In [27]:

```
#Please note that the code in this .py file was run in Jupyter notebook
#for the creation of a model, hence the inline imports. This code will
#only run in Jupyter notebook
#It is put into the same folder as the other .py files for the convenience
#of searching for the code
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
import os
```

In [28]:

```
#The keras library allows for an easier building of the
#Neural network model
import tensorflow as tf
from tensorflow import keras
```

In [29]:

```
#The training directories are located on my PC. Please change it according to your data
train_dir = r'C:\Users\Sahil\Desktop\Tries\ToBeProcessedDataset\train'
validation_dir = r'C:\Users\Sahil\Desktop\Tries\ToBeProcessedDataset\validation'
test_dir = r'C:\Users\Sahil\Desktop\Tries\ToBeProcessedDataset\test'
```

In [30]:

```
#The ImageDataGenerator library
#allows for the augmenting of data
#so that there isnt an
#overfitting of of the training data to the validation
#data
#Some steps include
#Data Preprocessing
#Read the picture files
#Decode the JPEG content to RGB grid of pixels
#Convert these into floating point tensors
#Rescale the pixel values(between 0 and 255) to the [0,1] interval
from tensorflow.keras.preprocessing.image import ImageDataGenerator
#height shift in the range 20 percent
train_datagen = ImageDataGenerator(
rescale = 1./255,
rotation_range = 40,
width_shift_range = 0.2,
height_shift_range = 0.2,
shear_range = 0.2,
zoom_range = 0.2,
horizontal_flip = True)
test_datagen = ImageDataGenerator(rescale = 1./ 255)
train_generator = train_datagen.flow_from_directory(
train_dir,
target_size=(150,150),
batch_size = 20,
class_mode = 'categorical')
validation_generator = test_datagen.flow_from_directory(
validation_dir,
target_size =(150, 150),
batch_size=2,
class_mode = 'categorical')
#link: https://keras.io/preprocessing/image/ for further reading
```

```
Found 706 images belonging to 10 classes.
Found 466 images belonging to 10 classes.
```

In [31]:

```
#Our model uses a VGG16 transfer learning model which won the ILSVRC competion
#We initially used a custom built CNN architecture,
#however, the VGG16 architecture showed far higher accuracies than the
#custom built architecture
#We only use the convolutional base
#as we prefer to use our own classification layer
#The input image shape is allowed to be 150*150 due to the
#150 input neurons
from tensorflow.keras.applications import VGG16
conv_base = VGG16(weights='imagenet',
include_top= False,
input_shape =(150,150,3))
```

In [32]:

```
#This method gives an indication of the
#look of the nn structure, and
#the arrangement of hyperparameter
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_3 (InputLayer)         [(None, 150, 150, 3)]     0         
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 150, 150, 64)      1792      
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 150, 150, 64)      36928     
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 75, 75, 64)        0         
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 75, 75, 128)       73856     
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 75, 75, 128)       147584    
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 37, 37, 128)       0         
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 37, 37, 256)       295168    
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 37, 37, 256)       590080    
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 37, 37, 256)       590080    
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 18, 18, 256)       0         
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 18, 18, 512)       1180160   
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 18, 18, 512)       2359808   
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 18, 18, 512)       2359808   
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 9, 9, 512)         0         
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 9, 9, 512)         2359808   
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 9, 9, 512)         2359808   
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 9, 9, 512)         2359808   
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0         
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

In [33]:

```
from tensorflow.keras import layers
from tensorflow.keras import models
```

In [34]:

```
#We use the VGG16 base in a
#sequential model for the base, and
#add a flattening layer for the dense
#layers to be compatible with the conv base input

model = models.Sequential()

model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

```

In [35]:

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688  
_________________________________________________________________
flatten_2 (Flatten)          (None, 8192)              0         
_________________________________________________________________
dense_10 (Dense)             (None, 256)               2097408   
_________________________________________________________________
dense_11 (Dense)             (None, 256)               65792     
_________________________________________________________________
dense_12 (Dense)             (None, 256)               65792     
_________________________________________________________________
dense_13 (Dense)             (None, 256)               65792     
_________________________________________________________________
dense_14 (Dense)             (None, 10)                2570      
=================================================================
Total params: 17,012,042
Trainable params: 17,012,042
Non-trainable params: 0
_________________________________________________________________
```

In [36]:

```
#We use the categorical crossentropy in optimizing our model
#The acc and AUC metrics represent accuracy and area under curve
from tensorflow.keras import optimizers
model.compile(loss='categorical_crossentropy',
optimizer= optimizers.RMSprop(lr= 2e-5),
metrics = ['acc', tf.keras.metrics.AUC()])
```

In [37]:

```
#can also use save best here, if you dont want to save all epochs
checkpoint_cb = keras.callbacks.ModelCheckpoint("CNN_Project_Model-{epoch:02d}.h5")
```

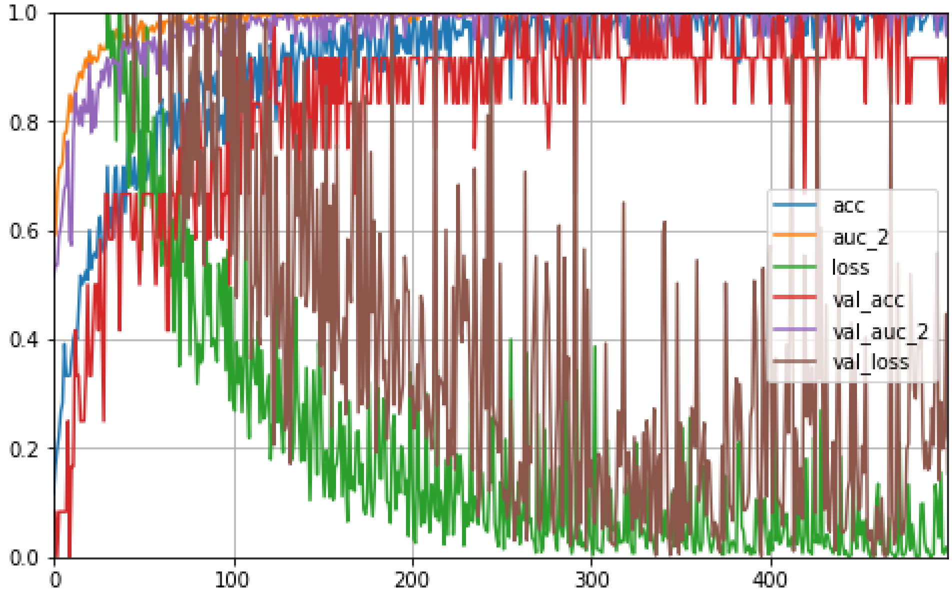
In [38]:

```
#This code starts the training of the network
history = model.fit_generator(
train_generator,
steps_per_epoch=6,
epochs =500,
validation_data = validation_generator,
validation_steps = 6,
callbacks=[checkpoint_cb])
```

```
Epoch 496/500
6/6 [==============================] - 92s 15s/step - loss: 0.1561 - acc:
0.9667 - auc_2: 0.9950 - val_loss: 0.2854 - val_acc: 0.8333 - val_auc_2:
0.9969
Epoch 497/500
6/6 [==============================] - 92s 15s/step - loss: 0.0090 - acc:
1.0000 - auc_2: 1.0000 - val_loss: 0.1988 - val_acc: 0.9167 - val_auc_2:
0.9977
Epoch 498/500
6/6 [==============================] - 91s 15s/step - loss: 0.0060 - acc:
1.0000 - auc_2: 1.0000 - val_loss: 0.2224 - val_acc: 0.8333 - val_auc_2:
0.9961
Epoch 499/500
6/6 [==============================] - 93s 16s/step - loss: 0.0184 - acc:
0.9917 - auc_2: 1.0000 - val_loss: 0.4467 - val_acc: 0.9167 - val_auc_2:
0.9564
Epoch 500/500
6/6 [==============================] - 82s 14s/step - loss: 0.0096 - acc:
1.0000 - auc_2: 1.0000 - val_loss: 0.1535 - val_acc: 0.9167 - val_auc_2:
0.9992
```

In [39]:

```
#This code graphically represents the accuracy,
#validation accuracy, validation loss etc,
#as the model gets trained.
#we see how the training data fits the validation
#data and the accuracy increasing as time moves forward
pd.DataFrame(history.history).plot(figsize = (8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show()
```



In [40]:

```
hist_df = pd.DataFrame(history.history)
#save history variable into a csv file
hist_csv_file = 'history.csv'
with open(hist_csv_file, mode='w') as f:
    hist_df.to_csv(f)
```

In [41]:

```
#We were only calculating accuracies on validation set
#lets see how it performs on test set
#test_datagen is same object for validation. Reshapping data from 0 to 255 to 0 to 1
test_generator = test_datagen.flow_from_directory(
test_dir,
target_size= (150,150),
batch_size=2,
class_mode = 'categorical')
```

```
Found 250 images belonging to 10 classes.
```

In [42]:

```
model.evaluate_generator(test_generator, steps = 2)
```

```
WARNING:tensorflow:From <ipython-input-42-126d51bbf105>:1: Model.evaluate_ge
nerator (from tensorflow.python.keras.engine.training) is deprecated and wil
l be removed in a future version.
Instructions for updating:
Please use Model.evaluate, which supports generators.

WARNING:tensorflow:From <ipython-input-42-126d51bbf105>:1: Model.evaluate_ge
nerator (from tensorflow.python.keras.engine.training) is deprecated and wil
l be removed in a future version.
Instructions for updating:
Please use Model.evaluate, which supports generators.

WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']

WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
```

Out[42]:

```
[0.03480612859129728, 1.0, 1.0]
```

In [43]:

```
#We save the trained network as an .h5
#file here, to allow for the model to be used in other applications
model.save('banknoteauthdentest.h5')
```

In [44]:

```
from tensorflow.keras.preprocessing import image
```

In [3]:

```python
from tkinter import *
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np
root = Tk()

e = Entry(root, width =50, borderwidth =5)
e.pack()

# dimensions of our images    -----   are these then grayscale (black and white)?
img_width, img_height = 150, 150

# load the model we saved
model = load_model('banknoteauthdentest.h5')
link =""

def myClick():
    link = e.get()
    link = link.replace('\\','/')
    # predicting images
    img = image.load_img(link
        ,
        target_size=(img_width, img_height))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)

    images = np.vstack([x])
    classes = model.predict_classes(images, batch_size=10)
    print(classes)

    # predicting multiple images at once
    img = image.load_img(link
         ,
        target_size=(img_width, img_height))
    y = image.img_to_array(img)
    y = np.expand_dims(y, axis=0)

    # pass the list of multiple images np.vstack()
    images = np.vstack([x, y])
    classes = model.predict_classes(images, batch_size=10)

    # print the classes, the images belong to
    print(classes)
    print(classes[0])

    prediction = 'cant process'
    if classes[0] == 0:
        prediction = 'fifty'
    elif classes[0] == 1:
        prediction = 'fake fifty'
    elif classes[0] == 2:
        prediction = 'hundred'
    elif classes[0] == 3:
        prediction = 'fake hundred'
    elif classes[0] == 4:
        prediction = 'ten'
    elif classes[0] == 5:
        prediction = 'fake ten'
    elif classes[0] == 6:
        prediction = 'twenty'
    elif classes[0] == 7:
        prediction = 'fake twenty'
    elif classes[0] == 8:
        prediction = 'two hundred'
    elif classes[0] == 9:
        prediction = 'fake two hundred'
    print(prediction)


    myLabel = Label(root, text = prediction)
    myLabel.pack()




myButton = Button(root, text = "Process", command= myClick)
myButton.pack()

root.mainloop()
```

```
[0]
[0 0]
0
fifty
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: