```
# Imports

import os, shutil
import pandas as pd

from sklearn.model_selection import train_test_split

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import plot_model
from tensorflow.keras.utils import image_dataset_from_directory


import numpy as np
import matplotlib.pyplot as plt
```

Note: Throughout this, I'll be omitting some of the output to make the PDF shorter

```
def data_build():
    '''
    It seems that all of the images have different dimensions.
    I will use the resize_with_pad to push everything to the largest m
x n dim,
    and then I will pool down from there with the goal of the padding
    pixels falling out in the CNN.
    '''
    image_h = 500
    image_w = 500
    batch_size = 16  #GPU Saturated and memory issues if I go to 32...

    # location on disk of the image data
    loc =
'C:/Users/btb51/Documents/GitHub/DeepLearning_DAAN570/DAAN570_Instruct
or_Sample_Codes/Lesson_08_Code/Assignment2_ZooClassifier/Zoo
Classifier project - images/images'

    #datasets will be a tuple of the train and validation data
    train_ds, val_ds = image_dataset_from_directory(loc,
                                  batch_size=batch_size,
                                  image_size = (image_h,image_w),  #
set as largest dims
                                  shuffle = True,
                                  seed = 570,
                                  validation_split = 0.2,
                                  subset = 'both')

    return train_ds, val_ds

train_ds, val_ds = data_build()
class_names = train_ds.class_names
num_classes = len(class_names)
```

```
print(class_names, num_classes)
```

```
Found 3000 files belonging to 3 classes.
Using 2400 files for training.
Using 600 files for validation.
['cats', 'dogs', 'panda'] 3
```

```
#Check for balanced dataset; maybe this is giving me my problem
def balance(val_ds):

    cat = 0
    dog =0
    pan = 0
    for _, labels in val_ds:
        for each in labels:
            if each == 0:
               cat = cat +1
            elif each ==1:
                dog = dog +1
            elif each ==2:
                pan = pan +1
    print("Cats, dogs, pandas")
    print(cat, dog, pan)

#datasets are balanced... this is not the problem

balance(train_ds)
balance(val_ds)
```

```
Cats, dogs, pandas
810 798 792
Cats, dogs, pandas
190 202 208
```

```
def plot_figs(train_ds):
    plt.figure(figsize=(10, 10))
    for images, labels in train_ds.take(1):
      for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(int(labels[i]))
        plt.axis("on")


plot_figs(train_ds)
```

```
def cnn_net(num_classes):

    #Build a Generic CNN with a set number of classes as the
classifing output

    net = Sequential([
    tf.keras.layers.Rescaling(1./255, input_shape=(500, 500, 3)),
    tf.keras.layers.Conv2D(16,3,padding='same', activation = 'relu'),
    tf.keras.layers.MaxPool2D(),
    tf.keras.layers.Conv2D(32, 3, padding='same', activation =
'relu'),
    tf.keras.layers.MaxPool2D(),
    tf.keras.layers.Conv2D(64, 3, padding='same', activation= 'relu'),
    tf.keras.layers.MaxPool2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation = 'relu'),
```

```
    tf.keras.layers.Dense(num_classes)
    ])

    return net

net_first = cnn_net(num_classes)

lr = 1e-3
optimizer = tf.keras.optimizers.Adam(learning_rate=lr)
loss_fn =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metrics = ['accuracy']

net_first.compile(optimizer=optimizer, loss=loss_fn, metrics=metrics)

net_first.summary()
```

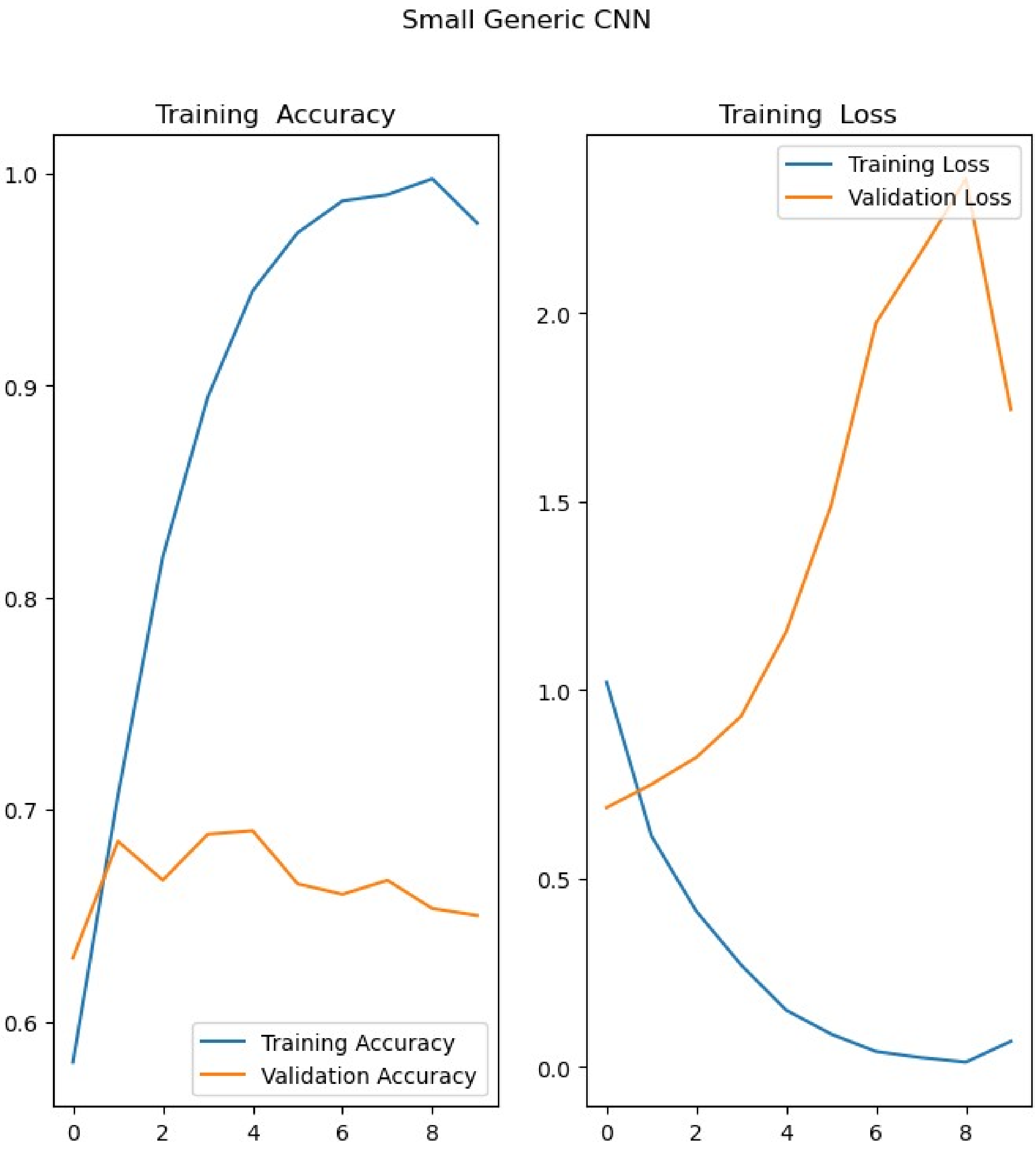
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 500, 500, 3)       0

 conv2d (Conv2D)             (None, 500, 500, 16)      448

 max_pooling2d (MaxPooling2D  (None, 250, 250, 16)     0
 )

 conv2d_1 (Conv2D)           (None, 250, 250, 32)      4640

 max_pooling2d_1 (MaxPooling  (None, 125, 125, 32)     0
 2D)

 conv2d_2 (Conv2D)           (None, 125, 125, 64)      18496

 max_pooling2d_2 (MaxPooling  (None, 62, 62, 64)       0
 2D)

 flatten (Flatten)           (None, 246016)            0

 dense (Dense)               (None, 128)               31490176

 dense_1 (Dense)             (None, 3)                 387

=================================================================
Total params: 31,514,147
Trainable params: 31,514,147
Non-trainable params: 0
_________________________________________________________________
```

```python
epochs = 10

history_first = net_first.fit(train_ds,
                              validation_data=val_ds,
                              epochs=epochs)
```

```
Epoch 1/10
150/150 [==============================] - 23s 103ms/step - loss:
1.0206 - accuracy: 0.5808 - val_loss: 0.6882 - val_accuracy: 0.6300
Epoch 2/10
150/150 [==============================] - 14s 94ms/step - loss:
0.6127 - accuracy: 0.7067 - val_loss: 0.7497 - val_accuracy: 0.6850
Epoch 3/10
150/150 [==============================] - 16s 106ms/step - loss:
0.4128 - accuracy: 0.8192 - val_loss: 0.8214 - val_accuracy: 0.6667
Epoch 4/10
150/150 [==============================] - 20s 129ms/step - loss:
0.2703 - accuracy: 0.8946 - val_loss: 0.9310 - val_accuracy: 0.6883
Epoch 5/10
150/150 [==============================] - 19s 123ms/step - loss:
0.1504 - accuracy: 0.9446 - val_loss: 1.1552 - val_accuracy: 0.6900
Epoch 6/10
150/150 [==============================] - 16s 106ms/step - loss:
0.0871 - accuracy: 0.9721 - val_loss: 1.4893 - val_accuracy: 0.6650
Epoch 7/10
150/150 [==============================] - 17s 109ms/step - loss:
0.0410 - accuracy: 0.9871 - val_loss: 1.9740 - val_accuracy: 0.6600
Epoch 8/10
150/150 [==============================] - 17s 108ms/step - loss:
0.0248 - accuracy: 0.9900 - val_loss: 2.1605 - val_accuracy: 0.6667
Epoch 9/10
150/150 [==============================] - 16s 105ms/step - loss:
0.0126 - accuracy: 0.9975 - val_loss: 2.3563 - val_accuracy: 0.6533
Epoch 10/10
150/150 [==============================] - 16s 107ms/step - loss:
0.0681 - accuracy: 0.9767 - val_loss: 1.7443 - val_accuracy: 0.6500
```

```python
def plot_acc_metric(history, epochs, title:str):

    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']

    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs_range = range(epochs)

    plt.figure(figsize=(8, 8))

    plt.suptitle(title)
```

```
    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, acc, label='Training Accuracy')
    plt.plot(epochs_range, val_acc, label='Validation Accuracy')
    plt.legend(loc='lower right')
    plt.title('Training  Accuracy')

    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, loss, label='Training Loss')
    plt.plot(epochs_range, val_loss, label='Validation Loss')
    plt.legend(loc='upper right')
    plt.title('Training  Loss')
    plt.show()

plot_acc_metric(history_first, 'Small Generic CNN')
```



This looks like it is overfitting here. We see the training accuracy climb while the validation does nothing for accuracy. Furthermore, we see the training and validation loss diverge which typically indicates an overfit.

Maybe some data augmentation will help me out here.

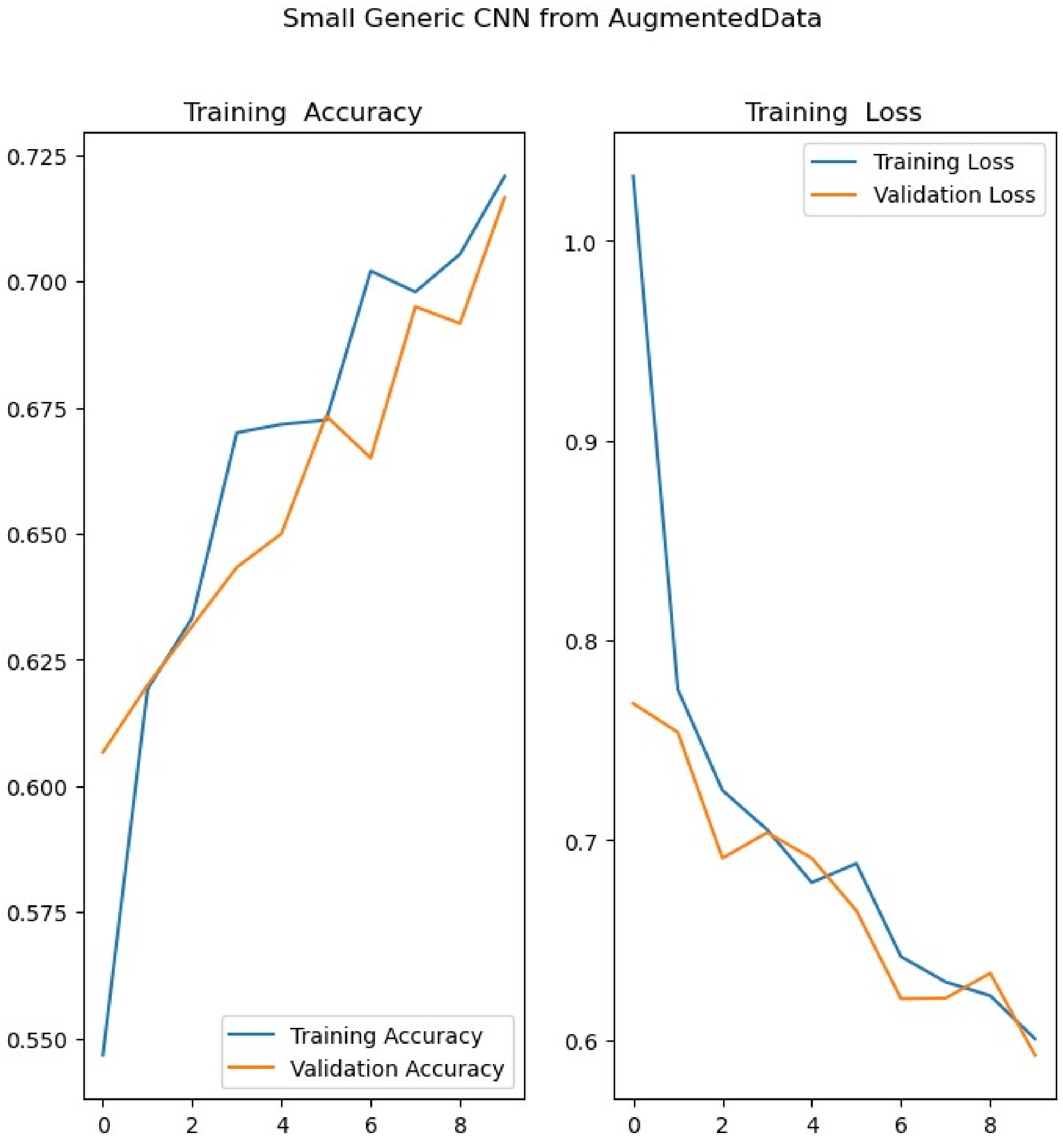
```
# Data Augment layers

data_augment = tf.keras.Sequential([
    tf.keras.layers.RandomFlip(mode="horizontal_and_vertical"),
    tf.keras.layers.RandomRotation(0.5)
])
```

```
# Note: I had tested out a random crop at 256 x 256 pixels but it
yielded no relevant change to the metrics.

# Run the data_augment function on each image with its label
# NOTE: I am NOT augmenting the label!
# NOTE 2: I am doing image normalization in the CNN itself
train_ds_aug = train_ds.map(
    lambda image, label: (data_augment(image), label),
    num_parallel_calls=tf.data.AUTOTUNE)

# I'm going to have to call this a lot


def trainer(net, train_ds, val_ds, epochs):

    lr = 1e-3
    optimizer = tf.keras.optimizers.Adam(learning_rate=lr)
    loss_fn =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
    metrics = ['accuracy']

    net.compile(optimizer=optimizer, loss=loss_fn, metrics=metrics)

    history = net.fit(train_ds, validation_data=val_ds, epochs=epochs)

    return history

net_2 = cnn_net(num_classes)
# forgive the misspelling of history...
hisotry_2 = trainer(net_2, train_ds_aug, val_ds, epochs=10)
```

```
Epoch 1/10
150/150 [==============================] - 41s 266ms/step - loss:
1.0323 - accuracy: 0.5467 - val_loss: 0.7684 - val_accuracy: 0.6067
Epoch 2/10
150/150 [==============================] - 41s 271ms/step - loss:
0.7754 - accuracy: 0.6192 - val_loss: 0.7540 - val_accuracy: 0.6200
Epoch 3/10
150/150 [==============================] - 40s 264ms/step - loss:
0.7249 - accuracy: 0.6333 - val_loss: 0.6909 - val_accuracy: 0.6317
Epoch 4/10
150/150 [==============================] - 40s 261ms/step - loss:
0.7052 - accuracy: 0.6700 - val_loss: 0.7037 - val_accuracy: 0.6433
Epoch 5/10
150/150 [==============================] - 41s 268ms/step - loss:
0.6787 - accuracy: 0.6717 - val_loss: 0.6910 - val_accuracy: 0.6500
Epoch 6/10
150/150 [==============================] - 42s 274ms/step - loss:
0.6882 - accuracy: 0.6725 - val_loss: 0.6647 - val_accuracy: 0.6733
Epoch 7/10
150/150 [==============================] - 41s 270ms/step - loss:
0.6417 - accuracy: 0.7021 - val_loss: 0.6206 - val_accuracy: 0.6650
```

```
Epoch 8/10
150/150 [==============================] - 40s 262ms/step - loss:
0.6290 - accuracy: 0.6979 - val_loss: 0.6209 - val_accuracy: 0.6950
Epoch 9/10
150/150 [==============================] - 39s 254ms/step - loss:
0.6221 - accuracy: 0.7054 - val_loss: 0.6334 - val_accuracy: 0.6917
Epoch 10/10
150/150 [==============================] - 40s 265ms/step - loss:
0.6005 - accuracy: 0.7208 - val_loss: 0.5924 - val_accuracy: 0.7167
```

```python
plot_acc_metric(hisotry_2, 'Small Generic CNN from AugmentedData') # and the misspelling of history is complete
```



At this point, it seems as if I am not getting better training and overfitting has been reduced. The validation loss is following the training loss.

I am going to run this for a longer duration and see what happens.

```
# Build the same net as last time and run for more epochs to see if
training improves
net_3 = cnn_net(num_classes)
# forgive the misspelling of history...
history_3 = trainer(net_3, train_ds_aug, val_ds, epochs=20)
```

```
Epoch 1/20
150/150 [==============================] - 41s 261ms/step - loss:
1.1542 - accuracy: 0.5600 - val_loss: 0.7600 - val_accuracy: 0.6350
Epoch 2/20
150/150 [==============================] - 44s 288ms/step - loss:
0.7553 - accuracy: 0.6296 - val_loss: 0.7093 - val_accuracy: 0.6367
Epoch 3/20
150/150 [==============================] - 44s 290ms/step - loss:
0.7269 - accuracy: 0.6379 - val_loss: 0.7296 - val_accuracy: 0.6567
Epoch 4/20
150/150 [==============================] - 43s 285ms/step - loss:
0.6968 - accuracy: 0.6654 - val_loss: 0.6646 - val_accuracy: 0.6867
Epoch 5/20
150/150 [==============================] - 42s 274ms/step - loss:
0.6801 - accuracy: 0.6837 - val_loss: 0.6741 - val_accuracy: 0.6883
Epoch 6/20
150/150 [==============================] - 46s 302ms/step - loss:
0.6693 - accuracy: 0.6908 - val_loss: 0.7060 - val_accuracy: 0.6683
Epoch 7/20
150/150 [==============================] - 40s 261ms/step - loss:
0.6652 - accuracy: 0.6883 - val_loss: 0.7425 - val_accuracy: 0.6650
Epoch 8/20
150/150 [==============================] - 46s 301ms/step - loss:
0.6530 - accuracy: 0.7050 - val_loss: 0.6806 - val_accuracy: 0.6833
Epoch 9/20
150/150 [==============================] - 43s 277ms/step - loss:
0.6294 - accuracy: 0.7021 - val_loss: 0.8065 - val_accuracy: 0.6783
Epoch 10/20
150/150 [==============================] - 42s 274ms/step - loss:
0.6068 - accuracy: 0.7088 - val_loss: 0.6495 - val_accuracy: 0.7017
Epoch 11/20
150/150 [==============================] - 43s 281ms/step - loss:
0.6105 - accuracy: 0.7217 - val_loss: 0.6118 - val_accuracy: 0.7233
Epoch 12/20
150/150 [==============================] - 43s 280ms/step - loss:
0.5872 - accuracy: 0.7283 - val_loss: 0.6325 - val_accuracy: 0.7233
Epoch 13/20
150/150 [==============================] - 44s 287ms/step - loss:
0.5713 - accuracy: 0.7325 - val_loss: 0.6147 - val_accuracy: 0.7200
Epoch 14/20
```

```
150/150 [==============================] - 45s 294ms/step - loss:
0.5535 - accuracy: 0.7462 - val_loss: 0.6493 - val_accuracy: 0.7183
Epoch 15/20
150/150 [==============================] - 40s 260ms/step - loss:
0.5878 - accuracy: 0.7387 - val_loss: 0.6171 - val_accuracy: 0.7183
Epoch 16/20
150/150 [==============================] - 42s 273ms/step - loss:
0.5481 - accuracy: 0.7450 - val_loss: 0.6408 - val_accuracy: 0.7167
Epoch 17/20
150/150 [==============================] - 41s 270ms/step - loss:
0.5563 - accuracy: 0.7408 - val_loss: 0.6084 - val_accuracy: 0.7283
Epoch 18/20
150/150 [==============================] - 41s 266ms/step - loss:
0.5420 - accuracy: 0.7554 - val_loss: 0.5826 - val_accuracy: 0.7483
Epoch 19/20
150/150 [==============================] - 41s 271ms/step - loss:
0.5544 - accuracy: 0.7558 - val_loss: 0.5850 - val_accuracy: 0.7333
Epoch 20/20
150/150 [==============================] - 43s 279ms/step - loss:
0.5405 - accuracy: 0.7487 - val_loss: 0.5644 - val_accuracy: 0.7467
```

```
plot_acc_metric(history_3, epochs=20, title='Small Generic CNN from
AugmentedData; 20 Epochs')
```

Well that continued to get better.

Let's see if I can make a learning rate schedule or the other option is to use Optuna to optimize a static learning rate.

Note to the astute reader, I have skipped net_4 and have moved directly on to net_5

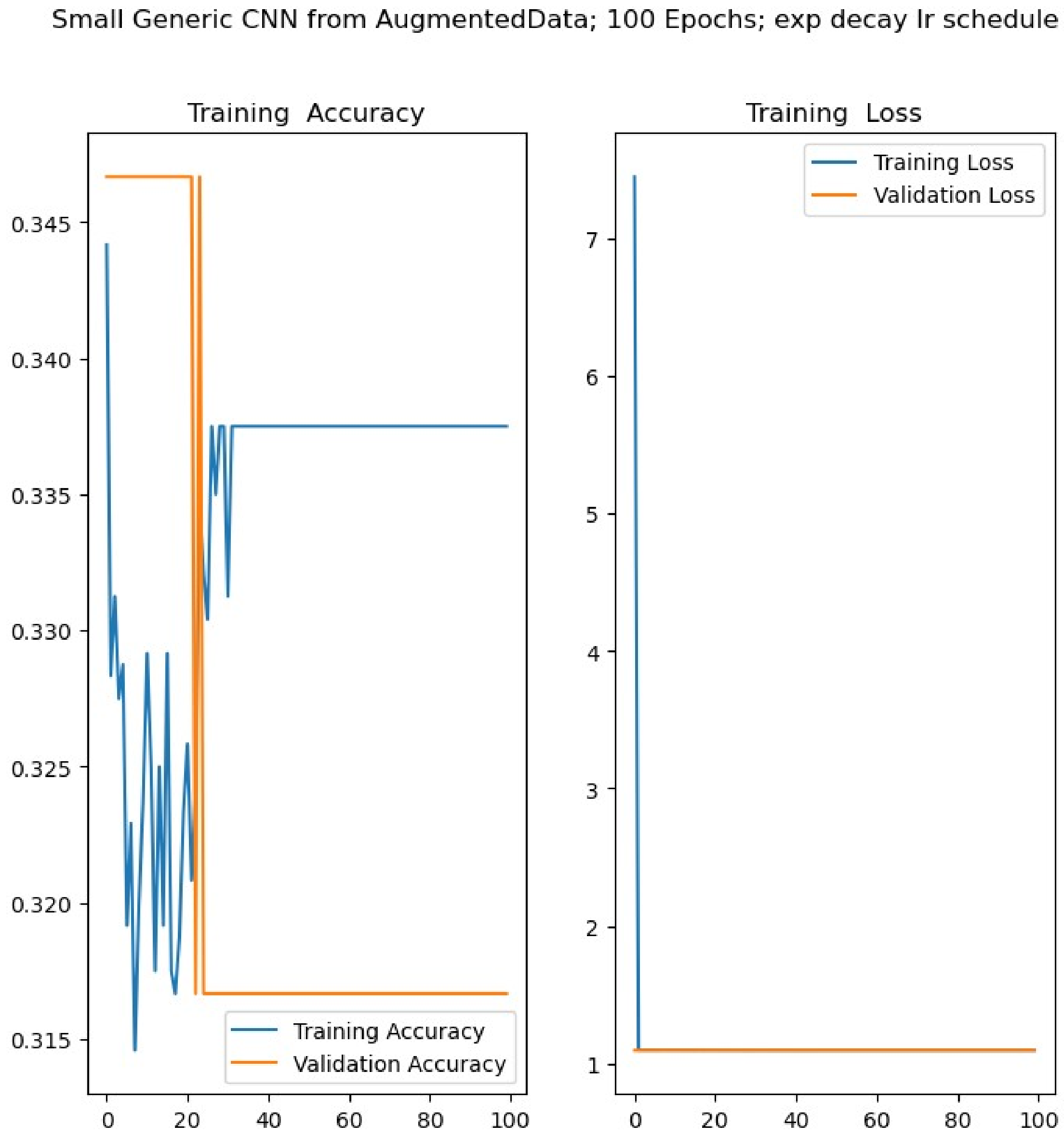
Using a learning rate scheduler: https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/LearningRateScheduler

```
# This function keeps the initial learning rate for the first ten epochs
# and decreases it exponentially after that.
def scheduler_expdec(epoch, lr):
      if epoch < 10:
```

```
        return lr
      else:
        return lr * tf.math.exp(-0.1)

# Build the model
net_5 = cnn_net(num_classes)

# Compile the model
lr = 1e-2 # this will be decayed via the scheduler callbacks
optimizer = tf.keras.optimizers.Adam(learning_rate=lr)
loss_fn =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metrics = ['accuracy']

net_5.compile(optimizer=optimizer, loss=loss_fn, metrics=metrics)

# LR Schedule Callbacks
learning_rates =
tf.keras.callbacks.LearningRateScheduler(scheduler_expdec)
callbacks_list = [learning_rates]

# Fit the model
history_5 = net_5.fit(train_ds_aug, validation_data=val_ds,
epochs=100, callbacks=callbacks_list)
```

```
Some outputs: Epoch 1/100 150/150 [==============================] - 40s 261ms/step - loss: 7.4465 - accuracy: 0.3442 - val_loss: 1.0980 - val_accuracy: 0.3467 - lr: 0.0100

Epoch 21/100 150/150 [==============================] - 41s 272ms/step - loss: 1.0992 - accuracy: 0.3258 - val_loss: 1.0986 - val_accuracy: 0.3467 - lr: 0.0033

150/150 [==============================] - 40s 263ms/step - loss: 1.0986 - accuracy: 0.3375 - val_loss: 1.0990 - val_accuracy: 0.3167 - lr: 1.2341e-06
```

```
plot_acc_metric(history_5, epochs=100, title='Small Generic CNN from
AugmentedData; 100 Epochs; exp decay lr schedule')
```



Well that didn't seem to work at all. From the very get go something went off in the optimizer and took this sideways...

However, My learning scheduler did work so that is at least a learning experience: 150/150 [==============================] - 40s 263ms/step - loss: 1.0986 - accuracy: 0.3375 - val_loss: 1.0990 - val_accuracy: 0.3167 - lr: 1.2341e-06

Time to try Optuna and see if I can get a solid static learning rate.

```
# Optuna imports
import optuna
from keras.backend import clear_session
```

```
C:\Users\btb51\anaconda3\envs\tf_LAtest\lib\site-packages\tqdm\
auto.py:22: TqdmWarning: IProgress not found. Please update jupyter
```

```
and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

```python
#make the next cnn_net
#net_6 = cnn_net(num_classes)  # no longer need this


def objective_cnn(trial):

    # Clear out the keras session
    clear_session()

    # Set the number of classes
    num_classes = 3

    # load data
    train_ds_opt, val_ds_opt = data_build()

    # run the data augmentation
    train_ds_opt_aug = train_ds_opt.map(
        lambda image, label: (data_augment(image), label),
        num_parallel_calls=tf.data.AUTOTUNE)

    # define the model (same as above but within this function)
    net_6 = Sequential([
    tf.keras.layers.Rescaling(1./255, input_shape=(image_h, image_w,
3)),
    tf.keras.layers.Conv2D(16,3,padding='same', activation = 'relu'),
    tf.keras.layers.MaxPool2D(),
    tf.keras.layers.Conv2D(32, 3, padding='same', activation =
'relu'),
    tf.keras.layers.MaxPool2D(),
    tf.keras.layers.Conv2D(64, 3, padding='same', activation= 'relu'),
    tf.keras.layers.MaxPool2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation = 'relu'),
    tf.keras.layers.Dense(num_classes)
    ])

    # set up trials for different learning rates
    lr_trial = trial.suggest_float("learning_rate", 1e-5, 1e-1,
log=True)

    # Compile items with the trial lr
    optimizer = tf.keras.optimizers.Adam(learning_rate=lr_trial)  #
note the trial lr
    loss_fn =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
    metrics = ['accuracy']
```

```
    # Compile the model
    net_6.compile(optimizer=optimizer, loss=loss_fn, metrics=metrics)

    # Fit the model
    net_6.fit(train_ds, validation_data=val_ds, epochs=10)

    # Evaluate accuracy on validation set (keep this quiet with
verbose = 0)
    eval_score = net_6.evaluate(val_ds_opt, verbose=0)

    # return the score
    return eval_score[1]


# Now that we have defined the objective above, we can run the optuna
optimizer below

# Create the study object
study = optuna.create_study(direction='maximize')

# run the study
study.optimize(objective_cnn, n_trials=40, timeout=6000)
```

Note: There is a ton of output for this. I have copied relevant items below but have deleted the large scrolling output so that the PDF is not needlessly long.

[I 2023-07-01 20:37:20,376] Trial 1 finished with value: 0.3466666638851166 and parameters: {'learning_rate': 0.006581063072314859}. Best is trial 0 with value: 0.3466666638851166.

Using a common output from the Optuna page: https://github.com/optuna/optuna-examples/blob/main/keras/keras_simple.py

```
# Print outputs of study

print("You completed {} trials.".format(len(study.trials)))

print("Best run:")
best = study.best_trial
print("    Value: {}".format(best.value))

# Now for the parameters
print("    Parameters: ")
for key, value in best.params.items():    # go through the dictionary
    print("    {}: {}".format(key,value))
```

```
You completed 2 trials.
Best run:
    Value: 0.3466666638851166
```

```
    Parameters:
    learning_rate: 0.005293065789786892
```

```python
# Lets try to optimize on the layers as well as the lr
# I'm running in to memory issues so I'm going to try and clean that up

import gc

gc.collect

tf.keras.backend.clear_session()


# Memory cleanup
class cleanup_callback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        gc.collect()


# Define the objective function to act on

def objective_everything(trial):

    # Clear out the keras session
    clear_session()

    # Set the number of classes
    num_classes = 3

    # load data
    #train_ds_opt, val_ds_opt = data_build()  # this brings in at 500 x 500

    #trying a smaller image size to prevent memory crashes
    batch_size = 16  #GPU Saturated and memory issues if I go to 32...

    # location on disk of the image data
    loc = 'C:/Users/btb51/Documents/GitHub/DeepLearning_DAAN570/DAAN570_Instructor_Sample_Codes/Lesson_08_Code/Assignment2_ZooClassifier/Zoo Classifier project - images/images'

    #datasets will be a tuple of the train and validation data
    train_ds_opt, val_ds_opt = image_dataset_from_directory(loc,
                                  batch_size=batch_size,
                                  image_size = (256,256),  # set to 256 to try and prevent memory crash
                                  shuffle = True,
                                  seed = 570,
```

```
                                  validation_split = 0.2,
                                  subset = 'both')

    # run the data augmentation
    train_ds_opt_aug = train_ds_opt.map(
        lambda image, label: (data_augment(image), label),
        num_parallel_calls=tf.data.AUTOTUNE)
    # testing how many conv layers should exist
    n_layers = trial.suggest_int("n_layers", 1, 5)

    #define the network
    net_7 = tf.keras.Sequential()

    # ensure the input shape is correct
    net_7.add(tf.keras.layers.Rescaling(1./255, input_shape=(256, 256,
3))) #changed input shape to 256 x 256

    # Make a number of layers (1-5) with either 32 or 64 filters of
kernel 3 or 5
    for i in range(n_layers):
        net_7.add(
            tf.keras.layers.Conv2D(
                filters=trial.suggest_categorical("filters", [32,
64]),
                kernel_size = trial.suggest_categorical("kernel_size",
[5]),  #took out 3
                activation = 'relu',
                padding = 'same'
            ))
        # Run max pooling on each layers as is typical of CNNs
        net_7.add(tf.keras.layers.MaxPool2D())
        # Test some dropout amoutns

net_7.add(tf.keras.layers.Dropout(trial.suggest_float("dropout", 0.2,
0.5)))

    # Final step in the model is to flatten, fully-connect, and
predict
    net_7.add(tf.keras.layers.Flatten())
    # Test different fc layers at the end

net_7.add(tf.keras.layers.Dense(trial.suggest_categorical("fc_dense",
[128]), activation = 'relu'))  #took out 64
    # number of classes is three here (dog, cat, panda)
    net_7.add(tf.keras.layers.Dense(3))


    # The softmax is applied in the SparseCatCrossEnt function
```

```
    # set up trials for different learning rates
    lr_trial = trial.suggest_float("learning_rate", 1e-5, 1e-1,
log=True)

    # Compile items with the trial lr
    optimizer = tf.keras.optimizers.Adam(learning_rate=lr_trial)  #
note the trial lr
    loss_fn =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
    metrics = ['accuracy']

    # Compile the model
    net_7.compile(optimizer=optimizer, loss=loss_fn, metrics=metrics)

    # Fit the model
    net_7.fit(train_ds_opt_aug, validation_data=val_ds_opt, epochs=10)
#took out the callback... had issues

    # Evaluate accuracy on validation set (keep this quiet with
verbose = 0)
    eval_score = net_7.evaluate(val_ds_opt, verbose=0)

    # return the score
    return eval_score[1]



# Now that we have defined the objective above, we can run the optuna
optimizer below

# Create the study object
study_big = optuna.create_study(direction='maximize')

# run the study
#probably going to hit 75/100 epochs in 9 hours (Tried to run 100
trials and had resource issues...)
study_big.optimize(objective_everything, n_trials=50, timeout=32400,
gc_after_trial=True)
```

For sake of brevity, I've only kept the final output:

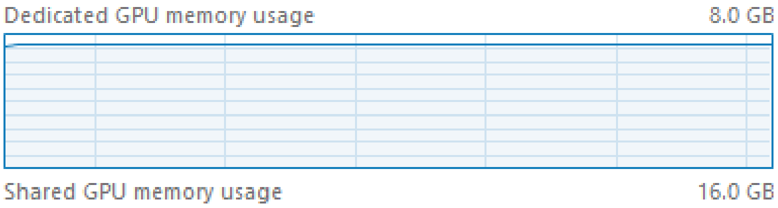
```
[I 2023-07-02 01:55:25,388] Trial 49 finished with value: 0.6783333420753479 and
parameters: {'n_layers': 3, 'filters': 32, 'kernel_size': 5, 'dropout': 0.25651711172768843,
'fc_dense': 128, 'learning_rate': 0.00011133696860382388}. Best is trial 31 with value:
0.7116666436195374.
```

```
# Print outputs of study

print("You completed {} trials.".format(len(study_big.trials)))
```

```
print("Best run:")
best = study_big.best_trial
print("    Accuracy Value: {}".format(best.value))

# Now for the parameters
print("    Parameters: ")
for key, value in best.params.items():    # go through the dictionary
    print("        {}: {}".format(key,value))
```

```
You completed 3 trials.
Best run:
    Accuracy Value: 0.6650000214576721
    Parameters:
        n_layers: 3
        filters: 64
        kernel_size: 5
        dropout: 0.3119410866475937
        fc_dense: 128
        learning_rate: 2.642900934170213e-05
```

My GPU memory consumption is maxing out. I'm not 100% sure how to free the GPU memory. I've tried a few things from gc and clear_session but neither seems to be freeing the GPU memory...



It seems I was luckly and was able to get a long run of Optuna to work without crashing. I'm not 100% that my memory issue was fixed but it worked.

```
# Print outputs of study

print("You completed {} trials.".format(len(study_big.trials)))

print("Best run:")
best = study_big.best_trial
print("    Accuracy Value: {}".format(best.value))

# Now for the parameters
print("    Parameters: ")
for key, value in best.params.items():    # go through the dictionary
    print("        {}: {}".format(key,value))
```

```
You completed 50 trials.
Best run:
    Accuracy Value: 0.7116666436195374
```

```
    Parameters:
        n_layers: 4
        filters: 64
        kernel_size: 5
        dropout: 0.22762135866715685
        fc_dense: 128
        learning_rate: 0.00044554658079259355
```

A quick note about the Best Parameters found here. The Conv2D filters design that was found here has 4 layers of the same filter width. This is opposed to what I did in the early General CNN where I have narrow filters that widden [16, 32, 64].

I will make a final CNN from the Best Params found here and run it through 50 epochs to see if I can get a best final model.

```
def cnn_best():

    #define the network
    net_best = tf.keras.Sequential()

    # ensure the input shape is correct
    net_best.add(tf.keras.layers.Rescaling(1./255, input_shape=(256, 
256, 3))) #changed input shape to 256 x 256

    n_layers = 4
    # Make a number of layers (1-5) with either 32 or 64 filters of 
kernel 3 or 5
    for i in range(n_layers):
        net_best.add(
            tf.keras.layers.Conv2D(64,5,  # 64 width of kernel shape 5 
x 5
                activation = 'relu',
                padding = 'same')
            )
        # Run max pooling on each layers as is typical of CNNs
        net_best.add(tf.keras.layers.MaxPool2D())
        # Test some dropout amoutns
        drop_value = 0.227
        net_best.add(tf.keras.layers.Dropout(drop_value))

    # Final step in the model is to flatten, fully-connect, and 
predict
    net_best.add(tf.keras.layers.Flatten())
    # Test different fc layers at the end
    net_best.add(tf.keras.layers.Dense(128, activation = 'relu'))
    # number of classes is three here (dog, cat, panda)
    net_best.add(tf.keras.layers.Dense(3)) # still hard coded the 3 
classes here
```

```
    return net_best


# Data in at 256 x 256 [data_build function brings in at 500 x 500]
#trying a smaller image size to prevent memory crashes
batch_size = 16  #GPU Saturated and memory issues if I go to 32...

# location on disk of the image data
loc = 
'C:/Users/btb51/Documents/GitHub/DeepLearning_DAAN570/DAAN570_Instruct
or_Sample_Codes/Lesson_08_Code/Assignment2_ZooClassifier/Zoo 
Classifier project - images/images'


#datasets will be a tuple of the train and validation data
train_ds, val_ds = image_dataset_from_directory(loc,
                              batch_size=batch_size,
                              image_size = (256,256),  # set to 256 to 
try and prevent memory crash
                              shuffle = True,
                              seed = 570,
                              validation_split = 0.2,
                              subset = 'both')

# run the data augmentation
train_ds_aug = train_ds_opt.map(
    lambda image, label: (data_augment(image), label),
    num_parallel_calls=tf.data.AUTOTUNE)
```

```
Found 3000 files belonging to 3 classes.
Using 2400 files for training.
Using 600 files for validation.
```

```
net_best = cnn_best()

# set up trials for different learning rates

lr = 0.00044554658079259355
# Compile items with the trial lr
optimizer = tf.keras.optimizers.Adam(learning_rate=lr)
loss_fn = 
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metrics = ['accuracy']

# Compile the model
net_best.compile(optimizer=optimizer, loss=loss_fn, metrics=metrics)
```

```
net_best.summary()
```

```
Model: "sequential_6"
```

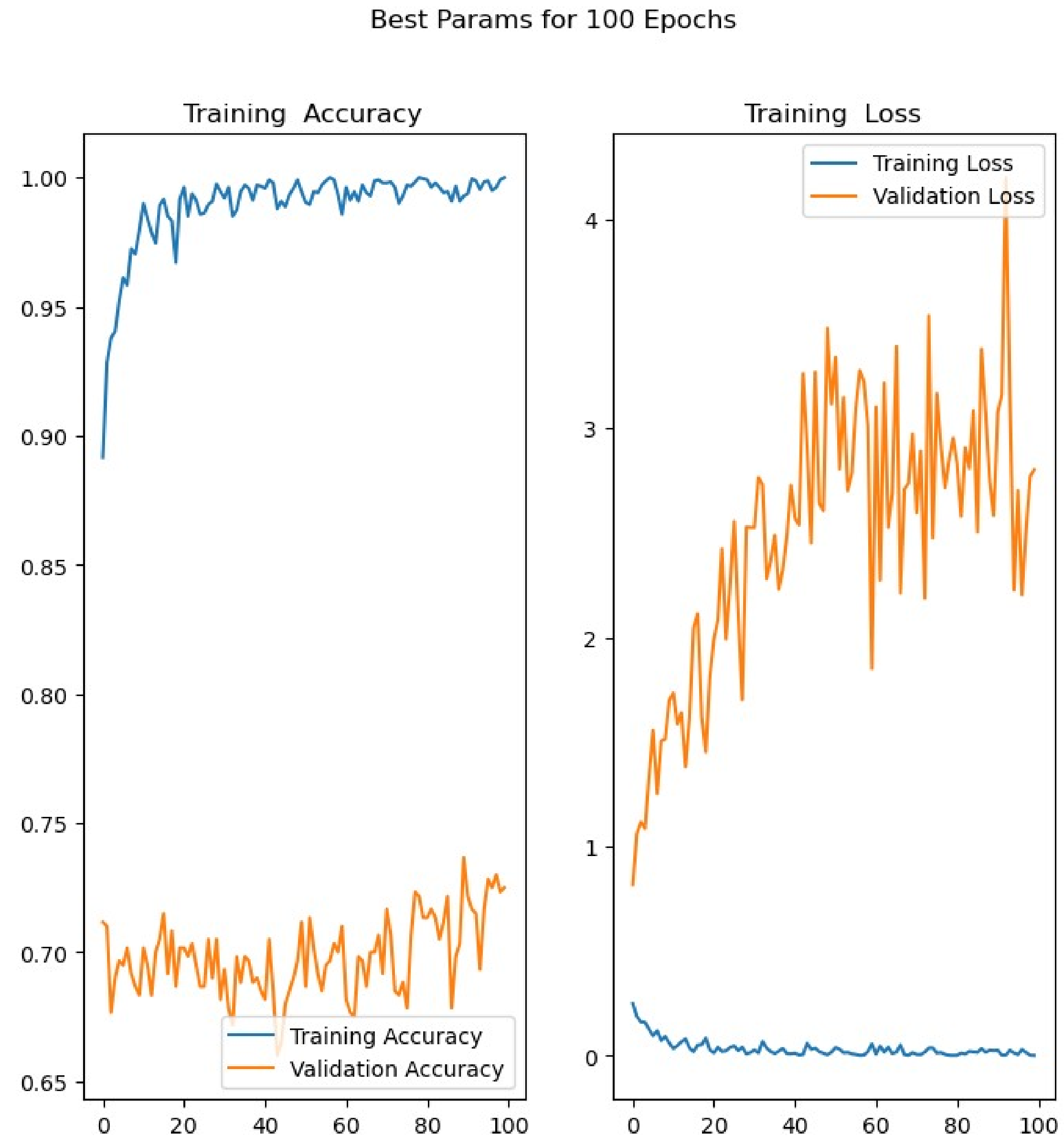
```
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_5 (Rescaling)     (None, 256, 256, 3)       0

 conv2d_16 (Conv2D)          (None, 256, 256, 64)      4864

 max_pooling2d_15 (MaxPoolin  (None, 128, 128, 64)     0
 g2D)

 dropout_15 (Dropout)        (None, 128, 128, 64)      0

 conv2d_17 (Conv2D)          (None, 128, 128, 64)      102464

 max_pooling2d_16 (MaxPoolin  (None, 64, 64, 64)       0
 g2D)

 dropout_16 (Dropout)        (None, 64, 64, 64)        0

 conv2d_18 (Conv2D)          (None, 64, 64, 64)        102464

 max_pooling2d_17 (MaxPoolin  (None, 32, 32, 64)       0
 g2D)

 dropout_17 (Dropout)        (None, 32, 32, 64)        0

 conv2d_19 (Conv2D)          (None, 32, 32, 64)        102464

 max_pooling2d_18 (MaxPoolin  (None, 16, 16, 64)       0
 g2D)

 dropout_18 (Dropout)        (None, 16, 16, 64)        0

 flatten_4 (Flatten)         (None, 16384)             0

 dense_8 (Dense)             (None, 128)               2097280

 dense_9 (Dense)             (None, 3)                 387

=================================================================
Total params: 2,409,923
Trainable params: 2,409,923
Non-trainable params: 0
_________________________________________________________________
```

```
# Fit the model
history_best = net_best.fit(train_ds_aug, validation_data=val_ds,
epochs=100)
```

Some Output:

```
Epoch 1/100 150/150 [==============================] - 12s 77ms/step - loss:
0.2491 - accuracy: 0.8917 - val_loss: 0.8166 - val_accuracy: 0.7117 Epoch 2/100 150/150
[==============================] - 11s 75ms/step - loss: 0.1870 - accuracy: 0.9283 -
val_loss: 1.0613 - val_accuracy: 0.7100 Epoch 3/100 150/150
[==============================] - 11s 74ms/step - loss: 0.1610 - accuracy: 0.9379 -
val_loss: 1.1171 - val_accuracy: 0.6767

...

Epoch 98/100 150/150 [==============================] - 11s 75ms/step - loss:
0.0148 - accuracy: 0.9962 - val_loss: 2.5153 - val_accuracy: 0.7300 Epoch 99/100 150/150
[==============================] - 11s 75ms/step - loss: 0.0017 - accuracy: 0.9992 -
val_loss: 2.7698 - val_accuracy: 0.7233 Epoch 100/100 150/150
[==============================] - 11s 75ms/step - loss: 6.4478e-04 - accuracy:
1.0000 - val_loss: 2.8012 - val_accuracy: 0.7250
```

```python
plot_acc_metric(history_best, title='Best Params for 100 Epochs',
epochs = 100)
```



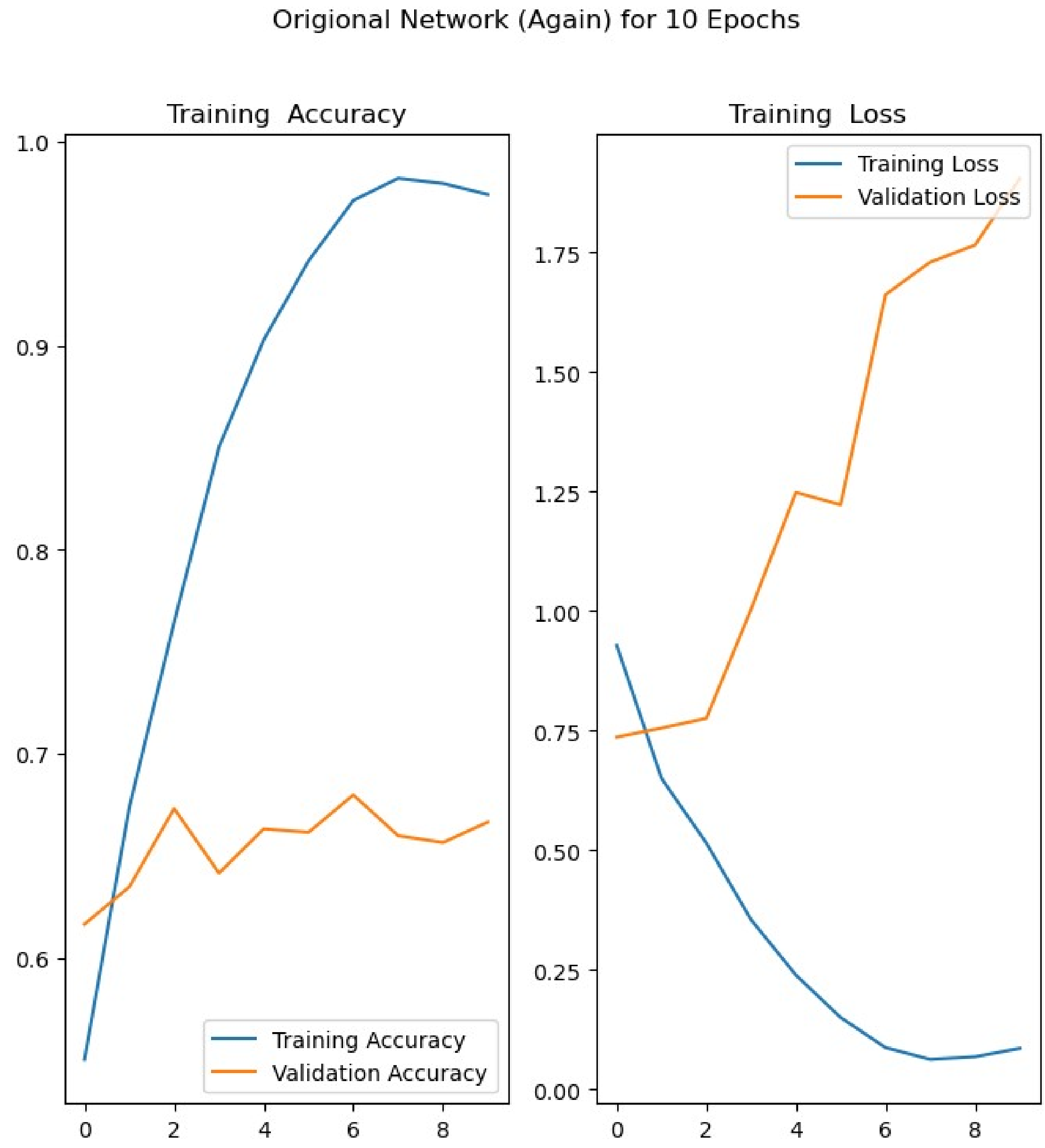
Well that looks like garbage... The different sized Conv layers from the second run were very helpful. I'm going to run that again for more Epochs and see what happens.

```
def cnn_net_small(num_classes):

    #Build a Generic CNN with a set number of classes as the
classifing output

    net = Sequential([
    tf.keras.layers.Rescaling(1./255, input_shape=(256, 256, 3)),
    tf.keras.layers.Conv2D(16,3,padding='same', activation = 'relu'),
    tf.keras.layers.MaxPool2D(),
    tf.keras.layers.Conv2D(32, 3, padding='same', activation =
'relu'),
```

```
    tf.keras.layers.MaxPool2D(),
    tf.keras.layers.Conv2D(64, 3, padding='same', activation= 'relu'),
    tf.keras.layers.MaxPool2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation = 'relu'),
    tf.keras.layers.Dense(num_classes)
    ])

    return net

net_hope = cnn_net_small(3)
# forgive the misspelling of history...
hisotry_hope = trainer(net_hope, train_ds_aug, val_ds, epochs=10)
```

```
Epoch 1/10
150/150 [==============================] - 6s 39ms/step - loss: 0.9282
- accuracy: 0.5504 - val_loss: 0.7368 - val_accuracy: 0.6167
Epoch 2/10
150/150 [==============================] - 5s 36ms/step - loss: 0.6507
- accuracy: 0.6742 - val_loss: 0.7556 - val_accuracy: 0.6350
Epoch 3/10
150/150 [==============================] - 5s 34ms/step - loss: 0.5151
- accuracy: 0.7646 - val_loss: 0.7760 - val_accuracy: 0.6733
Epoch 4/10
150/150 [==============================] - 5s 33ms/step - loss: 0.3548
- accuracy: 0.8504 - val_loss: 1.0038 - val_accuracy: 0.6417
Epoch 5/10
150/150 [==============================] - 5s 34ms/step - loss: 0.2391
- accuracy: 0.9029 - val_loss: 1.2484 - val_accuracy: 0.6633
Epoch 6/10
150/150 [==============================] - 5s 35ms/step - loss: 0.1502
- accuracy: 0.9417 - val_loss: 1.2225 - val_accuracy: 0.6617
Epoch 7/10
150/150 [==============================] - 5s 33ms/step - loss: 0.0875
- accuracy: 0.9712 - val_loss: 1.6620 - val_accuracy: 0.6800
Epoch 8/10
150/150 [==============================] - 5s 34ms/step - loss: 0.0628
- accuracy: 0.9821 - val_loss: 1.7299 - val_accuracy: 0.6600
Epoch 9/10
150/150 [==============================] - 5s 32ms/step - loss: 0.0683
- accuracy: 0.9796 - val_loss: 1.7656 - val_accuracy: 0.6567
Epoch 10/10
150/150 [==============================] - 5s 33ms/step - loss: 0.0859
- accuracy: 0.9742 - val_loss: 1.9051 - val_accuracy: 0.6667
```

```
plot_acc_metric(hisotry_hope, title='Origional Network (Again) for 10
Epochs', epochs = 10)
```



And I'm back to the overfit in just 10 epochs... I've come full circle. That second CNN I ran above (runs 2 and 3) had to just find the perfect path to optimize on. That or the data augmentation (which is random) augmented things in the perfect way for the CNN to tease out the proper features.

I'm not sure what I could do at this point other than a mega combination of what I've done thus far: 1) Optuna on the first CNN build with each layer of the CNN getting a search space on the width and kernal size 2) Optuna on different lr schedules [not 100% sure how to do this or if it is possible] 3) Develop better data augmentation past what I already have

Helpful References and Tutorials I used to guide me:

Optuna:
https://github.com/optuna/optuna-examples/blob/main/tensorflow/tensorflow_eager_simple.py https://github.com/optuna/optuna-examples/blob/main/keras/keras_simple.py

Keras:
https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/LearningRateScheduler

General Image Classification: https://www.tensorflow.org/tutorials/images/classification