ECE 595 Homework #3
7/22/2021
Christopher Couture Del Valle

Introduction:

For the first objective of homework #3, students were tasked with understanding the Transfer Learning (TL) approach. This was done by implementing and fine-tuning the state-of-the-art Deep Convolutional Neural Networks (DCNN) model called ResNet. The second objective was to ensemble models and achieve better testing accuracy for object recognition tasks.

Methodology:

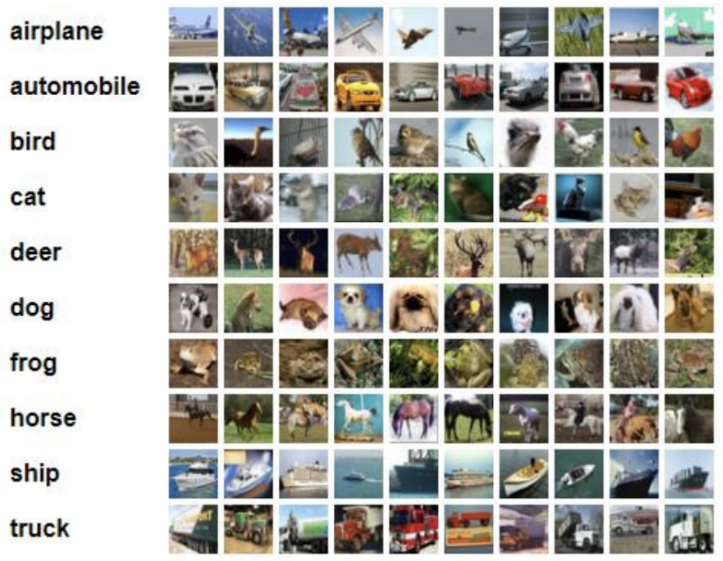


Figure 1: Example images from CIFAR 100 dataset

The methodology for implementing the above tasks follows as such. First ResNet50 architecture was implemented and the performance of the model evaluated on the CIFAR-100 dataset. Next the pretrained weights of the ResNet50 model were downloaded on the ImageNet dataset. The pretrained weights were utilized to fine-tune the model.

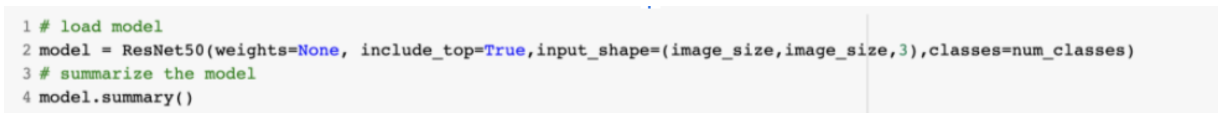
Testing results were provided for the experiment of the first two tasks. Code from lectures 7, 8 and 9 were referenced for the development of the three models. Next the best model was selected from assignment and trained/tested on the CIFAR 100 dataset. Finally the two models were put in ensemble and results compared against the single model implementation.

Deep Learning Architecture:

Untrained Resnet

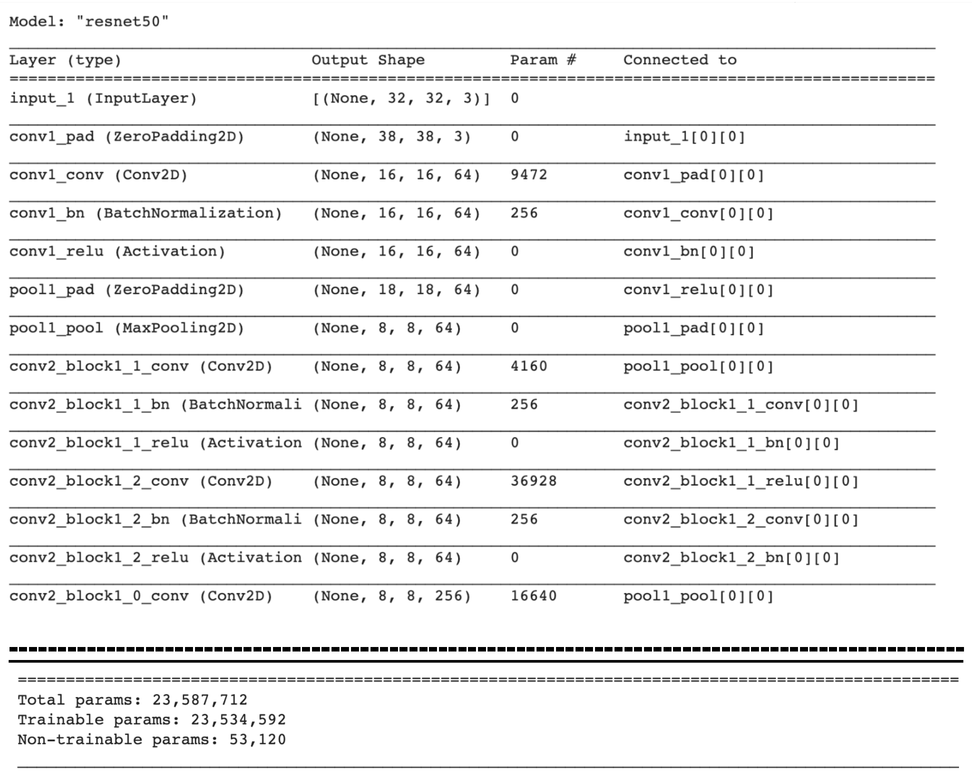
______

______The untrained resnet required the ResNet50 model to be imported from keras with some predefined input variables. The weights were set to None as imagenet pretrained weights were not being utilized. Include_top would therefore be set to True (aka top layers), the input shape was defined along with num_classes.



```python
# load model
model = ResNet50(weights=None, include_top=True,input_shape=(image_size,image_size,3),classes=num_classes)
# summarize the model
model.summary()
```

Figure 2: Untrained Resnet python syntax

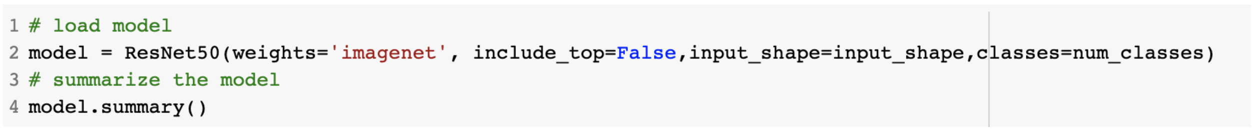


```
Model: "resnet50"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 32, 32, 3)]  0
__________________________________________________________________________________________________
conv1_pad (ZeroPadding2D)       (None, 38, 38, 3)    0           input_1[0][0]
__________________________________________________________________________________________________
conv1_conv (Conv2D)             (None, 16, 16, 64)   9472        conv1_pad[0][0]
__________________________________________________________________________________________________
conv1_bn (BatchNormalization)   (None, 16, 16, 64)   256         conv1_conv[0][0]
__________________________________________________________________________________________________
conv1_relu (Activation)         (None, 16, 16, 64)   0           conv1_bn[0][0]
__________________________________________________________________________________________________
pool1_pad (ZeroPadding2D)       (None, 18, 18, 64)   0           conv1_relu[0][0]
__________________________________________________________________________________________________
pool1_pool (MaxPooling2D)       (None, 8, 8, 64)     0           pool1_pad[0][0]
__________________________________________________________________________________________________
conv2_block1_1_conv (Conv2D)    (None, 8, 8, 64)     4160        pool1_pool[0][0]
__________________________________________________________________________________________________
conv2_block1_1_bn (BatchNormali (None, 8, 8, 64)     256         conv2_block1_1_conv[0][0]
__________________________________________________________________________________________________
conv2_block1_1_relu (Activation (None, 8, 8, 64)     0           conv2_block1_1_bn[0][0]
__________________________________________________________________________________________________
conv2_block1_2_conv (Conv2D)    (None, 8, 8, 64)     36928       conv2_block1_1_relu[0][0]
__________________________________________________________________________________________________
conv2_block1_2_bn (BatchNormali (None, 8, 8, 64)     256         conv2_block1_2_conv[0][0]
__________________________________________________________________________________________________
conv2_block1_2_relu (Activation (None, 8, 8, 64)     0           conv2_block1_2_bn[0][0]
__________________________________________________________________________________________________
conv2_block1_0_conv (Conv2D)    (None, 8, 8, 256)    16640       pool1_pool[0][0]

--------------------------------------------------------------------------------------------------
==================================================================================================
Total params: 23,587,712
Trainable params: 23,534,592
Non-trainable params: 53,120
__________________________________________________________________________________________________
```

Figure 3: Untrained Resnet Model Architecture Snippet

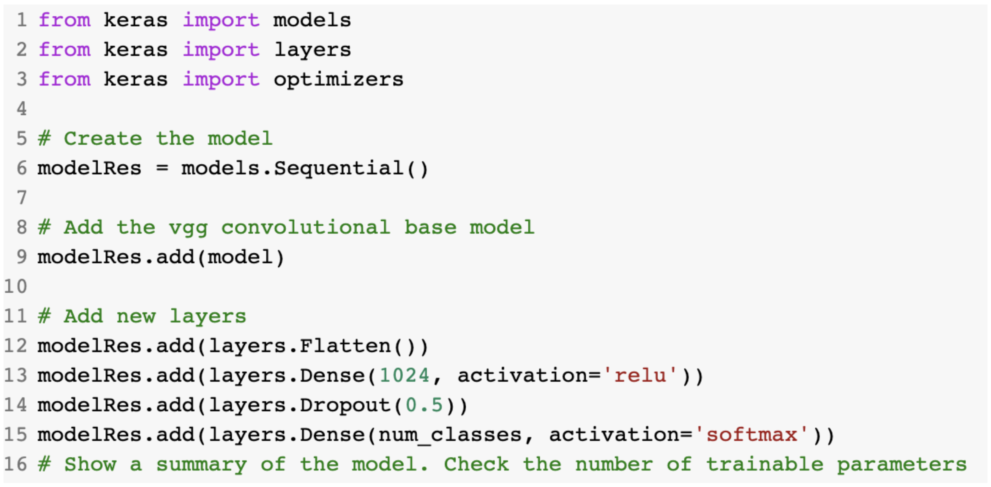
Trained Resnet

______The trained resnet required the ResNet50 model to be imported from keras with some predefined input variables along with imagenet being downloaded. The weights were set to imagenet as imagenet pretrained weights were being utilized. Include_top would therefore be set to False (aka top layers not showing), the input shape was defined along with num_classes. Additionally since the top 4 layers weren't present a sequential model with 4 additional layers were added to complete and improve the Resnet model.



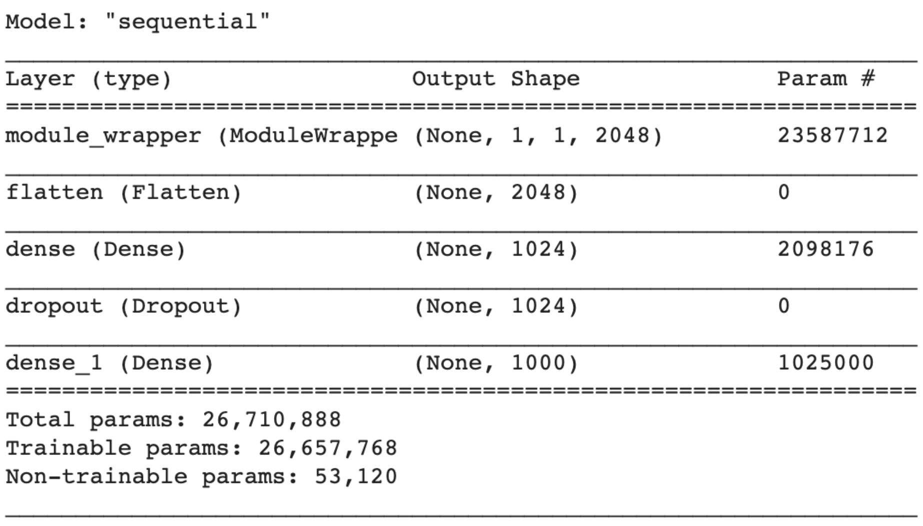
```
# load model
model = ResNet50(weights='imagenet', include_top=False,input_shape=input_shape,classes=num_classes)
# summarize the model
model.summary()
```

Figure 4: Trained Resnet Model syntax A



```
from keras import models
from keras import layers
from keras import optimizers

# Create the model
modelRes = models.Sequential()

# Add the vgg convolutional base model
modelRes.add(model)

# Add new layers
modelRes.add(layers.Flatten())
modelRes.add(layers.Dense(1024, activation='relu'))
modelRes.add(layers.Dropout(0.5))
modelRes.add(layers.Dense(num_classes, activation='softmax'))
# Show a summary of the model. Check the number of trainable parameters
```

Figure 5: Trained Resnet Model syntax B

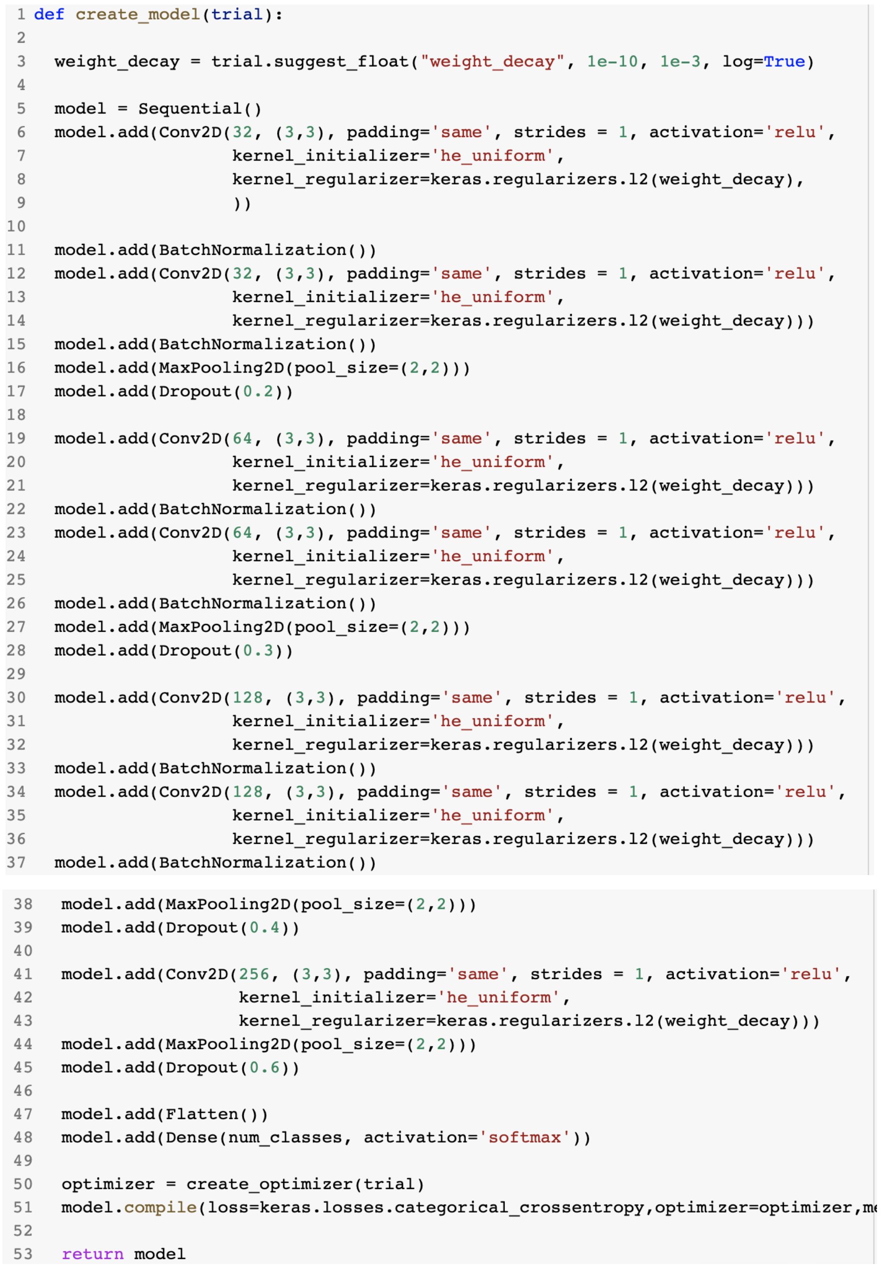


```
Model: "sequential"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
module_wrapper (ModuleWrappe (None, 1, 1, 2048)       23587712

_________________________________________________________________
flatten (Flatten)           (None, 2048)              0

_________________________________________________________________
dense (Dense)               (None, 1024)              2098176

_________________________________________________________________
dropout (Dropout)           (None, 1024)              0

_________________________________________________________________
dense_1 (Dense)             (None, 1000)              1025000
=================================================================
Total params: 26,710,888
Trainable params: 26,657,768
Non-trainable params: 53,120
_________________________________________________________________
```

Figure 6: Trained Resnet Model Architecture Snippet

Assignment 2 Model

The DCNN from assignment 2 was pulled and trained on the Cifar100 dataset like the previous ResNet models. Optuna was utilized to find the optimal weight decay and optimizer lr of the model. Following the optimal hyperparameter values the values would be frozen. The model from assignment 2 was created to be deep and wide being optimized by Optuna.

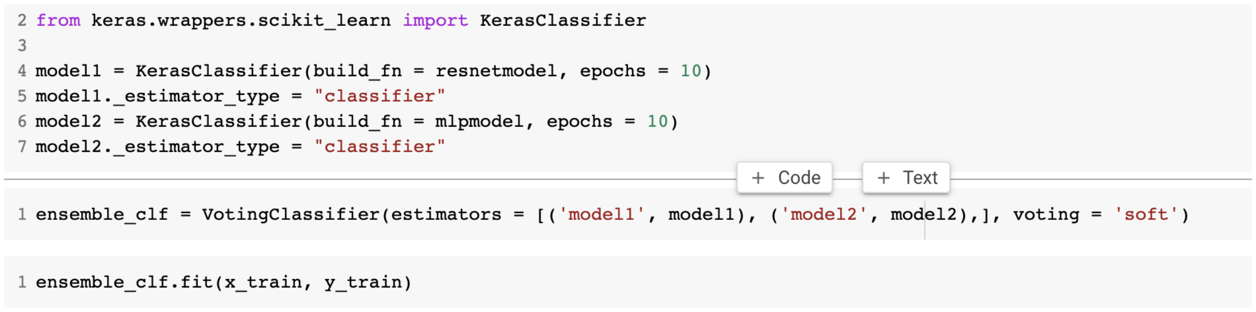


```python
def create_model(trial):

  weight_decay = trial.suggest_float("weight_decay", 1e-10, 1e-3, log=True)

  model = Sequential()
  model.add(Conv2D(32, (3,3), padding='same', strides = 1, activation='relu',
                   kernel_initializer='he_uniform',
                   kernel_regularizer=keras.regularizers.l2(weight_decay),
                   ))

  model.add(BatchNormalization())
  model.add(Conv2D(32, (3,3), padding='same', strides = 1, activation='relu',
                   kernel_initializer='he_uniform',
                   kernel_regularizer=keras.regularizers.l2(weight_decay)))
  model.add(BatchNormalization())
  model.add(MaxPooling2D(pool_size=(2,2)))
  model.add(Dropout(0.2))

  model.add(Conv2D(64, (3,3), padding='same', strides = 1, activation='relu',
                   kernel_initializer='he_uniform',
                   kernel_regularizer=keras.regularizers.l2(weight_decay)))
  model.add(BatchNormalization())
  model.add(Conv2D(64, (3,3), padding='same', strides = 1, activation='relu',
                   kernel_initializer='he_uniform',
                   kernel_regularizer=keras.regularizers.l2(weight_decay)))
  model.add(BatchNormalization())
  model.add(MaxPooling2D(pool_size=(2,2)))
  model.add(Dropout(0.3))

  model.add(Conv2D(128, (3,3), padding='same', strides = 1, activation='relu',
                   kernel_initializer='he_uniform',
                   kernel_regularizer=keras.regularizers.l2(weight_decay)))
  model.add(BatchNormalization())
  model.add(Conv2D(128, (3,3), padding='same', strides = 1, activation='relu',
                   kernel_initializer='he_uniform',
                   kernel_regularizer=keras.regularizers.l2(weight_decay)))
  model.add(BatchNormalization())
  model.add(MaxPooling2D(pool_size=(2,2)))
  model.add(Dropout(0.4))

  model.add(Conv2D(256, (3,3), padding='same', strides = 1, activation='relu',
                   kernel_initializer='he_uniform',
                   kernel_regularizer=keras.regularizers.l2(weight_decay)))
  model.add(MaxPooling2D(pool_size=(2,2)))
  model.add(Dropout(0.6))

  model.add(Flatten())
  model.add(Dense(num_classes, activation='softmax'))

  optimizer = create_optimizer(trial)
  model.compile(loss=keras.losses.categorical_crossentropy,optimizer=optimizer,m

  return model
```

Figure 7: Assignment 2 DNN Model Python Syntax

Ensemble Model

The ensemble model was created utilizing the DCNN model from assignment 2 and the imaged trained ResNet50 model. The syntax from lecture 9 was utilized to help create the ensemble mode from the two models. Some syntax had to be changed as it was dated with the estimator types having to be defined. Both models were placed in functions for proper coding etiquette. The models were the exact models as seen previously with the only change being the DCNN hyperparameter values being frozen.



```python
from keras.wrappers.scikit_learn import KerasClassifier

model1 = KerasClassifier(build_fn = resnetmodel, epochs = 10)
model1._estimator_type = "classifier"
model2 = KerasClassifier(build_fn = mlpmodel, epochs = 10)
model2._estimator_type = "classifier"
```

```python
ensemble_clf = VotingClassifier(estimators = [('model1', model1), ('model2', model2),], voting = 'soft')
```

```python
ensemble_clf.fit(x_train, y_train)
```

Figure 8: Ensemble Model Python Syntax

Results + Discussion & Comparison:

Untrained Resnet

______The untrained resnet model was run for 20 epochs on the Cifar100 dataset with its training and validation accuracy being plotted. The untrained model started with very low initial training and testing accuracy values. It then began to slowly progress and ended with a training accuracy of 48% and validation accuracy of 40%. The training loss continuously decreased whereas the validation loss was rather volatile.

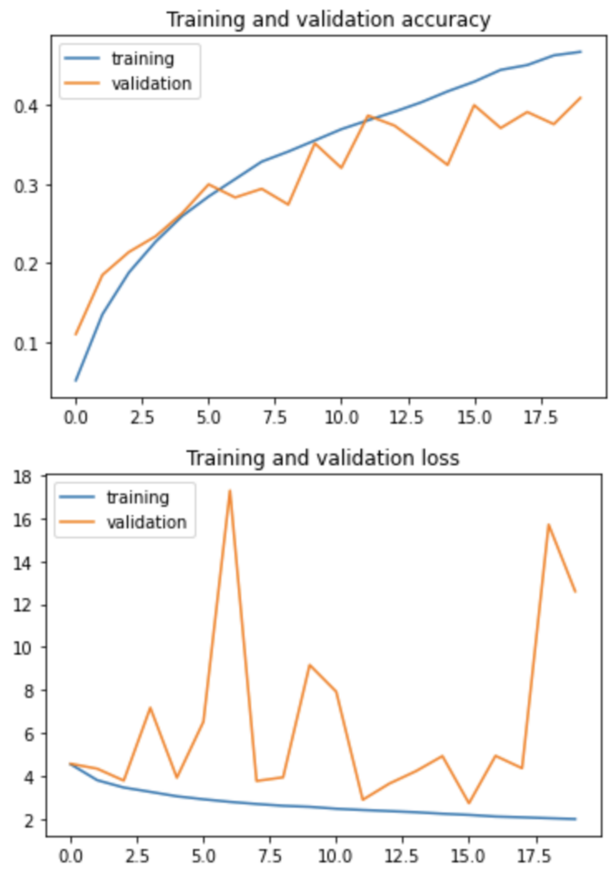


Figure 9: Untrained Resnet Accuracy and Loss Plots

Trained Resnet

The trained resnet model was also run for 20 epochs on the Cifar100 dataset with its training and validation accuracy being plotted. When compared to the untrained ResNet model, the trained model performed much better from the get go. The trained ResNet model started with much higher initial training and testing accuracy values. The training accuracy ended at 60% with the validation accuracy starting at 40% and ending at 54%. This better performance was most definitely attributed to the pretrained weights of imagenet. The training loss continuously decreased as expected whereas the validation loss increased. This was not perfect but performed better than the untrained model as expected.

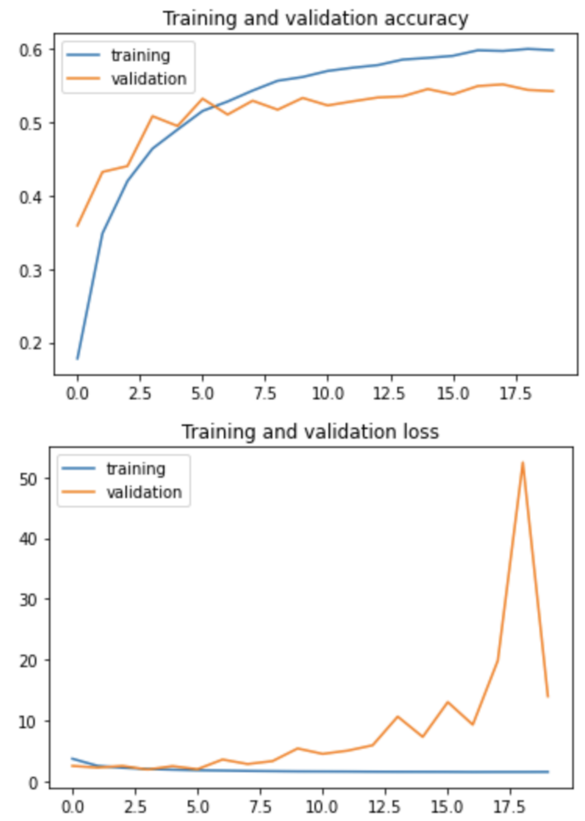
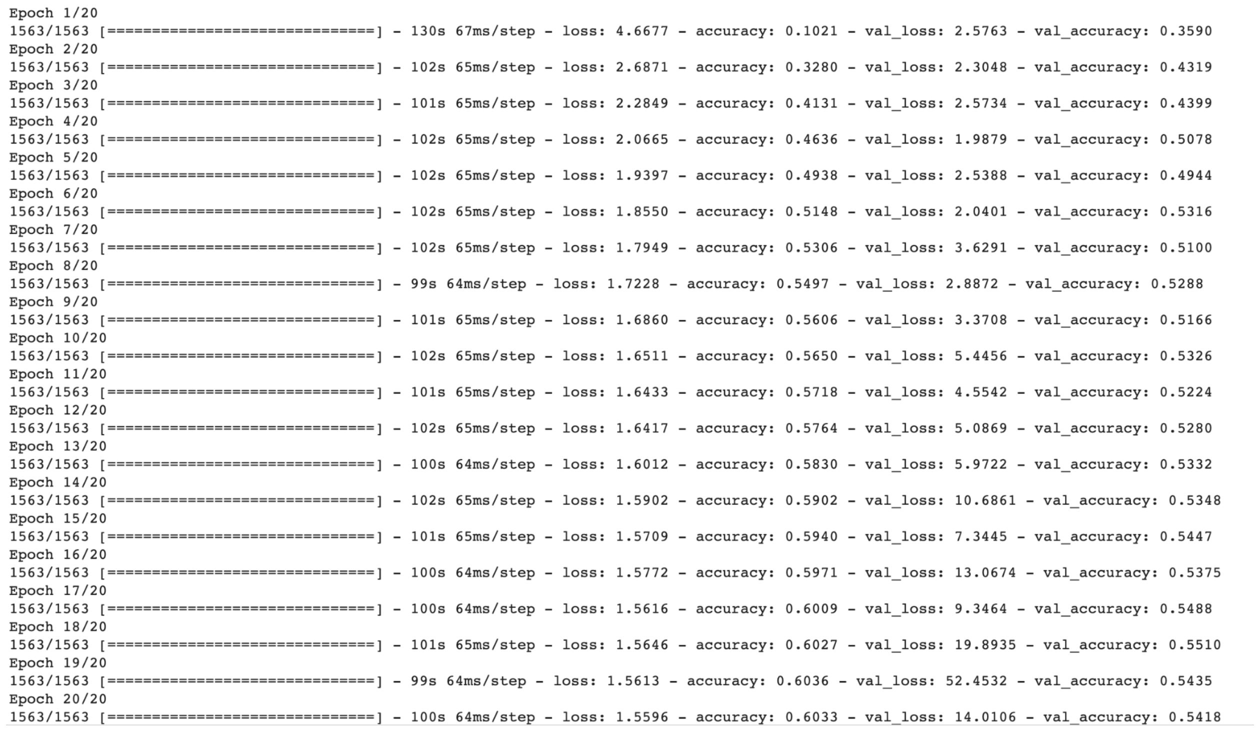


Figure 10: Trained Resnet Accuracy and Loss Plots

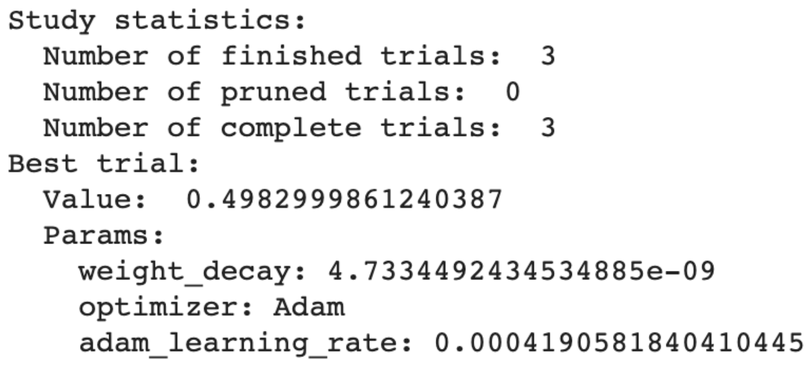


```
Epoch 1/20
1563/1563 [==============================] - 130s 67ms/step - loss: 4.6677 - accuracy: 0.1021 - val_loss: 2.5763 - val_accuracy: 0.3590
Epoch 2/20
1563/1563 [==============================] - 102s 65ms/step - loss: 2.6871 - accuracy: 0.3280 - val_loss: 2.3048 - val_accuracy: 0.4319
Epoch 3/20
1563/1563 [==============================] - 101s 65ms/step - loss: 2.2849 - accuracy: 0.4131 - val_loss: 2.5734 - val_accuracy: 0.4399
Epoch 4/20
1563/1563 [==============================] - 102s 65ms/step - loss: 2.0665 - accuracy: 0.4636 - val_loss: 1.9879 - val_accuracy: 0.5078
Epoch 5/20
1563/1563 [==============================] - 102s 65ms/step - loss: 1.9397 - accuracy: 0.4938 - val_loss: 2.5388 - val_accuracy: 0.4944
Epoch 6/20
1563/1563 [==============================] - 102s 65ms/step - loss: 1.8550 - accuracy: 0.5148 - val_loss: 2.0401 - val_accuracy: 0.5316
Epoch 7/20
1563/1563 [==============================] - 102s 65ms/step - loss: 1.7949 - accuracy: 0.5306 - val_loss: 3.6291 - val_accuracy: 0.5100
Epoch 8/20
1563/1563 [==============================] - 99s 64ms/step - loss: 1.7228 - accuracy: 0.5497 - val_loss: 2.8872 - val_accuracy: 0.5288
Epoch 9/20
1563/1563 [==============================] - 101s 65ms/step - loss: 1.6860 - accuracy: 0.5606 - val_loss: 3.3708 - val_accuracy: 0.5166
Epoch 10/20
1563/1563 [==============================] - 102s 65ms/step - loss: 1.6511 - accuracy: 0.5650 - val_loss: 5.4456 - val_accuracy: 0.5326
Epoch 11/20
1563/1563 [==============================] - 101s 65ms/step - loss: 1.6433 - accuracy: 0.5718 - val_loss: 4.5542 - val_accuracy: 0.5224
Epoch 12/20
1563/1563 [==============================] - 102s 65ms/step - loss: 1.6417 - accuracy: 0.5764 - val_loss: 5.0869 - val_accuracy: 0.5280
Epoch 13/20
1563/1563 [==============================] - 100s 64ms/step - loss: 1.6012 - accuracy: 0.5830 - val_loss: 5.9722 - val_accuracy: 0.5332
Epoch 14/20
1563/1563 [==============================] - 102s 65ms/step - loss: 1.5902 - accuracy: 0.5902 - val_loss: 10.6861 - val_accuracy: 0.5348
Epoch 15/20
1563/1563 [==============================] - 101s 65ms/step - loss: 1.5709 - accuracy: 0.5940 - val_loss: 7.3445 - val_accuracy: 0.5447
Epoch 16/20
1563/1563 [==============================] - 100s 64ms/step - loss: 1.5772 - accuracy: 0.5971 - val_loss: 13.0674 - val_accuracy: 0.5375
Epoch 17/20
1563/1563 [==============================] - 100s 64ms/step - loss: 1.5616 - accuracy: 0.6009 - val_loss: 9.3464 - val_accuracy: 0.5488
Epoch 18/20
1563/1563 [==============================] - 101s 65ms/step - loss: 1.5646 - accuracy: 0.6027 - val_loss: 19.8935 - val_accuracy: 0.5510
Epoch 19/20
1563/1563 [==============================] - 99s 64ms/step - loss: 1.5613 - accuracy: 0.6036 - val_loss: 52.4532 - val_accuracy: 0.5435
Epoch 20/20
1563/1563 [==============================] - 100s 64ms/step - loss: 1.5596 - accuracy: 0.6033 - val_loss: 14.0106 - val_accuracy: 0.5418
```

Figure 11: Trained Resnet Accuracy and Loss 20 Epoch Output

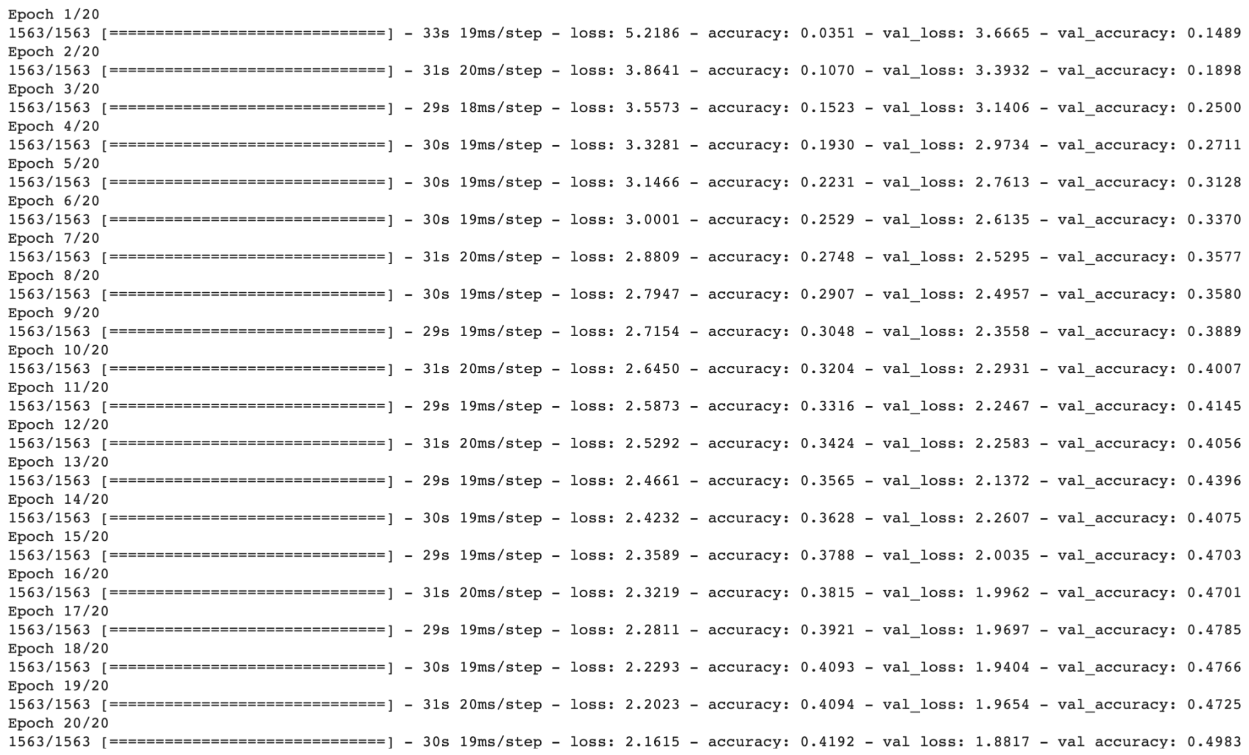
Assignment 2

The Assignment 2 DCNN was trained using the Cifar100 dataset for 20 epochs like the previous runs. Some changes were made to the syntax to accommodate for the use of Cifar100 and Optuna was utilized to optimize certain hyperparameters. A smaller amount of trials were utilized though Optuna as I was running out of Collab processing power but regardless a solid result was achieved. The training accuracy ended at 41% with the validation accuracy ending at almost 50%. The performance was lesser than that of the trained ResNet as expected.



```
Study statistics:
  Number of finished trials:  3
  Number of pruned trials:  0
  Number of complete trials:  3
Best trial:
  Value:  0.4982999861240387
  Params:
    weight_decay: 4.7334492434534885e-09
    optimizer: Adam
    adam_learning_rate: 0.00041905818404104445
```

Figure 12: Optuna training results

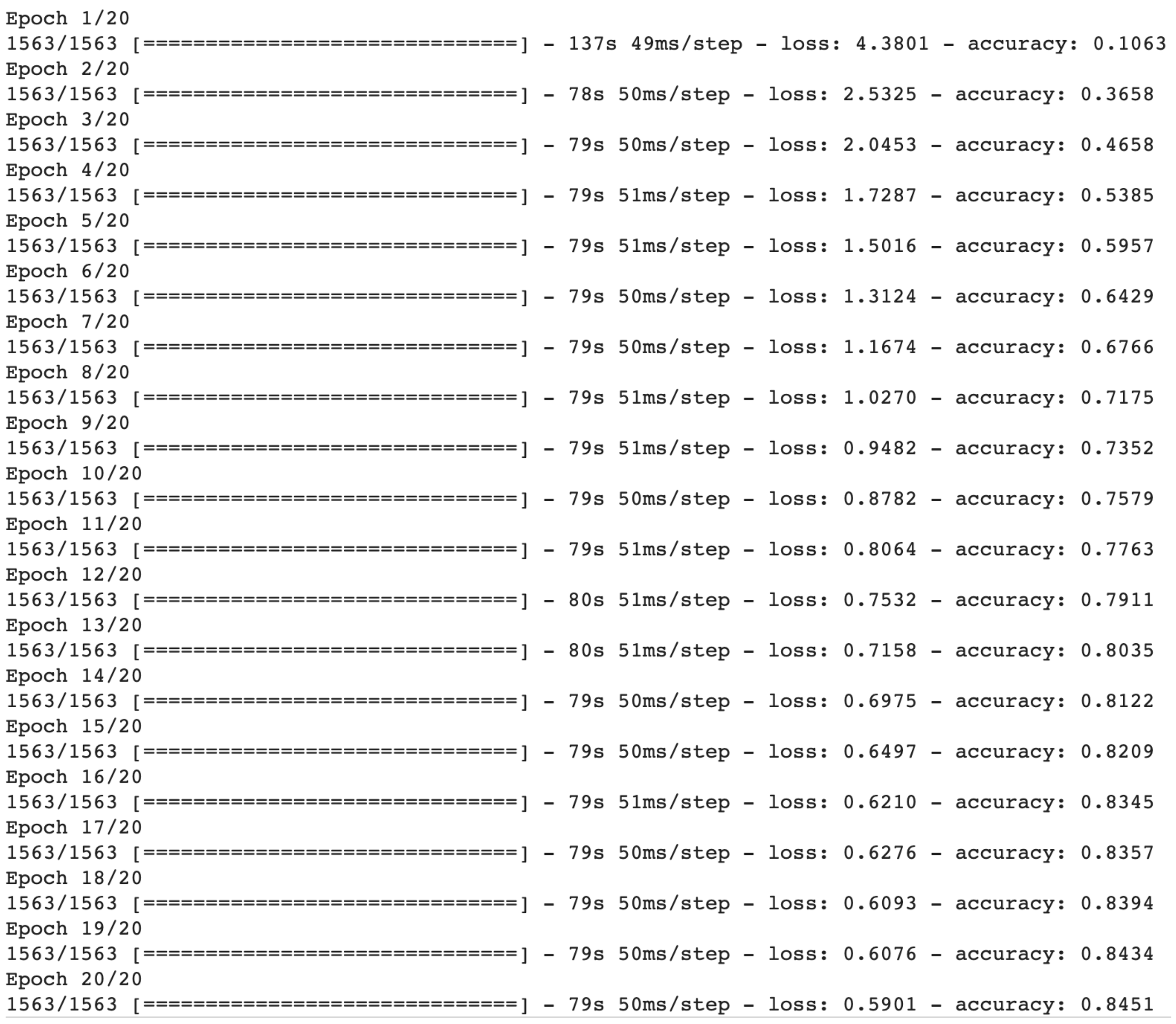


```
Epoch 1/20
1563/1563 [==============================] - 33s 19ms/step - loss: 5.2186 - accuracy: 0.0351 - val_loss: 3.6665 - val_accuracy: 0.1489
Epoch 2/20
1563/1563 [==============================] - 31s 20ms/step - loss: 3.8641 - accuracy: 0.1070 - val_loss: 3.3932 - val_accuracy: 0.1898
Epoch 3/20
1563/1563 [==============================] - 29s 18ms/step - loss: 3.5573 - accuracy: 0.1523 - val_loss: 3.1406 - val_accuracy: 0.2500
Epoch 4/20
1563/1563 [==============================] - 30s 19ms/step - loss: 3.3281 - accuracy: 0.1930 - val_loss: 2.9734 - val_accuracy: 0.2711
Epoch 5/20
1563/1563 [==============================] - 30s 19ms/step - loss: 3.1466 - accuracy: 0.2231 - val_loss: 2.7613 - val_accuracy: 0.3128
Epoch 6/20
1563/1563 [==============================] - 30s 19ms/step - loss: 3.0001 - accuracy: 0.2529 - val_loss: 2.6135 - val_accuracy: 0.3370
Epoch 7/20
1563/1563 [==============================] - 31s 20ms/step - loss: 2.8809 - accuracy: 0.2748 - val_loss: 2.5295 - val_accuracy: 0.3577
Epoch 8/20
1563/1563 [==============================] - 30s 19ms/step - loss: 2.7947 - accuracy: 0.2907 - val_loss: 2.4957 - val_accuracy: 0.3580
Epoch 9/20
1563/1563 [==============================] - 29s 19ms/step - loss: 2.7154 - accuracy: 0.3048 - val_loss: 2.3558 - val_accuracy: 0.3889
Epoch 10/20
1563/1563 [==============================] - 31s 20ms/step - loss: 2.6450 - accuracy: 0.3204 - val_loss: 2.2931 - val_accuracy: 0.4007
Epoch 11/20
1563/1563 [==============================] - 29s 19ms/step - loss: 2.5873 - accuracy: 0.3316 - val_loss: 2.2467 - val_accuracy: 0.4145
Epoch 12/20
1563/1563 [==============================] - 31s 20ms/step - loss: 2.5292 - accuracy: 0.3424 - val_loss: 2.2583 - val_accuracy: 0.4056
Epoch 13/20
1563/1563 [==============================] - 29s 19ms/step - loss: 2.4661 - accuracy: 0.3565 - val_loss: 2.1372 - val_accuracy: 0.4396
Epoch 14/20
1563/1563 [==============================] - 30s 19ms/step - loss: 2.4232 - accuracy: 0.3628 - val_loss: 2.2607 - val_accuracy: 0.4075
Epoch 15/20
1563/1563 [==============================] - 29s 19ms/step - loss: 2.3589 - accuracy: 0.3788 - val_loss: 2.0035 - val_accuracy: 0.4703
Epoch 16/20
1563/1563 [==============================] - 31s 20ms/step - loss: 2.3219 - accuracy: 0.3815 - val_loss: 1.9962 - val_accuracy: 0.4701
Epoch 17/20
1563/1563 [==============================] - 29s 19ms/step - loss: 2.2811 - accuracy: 0.3921 - val_loss: 1.9697 - val_accuracy: 0.4785
Epoch 18/20
1563/1563 [==============================] - 30s 19ms/step - loss: 2.2293 - accuracy: 0.4093 - val_loss: 1.9404 - val_accuracy: 0.4766
Epoch 19/20
1563/1563 [==============================] - 31s 20ms/step - loss: 2.2023 - accuracy: 0.4094 - val_loss: 1.9654 - val_accuracy: 0.4725
Epoch 20/20
1563/1563 [==============================] - 30s 19ms/step - loss: 2.1615 - accuracy: 0.4192 - val loss: 1.8817 - val accuracy: 0.4983
```

Figure 13: Assignment 2 Accuracy and Loss 20 Epoch Output

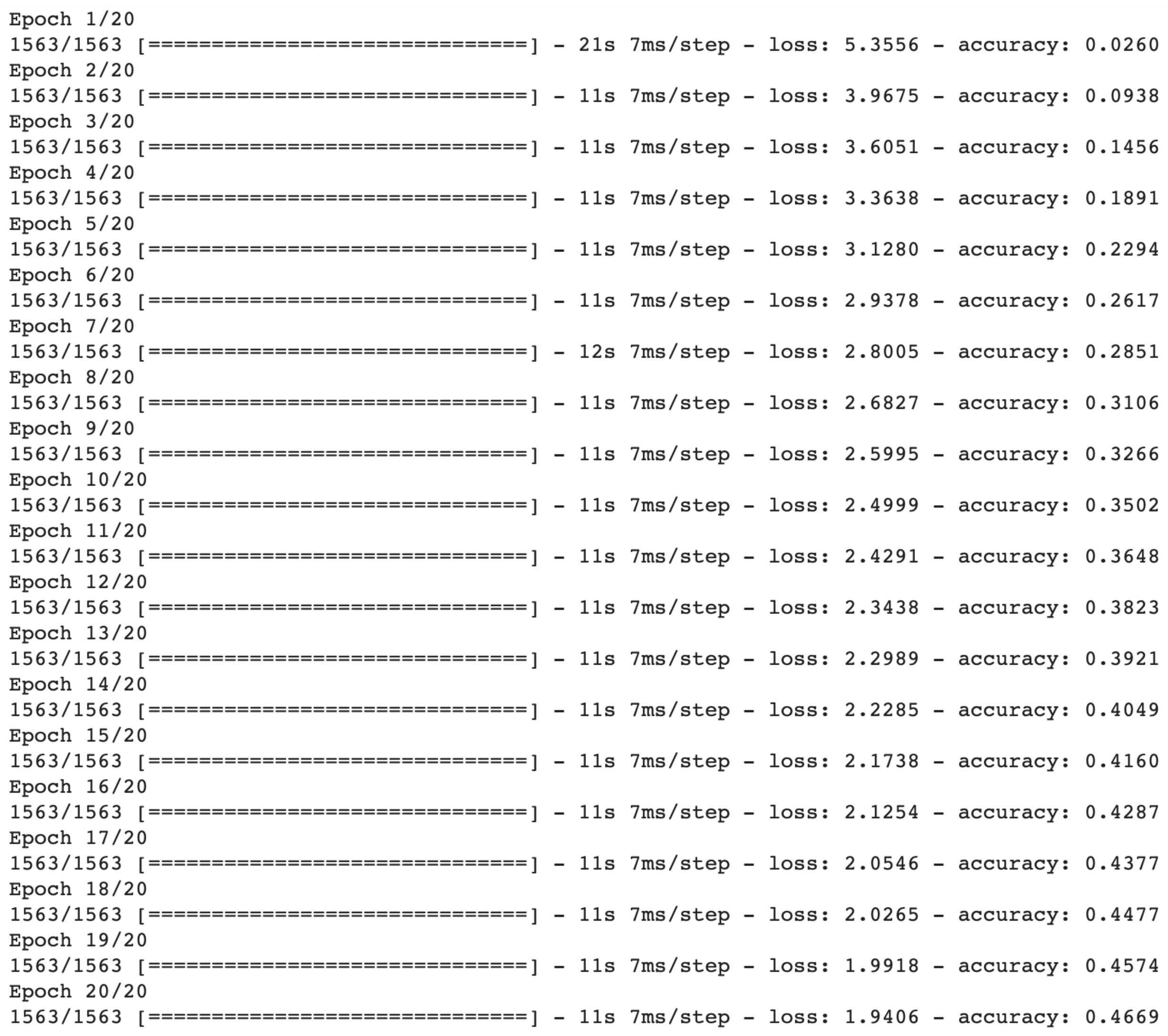
<u>Ensemble Model</u>

The ensemble model was run on the Cifar100 dataset at 20 epochs for each model. Both models performed much better than their previous counterparts when run in ensemble. The Resnet50 model had a final training accuracy of 84% or almost a 20% increase from the previous run. The Assignment 2 model has a final training accuracy of 46% or over a 6% increase from the previous run. The final average value was 54% when in ensemble.



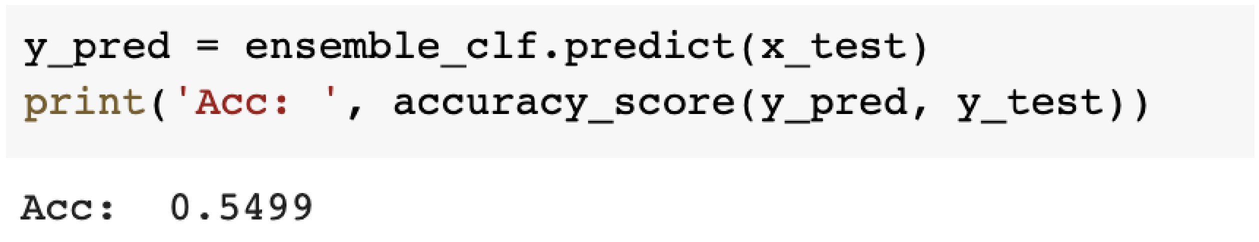
```
Epoch 1/20
1563/1563 [==============================] - 137s 49ms/step - loss: 4.3801 - accuracy: 0.1063
Epoch 2/20
1563/1563 [==============================] - 78s 50ms/step - loss: 2.5325 - accuracy: 0.3658
Epoch 3/20
1563/1563 [==============================] - 79s 50ms/step - loss: 2.0453 - accuracy: 0.4658
Epoch 4/20
1563/1563 [==============================] - 79s 51ms/step - loss: 1.7287 - accuracy: 0.5385
Epoch 5/20
1563/1563 [==============================] - 79s 51ms/step - loss: 1.5016 - accuracy: 0.5957
Epoch 6/20
1563/1563 [==============================] - 79s 50ms/step - loss: 1.3124 - accuracy: 0.6429
Epoch 7/20
1563/1563 [==============================] - 79s 50ms/step - loss: 1.1674 - accuracy: 0.6766
Epoch 8/20
1563/1563 [==============================] - 79s 51ms/step - loss: 1.0270 - accuracy: 0.7175
Epoch 9/20
1563/1563 [==============================] - 79s 51ms/step - loss: 0.9482 - accuracy: 0.7352
Epoch 10/20
1563/1563 [==============================] - 79s 50ms/step - loss: 0.8782 - accuracy: 0.7579
Epoch 11/20
1563/1563 [==============================] - 79s 51ms/step - loss: 0.8064 - accuracy: 0.7763
Epoch 12/20
1563/1563 [==============================] - 80s 51ms/step - loss: 0.7532 - accuracy: 0.7911
Epoch 13/20
1563/1563 [==============================] - 80s 51ms/step - loss: 0.7158 - accuracy: 0.8035
Epoch 14/20
1563/1563 [==============================] - 79s 50ms/step - loss: 0.6975 - accuracy: 0.8122
Epoch 15/20
1563/1563 [==============================] - 79s 50ms/step - loss: 0.6497 - accuracy: 0.8209
Epoch 16/20
1563/1563 [==============================] - 79s 51ms/step - loss: 0.6210 - accuracy: 0.8345
Epoch 17/20
1563/1563 [==============================] - 79s 50ms/step - loss: 0.6276 - accuracy: 0.8357
Epoch 18/20
1563/1563 [==============================] - 79s 50ms/step - loss: 0.6093 - accuracy: 0.8394
Epoch 19/20
1563/1563 [==============================] - 79s 50ms/step - loss: 0.6076 - accuracy: 0.8434
Epoch 20/20
1563/1563 [==============================] - 79s 50ms/step - loss: 0.5901 - accuracy: 0.8451
```

<u>Figure 14: Trained ResNet Ensemble Accuracy and Loss 20 Epoch Output</u>



```
Epoch 1/20
1563/1563 [==============================] - 21s 7ms/step - loss: 5.3556 - accuracy: 0.0260
Epoch 2/20
1563/1563 [==============================] - 11s 7ms/step - loss: 3.9675 - accuracy: 0.0938
Epoch 3/20
1563/1563 [==============================] - 11s 7ms/step - loss: 3.6051 - accuracy: 0.1456
Epoch 4/20
1563/1563 [==============================] - 11s 7ms/step - loss: 3.3638 - accuracy: 0.1891
Epoch 5/20
1563/1563 [==============================] - 11s 7ms/step - loss: 3.1280 - accuracy: 0.2294
Epoch 6/20
1563/1563 [==============================] - 11s 7ms/step - loss: 2.9378 - accuracy: 0.2617
Epoch 7/20
1563/1563 [==============================] - 12s 7ms/step - loss: 2.8005 - accuracy: 0.2851
Epoch 8/20
1563/1563 [==============================] - 11s 7ms/step - loss: 2.6827 - accuracy: 0.3106
Epoch 9/20
1563/1563 [==============================] - 11s 7ms/step - loss: 2.5995 - accuracy: 0.3266
Epoch 10/20
1563/1563 [==============================] - 11s 7ms/step - loss: 2.4999 - accuracy: 0.3502
Epoch 11/20
1563/1563 [==============================] - 11s 7ms/step - loss: 2.4291 - accuracy: 0.3648
Epoch 12/20
1563/1563 [==============================] - 11s 7ms/step - loss: 2.3438 - accuracy: 0.3823
Epoch 13/20
1563/1563 [==============================] - 11s 7ms/step - loss: 2.2989 - accuracy: 0.3921
Epoch 14/20
1563/1563 [==============================] - 11s 7ms/step - loss: 2.2285 - accuracy: 0.4049
Epoch 15/20
1563/1563 [==============================] - 11s 7ms/step - loss: 2.1738 - accuracy: 0.4160
Epoch 16/20
1563/1563 [==============================] - 11s 7ms/step - loss: 2.1254 - accuracy: 0.4287
Epoch 17/20
1563/1563 [==============================] - 11s 7ms/step - loss: 2.0546 - accuracy: 0.4377
Epoch 18/20
1563/1563 [==============================] - 11s 7ms/step - loss: 2.0265 - accuracy: 0.4477
Epoch 19/20
1563/1563 [==============================] - 11s 7ms/step - loss: 1.9918 - accuracy: 0.4574
Epoch 20/20
1563/1563 [==============================] - 11s 7ms/step - loss: 1.9406 - accuracy: 0.4669
```

Figure 15: Assignment 2 Ensemble Accuracy and Loss 20 Epoch Output



```
y_pred = ensemble_clf.predict(x_test)
print('Acc: ', accuracy_score(y_pred, y_test))
```

```
Acc:  0.5499
```

Figure 16: Ensemble Accuracy Output

Conclusion:

In conclusion this assignment was an astounding success with all hypotheses being proven correct. The trained ResNet50 performed better than the untrained and previous DCNN from assignment 2. This was because it was much deeper, wider and had the benefit of being pretrained. Finally the ensemble model saw both models perform better in the ensemble setting with a final average accuracy of 54% between the two models.