EECE 8740-Neural Networks HW #3

(Solutions)

University of Memphis

Written by

S. Parisa Daj.
U00743495
(s.dajkhosh@memphis.edu)

October 11, 2021

1 Introduction

The goal of this assignment is to implement either a ResNet32 or a ResNet50 model. The data used for this task is the Tiny ImageNet dataset which includes 100000 RGB images of 200 different classes. Each image is 64x64 square-sized. In the first question, we are evaluating the said dataset on the ResNet50 model. ResNet50 is a residual neural network designed and pre-trained on millions of images in 1000 various classes. This model is 50 layers deep. After the evaluation, the model will be fine-tuned with the new images. The next step is to ensemble and compare the result of the two previous layers with the best model trained in assignment 2. It is expected to have the highest accuracy from the second task because we fine-tune a very powerful pre-trained model.

2 Methodology and Deep Learning Architecture

The code is accessible by this link on google colaboratory using GPU.

To perform the evaluation, test inputs should enter the model to be predicted by the model. The challenge here is that images come from Tiny ImageNet with a size of 64x64. However, ResNet is pre-trained on ImageNet which has images 224x224 pixels wide. Therefore, it is not possible to predict the outputs directly from our data. To resolve the challenge, a zero-padding layer with the size of 80x80 is added to the inputs to resize them into the required 224x224 size for the ResNet. Also, in order to be able to use all the available RAM capacity from 12 GB RAM of Google Colaboratory, only one-fifth of samples are being used (20000 samples chosen equally from each class).

For the fine-tuning part, two different experiments are done. In the first experiment, all layers of the ResNet model are frozen except the last 3 layers including the activation layer, max-pooling, and a dense layer. Using RMSProp optimizer, in 25 epochs with a learning rate of 0.001 and a batch size of 500. The second experiment trains on 6 layers including the convolutional layer, the batch normalization layer, followed by the layers trained in the previous experiment. Adam optimizer is chosen this time training for 30 iterations. However, the training could not get completed because of the Google Colaboratory time limit. Therefore, the training ended after 10 epochs. These layers are followed by two dense layers to get the 200 classes coming from Tiny Imagenet.

In the third section, the Deep Convolutional Neural Network model from assignment 2 is loaded with initialization and batch normalization layers. The training takes place in 50 iterations with augmented inputs, Adam optimizer, and a decaying learning rate starting from 0.002. To do the model ensemble, and comparing all three models described, ”VotingClassifier” from ”sklearn” is being used to classify these three models in a ”soft” voting manner.

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	(None, 224, 224, 3)	0	
conv1_pad (ZeroPadding2D)	(None, 230, 230, 3)	0	input_2[0][0]
conv1_conv (Conv2D)	(None, 112, 112, 64)	9472	conv1_pad[0][0]
pool1_pad (ZeroPadding2D)	(None, 114, 114, 64)	0	conv1_relu[0][0]
pool1_pool (MaxPooling2D)	(None, 56, 56, 64)	0	pool1_pad[0][0]
conv2_block1_1_conv (Conv2D)	(None, 56, 56, 64)	4160	pool1_pool[0][0]
conv2_block1_2_conv (Conv2D)	(None, 56, 56, 64)	36928	conv2_block1_1_relu[0][0]
conv2_block1_0_conv (Conv2D)	(None, 56, 56, 256)	16640	pool1_pool[0][0]
conv2_block1_3_conv (Conv2D)	(None, 56, 56, 256)	16640	conv2_block1_2_relu[0][0]
conv2_block2_1_conv (Conv2D)	(None, 56, 56, 64)	16448	conv2_block1_out[0][0]
conv2_block2_2_conv (Conv2D)	(None, 56, 56, 64)	36928	conv2_block2_1_relu[0][0]
conv2_block2_3_conv (Conv2D)	(None, 56, 56, 256)	16640	conv2_block2_2_relu[0][0]
conv2_block3_1_conv (Conv2D)	(None, 56, 56, 64)	16448	conv2_block2_out[0][0]
conv2_block3_2_conv (Conv2D)	(None, 56, 56, 64)	36928	conv2_block3_1_relu[0][0]
conv2_block3_3_conv (Conv2D)	(None, 56, 56, 256)	16640	conv2_block3_2_relu[0][0]
conv3_block1_1_conv (Conv2D)	(None, 28, 28, 128)	32896	conv2_block3_out[0][0]
conv3_block1_2_conv (Conv2D)	(None, 28, 28, 128)	147584	conv3_block1_1_relu[0][0]
conv3_block1_0_conv (Conv2D)	(None, 28, 28, 512)	131584	conv2_block3_out[0][0]
conv3_block1_3_conv (Conv2D)	(None, 28, 28, 512)	66048	conv3_block1_2_relu[0][0]
conv3_block2_1_conv (Conv2D)	(None, 28, 28, 128)	65664	conv3_block1_out[0][0]
conv3_block2_2_conv (Conv2D)	(None, 28, 28, 128)	147584	conv3_block2_1_relu[0][0]
conv3_block2_3_conv (Conv2D)	(None, 28, 28, 512)	66048	conv3_block2_2_relu[0][0]
conv3_block3_1_conv (Conv2D)	(None, 28, 28, 128)	65664	conv3_block2_out[0][0]
conv3_block3_2_conv (Conv2D)	(None, 28, 28, 128)	147584	conv3_block3_1_relu[0][0]
conv3_block3_3_conv (Conv2D)	(None, 28, 28, 512)	66048	conv3_block3_2_relu[0][0]
conv3_block4_1_conv (Conv2D)	(None, 28, 28, 128)	65664	conv3_block3_out[0][0]
conv3_block4_2_conv (Conv2D)	(None, 28, 28, 128)	147584	conv3_block4_1_relu[0][0]
conv3_block4_3_conv (Conv2D)	(None, 28, 28, 512)	66048	conv3_block4_2_relu[0][0]
conv4_block1_1_conv (Conv2D)	(None, 14, 14, 256)	131328	conv3_block4_out[0][0]
conv4_block1_2_conv (Conv2D)	(None, 14, 14, 256)	590080	conv4_block1_1_relu[0][0]
conv4_block1_0_conv (Conv2D)	(None, 14, 14, 1024)	525312	conv3_block4_out[0][0]
conv4_block1_3_conv (Conv2D)	(None, 14, 14, 1024)	263168	conv4_block1_2_relu[0][0]
conv4_block2_1_conv (Conv2D)	(None, 14, 14, 256)	262400	conv4_block1_out[0][0]
conv4_block2_2_conv (Conv2D)	(None, 14, 14, 256)	590080	conv4_block2_1_relu[0][0]
conv4_block2_3_conv (Conv2D)	(None, 14, 14, 1024)	263168	conv4_block2_2_relu[0][0]
conv4_block3_1_conv (Conv2D)	(None, 14, 14, 256)	262400	conv4_block2_out[0][0]
conv4_block3_2_conv (Conv2D)	(None, 14, 14, 256)	590080	conv4_block3_1_relu[0][0]
conv4_block3_3_conv (Conv2D)	(None, 14, 14, 1024)	263168	conv4_block3_2_relu[0][0]
conv4_block4_1_conv (Conv2D)	(None, 14, 14, 256)	262400	conv4_block3_out[0][0]
conv4_block4_2_conv (Conv2D)	(None, 14, 14, 256)	590080	conv4_block4_1_relu[0][0]
conv4_block4_3_conv (Conv2D)	(None, 14, 14, 1024)	263168	conv4_block4_2_relu[0][0]
conv4_block5_1_conv (Conv2D)	(None, 14, 14, 256)	262400	conv4_block4_out[0][0]
conv4_block5_2_conv (Conv2D)	(None, 14, 14, 256)	590080	conv4_block5_1_relu[0][0]
conv4_block5_3_conv (Conv2D)	(None, 14, 14, 1024)	263168	conv4_block5_2_relu[0][0]
conv4_block6_1_conv (Conv2D)	(None, 14, 14, 256)	262400	conv4_block5_out[0][0]
conv4_block6_2_conv (Conv2D)	(None, 14, 14, 256)	590080	conv4_block6_1_relu[0][0]
conv4_block6_3_conv (Conv2D)	(None, 14, 14, 1024)	263168	conv4_block6_2_relu[0][0]
conv5_block1_1_conv (Conv2D)	(None, 7, 7, 512)	524800	conv4_block6_out[0][0]
conv5_block1_2_conv (Conv2D)	(None, 7, 7, 512)	2359808	conv5_block1_1_relu[0][0]
conv5_block1_0_conv (Conv2D)	(None, 7, 7, 2048)	2099200	conv4_block6_out[0][0]
conv5_block1_3_conv (Conv2D)	(None, 7, 7, 2048)	1050624	conv5_block1_2_relu[0][0]

3 Experimental and Test Results

The first model is evaluated using the top 10 predictions listed below:

$$\begin{gathered}
Predicted: \\
[('n01930112','nematode',0.8054854), \\
('n04153751','screw',0.09041874), \\
('n02840245','binder',0.08625883), \\
('n03041632','cleaver',0.017806126), \\
('n01774384','black_widow',4.2166891e-07), \\
('n04286575','spotlight',1.8185496e-07), \\
('n03657121','lens_cap',1.3376867e-07), \\
('n04118776','rule',1.2237993e-07), \\
('n03126707','crane',1.11845544e-07), \\
('n04376876','syringe',1.1049992e-07)]
\end{gathered} \tag{1}$$

As we can see, "nematode" is recognized with an accuracy of %80 which is a good one. However, the rest of them do not have acceptable accuracy. The code for this section is found in this link. The results show that there is not a big similarity between the ImageNet dataset and the Tiny ImageNet dataset. Or, maybe due to the difference in size, and zero padding, it is difficult for the network to recognize the images. To have better accuracy, it is recommended to keep most of the layers frozen and fine-tune some of the last layers. That is where the second task starts. With the code available here the result is as seen in (figure 1. The validation accuracy in the best point reaches only %1.4 which is embarrassing. As a result, the second experiment is going to be tested which seems to be better. The final accuracy after all the epochs is still too low even after the fine-tuning (%1.1 shown in figure 2) The slope of the loss function, and the rate of increase in the second task indicates that if the training would have continued for 25 epochs, maybe we could achieve higher accuracy than the first experiment. However, the accuracy is still too small and requires some discussions around this topic.

On the other hand, the third task shows a higher accuracy of %26.4. It was expected that the accuracy for this model will be smaller than the accuracy achieved by the second task. As the second task is being fine-tuned on a very strong pre-trained model. To do the model ensemble classification, unfortunately, I was not able to complete the task because of receiving this error message: "NotImplementedError: Multilabel and multi-output classification is not supported." when I was trying to fit the model on training input and labels. And, unfortunately, I could not find a proper solution for that through the network and did not find more time to ask about it. Link for the third question

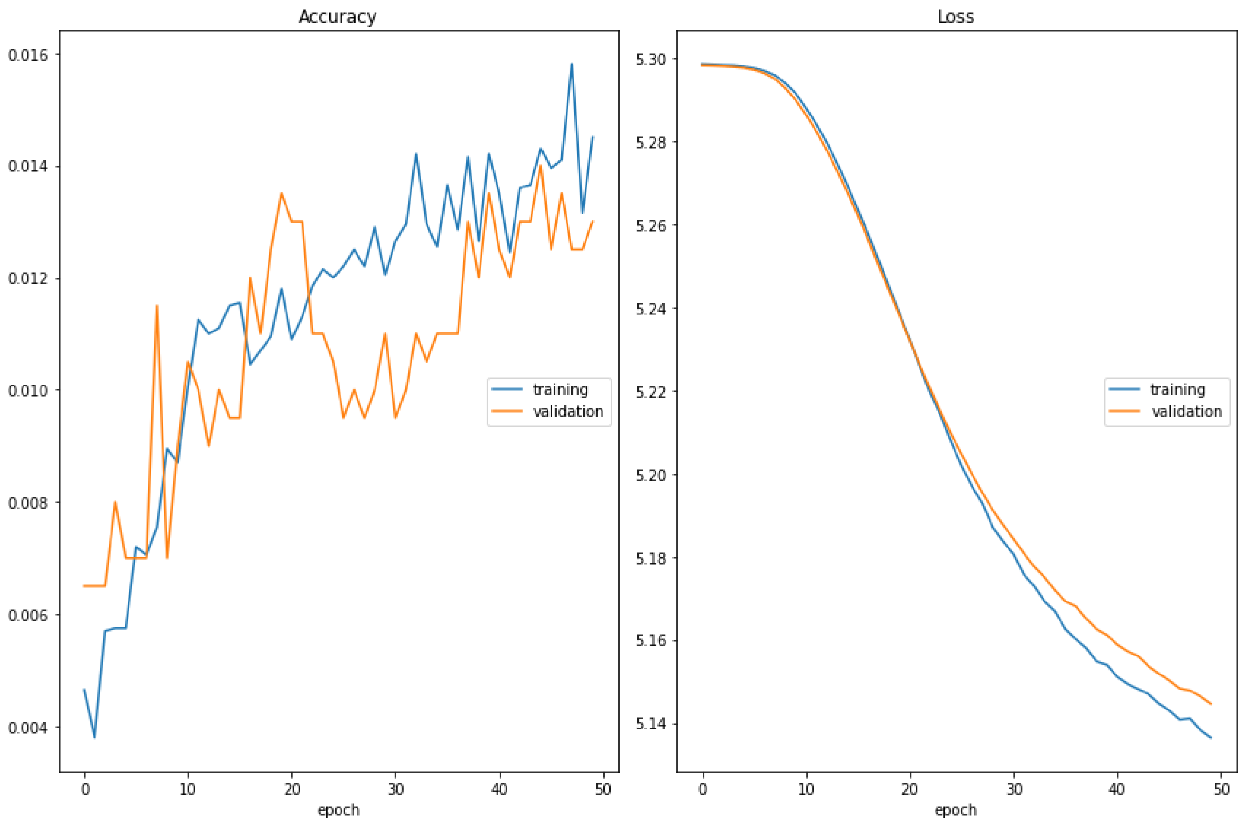

Figure 1: Network train and validation accuracy and loss logs along the epochs for the fine tunning task. (first experiment)

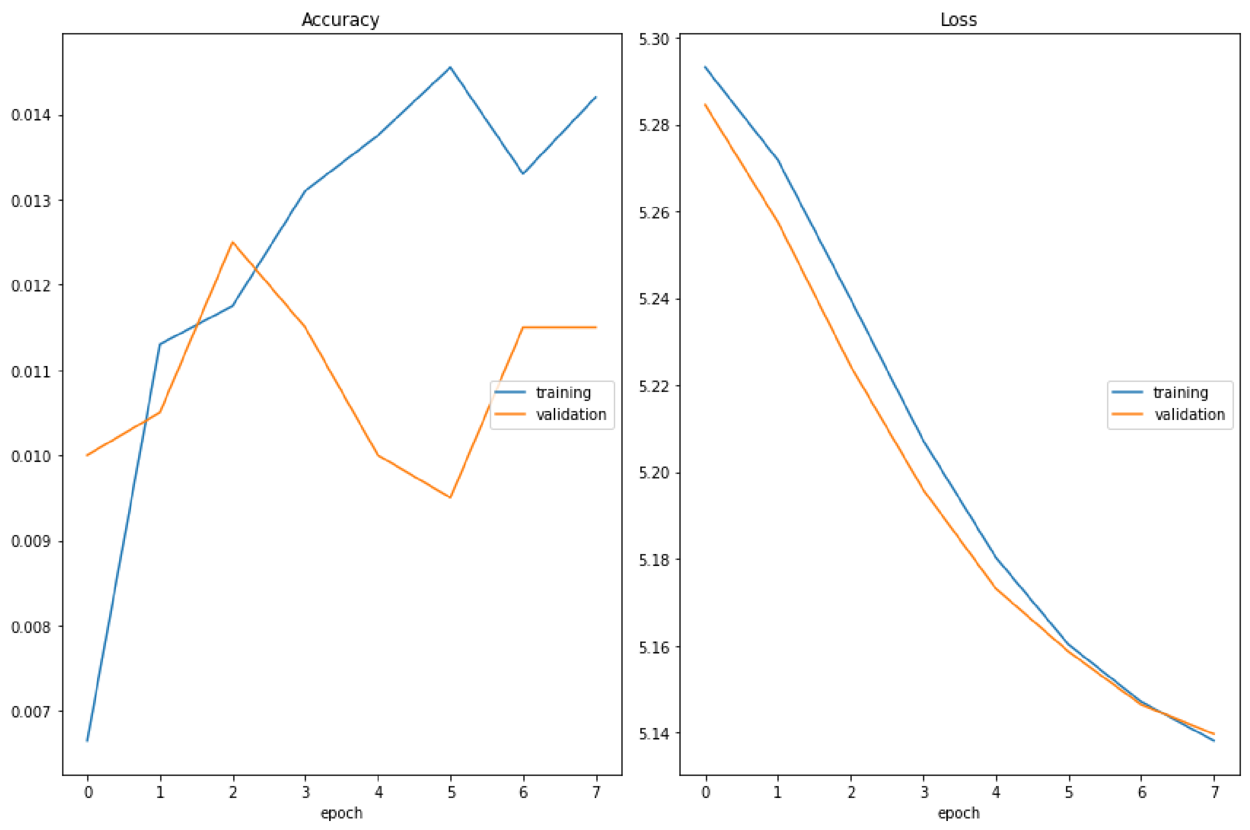

Figure 2: Network train and validation accuracy and loss logs along the epochs for the fine tuning task (second experiment).

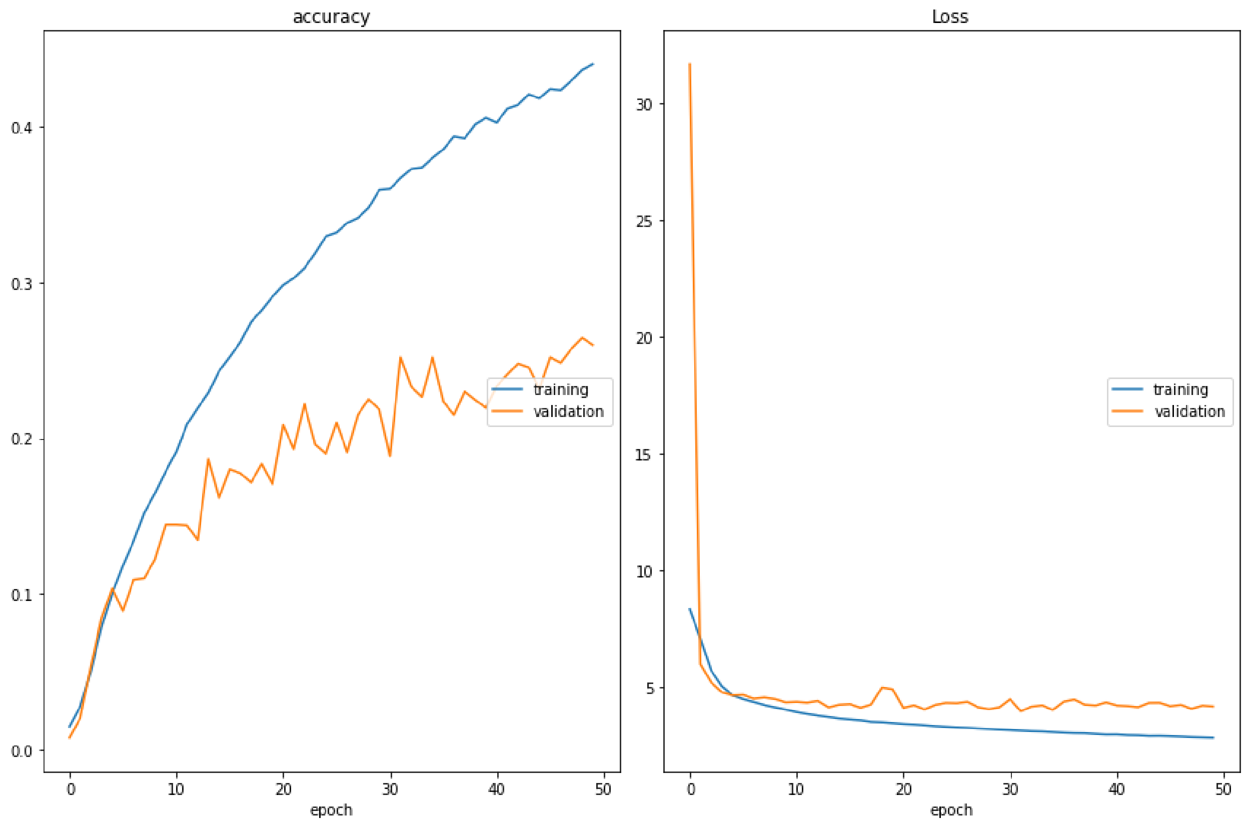

Figure 3: Network train and validation accuracy and loss logs along the epochs for the best model from assignment 2.

4 Conclusion

The point of this assignment was to be able to implement three different models: 1- just using a pre-trained model on a new dataset, 2- fine-tune the pre-trained model with the dataset, and 3- train the data on a DCNN model. Afterward, we could compare these three models together by ensembling the models. I expected to have a high accuracy in task 2 due to the power of pre-trained ResNet50 in addition to the time and effort spent to retrain the final layers. The two experiments done on the second task show that the result is not close to over-fitting as the training and validation accuracy and losses are too close to each other. Thus, having a much bigger iteration number may change the maximum accuracy significantly. Furthermore, the zero-padding method to increase the size of input images can be one of the sources of trouble. As it can be seen there are different parameters responsible for choosing one of the three models such as the number of epochs, resize methods, the similarity between our dataset and the dataset that our model is trained with, also the size of the dataset can have an essential impact.