Assignment 3: CNN Cancer Detection Kaggle Mini-Project

Description

This project is a binary image classification problem. The goal of this project is to develop a convolutional neural network model that can accurately identify metastatic cancer patches in an image. The images are provided by the Kaggle Histopathologic Cancer Detection Competition and located at https://www.kaggle.com/competitions/histopathologic-cancer-detection/data. Miltilayer convolutional neural networks will be developed, trained and validated. The best model will then be used to predict the labels on the test images.

Natural Language Processing (NLP)

NLP is a machine learning technique that seeks to understand and make sense of the humnan language in its different forms, text, speech, etc. NLP processes a series of words, text or images and attempts to analyze the input to produce the desired output.

The challenge in NLP lies in capturing the intended meaning of the input, given that the same input can have different meanings, i.e. different outputs. The trick lies in capturing the context of the input so the the correct output can be derived.

Data Summary

The data consists of training and test images with image ids provided as the filename. There are 220,025 training images and 57,458 test images. The train_labels.csv contains 220,025 rows that house the image id and assoiciated ground truth lables for the training images. The sample_submission.csv contains the image ids of the test images and sample ground truth labels. The labels are to be replaced with test results and submitted for assessment of the model.

In [1]:
```
#Set Page Width to 100%
from IPython.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

In [19]:
```
#Load Required Resources
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import seaborn as sns
import math
import time
import cv2
from sklearn import metrics
from sklearn.model_selection import train_test_split
```

```
import gc

import tensorflow as tf
from tensorflow.keras import models
from keras_preprocessing.image import ImageDataGenerator
from PIL import Image
from tensorflow.python.keras.layers import LSTM, Dense, Conv2D, MaxPooling2D, Dropout,
from tensorflow.python.keras.models import Sequential
from tensorflow.keras.layers import BatchNormalization
from keras.optimizers import Adam
```

In [3]:
```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
print(tf.__version__)
```

```
Num GPUs Available:  0
2.10.1
```

GPU Limitation :As noted above this project does not have GPU access. Some models and simulations were reduced in complexity based on actual run times.

In [20]:
```
## Import Data
train_image_df = pd.read_csv('train/train_labels.csv')
test_image_df = pd.read_csv("sample_submission.csv" )

print(train_image_df.head(), '\n')
print(train_image_df.info(), '\n')
print(test_image_df.head(), '\n')
print(test_image_df.info())
```

```
                                         id  label
0  f38a6374c348f90b587e046aac6079959adf3835      0
1  c18f2d887b7ae4f6742ee445113fa1aef383ed77      1
2  755db6279dae599ebb4d39a9123cce439965282d      0
3  bc3f0c64fb968ff4a8bd33af6971ecae77c75e08      0
4  068aba587a4950175d04c680d38943fd488d6a9d      0

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 220025 entries, 0 to 220024
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype 
---  ------  --------------   ----- 
 0   id      220025 non-null  object
 1   label   220025 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 3.4+ MB
None

                                         id  label
0  0b2ea2a822ad23fdb1b5dd26653da899fbd2c0d5      0
1  95596b92e5066c5c52466c90b69ff089b39f2737      0
2  248e6738860e2ebcf6258cdc1f32f299e0c76914      0
3  2c35657e312966e9294eac6841726ff3a748febf      0
4  145782eb7caa1c516acbe2eda34d9a3f31c41fd6      0

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 57458 entries, 0 to 57457
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   id      57458 non-null  object
 1   label   57458 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 897.9+ KB
None
```

Data

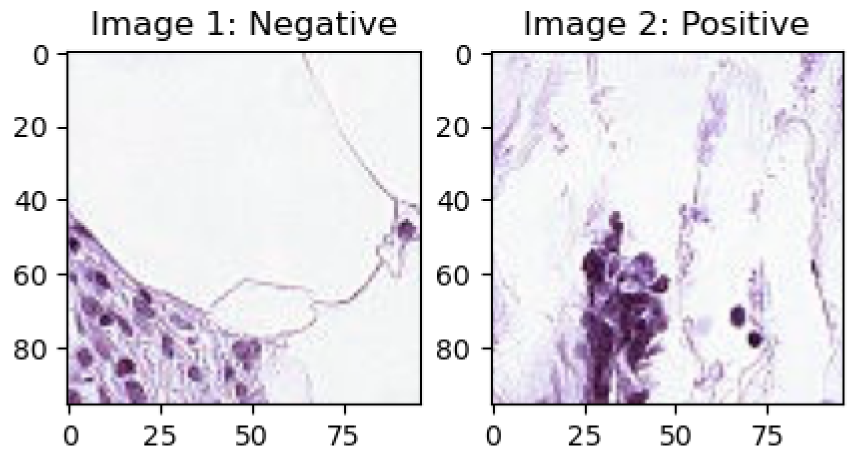
See Images 1 and 2 for examples of negative and positive instances, respectively. A brief review of the images, image ids and ground truth labels showed no anomolous cases. It is assumed that the provided data set is in proper order.

In [21]:

```
#Display Sample Images

fig, ax = plt.subplots(1, 2, figsize = (5, 5))

ax[0].imshow(cv2.imread('train/' + train_image_df['id'][0] + '.tif'))
ax[0].set_title('Image 1: Negative')
ax[1].imshow(cv2.imread('train/' + train_image_df['id'][1] + '.tif'))
ax[1].set_title('Image 2: Positive')
```

Out[21]:

```
Text(0.5, 1.0, 'Image 2: Positive')
```



EDA

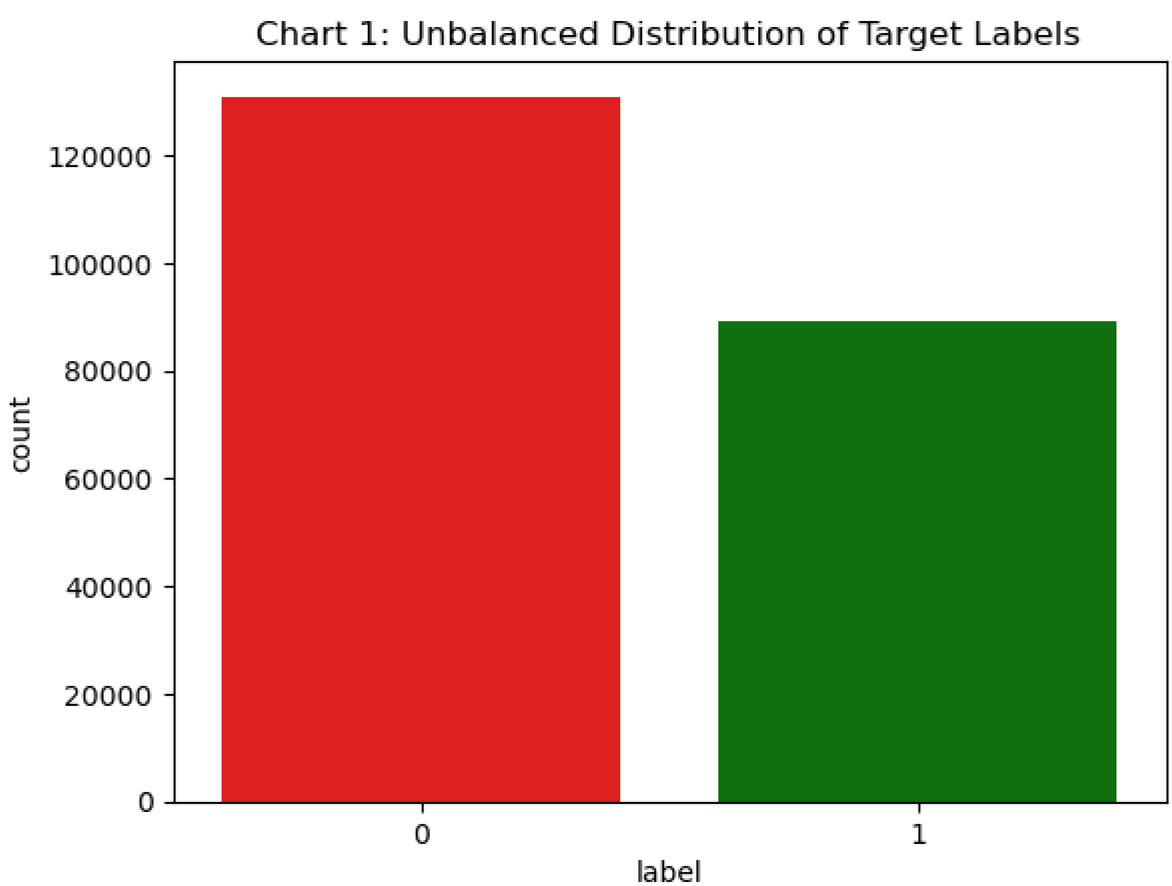
As shown in Chart 1, 59.5% of the training examples are negative while 40.5% positive. This imbalance could result in biased training so the training set is rebalanced to a 50-50 split. Plot 2 shows the new distribution to be used as the training set. As noted above, a review of the images, image ids and labels revealed no anomolies. No additional data cleansing or adjustments are required.

In [22]:
```
#Add tif file extension
train_image_df['id'] = train_image_df['id'].astype('str') + '.tif'
print(train_image_df.head())

test_image_df['id'] = test_image_df['id'].astype('str') + '.tif'
print(test_image_df.head())
print(test_image_df.info())
```

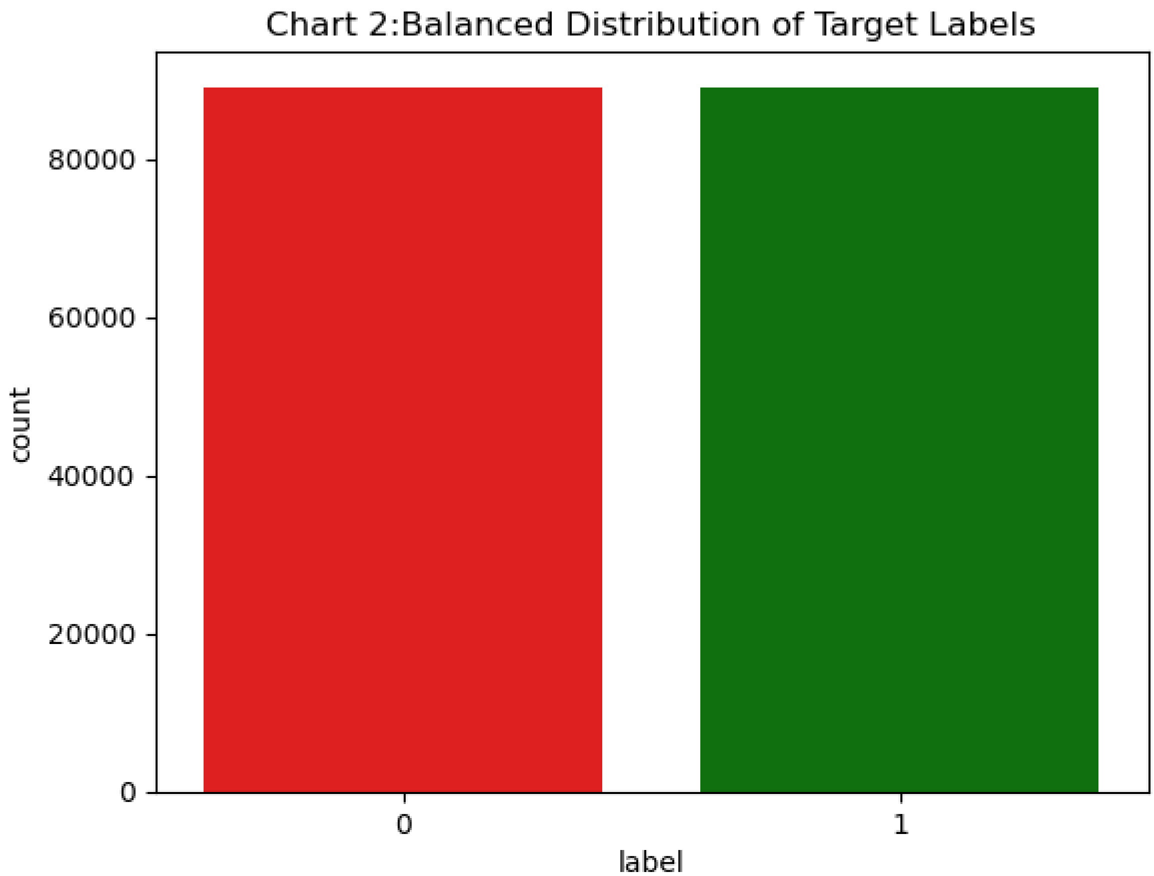
```
                                             id  label
0  f38a6374c348f90b587e046aac6079959adf3835.tif      0
1  c18f2d887b7ae4f6742ee445113fa1aef383ed77.tif      1
2  755db6279dae599ebb4d39a9123cce439965282d.tif      0
3  bc3f0c64fb968ff4a8bd33af6971ecae77c75e08.tif      0
4  068aba587a4950175d04c680d38943fd488d6a9d.tif      0
                                             id  label
0  0b2ea2a822ad23fdb1b5dd26653da899fbd2c0d5.tif      0
1  95596b92e5066c5c52466c90b69ff089b39f2737.tif      0
2  248e6738860e2ebcf6258cdc1f32f299e0c76914.tif      0
3  2c35657e312966e9294eac6841726ff3a748febf.tif      0
4  145782eb7caa1c516acbe2eda34d9a3f31c41fd6.tif      0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 57458 entries, 0 to 57457
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   id      57458 non-null  object
 1   label   57458 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 897.9+ KB
None
```

In [23]:
```
# Charts, data shape
```

```
plt.figure()
sns.countplot(data=train_image_df, x='label', palette=['#ff0000',"#008000"])
plt.title('Chart 1: Unbalanced Distribution of Target Labels')
plt.show()

# Rebalance and training arrays

#Remove 31% of negative images to balance dataset
train_image_df = train_image_df.drop(train_image_df[train_image_df['label'] == 0].samp

train_negative_df = train_image_df[train_image_df['label'] == 0]
print('Negative Train Samples: ', train_negative_df.shape)

train_positive_df = train_image_df[train_image_df['label'] == 1]
print('Positive Train Samples: ', train_positive_df.shape)

#Convert to Numpy Array
train_image_id = train_image_df["id"].to_numpy()
test_image_id =  test_image_df["id"].to_numpy()
train_image_label = train_image_df["label"].to_numpy()

print('Train Image Id array shape: ', train_image_id.shape)
print('Test Image Id array shape: ', test_image_id.shape )

# Charts, new data shape
plt.figure()
sns.countplot(data=train_image_df, x='label', palette=['#ff0000',"#008000"])
plt.title('Chart 2:Balanced Distribution of Target Labels')
plt.show()
```



```
Negative Train Samples:  (89017, 2)
Positive Train Samples:  (89117, 2)
Train Image Id array shape:  (178134,)
Test Image Id array shape:  (57458,)
```



Approach

Based on the EDA the following approach will be deployed.

1. Establish train and test sets. 25,000 images of each classification.
2. Preprocess the data (scaling, normalization).
3. Compare two models where the Batch Normalization layers are in different positions. Before and after the Activation layer (optimizer - adam, metric - accuracy).
4. Improve best model through hyperparameter tuning

Model Architecture

The core of the models will be the Convolutional layers. The Convolutional model is extremely adapt at processing images to various levels. Multiple Convolution layers (3) will be used in an effort to increase accuracy. The Convolutional layers are bracketed by Batch Normalization and Max pooling layers. These layers are added to improve both model performance and efficiency. Relu activation will be used in the first three Activation layers as this method is best suited for this type of analysis. A Dense layer is placed in front of the final Activation layer. The final Activation layer will use sigmoid activation for classification purposes.

While designing the model, the question arose as where to place the Batch Normalization layers. With that in mind, it was determined that two different models, with the Batch Normalization layers placed in different places would be a good basis for model comparison.

The first model has the Batch Normalization layers placed before the Activation layers. The second model has the Batch Normalization layers placed after the Activation layers. The models will be compared to determine the optimal location for the Batch Normalization layers.

In [24]:
```
#### Pre-Process Data

#Set Training set size and split
negative_training_samples = train_negative_df.sample(n=25000, random_state=25)
positive_training_samples = train_positive_df.sample(n=25000, random_state=25)

full_train_set = pd.concat([negative_training_samples, positive_training_samples], axi

full_train_set["label"].value_counts()

print(full_train_set.head())
print(full_train_set.info())

training_set, validation_set = train_test_split(full_train_set, random_state=25, test_


#Generate and scale images
data_image_generator = ImageDataGenerator(featurewise_center=False, zoom_range = 0.2,

generator_train_set = data_image_generator.flow_from_dataframe(
                            dataframe=training_set,
                            directory='train/',
                            x_col="id",
                            y_col="label",
                            batch_size=32,
                            seed=25,
                            #class_mode="binary",
                            class_mode="raw",
                            target_size=(32,32))

generator_validation_set = data_image_generator.flow_from_dataframe(
                            dataframe=validation_set,
                            directory='train/',
                            x_col="id",
                            y_col="label",
                            batch_size=32,
                            seed=25,
                            #class_mode="binary",
                            class_mode="raw",
                            target_size=(32,32))

generator_test_set = test_image_generator.flow_from_dataframe(
                            dataframe=test_image_df,
                            directory='test/',
                            x_col='id',
                            y_col= 'label',
                            batch_size=100,
                            #shuffle = False,
                            seed=25,
```

```
                             class_mode="raw",
                             target_size=(32,32),
                             validation_steps=100)
```

```
                                             id  label
0  35a93e5612821c37670e70af150238932e20a942.tif      0
1  a3fa8742c25a40834271ecb40e23b28a323b27ff.tif      0
2  66bf34cfaa615793d2266e0668f0e81ba384f9a3.tif      0
3  a55f0f48559ed6328c6f666e6f32f8c4d13c52d9.tif      0
4  3e571e7b6cb092a08fb7d6592d54a7c5c47b8302.tif      0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   id      50000 non-null  object
 1   label   50000 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 781.4+ KB
None
Found 35000 validated image filenames.
Found 15000 validated image filenames.
Found 57458 validated image filenames.
```

Models

Batch Normalization Before Activativation Layer

In [54]:

```
#Before Model - two convolution layers, flatten layer, dense layer, batch normalizatio
number_of_epochs = 20

before_model = Sequential()

before_model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3)))

before_model.add(BatchNormalization())
before_model.add(Activation('relu'))

before_model.add(MaxPooling2D(pool_size=(3, 3)))
before_model.add(Conv2D(64, (3, 3)))

before_model.add(BatchNormalization())
before_model.add(Activation('relu'))

before_model.add(MaxPooling2D(pool_size=(3, 3)))
before_model.add(Flatten())
before_model.add(Dense(128))

before_model.add(BatchNormalization())
before_model.add(Activation('relu'))

before_model.add(Dense(1))
before_model.add(Activation('sigmoid'))

before_model.summary()

before_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'
```

```
before_history = before_model.fit(generator_train_set,
                    epochs = number_of_epochs,
                    steps_per_epoch=(generator_train_set.n//generator_train_set.batch_
                    validation_data = generator_validation_set,
                    validation_steps=(generator_validation_set.n//generator_validatior

before_model.save("before_model")
```

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_33 (Conv2D)           (None, 30, 30, 32)        896
_________________________________________________________________
module_wrapper_47 (ModuleWra (None, 30, 30, 32)        128
_________________________________________________________________
activation_61 (Activation)   (None, 30, 30, 32)        0
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 10, 10, 32)        0
_________________________________________________________________
conv2d_34 (Conv2D)           (None, 8, 8, 64)          18496
_________________________________________________________________
module_wrapper_48 (ModuleWra (None, 8, 8, 64)          256
_________________________________________________________________
activation_62 (Activation)   (None, 8, 8, 64)          0
_________________________________________________________________
max_pooling2d_34 (MaxPooling (None, 2, 2, 64)          0
_________________________________________________________________
flatten_14 (Flatten)         (None, 256)               0
_________________________________________________________________
dense_28 (Dense)             (None, 128)               32896
_________________________________________________________________
module_wrapper_49 (ModuleWra (None, 128)               512
_________________________________________________________________
activation_63 (Activation)   (None, 128)               0
_________________________________________________________________
dense_29 (Dense)             (None, 1)                 129
_________________________________________________________________
activation_64 (Activation)   (None, 1)                 0
=================================================================
Total params: 53,313
Trainable params: 52,865
Non-trainable params: 448
_________________________________________________________________
Epoch 1/20
546/546 [==============================] - 83s 151ms/step - loss: 0.4874 - accuracy: 
0.7683 - val_loss: 0.5213 - val_accuracy: 0.7301
Epoch 2/20
546/546 [==============================] - 93s 171ms/step - loss: 0.4383 - accuracy: 
0.8002 - val_loss: 0.5845 - val_accuracy: 0.7227
Epoch 3/20
546/546 [==============================] - 81s 148ms/step - loss: 0.4268 - accuracy: 
0.8067 - val_loss: 0.4928 - val_accuracy: 0.7714
Epoch 4/20
546/546 [==============================] - 84s 154ms/step - loss: 0.4205 - accuracy: 
0.8099 - val_loss: 0.4599 - val_accuracy: 0.7872
Epoch 5/20
546/546 [==============================] - 85s 157ms/step - loss: 0.4089 - accuracy: 
0.8174 - val_loss: 0.5329 - val_accuracy: 0.7623
Epoch 6/20
546/546 [==============================] - 77s 141ms/step - loss: 0.4075 - accuracy: 
0.8167 - val_loss: 1.3158 - val_accuracy: 0.5690
Epoch 7/20
546/546 [==============================] - 77s 142ms/step - loss: 0.4004 - accuracy: 
0.8203 - val_loss: 1.1754 - val_accuracy: 0.5135
Epoch 8/20
546/546 [==============================] - 144s 263ms/step - loss: 0.4031 - accuracy: 
0.8168 - val_loss: 0.4541 - val_accuracy: 0.7892
```

```
Epoch 9/20
546/546 [==============================] - 93s 170ms/step - loss: 0.4022 - accuracy:
0.8221 - val_loss: 0.3894 - val_accuracy: 0.8247
Epoch 10/20
546/546 [==============================] - 108s 198ms/step - loss: 0.3832 - accuracy:
0.8297 - val_loss: 0.4053 - val_accuracy: 0.8187
Epoch 11/20
546/546 [==============================] - 114s 208ms/step - loss: 0.3813 - accuracy:
0.8308 - val_loss: 0.5363 - val_accuracy: 0.7361
Epoch 12/20
546/546 [==============================] - 97s 177ms/step - loss: 0.3839 - accuracy:
0.8301 - val_loss: 0.5687 - val_accuracy: 0.7489
Epoch 13/20
546/546 [==============================] - 81s 148ms/step - loss: 0.3824 - accuracy:
0.8318 - val_loss: 0.6038 - val_accuracy: 0.7202
Epoch 14/20
546/546 [==============================] - 91s 166ms/step - loss: 0.3719 - accuracy:
0.8367 - val_loss: 0.4316 - val_accuracy: 0.7952
Epoch 15/20
546/546 [==============================] - 81s 148ms/step - loss: 0.3687 - accuracy:
0.8353 - val_loss: 0.4084 - val_accuracy: 0.8160
Epoch 16/20
546/546 [==============================] - 83s 153ms/step - loss: 0.3721 - accuracy:
0.8357 - val_loss: 0.6459 - val_accuracy: 0.6981
Epoch 17/20
546/546 [==============================] - 73s 135ms/step - loss: 0.3587 - accuracy:
0.8425 - val_loss: 0.8854 - val_accuracy: 0.6485
Epoch 18/20
546/546 [==============================] - 79s 145ms/step - loss: 0.3641 - accuracy:
0.8400 - val_loss: 0.4916 - val_accuracy: 0.7755
Epoch 19/20
546/546 [==============================] - 77s 141ms/step - loss: 0.3587 - accuracy:
0.8448 - val_loss: 0.7193 - val_accuracy: 0.7123
Epoch 20/20
546/546 [==============================] - 78s 144ms/step - loss: 0.3534 - accuracy:
0.8444 - val_loss: 0.5166 - val_accuracy: 0.7460
INFO:tensorflow:Assets written to: before_model\assets
```

Batch Normalization After Activativation Layer

In [55]:

```
#Train models
#After Model - two convolution layers, flatten layer, dense layer, batch normalizatior
after_model = Sequential()

after_model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3)))

after_model.add(Activation('relu'))
after_model.add(BatchNormalization())

after_model.add(MaxPooling2D(pool_size=(3, 3)))
after_model.add(Conv2D(64, (3, 3)))

after_model.add(Activation('relu'))
after_model.add(BatchNormalization())

after_model.add(MaxPooling2D(pool_size=(3, 3)))
after_model.add(Flatten())
after_model.add(Dense(128))
```

```
after_model.add(Activation('relu'))
after_model.add(BatchNormalization())

after_model.add(Dense(1))
after_model.add(Activation('sigmoid'))

after_model.summary()

after_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy']

after_history = after_model.fit(generator_train_set,
                    epochs = number_of_epochs,
                    steps_per_epoch=(generator_train_set.n//generator_train_set.batch_
                    validation_data = generator_validation_set,
                    validation_steps=(generator_validation_set.n//generator_validation

after_model.save("after_model")
```

```
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_35 (Conv2D)           (None, 30, 30, 32)        896
_________________________________________________________________
activation_65 (Activation)   (None, 30, 30, 32)        0
_________________________________________________________________
module_wrapper_50 (ModuleWra (None, 30, 30, 32)        128
_________________________________________________________________
max_pooling2d_35 (MaxPooling (None, 10, 10, 32)        0
_________________________________________________________________
conv2d_36 (Conv2D)           (None, 8, 8, 64)          18496
_________________________________________________________________
activation_66 (Activation)   (None, 8, 8, 64)          0
_________________________________________________________________
module_wrapper_51 (ModuleWra (None, 8, 8, 64)          256
_________________________________________________________________
max_pooling2d_36 (MaxPooling (None, 2, 2, 64)          0
_________________________________________________________________
flatten_15 (Flatten)         (None, 256)               0
_________________________________________________________________
dense_30 (Dense)             (None, 128)               32896
_________________________________________________________________
activation_67 (Activation)   (None, 128)               0
_________________________________________________________________
module_wrapper_52 (ModuleWra (None, 128)               512
_________________________________________________________________
dense_31 (Dense)             (None, 1)                 129
_________________________________________________________________
activation_68 (Activation)   (None, 1)                 0
=================================================================
Total params: 53,313
Trainable params: 52,865
Non-trainable params: 448
_________________________________________________________________
Epoch 1/20
546/546 [==============================] - 82s 149ms/step - loss: 0.5161 - accuracy:
0.7529 - val_loss: 0.6983 - val_accuracy: 0.6525
Epoch 2/20
546/546 [==============================] - 85s 156ms/step - loss: 0.4799 - accuracy:
0.7731 - val_loss: 1.0931 - val_accuracy: 0.5390
Epoch 3/20
546/546 [==============================] - 78s 142ms/step - loss: 0.4644 - accuracy:
0.7820 - val_loss: 0.4655 - val_accuracy: 0.7869
Epoch 4/20
546/546 [==============================] - 89s 164ms/step - loss: 0.4564 - accuracy:
0.7892 - val_loss: 0.7704 - val_accuracy: 0.6044
Epoch 5/20
546/546 [==============================] - 85s 155ms/step - loss: 0.4433 - accuracy:
0.7922 - val_loss: 0.6654 - val_accuracy: 0.6768
Epoch 6/20
546/546 [==============================] - 87s 160ms/step - loss: 0.4414 - accuracy:
0.7986 - val_loss: 0.8334 - val_accuracy: 0.5504
Epoch 7/20
546/546 [==============================] - 76s 139ms/step - loss: 0.4368 - accuracy:
0.8016 - val_loss: 0.5505 - val_accuracy: 0.7622
Epoch 8/20
546/546 [==============================] - 77s 141ms/step - loss: 0.4319 - accuracy:
0.8027 - val_loss: 0.6268 - val_accuracy: 0.7430
```

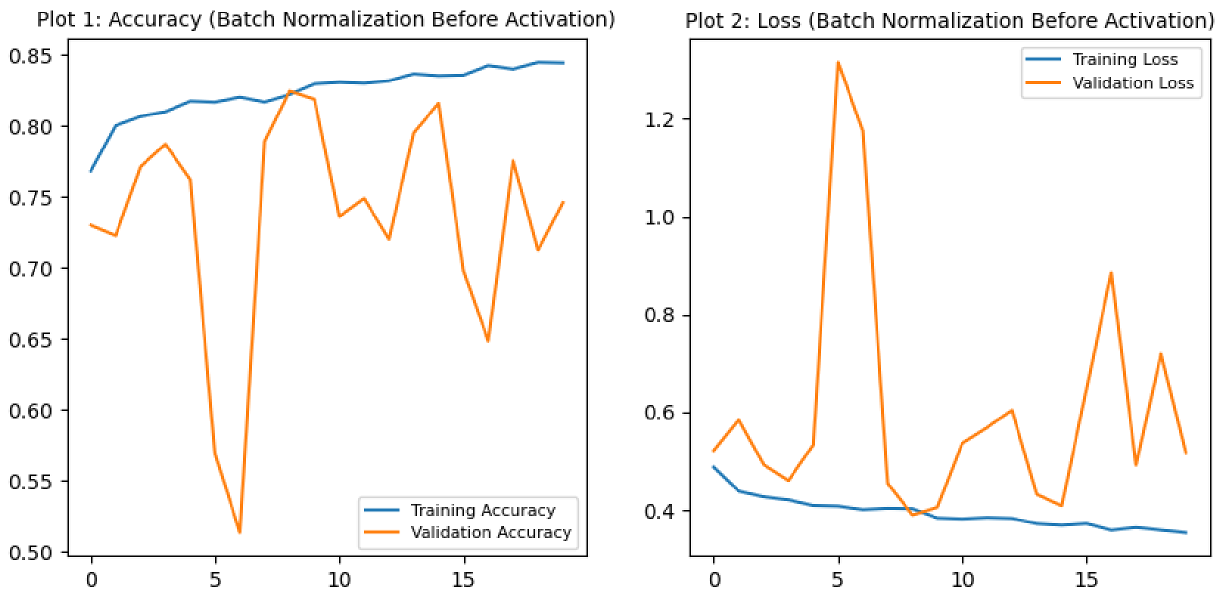
```
Epoch 9/20
546/546 [==============================] - 75s 138ms/step - loss: 0.4282 - accuracy: 0.8064 - val_loss: 0.6375 - val_accuracy: 0.6879
Epoch 10/20
546/546 [==============================] - 76s 139ms/step - loss: 0.4251 - accuracy: 0.8090 - val_loss: 0.4326 - val_accuracy: 0.8049
Epoch 11/20
546/546 [==============================] - 74s 135ms/step - loss: 0.4241 - accuracy: 0.8092 - val_loss: 2.4545 - val_accuracy: 0.4742
Epoch 12/20
546/546 [==============================] - 77s 142ms/step - loss: 0.4160 - accuracy: 0.8135 - val_loss: 0.5557 - val_accuracy: 0.7512
Epoch 13/20
546/546 [==============================] - 129s 237ms/step - loss: 0.4179 - accuracy: 0.8090 - val_loss: 1.2267 - val_accuracy: 0.5585
Epoch 14/20
546/546 [==============================] - 226s 414ms/step - loss: 0.4105 - accuracy: 0.8147 - val_loss: 0.6451 - val_accuracy: 0.7112
Epoch 15/20
546/546 [==============================] - 183s 335ms/step - loss: 0.4103 - accuracy: 0.8159 - val_loss: 0.5352 - val_accuracy: 0.7428
Epoch 16/20
546/546 [==============================] - 168s 308ms/step - loss: 0.4104 - accuracy: 0.8141 - val_loss: 0.4273 - val_accuracy: 0.8064
Epoch 17/20
546/546 [==============================] - 125s 230ms/step - loss: 0.4015 - accuracy: 0.8181 - val_loss: 0.4554 - val_accuracy: 0.8006
Epoch 18/20
546/546 [==============================] - 77s 142ms/step - loss: 0.3934 - accuracy: 0.8271 - val_loss: 0.7176 - val_accuracy: 0.6309
Epoch 19/20
546/546 [==============================] - 70s 129ms/step - loss: 0.3920 - accuracy: 0.8248 - val_loss: 1.1621 - val_accuracy: 0.5946
Epoch 20/20
546/546 [==============================] - 71s 131ms/step - loss: 0.3988 - accuracy: 0.8206 - val_loss: 0.4312 - val_accuracy: 0.8072
INFO:tensorflow:Assets written to: after_model\assets
```

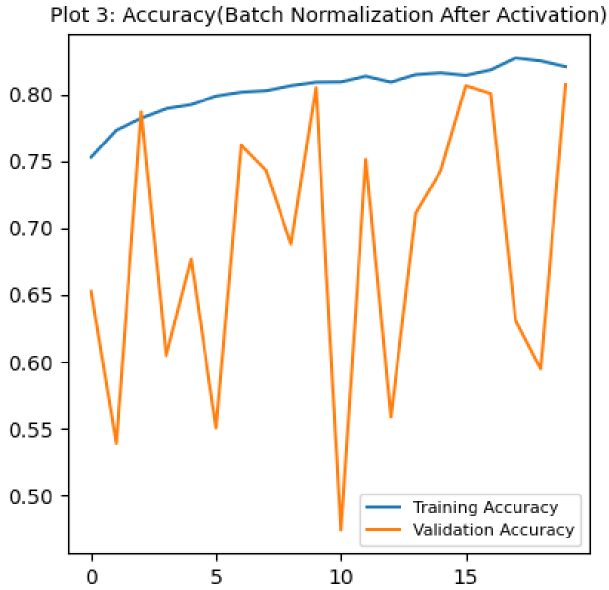
Model Comparison Results/Analysis

In [56]:

```python
#Before Model
before_model_accuracy = before_history.history['accuracy']
before_model_val_acc = before_history.history['val_accuracy']
before_model_loss = before_history.history['loss']
before_model_val_loss = before_history.history['val_loss']

before_epochs_range = range(number_of_epochs)

plt.figure(figsize=(10, 10))
plt.subplot(2, 2, 1)
plt.plot(before_epochs_range, before_model_accuracy, label='Training Accuracy')
plt.plot(before_epochs_range, before_model_val_acc, label='Validation Accuracy')
plt.legend(loc='lower right', fontsize = 8)
plt.title('Plot 1: Accuracy (Batch Normalization Before Activation)', fontsize = 10)

plt.subplot(2, 2, 2)
plt.plot(before_epochs_range, before_model_loss, label='Training Loss')
plt.plot(before_epochs_range, before_model_val_loss, label='Validation Loss')
plt.legend(loc='upper right', fontsize = 8)
```

```
plt.title('Plot 2: Loss (Batch Normalization Before Activation)', fontsize = 10)
plt.show()

#After Model
after_model_accuracy = after_history.history['accuracy']
after_model_val_acc = after_history.history['val_accuracy']
after_model_loss = after_history.history['loss']
after_model_val_loss = after_history.history['val_loss']


after_epochs_range = range(number_of_epochs)

plt.figure(figsize=(10, 10))
plt.subplot(2, 2, 1)
plt.plot(after_epochs_range, after_model_accuracy, label='Training Accuracy')
plt.plot(after_epochs_range, after_model_val_acc, label='Validation Accuracy')
plt.legend(loc='lower right', fontsize = 8)
plt.title('Plot 3: Accuracy(Batch Normalization After Activation)', fontsize = 10)

plt.subplot(2, 2, 2)
plt.plot(after_epochs_range, after_model_loss, label='Training Loss')
plt.plot(after_epochs_range, after_model_val_loss, label='Validation Loss')
plt.legend(loc='upper left', fontsize = 8)
plt.title('Plot 4: Loss (Batch Normalization After Activation)', fontsize = 10)
plt.show()
```





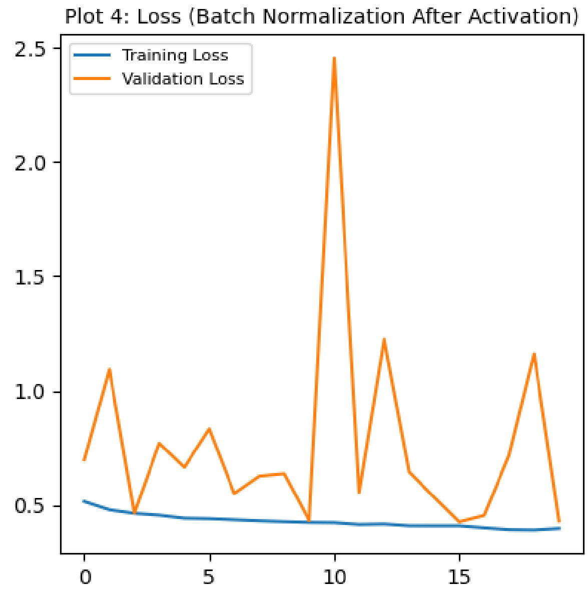


Table 1: Before and After Activation Layer vs Accuracy*

Model	Accuracy	Validation Accuracy
Before	.8444	.7460
After	.8206	.8070

Plots 1 thru 4 provide a view of the training accuracy and loss metrics between the models with the Batch Normalization layer before and after the activation layers.

Table 1 indicates that the before model has an accuracy of 84.44%, 2.36% better then the after model at 82.06%

However, when looking closely at the plots, at around the 10th epoch, the before model's training and validation accuracy begin to diverge, while the after model's accuracy trend continues to track and converge. This is an indication of potential overfitting in the before model.

The loss plots show a similiar pattern with the training and validation loss diverging in the before model while the after model's losses track and continue to exhibit a converging pattern. Again, another indication of overfitting of the before model.

Taking these two observations together it appears that the before model is showing signs of overfitting while the after model is stable and continues to improve. The implication being that placing the Batch Normalization layer after the Activation layer aided in protecting the after model from overfitting. In the end analysis, given the marginal difference in accuracy between the models, and the fact the the before model shows signs of overfitting, the after model is deemed the better of the two.

Test Submission

In [ ]:
```
print(test_image_df)

print(test_image_df.head())
print(test_image_df.info())


test_results = before_model.predict(generator_test_set,  batch_size = (generator_test_
print(test_results)
test_results

#3500/3500 [==============================] - 108s 19ms/step
```

In [ ]:
```
#Generate submission file
test_submission = np.where(test_results <= 0.5, 0, 1)

final_submission = np.transpose(test_submission)[0]
final_final_submission = pd.DataFrame()
final_submission['id'] = test_image_df['id'].apply(lambda x: x.split('.')[0])
final_submission['label'] = test_submission
final_submission.head()

final_submission.to_csv('submission.csv', index=False)
```

Hyperparameter Tuning

Momentum on the Batch Normalization Layers will be varied and the impact on accruacy assessed.

In [39]:
```
mv2 = [.99, .75, .50, .25]
history = []

for m in mv2:
    before_model = Sequential()

    before_model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3)))
    before_model.add(BatchNormalization(momentum = m))
    before_model.add(Activation('relu'))
    before_model.add(MaxPooling2D(pool_size=(3, 3)))

    before_model.add(Conv2D(64, (3, 3)))
    before_model.add(BatchNormalization(momentum = m))
    before_model.add(Activation('relu'))
    before_model.add(MaxPooling2D(pool_size=(3, 3)))

    before_model.add(Flatten())
    before_model.add(Dense(128))
    before_model.add(BatchNormalization(momentum = m))
    before_model.add(Activation('relu'))


    before_model.add(Dense(1))
    before_model.add(Activation('sigmoid'))

    before_model.summary()

    before_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accur
```

```
    number_of_epochs = 20


    tune_history = before_model.fit(generator_train_set,
                    epochs = number_of_epochs,
                    steps_per_epoch=(generator_train_set.n//generator_train_set.batch_
                    validation_data = generator_validation_set,
                    validation_steps=(generator_validation_set.n//generator_validatior




    print('Momentum: ', m)
    print('History: ', tune_history.history)
    history.append(tune_history.history)
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 30, 30, 32)        896
_________________________________________________________________
module_wrapper_15 (ModuleWra (None, 30, 30, 32)        128
_________________________________________________________________
activation_20 (Activation)   (None, 30, 30, 32)        0
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 10, 10, 32)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 8, 8, 64)          18496
_________________________________________________________________
module_wrapper_16 (ModuleWra (None, 8, 8, 64)          256
_________________________________________________________________
activation_21 (Activation)   (None, 8, 8, 64)          0
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 2, 2, 64)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_10 (Dense)             (None, 128)               32896
_________________________________________________________________
module_wrapper_17 (ModuleWra (None, 128)               512
_________________________________________________________________
activation_22 (Activation)   (None, 128)               0
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 129
_________________________________________________________________
activation_23 (Activation)   (None, 1)                 0
=================================================================
Total params: 53,313
Trainable params: 52,865
Non-trainable params: 448
_________________________________________________________________
Epoch 1/20
546/546 [==============================] - 312s 567ms/step - loss: 0.4850 - accuracy:
0.7683 - val_loss: 1.0514 - val_accuracy: 0.5354
Epoch 2/20
546/546 [==============================] - 248s 456ms/step - loss: 0.4522 - accuracy:
0.7924 - val_loss: 0.7236 - val_accuracy: 0.6920
Epoch 3/20
546/546 [==============================] - 123s 225ms/step - loss: 0.4332 - accuracy:
0.8035 - val_loss: 0.4410 - val_accuracy: 0.7863
Epoch 4/20
546/546 [==============================] - 175s 321ms/step - loss: 0.4219 - accuracy:
0.8085 - val_loss: 1.3560 - val_accuracy: 0.6011
Epoch 5/20
546/546 [==============================] - 152s 278ms/step - loss: 0.4178 - accuracy:
0.8110 - val_loss: 0.3918 - val_accuracy: 0.8206
Epoch 6/20
546/546 [==============================] - 265s 485ms/step - loss: 0.4045 - accuracy:
0.8192 - val_loss: 0.4177 - val_accuracy: 0.8064
Epoch 7/20
546/546 [==============================] - 196s 359ms/step - loss: 0.3994 - accuracy:
0.8185 - val_loss: 0.5115 - val_accuracy: 0.7640
Epoch 8/20
546/546 [==============================] - 126s 231ms/step - loss: 0.4000 - accuracy:
0.8226 - val_loss: 0.4740 - val_accuracy: 0.7631
```

```
Epoch 9/20
546/546 [==============================] - 107s 196ms/step - loss: 0.3916 - accuracy:
0.8250 - val_loss: 0.4882 - val_accuracy: 0.7515
Epoch 10/20
546/546 [==============================] - 116s 213ms/step - loss: 0.3861 - accuracy:
0.8279 - val_loss: 0.5286 - val_accuracy: 0.7489
Epoch 11/20
546/546 [==============================] - 97s 179ms/step - loss: 0.3860 - accuracy:
0.8280 - val_loss: 0.9395 - val_accuracy: 0.6245
Epoch 12/20
546/546 [==============================] - 115s 210ms/step - loss: 0.3784 - accuracy:
0.8328 - val_loss: 0.4330 - val_accuracy: 0.8098
Epoch 13/20
546/546 [==============================] - 96s 176ms/step - loss: 0.3804 - accuracy:
0.8304 - val_loss: 0.5197 - val_accuracy: 0.7423
Epoch 14/20
546/546 [==============================] - 113s 207ms/step - loss: 0.3748 - accuracy:
0.8372 - val_loss: 0.8836 - val_accuracy: 0.5774
Epoch 15/20
546/546 [==============================] - 104s 191ms/step - loss: 0.3692 - accuracy:
0.8372 - val_loss: 1.0588 - val_accuracy: 0.5222
Epoch 16/20
546/546 [==============================] - 121s 221ms/step - loss: 0.3689 - accuracy:
0.8401 - val_loss: 0.6488 - val_accuracy: 0.7425
Epoch 17/20
546/546 [==============================] - 121s 223ms/step - loss: 0.3624 - accuracy:
0.8398 - val_loss: 0.4285 - val_accuracy: 0.8122
Epoch 18/20
546/546 [==============================] - 121s 222ms/step - loss: 0.3588 - accuracy:
0.8439 - val_loss: 0.4089 - val_accuracy: 0.8095
Epoch 19/20
546/546 [==============================] - 113s 208ms/step - loss: 0.3576 - accuracy:
0.8444 - val_loss: 0.5920 - val_accuracy: 0.7465
Epoch 20/20
546/546 [==============================] - 113s 208ms/step - loss: 0.3602 - accuracy:
0.8405 - val_loss: 0.6725 - val_accuracy: 0.6893
Momentum:  0.99
History:  {'loss': [0.4849769175052643, 0.4521576464176178, 0.4332250654697418, 0.421
90587520599365, 0.41779088973999023, 0.4045145511627197, 0.39940616488456726, 0.40003
01659107208, 0.39164865016937256, 0.38612380623817444, 0.386012464761734, 0.378361195
32585144, 0.3803672790527344, 0.3748093247413635, 0.3691815435886383, 0.3688513040542
6025, 0.3624342978000641, 0.3588135242462158, 0.3576175272464752, 0.3602410256862640
4], 'accuracy': [0.7682661414146423, 0.7923534512519836, 0.8034814596176147, 0.808493
6141967773, 0.8109825849533081, 0.8191964030265808, 0.8184837102890015, 0.82263046503
06702, 0.8249542117118835, 0.8278960585594177, 0.8279889822006226, 0.832760989665985
1, 0.8303939700126648, 0.8372252583503723, 0.8371506929397583, 0.8400869965553284, 0.
8397846817970276, 0.8438644409179688, 0.8444228172302246, 0.8404876589775085], 'val_l
oss': [1.0514280796051025, 0.7236369252204895, 0.4410248100757599, 1.356036305427551
3, 0.39180848002433777, 0.4176807701587677, 0.511466920375824, 0.4740268290042877, 0.
4882241487503052, 0.5286386013031006, 0.939549446105957, 0.4330059885978699, 0.519674
0031242371, 0.8835945725440979, 1.0587953329086304, 0.6488038897514343, 0.42846411466
59851, 0.4088684618473053, 0.5919619202613831, 0.672546923160553], 'val_accuracy':
[0.5353899598121643, 0.6919786334037781, 0.7863247990608215, 0.6010695099830627, 0.82
0646345615387, 0.8064171075820923, 0.7640224099159241, 0.7631015777587891, 0.75146901
60751343, 0.7489304542541504, 0.624465823173523, 0.8097593784332275, 0.74225425720214
84, 0.5774064064025879, 0.5221688151359558, 0.7425133585929871, 0.8122329115867615,
0.8094919919967651, 0.7465277910232544, 0.6893048286437988]}
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
conv2d_12 (Conv2D)           (None, 30, 30, 32)        896
_________________________________________________________________
module_wrapper_18 (ModuleWra (None, 30, 30, 32)        128
_________________________________________________________________
activation_24 (Activation)   (None, 30, 30, 32)        0
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 10, 10, 32)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 8, 8, 64)          18496
_________________________________________________________________
module_wrapper_19 (ModuleWra (None, 8, 8, 64)          256
_________________________________________________________________
activation_25 (Activation)   (None, 8, 8, 64)          0
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 2, 2, 64)          0
_________________________________________________________________
flatten_6 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_12 (Dense)             (None, 128)               32896
_________________________________________________________________
module_wrapper_20 (ModuleWra (None, 128)               512
_________________________________________________________________
activation_26 (Activation)   (None, 128)               0
_________________________________________________________________
dense_13 (Dense)             (None, 1)                 129
_________________________________________________________________
activation_27 (Activation)   (None, 1)                 0
=================================================================
Total params: 53,313
Trainable params: 52,865
Non-trainable params: 448
_________________________________________________________________
Epoch 1/20
546/546 [==============================] - 116s 211ms/step - loss: 0.4919 - accuracy:
0.7659 - val_loss: 0.5644 - val_accuracy: 0.7285
Epoch 2/20
546/546 [==============================] - 107s 197ms/step - loss: 0.4530 - accuracy:
0.7923 - val_loss: 0.5167 - val_accuracy: 0.7654
Epoch 3/20
546/546 [==============================] - 96s 176ms/step - loss: 0.4406 - accuracy:
0.7970 - val_loss: 0.5783 - val_accuracy: 0.7190
Epoch 4/20
546/546 [==============================] - 106s 195ms/step - loss: 0.4201 - accuracy:
0.8094 - val_loss: 0.4414 - val_accuracy: 0.7971
Epoch 5/20
546/546 [==============================] - 95s 174ms/step - loss: 0.4116 - accuracy:
0.8178 - val_loss: 0.3914 - val_accuracy: 0.8268
Epoch 6/20
546/546 [==============================] - 105s 193ms/step - loss: 0.4132 - accuracy:
0.8136 - val_loss: 0.6397 - val_accuracy: 0.6877
Epoch 7/20
546/546 [==============================] - 92s 168ms/step - loss: 0.4006 - accuracy:
0.8197 - val_loss: 0.5008 - val_accuracy: 0.7718
Epoch 8/20
546/546 [==============================] - 103s 189ms/step - loss: 0.4020 - accuracy:
0.8211 - val_loss: 0.6255 - val_accuracy: 0.7083
Epoch 9/20
546/546 [==============================] - 90s 164ms/step - loss: 0.3875 - accuracy:
0.8275 - val_loss: 0.4052 - val_accuracy: 0.8228
```

```
Epoch 10/20
546/546 [==============================] - 102s 187ms/step - loss: 0.3901 - accuracy:
0.8257 - val_loss: 0.3849 - val_accuracy: 0.8253
Epoch 11/20
546/546 [==============================] - 90s 166ms/step - loss: 0.3813 - accuracy:
0.8335 - val_loss: 0.3837 - val_accuracy: 0.8317
Epoch 12/20
546/546 [==============================] - 105s 193ms/step - loss: 0.3825 - accuracy:
0.8285 - val_loss: 0.3756 - val_accuracy: 0.8306
Epoch 13/20
546/546 [==============================] - 92s 170ms/step - loss: 0.3671 - accuracy:
0.8401 - val_loss: 0.5295 - val_accuracy: 0.7519
Epoch 14/20
546/546 [==============================] - 104s 190ms/step - loss: 0.3829 - accuracy:
0.8311 - val_loss: 0.5657 - val_accuracy: 0.7541
Epoch 15/20
546/546 [==============================] - 91s 166ms/step - loss: 0.3727 - accuracy:
0.8365 - val_loss: 0.4875 - val_accuracy: 0.7963
Epoch 16/20
546/546 [==============================] - 109s 199ms/step - loss: 0.3683 - accuracy:
0.8366 - val_loss: 0.3829 - val_accuracy: 0.8318
Epoch 17/20
546/546 [==============================] - 102s 187ms/step - loss: 0.3620 - accuracy:
0.8413 - val_loss: 0.3756 - val_accuracy: 0.8313
Epoch 18/20
546/546 [==============================] - 108s 197ms/step - loss: 0.3586 - accuracy:
0.8414 - val_loss: 0.3940 - val_accuracy: 0.8277
Epoch 19/20
546/546 [==============================] - 109s 200ms/step - loss: 0.3627 - accuracy:
0.8406 - val_loss: 0.3477 - val_accuracy: 0.8536
Epoch 20/20
546/546 [==============================] - 116s 212ms/step - loss: 0.3566 - accuracy:
0.8423 - val_loss: 0.3610 - val_accuracy: 0.8469
Momentum:  0.75
History:  {'loss': [0.491936594247818, 0.4529515504837036, 0.44059401750564575, 0.420
05667090415955, 0.4116076529026315, 0.41323259472846985, 0.40060049295425415, 0.4019
608199596405, 0.3874862492084503, 0.3901159465312958, 0.38133299350738525, 0.38254237
174987793, 0.3670882284641266, 0.3828747868537903, 0.372717946767807, 0.3683088123798
3704, 0.3619937300682068, 0.35858792066574097, 0.3626682758331299, 0.356631845235824
6], 'accuracy': [0.7659184336662292, 0.7922962307929993, 0.7969537377357483, 0.809409
3203544617, 0.8177965879440308, 0.8136447072029114, 0.819686233997345, 0.821085155010
2234, 0.8274736404418945, 0.8257211446762085, 0.8335433006286621, 0.828468382358551,
0.8400710225105286, 0.8311011791229248, 0.8364635705947876, 0.8365957140922546, 0.841
3307666778564, 0.8414033651351929, 0.8405863642692566, 0.8422619104385376], 'val_los
s': [0.5643996000289917, 0.5167384147644043, 0.5783010125160217, 0.4413726329803467,
0.39136797189712524, 0.639728844165802, 0.5008347034454346, 0.6255342364311218, 0.405
185729265213, 0.3849208354949951, 0.3837343156337738, 0.3756384551525116, 0.529545724
3919373, 0.5656939744949341, 0.4875234365463257, 0.3828989863395691, 0.37563121318817
14, 0.3940175473690033, 0.3477363884449005, 0.3609672486782074], 'val_accuracy': [0.7
284989356994629, 0.7653743028640747, 0.7190170884132385, 0.7970588207244873, 0.826789
5579338074, 0.6877005100250244, 0.7717681527137756, 0.7082887887954712, 0.82278311252
59399, 0.8252673745155334, 0.8317307829856873, 0.8306149840354919, 0.751869678497314
5, 0.7541443705558777, 0.796340823173523, 0.831818163394928, 0.8313301205635071, 0.82
76737928390503, 0.8536324501037598, 0.8469251394271851]}
Model: "sequential_7"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
conv2d_14 (Conv2D)          (None, 30, 30, 32)        896

_________________________________________________________________
```

```
module_wrapper_21 (ModuleWra (None, 30, 30, 32)        128
_________________________________________________________________
activation_28 (Activation)   (None, 30, 30, 32)        0
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 10, 10, 32)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 8, 8, 64)          18496
_________________________________________________________________
module_wrapper_22 (ModuleWra (None, 8, 8, 64)          256
_________________________________________________________________
activation_29 (Activation)   (None, 8, 8, 64)          0
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 2, 2, 64)          0
_________________________________________________________________
flatten_7 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_14 (Dense)             (None, 128)               32896
_________________________________________________________________
module_wrapper_23 (ModuleWra (None, 128)               512
_________________________________________________________________
activation_30 (Activation)   (None, 128)               0
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 129
_________________________________________________________________
activation_31 (Activation)   (None, 1)                 0
=================================================================
Total params: 53,313
Trainable params: 52,865
Non-trainable params: 448
_________________________________________________________________
Epoch 1/20
546/546 [==============================] - 100s 181ms/step - loss: 0.4783 - accuracy:
0.7747 - val_loss: 0.4534 - val_accuracy: 0.7905
Epoch 2/20
546/546 [==============================] - 108s 199ms/step - loss: 0.4430 - accuracy:
0.7969 - val_loss: 0.4436 - val_accuracy: 0.7955
Epoch 3/20
546/546 [==============================] - 179s 328ms/step - loss: 0.4254 - accuracy:
0.8059 - val_loss: 0.4087 - val_accuracy: 0.8145
Epoch 4/20
546/546 [==============================] - 270s 495ms/step - loss: 0.4230 - accuracy:
0.8078 - val_loss: 0.4225 - val_accuracy: 0.8075
Epoch 5/20
546/546 [==============================] - 152s 279ms/step - loss: 0.4093 - accuracy:
0.8149 - val_loss: 0.4600 - val_accuracy: 0.7768
Epoch 6/20
546/546 [==============================] - 151s 276ms/step - loss: 0.4120 - accuracy:
0.8132 - val_loss: 0.3939 - val_accuracy: 0.8207
Epoch 7/20
546/546 [==============================] - 82s 150ms/step - loss: 0.4052 - accuracy:
0.8162 - val_loss: 0.4195 - val_accuracy: 0.8124
Epoch 8/20
546/546 [==============================] - 80s 147ms/step - loss: 0.4048 - accuracy:
0.8174 - val_loss: 0.4297 - val_accuracy: 0.7983
Epoch 9/20
546/546 [==============================] - 76s 140ms/step - loss: 0.3873 - accuracy:
0.8247 - val_loss: 0.4254 - val_accuracy: 0.8061
Epoch 10/20
546/546 [==============================] - 82s 150ms/step - loss: 0.3910 - accuracy:
0.8277 - val_loss: 0.3852 - val_accuracy: 0.8333
```

```
Epoch 11/20
546/546 [==============================] - 75s 137ms/step - loss: 0.3797 - accuracy: 0.8310 - val_loss: 0.3611 - val_accuracy: 0.8451
Epoch 12/20
546/546 [==============================] - 78s 143ms/step - loss: 0.3891 - accuracy: 0.8263 - val_loss: 0.3924 - val_accuracy: 0.8223
Epoch 13/20
546/546 [==============================] - 81s 148ms/step - loss: 0.3823 - accuracy: 0.8278 - val_loss: 0.4996 - val_accuracy: 0.7730
Epoch 14/20
546/546 [==============================] - 86s 158ms/step - loss: 0.3770 - accuracy: 0.8325 - val_loss: 0.3968 - val_accuracy: 0.8247
Epoch 15/20
546/546 [==============================] - 87s 159ms/step - loss: 0.3695 - accuracy: 0.8376 - val_loss: 0.3887 - val_accuracy: 0.8360
Epoch 16/20
546/546 [==============================] - 85s 156ms/step - loss: 0.3650 - accuracy: 0.8393 - val_loss: 0.3647 - val_accuracy: 0.8382
Epoch 17/20
546/546 [==============================] - 82s 150ms/step - loss: 0.3595 - accuracy: 0.8436 - val_loss: 0.4219 - val_accuracy: 0.8070
Epoch 18/20
546/546 [==============================] - 97s 179ms/step - loss: 0.3588 - accuracy: 0.8438 - val_loss: 0.3786 - val_accuracy: 0.8293
Epoch 19/20
546/546 [==============================] - 96s 177ms/step - loss: 0.3543 - accuracy: 0.8470 - val_loss: 0.3733 - val_accuracy: 0.8415
Epoch 20/20
546/546 [==============================] - 104s 191ms/step - loss: 0.3508 - accuracy: 0.8494 - val_loss: 0.3790 - val_accuracy: 0.8341
Momentum:  0.5
History:  {'loss': [0.4782549738883972, 0.44297072291374207, 0.42543235421180725, 0.4230184555053711, 0.40933364629745483, 0.4120127856731415, 0.4052249491214752, 0.40475207567214966, 0.38731464743614197, 0.39102041721343994, 0.3796846568584442, 0.38907477259635925, 0.3822660446166992, 0.37701615691185, 0.36950573325157166, 0.3649819493293762, 0.3595219552516937, 0.35878658294677734, 0.35427695512771606, 0.35083621740341187], 'accuracy': [0.7747365832328796, 0.7969322204589844, 0.805886387825012 2, 0.8078067898750305, 0.8148763179779053, 0.8131868243217468, 0.816193282604217 5, 0.8173649311065674, 0.824725151062011 7, 0.8277243375778198, 0.8310238122940063, 0.8262934684753418, 0.827817201614379 9, 0.8325320482254028, 0.8375515341758728, 0.8392857313156128, 0.843563914299011 2, 0.84375, 0.8469995260238647, 0.8493589758872986], 'val_loss': [0.4534325301647 1863, 0.4435521364212036, 0.4087141752243042, 0.4224659502506256, 0.4600068406486 51, 0.3938656747341156, 0.419475793838501, 0.42965251207351685, 0.4254249036312103, 0.38515013456344604, 0.3611338436603546, 0.3924413025379181, 0.4995637536048889, 0.396753191947937, 0.38871046900749207, 0.3647327125072479, 0.4219432771205902, 0.378569632768631, 0.3733358383178711, 0.3789542615413666], 'val_accuracy': [0.7904647588729858, 0.7954545617103577, 0.8145031929016113, 0.8074866533279419, 0.7768429517745972, 0.8207219243049622, 0.8123664259910583, 0.7982620596885681, 0.8060897588729858, 0.8332887887954712, 0.8450854420661926, 0.8223261833190918, 0.7729700803756714, 0.8247326016426086, 0.8360042572021484, 0.8382353186607361, 0.8070245981216431, 0.8292780518531799, 0.8414797186851501, 0.8340908885002136]}
Model: "sequential_8"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)          (None, 30, 30, 32)        896
_________________________________________________________________
module_wrapper_24 (ModuleWra (None, 30, 30, 32)       128
_________________________________________________________________
activation_32 (Activation)  (None, 30, 30, 32)        0
```

```
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 10, 10, 32)        0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 8, 8, 64)          18496
_________________________________________________________________
module_wrapper_25 (ModuleWra (None, 8, 8, 64)          256
_________________________________________________________________
activation_33 (Activation)   (None, 8, 8, 64)          0
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 2, 2, 64)          0
_________________________________________________________________
flatten_8 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_16 (Dense)             (None, 128)               32896
_________________________________________________________________
module_wrapper_26 (ModuleWra (None, 128)               512
_________________________________________________________________
activation_34 (Activation)   (None, 128)               0
_________________________________________________________________
dense_17 (Dense)             (None, 1)                 129
_________________________________________________________________
activation_35 (Activation)   (None, 1)                 0
=================================================================
Total params: 53,313
Trainable params: 52,865
Non-trainable params: 448
_________________________________________________________________
Epoch 1/20
546/546 [==============================] - 102s 185ms/step - loss: 0.4946 - accuracy: 0.7656 - val_loss: 0.4714 - val_accuracy: 0.7804
Epoch 2/20
546/546 [==============================] - 124s 227ms/step - loss: 0.4540 - accuracy: 0.7895 - val_loss: 0.4138 - val_accuracy: 0.8099
Epoch 3/20
546/546 [==============================] - 94s 172ms/step - loss: 0.4432 - accuracy: 0.7977 - val_loss: 0.4295 - val_accuracy: 0.8010
Epoch 4/20
546/546 [==============================] - 94s 172ms/step - loss: 0.4223 - accuracy: 0.8088 - val_loss: 0.4107 - val_accuracy: 0.8045
Epoch 5/20
546/546 [==============================] - 89s 163ms/step - loss: 0.4161 - accuracy: 0.8120 - val_loss: 0.3846 - val_accuracy: 0.8291
Epoch 6/20
546/546 [==============================] - 102s 186ms/step - loss: 0.4104 - accuracy: 0.8149 - val_loss: 0.3833 - val_accuracy: 0.8345
Epoch 7/20
546/546 [==============================] - 120s 220ms/step - loss: 0.3966 - accuracy: 0.8230 - val_loss: 0.4058 - val_accuracy: 0.8185
Epoch 8/20
546/546 [==============================] - 134s 245ms/step - loss: 0.4013 - accuracy: 0.8206 - val_loss: 0.3910 - val_accuracy: 0.8274
Epoch 9/20
546/546 [==============================] - 134s 247ms/step - loss: 0.3927 - accuracy: 0.8220 - val_loss: 0.3886 - val_accuracy: 0.8288
Epoch 10/20
546/546 [==============================] - 134s 245ms/step - loss: 0.3934 - accuracy: 0.8242 - val_loss: 0.4313 - val_accuracy: 0.7984
Epoch 11/20
546/546 [==============================] - 129s 237ms/step - loss: 0.3858 - accuracy: 0.8299 - val_loss: 0.3734 - val_accuracy: 0.8331
```

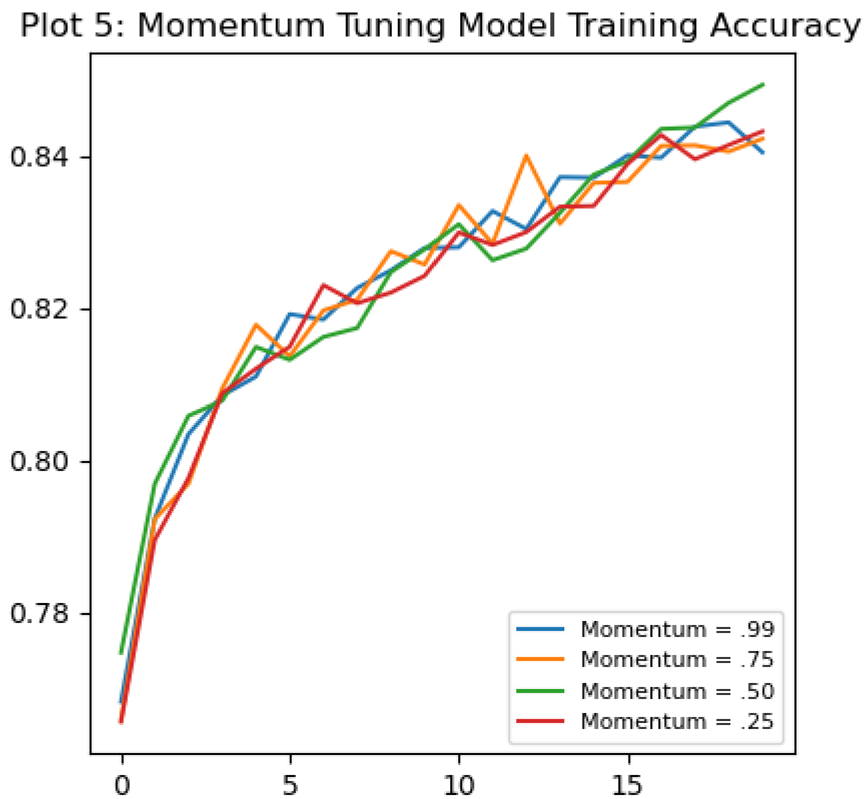
```
Epoch 12/20
546/546 [==============================] - 115s 210ms/step - loss: 0.3871 - accuracy:
0.8283 - val_loss: 0.3963 - val_accuracy: 0.8198
Epoch 13/20
546/546 [==============================] - 104s 191ms/step - loss: 0.3827 - accuracy:
0.8300 - val_loss: 0.3971 - val_accuracy: 0.8196
Epoch 14/20
546/546 [==============================] - 106s 194ms/step - loss: 0.3773 - accuracy:
0.8333 - val_loss: 0.4257 - val_accuracy: 0.8032
Epoch 15/20
546/546 [==============================] - 104s 191ms/step - loss: 0.3748 - accuracy:
0.8334 - val_loss: 0.3680 - val_accuracy: 0.8412
Epoch 16/20
546/546 [==============================] - 105s 193ms/step - loss: 0.3683 - accuracy:
0.8389 - val_loss: 0.3726 - val_accuracy: 0.8320
Epoch 17/20
546/546 [==============================] - 102s 186ms/step - loss: 0.3604 - accuracy:
0.8428 - val_loss: 0.3852 - val_accuracy: 0.8301
Epoch 18/20
546/546 [==============================] - 105s 192ms/step - loss: 0.3644 - accuracy:
0.8396 - val_loss: 0.3531 - val_accuracy: 0.8479
Epoch 19/20
546/546 [==============================] - 107s 196ms/step - loss: 0.3625 - accuracy:
0.8414 - val_loss: 0.3528 - val_accuracy: 0.8471
Epoch 20/20
546/546 [==============================] - 140s 256ms/step - loss: 0.3511 - accuracy:
0.8432 - val_loss: 0.3590 - val_accuracy: 0.8436
Momentum:  0.25
History:  {'loss': [0.49455761909484863, 0.45400887727737427, 0.4431801736354828, 0.4
2225244641304016, 0.41612425446510315, 0.4103754758834839, 0.39658114314079285, 0.401
31276845932007, 0.392747163772583, 0.39335402846336365, 0.38575831055641174, 0.387124
240398407, 0.3826524615287781, 0.3773442804813385, 0.3748388886451721, 0.368334293365
4785, 0.3604435622692108, 0.36442652344703674, 0.3624778091907501, 0.351147383451461
8], 'accuracy': [0.765632152557373, 0.7895489931106567, 0.7976981401443481, 0.8087797
76096344, 0.8120132684707642, 0.8149038553237915, 0.823007345199585, 0.82062727212905
88, 0.8220338821411133, 0.8242330551147461, 0.8299358487129211, 0.8282967209815979,
0.8299931287765503, 0.8333333134651184, 0.8334287405014038, 0.838942289352417, 0.8427
622318267822, 0.8395718932151794, 0.84144526720047, 0.8432348966598511], 'val_loss':
[0.47139647603034973, 0.41383710503578186, 0.4294649362564087, 0.41069862246513367,
0.38461852073669434, 0.38328418135643005, 0.4058365821838379, 0.3909931778907776, 0.3
885525166988373, 0.43133559823036194, 0.37339961528778076, 0.3963363766670227, 0.3970
6292748451233, 0.4257175922393799, 0.3679625988006592, 0.3725605607032776, 0.38522690
534591675, 0.35314831137657166, 0.3528195023536682, 0.3589937388896942], 'val_accurac
y': [0.7804487347602844, 0.8098930716514587, 0.8010149598121643, 0.8045454621315002,
0.8290598392486572, 0.8344919681549072, 0.8185096383094788, 0.8274064064025879, 0.828
7927508354187, 0.7983956933021545, 0.8330662250518799, 0.8197860717773438, 0.81957799
19624329, 0.8032085299491882, 0.8412126302719116, 0.8319518566131592, 0.8301281929016
113, 0.8478609919548035, 0.8470886945724487, 0.8435828685760498]}
```

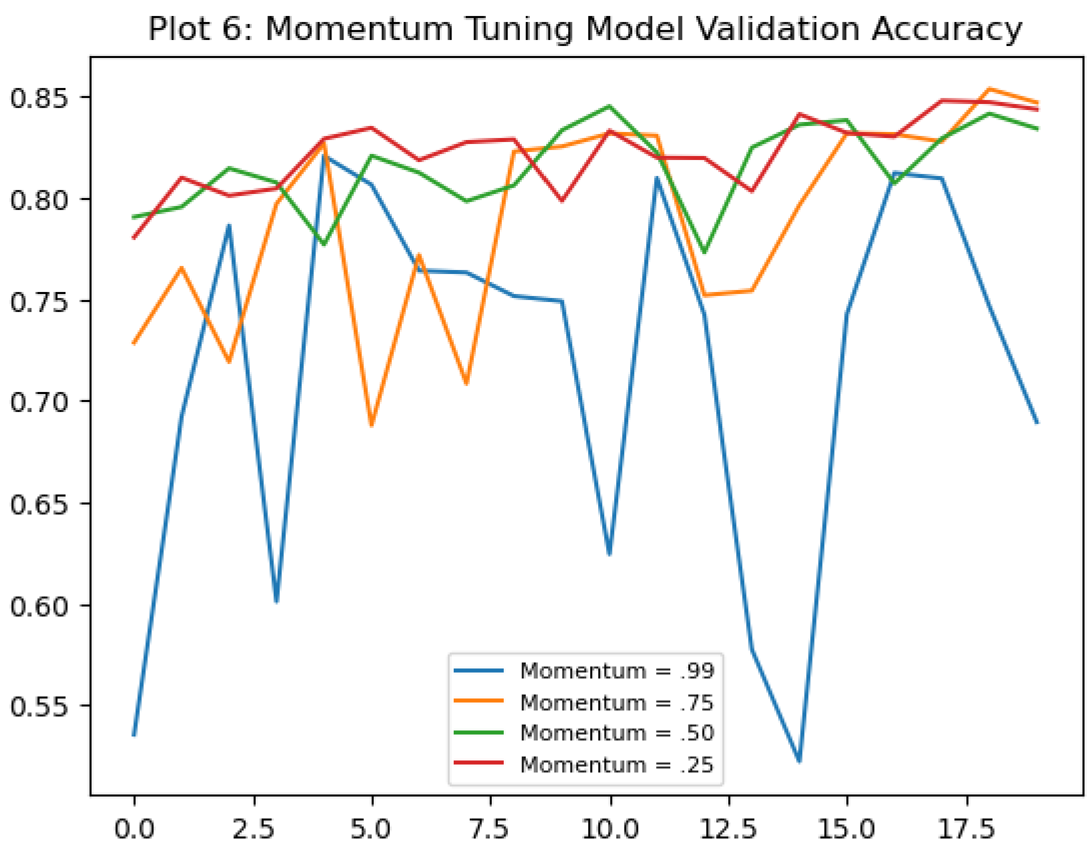
Hyperparameter Tuning Results / Analysis

In [44]:

```python
tune_epochs_range = range(number_of_epochs)

plt.figure(figsize=(10, 10))
plt.subplot(2, 2, 1)
plt.plot(tune_epochs_range, history[0]['accuracy'], label='Momentum = .99')
plt.plot(tune_epochs_range, history[1]['accuracy'], label='Momentum = .75')
plt.plot(tune_epochs_range, history[2]['accuracy'], label='Momentum = .50')
plt.plot(tune_epochs_range, history[3]['accuracy'], label='Momentum = .25')
```

```
plt.legend(loc='lower right', fontsize=8)
plt.title('Plot 5: Momentum Tuning Model Training Accuracy')
plt.show()
#plt.subplot(2, 2, 2)
plt.plot(tune_epochs_range, history[0]['val_accuracy'], label='Momentum = .99')
plt.plot(tune_epochs_range, history[1]['val_accuracy'], label='Momentum = .75')
plt.plot(tune_epochs_range, history[2]['val_accuracy'], label='Momentum = .50')
plt.plot(tune_epochs_range, history[3]['val_accuracy'], label='Momentum = .25')
plt.legend(loc='lower center', fontsize=8)
plt.title('Plot 6: Momentum Tuning Model Validation Accuracy')
plt.show()
```





Accuracy Comparison / Analysis

Table 2: Momentum vs Accuracy

Momentum	Accuracy	Validation Accuracy
.99	.8409	.7505
.75	.8375	.8374
.50	.8431	.8449
.25	.8379	.8389

As can be seen in Plot 5 and Table 2, 'Momentum vs Accuracy', all 4 momentum levels reach similar levels of accuracy. Model 3, momentum = .5, performed best with an accuracy of 84.31%. The worst performer was Model 2, momentum = .75, with accuracy of 83.75%. When comparing models strictly from an accuracy perspective there is no appreciable difference between models. The best and worst performers were separated by only .56%, essentially no difference.

However, it's how the models got there that is interesting. Plot 5 shows that Model 4, momentum = .25, exhibited the least amount of variability through its trajectory. On the other hand, Model 1, momentum = .99, varied wildly throughout its path. This is not unexpected as the larger momentum model would be expected to 'bounce' more than the lower momentum models.

Plot 6, Validation Accuracy, more clearly details how the higher momentums impact the loss trajectory. As expected, as the momentum of the models decreased, so did the variation in the loss path.

Given the marginal difference in accuracy in conjunction with the variation in the accuracy trajectories, the data shows that working with a lower momentum provides a more stable model in terms of accuracy and loss. Model 4, momentum = .25 is the preferred model.

Conclusion

Model Comparison

Over 15 epochs, comparisons of the before and after models showed marginal accuracy and loss differences. The before model performed slightly better with its final accuracy 2.38% better at 84.44%, then the after model at 80.20%.

However as outlined above, the before model began to show signs of overfitting at around the 10th epoch, while the after model did not. Based on the analysis and discussion above, the conclusion is that placing the Batch Normalization layer after the Activation layer proved to be beneficial. Given the marginal difference in performance and the overfitting protection in the after model, the after model is deemed the better of the two.

Hyperparamter Tuning

Hyperparameter tuning, varying the momentum parameter, provided some insight into how momentum impacted the model's performance. The total difference in performance across the four different momentum values resulted in a maximum accuracy difference of .56%. The middle two momentum values (.75 and .50) resulted in a .56% difference in accuracy.

When considering that the accurracies and accuracy trajectories are essentially identical, the deciding factor comes dowm to the variability in the accuracy paths. Given that Model 4, momentum of .25, exhibited the least amount of variability, Model 4 is considered the most stable and therefore deemed the best model of the four.

In [ ]: