CS 4372
ASSIGNMENT 3

Names of students in your group:
James Hooper
Hritik Panchasara

Number of free late days used: 0

Note: You are allowed a **total** of 4 free late days for the **entire semester**. You can use at most 2 for each assignment. After that, there will be a penalty of 10% for each late day.

Please list clearly all the sources/references that you have used in this assignment.

For model creation and image altering
https://www.tensorflow.org/
https://www.kite.com/python/docs/keras.backend.moving_averages.distribution_strategy_context.distribute_lib.dataset_ops.BatchDataset
For graph print outs
https://matplotlib.org/
For data manipulation
https://numpy.org/

LOG & REPORT

Explanation & Analysis of the Log File

- The log file has the regular print outs from the actual code. There are differences between each entry as needed to explain the parameter changes. Here is a quick overview of the results. The first entry was done with the epochs = 4, steps per epoch & validation steps set to default, weights = 'imagenet' (this isn't explicitly stated since it is the assumed default for the base model), and with no Dense layer added. This entry ended up having a 96.875% accuracy over the test dataset and is the main proof that the Dense layer isn't required for great results. This of course sacrifices the amount of time needed to train by a lot. The majority of the log file is random parameter changes that ended with mediocre to bad results compared to this initial entry. The main change comes when the Dense layer & a second dropout layer is finally added. At this point the model and parameters can reach 100% accuracy for the train accuracy, the validation accuracy, and the test accuracy. The very final entry showcases just how quick we can achieve this tuning. The second to last entry is an example of what happens if the weights are not initialized with the imagenet weights for the base model. Even with a Dense layer and second dropout layer the results become abysmal.
- The key takeaway is that the best results come from the base model weights being initialized with the imagenet weights & having a Dense layer alongside a second Dropout layer (all dropout layers had a 20% rate). The last entry showcases that with only 3 epochs, 8 steps per epoch, and 7 validation steps were needed for 100% accuracy results. Of course, with better fine tuning these results can be achieved most likely in a far quicker pace.
- This is also a testament to how powerful transferring learning is, that within 30 minutes and a few extra layers you can achieve a very powerful specific model.

Quick Notes and Assumptions

- The dataset originally had 1027 images belonging to 2 classes for the training dataset and 256 images belonging to 2 classes for the validation dataset.
- Each batch has 32 images.
- The training dataset originally had 33 batches and validation dataset had 8 batches.
- By augmenting the training dataset increased to be 3081 images thus increasing the batch size to be 99.
- The validation dataset was split into a validation dataset and a test dataset. The validation dataset has 7 batches whilst the test dataset has 1 batch.
- Instead of the 25 required test images, this report does 32 due to this.
- The shape for the images were (300, 300, 3).
- There is a layer specifically for rescaling the values for the model such that they are a value between 0 and 1 ([0,1]).
- The major layers added to the model were:
    - Preprocess layer
    - VGG16 base model layers

    - Dropout (.2)
    - Dense (4096, relu)
    - Dropout (.2)
    - Dense (2, softmax) (predictions)
- The Adam Optimizer was used in its default standard.
- Sparse Categorical Cross Entropy was used as the loss function.

4th to Last Log Info & Images

Model: "functional_5"

_____

Layer (type) Output Shape Param #

==========

input_6 (InputLayer) [(None, 300, 300, 3)] 0

_____

tf_op_layer_strided_slice_2 [(None, 300, 300, 3)] 0

_____

tf_op_layer_BiasAdd_2 (Tenso [(None, 300, 300, 3)] 0

_____

vgg16 (Functional) (None, 512) 14714688

_____

dropout_2 (Dropout) (None, 512) 0

_____

FC1 (Dense) (None, 4096) 2101248

_____

dropout_3 (Dropout) (None, 4096) 0

_____

Predictions (Dense) (None, 2) 8194

==========

Total params: 16,824,130

Trainable params: 2,109,442

Non-trainable params: 14,714,688

epochs = 4

steps per epoch = 8

validation steps = 7

```
inputs = tf.keras.Input(shape=(300,300,3))
x = preprocess_input(inputs)
x = base_model_VGG16(x, training=False)
x = tf.keras.layers.Dropout(0.2)(x)
x = Dense(4096, activation ='relu', name = 'FC1')(x)
x = tf.keras.layers.Dropout(0.2)(x)
x = tf.keras.layers.Dense(2, activation='softmax', name='Predictions')(x)
```

```
Epoch 1/4
8/8 [==============================] - 397s 50s/step - loss: 0.3553 - accuracy: 0.8750 - val_loss: 0.0521 - val_accuracy: 0.9955
Epoch 2/4
8/8 [==============================] - 401s 50s/step - loss: 0.0199 - accuracy: 0.9922 - val_loss: 0.0235 - val_accuracy: 0.9955
Epoch 3/4
8/8 [==============================] - 400s 50s/step - loss: 0.0035 - accuracy: 1.0000 - val_loss: 0.0147 - val_accuracy: 1.0000
Epoch 4/4
8/8 [==============================] - 396s 49s/step - loss: 0.0026 - accuracy: 1.0000 - val_loss: 0.0093 - val_accuracy: 1.0000
```

```
Prediction 0 0.0048591164 0.9951409
Prediction 1 0.029711256 0.97028875
Prediction 2 0.002500701 0.9974993
Prediction 3 0.99965525 0.0003447523
Prediction 4 0.99999726 2.7762287e-06
Prediction 5 1.0 1.6306474e-12
Prediction 6 0.014945514 0.9850545
Prediction 7 0.0024700817 0.9975299
Prediction 8 0.003939209 0.9960608
Prediction 9 1.0 1.5518793e-11
```

Prediction 10 0.99999905 1.0103028e-06

Prediction 11 0.007779476 0.99222046

Prediction 12 1.0 3.532495e-16

Prediction 13 0.013256529 0.98674345

Prediction 14 0.015739975 0.98426

Prediction 15 1.0 2.2785677e-12

Prediction 16 0.0037829229 0.9962171

Prediction 17 0.0015228769 0.99847716

Prediction 18 1.0 1.7232107e-08

Prediction 19 0.010636321 0.9893637

Prediction 20 0.002325197 0.99767476

Prediction 21 1.0 8.262565e-10

Prediction 22 0.03573277 0.9642672

Prediction 23 0.0025446173 0.9974554

Prediction 24 1.0 6.3280964e-11

Prediction 25 1.0 1.729644e-13

Prediction 26 0.006355565 0.9936445

Prediction 27 0.0021700521 0.99783

Prediction 28 1.0 1.2412626e-08

Prediction 29 0.9999367 6.327872e-05

Prediction 30 1.0 7.471545e-13

Prediction 31 0.012729587 0.9872705

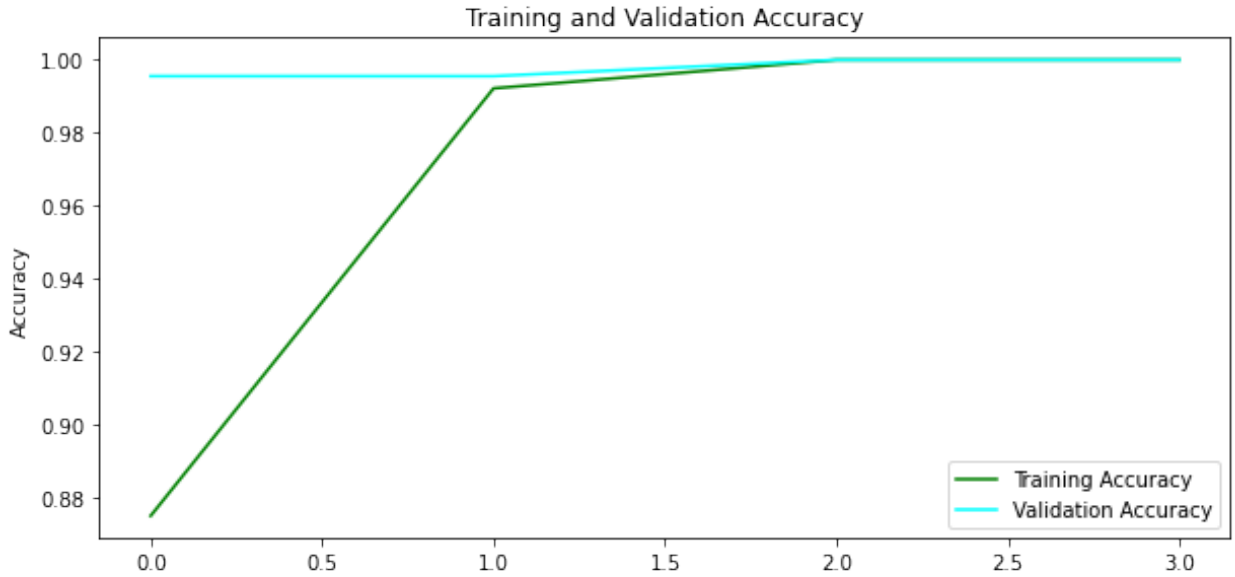
Predictions for Test Dataset:

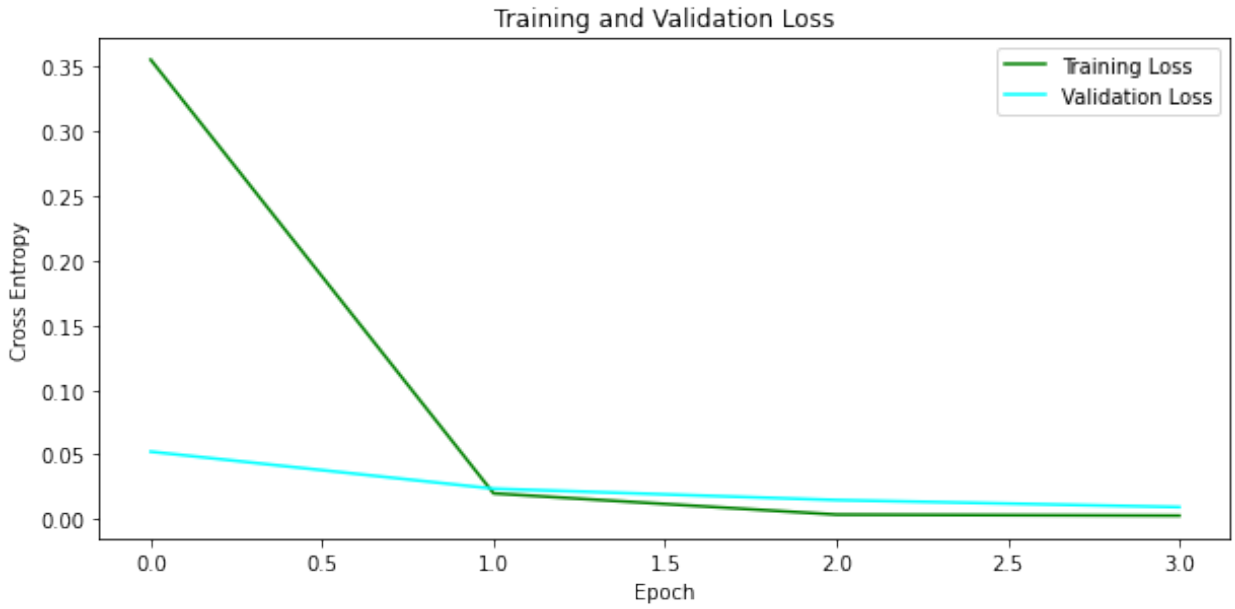
[1 1 1 0 0 0 1 1 1 0 0 1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 0 0 1]

Labels:

[1 1 1 0 0 0 1 1 1 0 0 1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 0 0 1]

Percent Correct: 100.0 %


Training and Validation Accuracy
Accuracy
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Training Accuracy
Validation Accuracy


Training and Validation Loss
Cross Entropy
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Epoch
Training Loss
Validation Loss

- As we can see with the prediction data for this log the results of the model are pretty sure of the class that an image belongs to. The predictions here are obviously from the softmax layer for all 32 test images (1 whole batch). This ‘confidence’ showcases just how accurate this model is.
- It can also be assumed that since the humans are slender figures, the only inaccuracy for horse predictions is if they were looking straight ahead hiding their mass & apparent 4 legs. This can be the assumed learned structures by the model. Similar to how humans perceive it: a horse is a 4 legged rather large creature, whilst a humanoid (obvious but still) figure is a human.