket Eshete Kebede
UID - U00827234

Graduate Student
The University of Memphis, Fall 2021
EECE 7740 Neural Networks

Transfer Learning (TL)

Instructor: Prof. Zahangir Alom
Assignment #3 Report

Introduction

In this report, we have two objectives. The first target is to try to understand transfer learning by fine-tuning ResNet Deep Convolutional Neural Network (DCNN). And the second target is ensembling neural models to accomplish a better accuracy for object recognition tasks. We use the TinyImage dataset for this recognition task. [1] TinyImage contains 80 million 32×32 pixel color images scaled down from images extracted from the World Wide Web in 2008 using automated web search queries and the words in the search terms were used as labels. [2]

The DCNN we will use for our task is called ResNet50. ResNet model is a widely used variant of the ResNet model and it has made it possible to resin ultra-deep neural networks. It has 48 convolutional layers with a Max pool and an Average pool layer. [3]

To achieve our objective of neural networks ensembles, I will introduce the concept of transfer learning. Transfer learning involves using pre-trained models as a starting point on a related problem. DCNN could take a lot of time while training large datasets, hence transfer learning provides a shortcut by diving the network into pieces and each can be trained individually and we can use transfer learning to stitch the networks we are interested in. In general transfer, learning has the advantage of reducing the training time and could result in lower generalization error. [4] The code associated with this report could be accessed at, https://github.com/bereketeshete/TransferLearning-ResNet50

Methodology

Three methodologies will be used for this report on transfer learning. The first experiment is pure training with ResNet50 network on the TinyImage dataset. To achieve this task we will utilize pre-implemented ResNet from Keras. The second experiment is saving the pre-trained weights from the first experiment and using those weights as a starting point to fine-tune our ResNet. The final experiment is an ensemble of the best models for object classification from these experiments and assignment 2 and we observe if the aggregation network performs better than individual models.

Deep Learning Architecture

There are many versions of ResNet, for this report, we will utilize a ResNet with 50 layers. The architecture detail is given below in the following table 1, along the fifth column is ResNet50 we are interested in.

ResNet50 Architecture

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Table 1. Layers used in ResNet Architecture

The architecture we will use for creating an ensemble network could be depicted as in the diagram below.

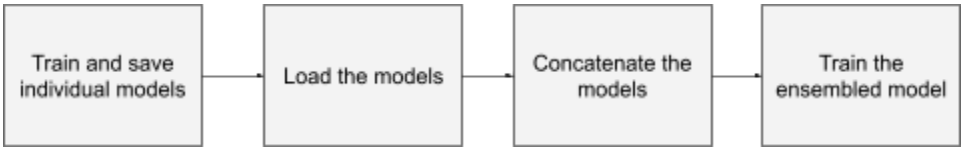

Figure 1. Illustration of an ensemble network from created from individual ones.

Experiment and Results

The experiment in this report was conducted with Tensorflow & Keras v2.6 tool on NVIDIA TITAN RTX GPU. The primary software used is this software is python and jupyter notebook. The training and testing results are discussed in the following section. In this experiment, to save the trained model, we use the model.save() whereas, to load the saved model weights for fine-tuning we use the following format below.

```
model.load_weights('model_resnet50.h5')
```

(a) Training and testing logs

- **Using ResNet50**

Training the ResNet50 model on the TinyImage dataset we get **56.5%** Testing Accuracy and **87.33%** Training Accuracy.

```
In [64]: scores = model.evaluate(X_test, Y_test, verbose=1)
         print('\nAccuracy:', scores[1])

         model.save('model_resnet50.h5')
         model.save_weights('pre_trained_weights_resnet50.h5')

         313/313 [==============================] - 4s 14ms/step - loss: 2.1671 - accuracy: 0.5651

         Accuracy: 0.5651000142097473
```

The training plot for accuracy and loss is depicted in the figure below.

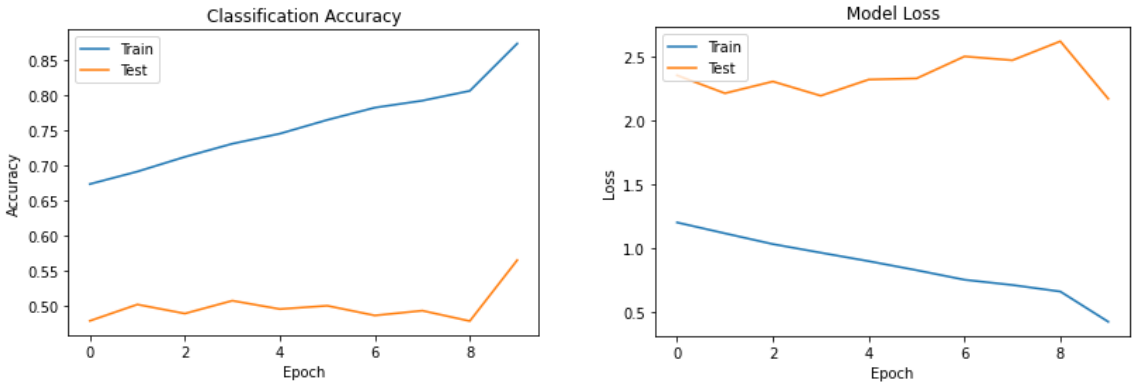

Figure 2. Classification accuracy and loss are plotted for ResNet classification performance on the TinyImage dataset.

- **Using pre-trained weights of ResNet50 and fine-tuning the model**

Training the ResNet50 model using weights from previous training and fine-tuning the model, we get an accuracy of **55.6%** Testing Accuracy and **95.37%** Training Accuracy. A small decrease in testing accuracy is observed but also a very big boost in training accuracy is observed compared to pre-fine tune, this could be due to overfitting while fine-tuning the ResNet50 model.

```
: scores = model.evaluate(X_test, Y_test, verbose=1)
  print('\nAccuracy:', scores[1])

  model.save('model_resnet50_fine_tuned.h5')

  313/313 [==============================] - 4s 14ms/step - loss: 2.5095 - accuracy: 0.5560

  Accuracy: 0.5559999942779541
```

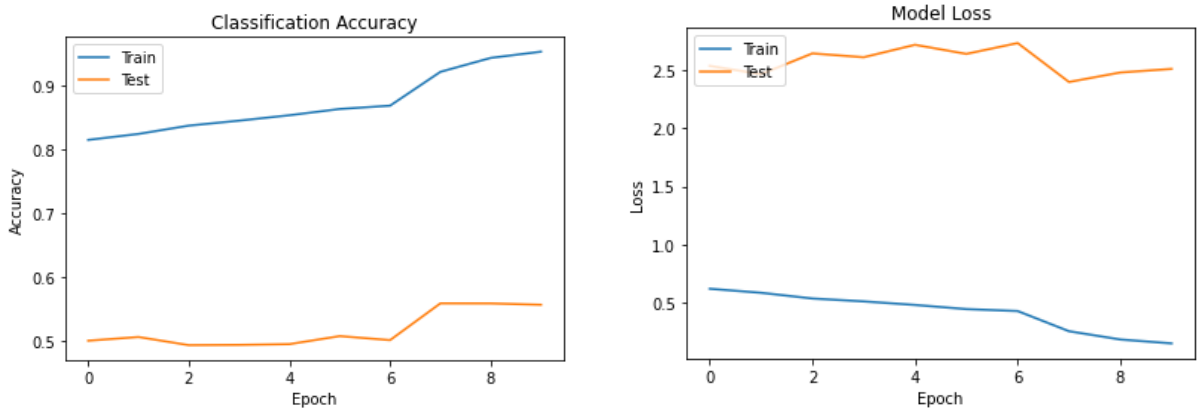

Figure 3. Classification accuracy and loss are plotted for fine-tuned ResNet50 classification performance on the TinyImage dataset.

❖ **Selecting the best DCNN performer from Assignment 2 for 15 Classification tasks**

A DCNN bundled with GlorotUniform Initializer and Adam optimizer with Training accuracy **96.95%** and Testing accuracy **80.58%**.

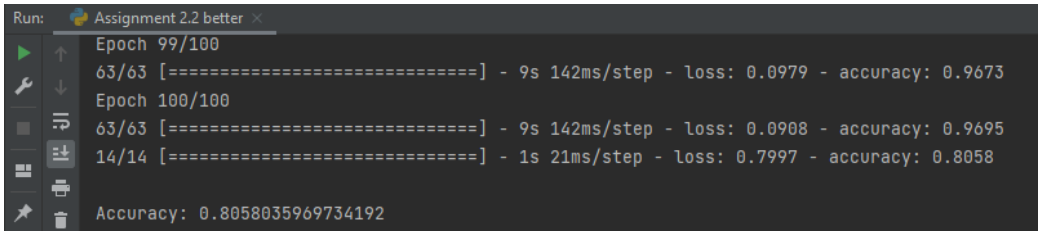

Figure 4. Using Adam results in Training accuracy **96.95%** and Testing accuracy **80.58%**. The computation time was 15 mins. Result From assignment 2.

- **Ensembling the models**

Finally we ensemble the models with the intention of getting better overall accuracy. First, to demonstrate the effectiveness of the ensemble model, we train ensemble 5 individuals MNIST and observe a 97% accuracy which is a 5% increase than assignment 1 MNIST classification. Now let's use a similar ensemble concept for ResNet50, and we obtain the following results.

```
scores = newModel.evaluate(X_test, Y_test, verbose=1)
print('\nAccuracy:', scores[1])
```

```
313/313 [==============================] - 7s 24ms/step - loss: 2.4365 - accuracy: 0.5663

Accuracy: 0.5662999749183655
```

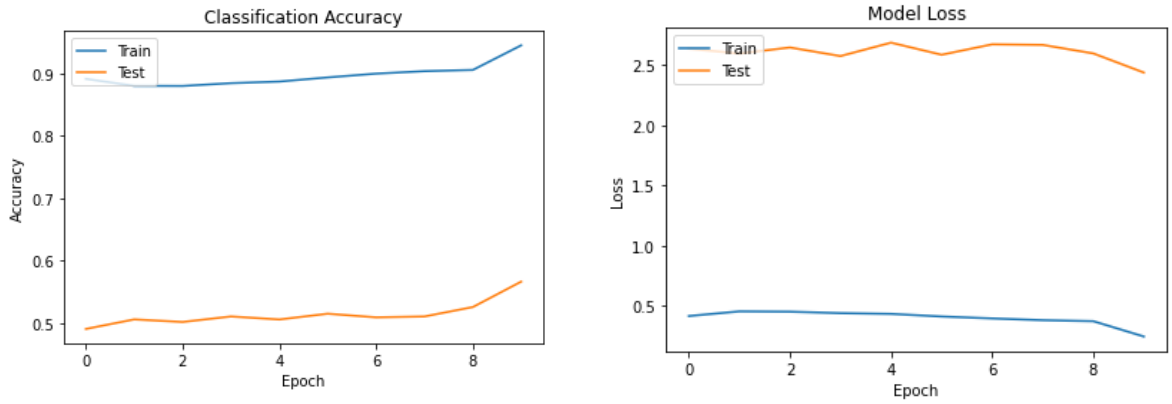

Figure 5. Training accuracy and model loss plot for Ensemble 1 & 2 (ResNet50 and fine-tuned ResNet50)

Ensembling the first two ResNet models we obtain **56.63%** testing accuracy and **94.49%** testing was observed. The ResNet couldn't be ensembled with the 15 DCNN from assignment 2, because the saved models have different input sizes and also KerasClassifier doesn't function on Multi-label and Multi-class problems. Hence only the first two ResNet was ensembled through concatenation of the layers.

(b) Discussion and comparison

The following table compares and contrasts the accuracy performance of the different image classification DNN models we discussed in this report.

Model	Accuracy
Resnet50	56.5%
Resnet50 fine-tuned	55.6%
DCNN 15 classifier	80.58%
Ensemble 1 & 2	56.6%

Table 2. Comparison of Testing accuracy for different neural networks used in this report.

Conclusion

In summary, ResNet50 is one of the best models for image classification tasks. We were able to obtain a 56.6% testing accuracy and up to 95.37% training accuracy. In addition, fine-tuning and ensembling DNN models could be used to boost the accuracy of the given neural networks through the method of transfer learning. DCNN has the best accuracy out of the models presented in table 2, and the combination of DCNN 15 with Ensemble 1 & 2 ResNet after modifying and matching input sizes of the saved models would yield the best accuracy.

References

[1]https://www.kaggle.com/akash2sharma/tiny-imagenet
[2]https://github.com/sonugiri1043/Train_ResNet_On_Tiny_ImageNet/blob/master/ResNet_TinyImageNet.ipynb
[3] https://iq.opengenus.org/resnet50-architecture/
[4]https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/
[5]https://towardsdatascience.com/destroy-image-classification-by-ensemble-of-pre-trained-models-f287513b7687
[6] https://github.com/zahangircse/COMP_EECE_7or8740_NNs/blob/main/Lecture_10_TL_1.ipynb
[7] https://github.com/zahangircse/COMP_EECE_7or8740_NNs/blob/main/Lecture_10_TL2.ipynb
[8]https://github.com/zahangircse/COMP_EECE_7or8740_NNs/blob/main/Lecture_10_model_ensemble.ipynb