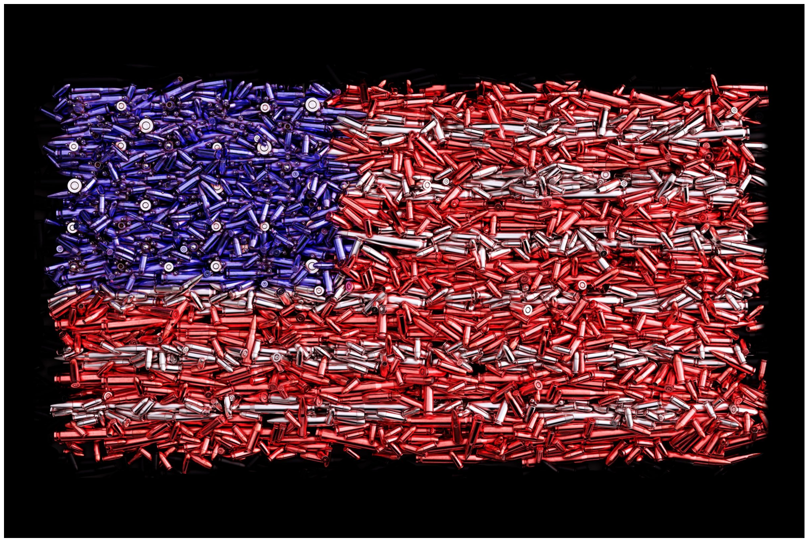
Gun Image Classification with CNNs

Eric Au & Henry Chung



Source: Paul Campbell - Getty Images

Abstract

Business Understanding

Unfortunately, we often see news reports of acts of violence involving guns in the United States. While it is difficult to identify potentially dangerous individuals from committing such crimes, there is the thought that these mass shootings we see on the news could have been prevented.

For instance, there has been evidence after the fact, that these dangerous individuals have exhibited violent behavior through online social media platforms (The Guardian) (https://www.theguardian.com/us-news/2022/may/28/texas-gunman-threats-behavior).

There have even been recent legislative efforts (https://www.cityandstateny.com/policy/2022/05/hochul-proposes-new-gun-laws-social-media-crackdowns-after-mass-shooting/367113/) in New York State following the mass shooting in Buffalo on May 2022 establishing Domestic Terrorism Units tasked with tracking down people looking to commit acts of violence.

With these preventative efforts in mind, we will explore whether image recognition technology has the ability to distinguish between images of guns and not guns.

Stakeholders: New York State Domestic Terrorism Unit

Is there a way to detect potentially dangerous individuals by image recognition of guns?

- Where is gun violence in America most prevalent?
- Has there been an increase in gun violence? Are there trends we are seeing over time?

For further exploration into these questions, please see the exploratory data analysis (EDA) notebook.

Data Sources:

- Center for Disease, Control, and Prevention (https://wonder.cdc.gov/ucd-icd10.html)

Image Data:

- Andalusian Research Institute in Data Science and Computational Intelligence (DaSCI) (https://dasci.es/transferencia/open-data/24705/)
- Kaggle (https://www.kaggle.com/datasets/ahmadahmadzada/images2000)

The dataset consisted of 65% of various gun types including handguns and rifles. The remaining 35% consisted of other hand-held objects such as knives, phones, bats, and money. In other words, the data was classified as follows: 0 for "not gun" and 1 for "gun".

The Task:

Perform an analysis on gun violence in the United States and create a machine learning model that can classify images of guns and not guns. The intention is to proactively detect potentially dangerous individuals on social media and for authorities to respond appropriately.

For reference, this project implements Keras library. Keras is one of the leading high-level neural networks APIs and does a great job in the classification of images.

Data Processing:

To prepare the images for modeling, the images in each data set were reshaped, normalized, and selected for modeling into the training, validation, and test sets. All 3 steps are accounted for using the `ImageDataGenerator` package from Keras.

3,000 images were used in the training set, the next 1,000 images were used in the validation set, and the remaining 1,000 images were used in the test set.

Modeling

The primary metric for assessing model performance was accuracy (classification of guns and not guns). However, we also considered the recall score (ratio of # of true positives of guns to the total image class of gun images) since the context of false negative images far outweigh the significance of accuracy for the purposes of this business problem.

The modeling process consisted of an iterative approach of attempting to build upon the previous best model. Training set images were fit into each respective model with validation performed to gauge performance on the testing data. A baseline model consisting of a simple dense neural network was instantiated as a benchmark.

Building off the baseline model, a number of different architectural modeling decisions were implemented and described more fully in this notebook. In general, we implemented different optimizers, introduced new layers, dropped layers, added max pooling layers, and dropout layers.

Evaluation

We determined that our best model, the CNN- V6 struck the best balance between accuracy and recall, scoring 88% and 96% respectively.

Modeling - Binary Classification (Gun vs. Not Gun)

In [56]:
```python
import joblib
import time
import matplotlib.pyplot as plt
import scipy
import numpy as np
from PIL import Image
from scipy import ndimage

import tensorflow as tf
from tensorflow import keras
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
from keras import models, layers
from tensorflow.keras.utils import array_to_img, load_img, img_to_array, to_categorical
from tensorflow.keras.callbacks import EarlyStopping
from keras.regularizers import l1, l2
from keras.layers import Dropout
from keras.applications import imagenet_utils

from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

import datetime
original_start = datetime.datetime.now()
start = datetime.datetime.now()

import os, shutil

import warnings
warnings.filterwarnings("ignore")

np.random.seed(42)
```

To start, we will load in the total images of guns and not guns into our directories.

In [57]:
```
# create directions for guns and not guns
data_gun_dir = 'image_data/gun'
data_not_gun_dir = 'image_data/not_gun/'

# new directory for the train test validation split
new_dir = 'split/'
```

In [58]:
```
# add gun images
imgs_gun = [file for file in os.listdir(data_gun_dir) if file.endswith('.jpg')]
```

In [59]:
```
# check first ten
imgs_gun[0:10]
```

Out[59]:
```
['120px-RugerMuzzelite.jpg',
 'armas (2311).jpg',
 '120px-NEF_B32.jpg',
 'armas (2741).jpg',
 '120px-NambuType14Pistol.jpg',
 'armas (695).jpg',
 'armas (1497).jpg',
 'armas (1182).jpg',
 'armas (380).jpg',
 'armas (1478).jpg']
```

In [60]:
```
len(imgs_gun)
```

Out[60]:
```
3967
```

In [61]:
```
# add not gun images
imgs_not_gun = [file for file in os.listdir(data_not_gun_dir) if file.endswith('.jpg')]
```

In [62]:
```
imgs_not_gun[0:10]
```

Out[62]:
```
['-478.jpg',
 'smartphone_0049_box1.jpg',
 'smartphone_0048_box1.jpg',
 '004_0101.jpg',
 '-322.jpg',
 'smartphone_0817_box1.jpg',
 'smartphone_0487_box1.jpg',
 '-444.jpg',
 '-450.jpg',
 'smartphone_0816_box1.jpg']
```

In [63]:
```
len(imgs_not_gun)
```

Out[63]:
```
2150
```

In [64]:
```
# below code creates new directory if not already created
# os.mkdir(new_dir)
```

In [65]:
```
# from the new directory, create new training, test, and val folders to store the guns and not guns
train_folder = os.path.join(new_dir, 'train')
train_gun = os.path.join(train_folder, 'gun')
train_not_gun = os.path.join(train_folder, 'not_gun')

test_folder = os.path.join(new_dir, 'test')
test_gun = os.path.join(test_folder, 'gun')
test_not_gun = os.path.join(test_folder, 'not_gun')

val_folder = os.path.join(new_dir, 'validation')
val_gun = os.path.join(val_folder, 'gun')
val_not_gun = os.path.join(val_folder, 'not_gun')
```

In [66]:
```
# os.mkdir(test_folder)
# os.mkdir(test_gun)
# os.mkdir(test_not_gun)

# os.mkdir(train_folder)
# os.mkdir(train_gun)
# os.mkdir(train_not_gun)

# os.mkdir(val_folder)
# os.mkdir(val_gun)
# os.mkdir(val_not_gun)
```

Now we will slice the images from `imgs_not_gun` into train, validation, and test folders and do the same for `imgs_gun` into train, validation, and test folders.

Not Gun Images

In [67]:
```
# train not_gun
imgs = imgs_not_gun[:1290]
for img in imgs:
    origin = os.path.join(data_not_gun_dir, img)
    destination = os.path.join(train_not_gun, img)
    shutil.copyfile(origin, destination)
# validation not_gun
imgs = imgs_not_gun[1290:1720]
for img in imgs:
    origin = os.path.join(data_not_gun_dir, img)
    destination = os.path.join(val_not_gun, img)
    shutil.copyfile(origin, destination)
# test not_gun
imgs = imgs_not_gun[1720:]
for img in imgs:
    origin = os.path.join(data_not_gun_dir, img)
    destination = os.path.join(test_not_gun, img)
    shutil.copyfile(origin, destination)
```

Gun Images

In [68]:
```
# train gun
imgs = imgs_gun[:2270]  # 2270 images
for img in imgs:
    origin = os.path.join(data_gun_dir, img)
    destination = os.path.join(train_gun, img)
    shutil.copyfile(origin, destination)

# validation gun
imgs = imgs_gun[2270:3027]  # 757 images
for img in imgs:
    origin = os.path.join(data_gun_dir, img)
    destination = os.path.join(val_gun, img)
    shutil.copyfile(origin, destination)

# test gun
imgs = imgs_gun[3027:]  # 757 images
for img in imgs:
    origin = os.path.join(data_gun_dir, img)
    destination = os.path.join(test_gun, img)
    shutil.copyfile(origin, destination)
```

Check how many images are in each set.

In [69]:
```
print('There are', len(os.listdir(train_gun)), 'gun images in the train set')
print('There are', len(os.listdir(val_gun)), 'gun images in the validation set')
print('There are', len(os.listdir(test_gun)), 'gun images in the test set')
```

```
There are 2270 gun images in the train set
There are 757 gun images in the validation set
There are 940 gun images in the test set
```

In [70]:
```
print('There are', len(os.listdir(train_not_gun)), 'not gun images in the train set')
print('There are', len(os.listdir(val_not_gun)), 'not gun images in the validation set')
print('There are', len(os.listdir(test_not_gun)), 'not gun images in the test set')
```

```
There are 1290 not gun images in the train set
There are 430 not gun images in the validation set
There are 430 not gun images in the test set
```

Preprocessing Images

To prepare the images for modeling, the following are the steps implemented to pass the images into each respective model.

- Normalize
- Set up image size
- Set up size of training, validation, and test sets

We can do this all with the `ImageDataGenerator`.

In [71]:
```
# get all the data in the directory split/train, and reshape them
# normalizees by rescaling
# set image size to 256 x 256
# batch size varies for each set size

train_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
                                                     train_folder,
                                                     target_size=(224, 224),
                                                     classes = ['not_gun', 'gun'],
                                                     batch_size=3000) # 3682 total from train

# get all the data in the directory split/validation, and reshape them
val_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
                                                   val_folder,
                                                   target_size=(224, 224),
                                                   classes = ['not_gun', 'gun'],
                                                   batch_size = 1000) # 1333 total from val

# get all the data in the directory split/test, and reshape them
test_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
                                                    test_folder,
                                                    target_size=(224, 224),
                                                    classes = ['not_gun', 'gun'],
                                                    batch_size = 1000) # 1390 total from test
```

```
Found 3560 images belonging to 2 classes.
Found 1187 images belonging to 2 classes.
Found 1370 images belonging to 2 classes.
```

In [72]:
```
# create the data sets and label the images as gun or not gun
train_images, train_labels = next(train_generator)
test_images, test_labels = next(test_generator)
val_images, val_labels = next(val_generator)
```

In [73]:
```
# check shape of images in train set
train_images.shape
```

Out[73]:
```
(3000, 224, 224, 3)
```

In [74]:
```
# check labels for train
train_labels
```

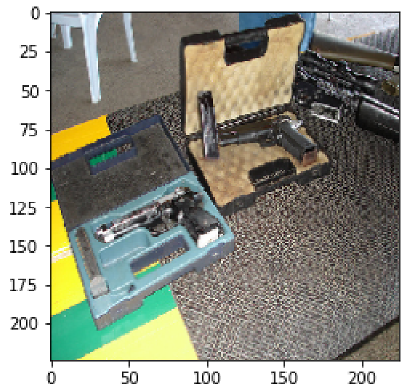
Out[74]:
```
array([[0., 1.],
       [1., 0.],
       [0., 1.],
       ...,
       [0., 1.],
       [1., 0.],
       [0., 1.]], dtype=float32)
```

In [75]:
```
train_generator.class_indices
```

Out[75]:
```
{'not_gun': 0, 'gun': 1}
```

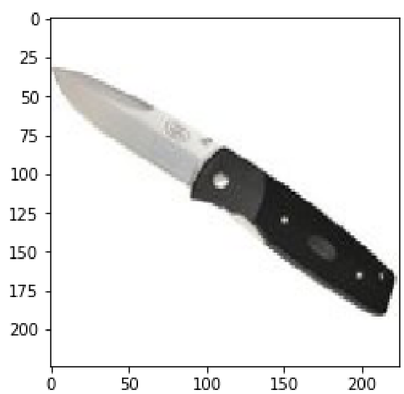
```
In [76]: # check an example gun image
sample_train_image = train_images[105]
sample_train_label = train_labels[105]
display(plt.imshow(sample_train_image))
print('Label: {}'.format(sample_train_label))
```

```
<matplotlib.image.AxesImage at 0x16803e770>

Label: [0. 1.]
```




```
In [77]: # check an example not gun image
sample_train_image = train_images[1]
sample_train_label = train_labels[1]
display(plt.imshow(sample_train_image))
print('Label: {}'.format(sample_train_label))
```

```
<matplotlib.image.AxesImage at 0x16807dea0>

Label: [1. 0.]
```



Visualize the Image Dataset

Lets take a closer look at the images in each set.

```
In [78]: # function that plots images and labels
def plots(ims, figsize = (20,4), rows = 1, interp = False, titles = None):
    """
    Takes in image set (recommend to slice for large sets); and image labels
    and plots a row of the images with associated labels.
    """
    if type(ims[0]) is np.ndarray:
        ims = np.array(ims).astype(np.uint8)
        if (ims.shape[-1] != 3):
            ims = ims.transpose((0,2,3,1))
    f = plt.figure(figsize=figsize)
    cols = len(ims)//rows if len(ims) % 2 == 0 else len(ims)//rows + 1
    for i in range(len(ims)):
        sp = f.add_subplot(rows, cols, i +1)
        sp.axis('Off')
        if titles is not None:
            sp.set_title(titles[i], fontsize = 16)
        plt.imshow(ims[i], interpolation = None if interp else 'none')
```

In [79]:
```python
# peek at 10 images in the train set
plots(train_images[0:10], titles = train_labels[0:10])
```

In [80]:
```python
# peek at 10 images in the test set
plots(test_images[500:510], titles = test_labels[500:510])
```

In [81]:
```python
# Explore dataset again
m_train = train_images.shape[0]  # number of images in train
num_px = train_images.shape[1]  # number of pixels
m_test = test_images.shape[0]  # number of images in test
m_val = val_images.shape[0]  # number of images in validation

print ("Number of training samples: " + str(m_train))
print ("Number of testing samples: " + str(m_test))
print ("Number of validation samples: " + str(m_val))
print('-'*40)
print ("train_images shape: " + str(train_images.shape))
print ("train_labels shape: " + str(train_labels.shape))
print('-'*40)
print ("test_images shape: " + str(test_images.shape))
print ("test_labels shape: " + str(test_labels.shape))
print('-'*40)
print ("val_images shape: " + str(val_images.shape))
print ("val_labels shape: " + str(val_labels.shape))
```

```
Number of training samples: 3000
Number of testing samples: 1000
Number of validation samples: 1000
----------------------------------------
train_images shape: (3000, 224, 224, 3)
train_labels shape: (3000, 2)
----------------------------------------
test_images shape: (1000, 224, 224, 3)
test_labels shape: (1000, 2)
----------------------------------------
val_images shape: (1000, 224, 224, 3)
val_labels shape: (1000, 2)
```

Reshaping the images for the model

In [82]:
```python
# reshapes the images to (num of images in set, num of pixels ie. 64 x 64 x 3 = 12288)
train_img = train_images.reshape(train_images.shape[0], -1)
test_img = test_images.reshape(test_images.shape[0], -1)
val_img = val_images.reshape(val_images.shape[0], -1)

print("Train", train_img.shape)
print("Validation", val_img.shape)
print("Test", test_img.shape)
```

```
Train (3000, 150528)
Validation (1000, 150528)
Test (1000, 150528)
```

Lets check the class balance for each image set:

In [83]:
```
# check train labels shape; currently as a binary tuple label
train_labels
```

Out[83]:
```
array([[0., 1.],
       [1., 0.],
       [0., 1.],
       ...,
       [0., 1.],
       [1., 0.],
       [0., 1.]], dtype=float32)
```

In [84]:
```
# get array of not gun vs. gun image labels
train_label_sum = sum(train_labels)
val_label_sum = sum(val_labels)
test_label_sum = sum(test_labels)

# get percentage of gun images in each set
train_gun_balance = round(train_label_sum[1] / len(train_labels),3)
val_gun_balance = round(val_label_sum[1] / len(val_labels),3)
test_gun_balance = round(test_label_sum[1] / len(test_labels),3)

print("Percentage of Gun Images in Train Set:", train_gun_balance)
print("Percentage of Gun Images in Validation Set:", val_gun_balance)
print("Percentage of Gun Images in Test Set:", test_gun_balance)
```

```
Percentage of Gun Images in Train Set: 0.635
Percentage of Gun Images in Validation Set: 0.65
Percentage of Gun Images in Test Set: 0.692
```

Finally, to model, we need to reshape the target variable so that it is in the correct shape to pass into the models.

In [85]:
```
# reshape the target, changes target values to binary (1 or 0)
train_y = np.reshape(train_labels[:,1], (3000,1))
test_y = np.reshape(test_labels[:,1], (1000,1))
val_y = np.reshape(val_labels[:,1], (1000,1))
```

In [86]:
```
# check test_y
test_y[0:15]
```

Out[86]:
```
array([[1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.]], dtype=float32)
```

In [87]:
```
# verify test_y labels are correct with test set
test_labels[0:15]
```

Out[87]:
```
array([[0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.]], dtype=float32)
```

In [88]:
```
# check change
print(train_y)
print(train_y.shape)
```

```
[[1.]
 [0.]
 [1.]
 ...
 [1.]
 [0.]
 [1.]]
(3000, 1)
```

Build Baseline Dense Network

Now that the images have been processed and are in the correct shape, we will now go through an iterative modeling process. To start, we'll develop a baseline dense neural network.

In [99]:
```
# Build a baseline fully connected model
np.random.seed(42)

baseline_model = models.Sequential()
baseline_model._name = "Baseline" # names the model

baseline_model.add(layers.Dense(64, activation='relu', input_shape=(150528,)))

# 2 hidden layers
baseline_model.add(layers.Dense(32, activation='relu'))
baseline_model.add(layers.Dense(16, activation='relu'))

baseline_model.add(layers.Dense(1, activation='sigmoid'))
```

In [100]:
```
baseline_model.summary()
```

```
Model: "Baseline"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_8 (Dense)             (None, 64)                9633856

 dense_9 (Dense)             (None, 32)                2080

 dense_10 (Dense)            (None, 16)                528

 dense_11 (Dense)            (None, 1)                 17

=================================================================
Total params: 9,636,481
Trainable params: 9,636,481
Non-trainable params: 0
_________________________________________________________________
```

In [101]:
```
# terminate training if doesnt improve on specified min_delta for 5 epochs
trainCallback = EarlyStopping(monitor='accuracy', min_delta = 1e-2, patience = 5)
```

⏰ **Below code runs for a few minutes**

In [102]:
```
baseline_model.compile(optimizer='adam',
              loss='binary_crossentropy', # for binary classification (gun or not gun)
              metrics=['accuracy'])

baseline_model = baseline_model.fit(train_img,
                                    train_y,
                                    epochs=50,
                                    batch_size=64,
                                    validation_data=(val_img, val_y),
                                    callbacks=[trainCallback])
```

```
- val_accuracy: 0.7150
Epoch 10/50
47/47 [==============================] - 4s 88ms/step - loss: 0.4606 - accuracy: 0.7860 - val_loss: 0.6999
- val_accuracy: 0.7050
Epoch 11/50
47/47 [==============================] - 4s 89ms/step - loss: 0.6603 - accuracy: 0.7053 - val_loss: 0.5245
- val_accuracy: 0.7320
Epoch 12/50
47/47 [==============================] - 4s 88ms/step - loss: 0.5623 - accuracy: 0.7327 - val_loss: 0.6644
- val_accuracy: 0.6620
Epoch 13/50
47/47 [==============================] - 4s 88ms/step - loss: 0.4306 - accuracy: 0.7923 - val_loss: 0.5774
- val_accuracy: 0.7260
Epoch 14/50
47/47 [==============================] - 4s 88ms/step - loss: 0.4645 - accuracy: 0.7797 - val_loss: 0.7305
- val_accuracy: 0.7190
Epoch 15/50
47/47 [==============================] - 4s 88ms/step - loss: 0.5961 - accuracy: 0.7287 - val_loss: 0.9279
- val_accuracy: 0.7040
```

.pkl the file

In [93]:
```
# # use the built-in open() function to open a file
# output_file = open("baseline_model.pkl", "wb") # "wb" means "write as bytes"

# # dump the variable's contents into the file
# joblib.dump(baseline_model, output_file)

# # close the file, ensuring nothing stays in the buffer
# output_file.close()
```

In [95]:
```
# use the built-in open() function again, this time to read
model_file = open("baseline_model.pkl", "rb") # "rb" means "read as bytes"
# load the variable's contents from the file into a variable
loaded_baseline_model = joblib.load(model_file)
# close the file
model_file.close()
```

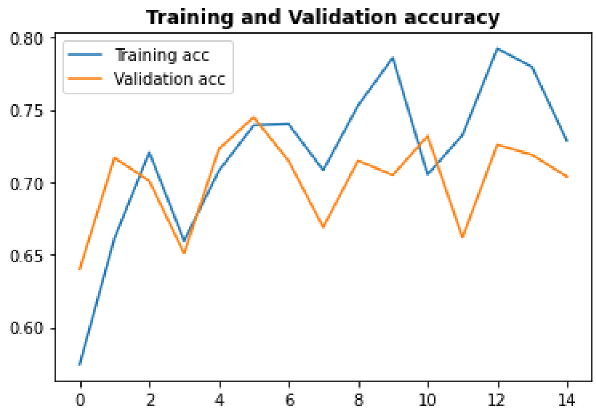
In [103]:

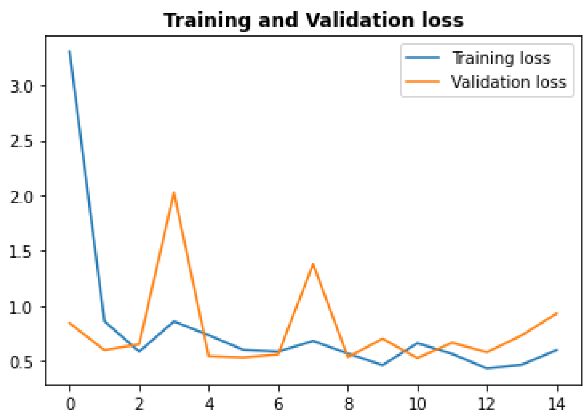
```python
# create a helper function that returns loss and accuracy results from model
# also plots the loss and accuracy

def model_results(mod, train_img, train_y, test_img, test_y):
    """ Takes in the model, image set, and array y of targets for training and test sets
        and returns the model's loss and accuracy scores.
        Also returns a plot of the training and validation scores.
    """
    # returns loss and accuracy scores for training and test sets
    results_train = mod.model.evaluate(train_img, train_y)
    results_test = mod.model.evaluate(test_img, test_y)

    # get the accuracy and loss for training and validation
    acc = mod.history['accuracy']
    val_acc = mod.history['val_accuracy']
    loss = mod.history['loss']
    val_loss = mod.history['val_loss']
    epochs = range(len(acc))

    # return train and test loss and accuracy
    print("Train Results Loss:", round(results_train[0],5))
    print("Train Results Accuracy:", round(results_train[1], 5))
    print("-"* 50)
    print("Test Results Loss:", round(results_test[0],5))
    print("Test Results Accuracy:", round(results_test[1], 5))

    # plot the Traininng and Validation Accuracy and Loss
    plt.plot(epochs, acc, label='Training acc')
    plt.plot(epochs, val_acc, label='Validation acc')
    plt.title('Training and Validation accuracy', fontweight = "bold")
    plt.legend()
    plt.figure()
    plt.plot(epochs, loss, label='Training loss')
    plt.plot(epochs, val_loss, label='Validation loss')
    plt.title('Training and Validation loss', fontweight = "bold")
    plt.legend()
    plt.show()
```

Baseline Model Results

In [104]:
```
# get baseline model results
model_results(loaded_baseline_model, train_img, train_y, test_img, test_y)
```

```
94/94 [==============================] - 5s 47ms/step - loss: 0.7325 - accuracy: 0.7347
32/32 [==============================] - 2s 46ms/step - loss: 0.8931 - accuracy: 0.7210
Train Results Loss: 0.73247
Train Results Accuracy: 0.73467
--------------------------------------------------
Test Results Loss: 0.89309
Test Results Accuracy: 0.721
```

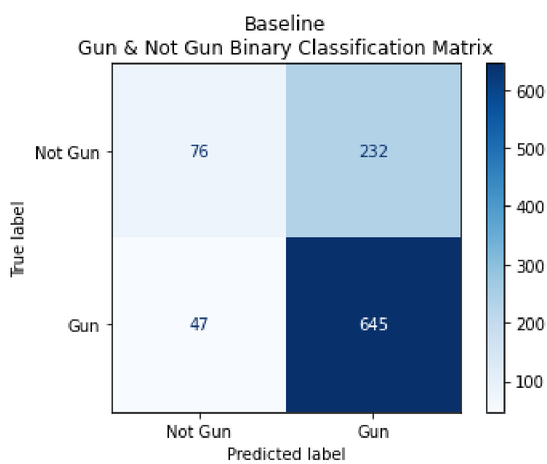




In [105]:
```python
# create helper function to plot test results as a confusion matrix
def get_test_results(mod, test_img, test_y):
    """
    Takes in the model, test image set, and test_y set
    and returns the model's accuracy and confusion matrix.
    """
    # return the loss and accuracy scores for the test set
    mod.model.evaluate(test_img, test_y)

    # get probabilites from the prediction on the test image set
    y_proba = mod.model.predict(test_img)

    # get assigned index values; ie. predicted labels
    predicted = y_proba.round()

    # plot confusion matrix on test set
    cm = confusion_matrix(test_y, predicted)

    disp = ConfusionMatrixDisplay(
        display_labels = ['Not Gun', 'Gun'],
        confusion_matrix=cm)

    disp.plot(cmap=plt.cm.Blues)

    model_name = mod.model.name

    # labels, title and ticks

    plt.title(model_name + "\nGun & Not Gun Binary Classification Matrix")
    plt.show()
```

In [106]:
```python
# get confusion matrix and test results for test image set
get_test_results(loaded_baseline_model, test_img, test_y)
```

```
32/32 [==============================] - 1s 45ms/step - loss: 0.8931 - accuracy: 0.7210
 2/32 [>.............................] - ETA: 1s

2022-08-05 16:30:10.566013: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

32/32 [==============================] - 2s 57ms/step
```



Overall, this model does a pretty good job with 72% accuracy when classifying between gun and not gun images. However, we see there are many false positive values.

Building a CNN

Convolutional neural networks should be a better model that can classify images. We'll now implement a baseline with several pooling and convolutional layers as a baseline.

Depending on the results, we will adjust the layers as appropriately following review of the results.

CNN Baseline V1

In [45]:
```
cnn_model = models.Sequential()
cnn_model._name = "CNN"

cnn_model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(224, 224, 3)))

cnn_model.add(layers.MaxPooling2D((2, 2)))

cnn_model.add(layers.Conv2D(32, (4, 4), activation='relu'))
cnn_model.add(layers.MaxPooling2D((2, 2)))

cnn_model.add(layers.Conv2D(64, (3, 3), activation='relu'))
cnn_model.add(layers.MaxPooling2D((2, 2)))

cnn_model.add(layers.Flatten())
cnn_model.add(layers.Dense(64, activation='relu'))
cnn_model.add(layers.Dense(1, activation='sigmoid'))

cnn_model.compile(loss='binary_crossentropy',
              optimizer="sgd",
              metrics=['accuracy'])
```

In [46]:
```
cnn_model.summary()
```

```
Model: "CNN"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 32)      896       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 111, 111, 32)     0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 108, 108, 32)      16416     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 54, 54, 32)       0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (None, 52, 52, 64)        18496     
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 26, 26, 64)       0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (None, 43264)             0         
                                                                 
 dense_4 (Dense)             (None, 64)                2768960   
                                                                 
 dense_5 (Dense)             (None, 1)                 65        
                                                                 
=================================================================
Total params: 2,804,833
Trainable params: 2,804,833
Non-trainable params: 0
_________________________________________________________________
```

⏰ **Below code runs for about 8 minutes**

In [47]:
```
# create a CNN model
cnn_model = cnn_model.fit(train_images,
                          train_y,
                          epochs=20,
                          batch_size=64,
                          validation_data=(val_images, val_y),
                          callbacks=[trainCallback])
```

```
19 - val_accuracy: 0.8300
Epoch 15/20
47/47 [==============================] - 17s 367ms/step - loss: 0.3594 - accuracy: 0.8447 - val_loss: 0.37
69 - val_accuracy: 0.8450
Epoch 16/20
47/47 [==============================] - 17s 369ms/step - loss: 0.3423 - accuracy: 0.8500 - val_loss: 0.40
88 - val_accuracy: 0.8000
Epoch 17/20
47/47 [==============================] - 17s 368ms/step - loss: 0.3329 - accuracy: 0.8523 - val_loss: 0.35
36 - val_accuracy: 0.8490
Epoch 18/20
47/47 [==============================] - 17s 369ms/step - loss: 0.3188 - accuracy: 0.8590 - val_loss: 0.53
05 - val_accuracy: 0.7230
Epoch 19/20
47/47 [==============================] - 17s 369ms/step - loss: 0.3199 - accuracy: 0.8553 - val_loss: 0.35
00 - val_accuracy: 0.8430
Epoch 20/20
47/47 [==============================] - 17s 368ms/step - loss: 0.3093 - accuracy: 0.8610 - val_loss: 0.37
41 - val_accuracy: 0.8220
```

.pkl the file

In [48]:
```
# # use the built-in open() function to open a file
# output_file = open("cnn_model.pkl", "wb") # "wb" means "write as bytes"

# # dump the variable's contents into the file
# joblib.dump(cnn_model, output_file)

# # close the file, ensuring nothing stays in the buffer
# output_file.close()
```

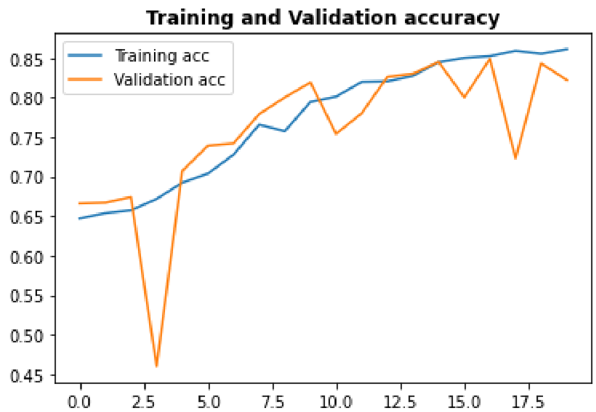
```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op,
_jit_compiled_convolution_op while saving (showing 3 of 3). These functions will not be directly callable
after loading.

INFO:tensorflow:Assets written to: ram://fb63b82b-31cd-4d68-9c8a-408a533f5ee4/assets

INFO:tensorflow:Assets written to: ram://fb63b82b-31cd-4d68-9c8a-408a533f5ee4/assets
```

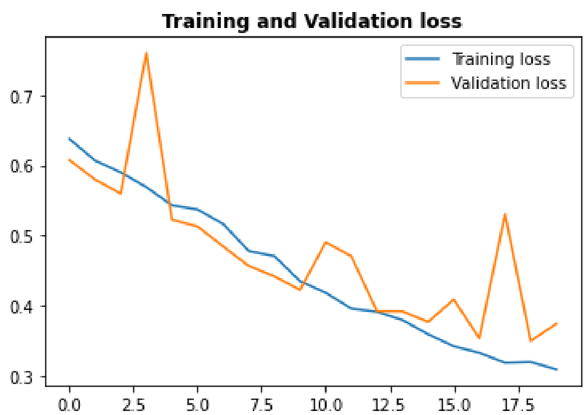
In [43]:
```
# use the built-in open() function again, this time to read
cnn_model_file = open("cnn_model.pkl", "rb") # "rb" means "read as bytes"
# load the variable's contents from the file into a variable
loaded_cnn_model = joblib.load(cnn_model_file)
# close the file
cnn_model_file.close()
```

CNN Baseline V1 Results

```
In [44]: # get model results
model_results(loaded_cnn_model, train_images, train_y, test_images, test_y)
```

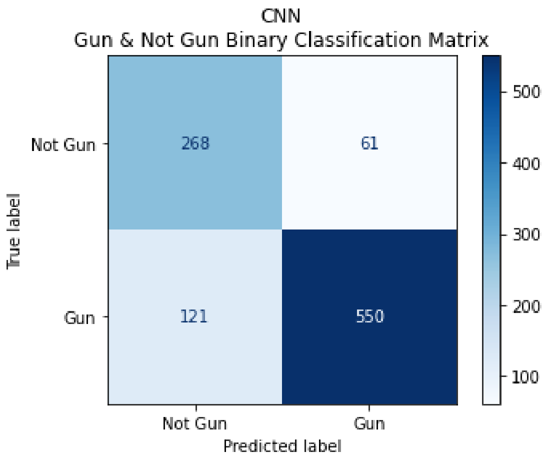
```
2022-08-03 10:21:42.576635: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

94/94 [==============================] - 5s 50ms/step - loss: 0.3130 - accuracy: 0.8650
32/32 [==============================] - 2s 69ms/step - loss: 0.3877 - accuracy: 0.8180
Train Results Loss: 0.31298
Train Results Accuracy: 0.865
--------------------------------------------------
Test Results Loss: 0.38771
Test Results Accuracy: 0.818
```





In [45]:
```
# get confusion matrix and test results for test image set
get_test_results(loaded_cnn_model, test_images, test_y)
```

```
32/32 [==============================] - 2s 69ms/step - loss: 0.3877 - accuracy: 0.8180
 2/32 [>.............................] - ETA: 3s

2022-08-03 10:21:53.261654: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

32/32 [==============================] - 2s 73ms/step
```



CNN Tuning V2

- Change to Adam Optimizer.

In [48]:
```
cnn_model_2 = models.Sequential()
cnn_model_2._name = "CNN2"

cnn_model_2.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(224, 224, 3)))

cnn_model_2.add(layers.MaxPooling2D((2, 2)))

cnn_model_2.add(layers.Conv2D(32, (4, 4), activation='relu'))
cnn_model_2.add(layers.MaxPooling2D((2, 2)))

cnn_model_2.add(layers.Conv2D(64, (3, 3), activation='relu'))
cnn_model_2.add(layers.MaxPooling2D((2, 2)))

cnn_model_2.add(layers.Flatten())
cnn_model_2.add(layers.Dense(64, activation='relu'))
cnn_model_2.add(layers.Dense(1, activation='sigmoid'))

cnn_model_2.compile(loss='binary_crossentropy',
              optimizer="adam",  # change to adam optimizer
              metrics=['accuracy'])
```

In [49]:
```
# tf.config.run_functions_eagerly(True)
```

⏰ **Below code runs for about 6 minutes**

In [50]:
```
# create a CNN model 2
cnn_model_2 = cnn_model_2.fit(train_images,
                              train_y,
                              epochs=20,
                              batch_size=64,
                              validation_data=(val_images, val_y),
                              callbacks=[trainCallback])
```

```
Epoch 1/20

2022-08-03 10:22:37.035808: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

47/47 [==============================] - ETA: 0s - loss: 0.6874 - accuracy: 0.6713

2022-08-03 10:22:55.810186: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

47/47 [==============================] - 21s 406ms/step - loss: 0.6874 - accuracy: 0.6713 - val_loss: 0.52
24 - val_accuracy: 0.7370
Epoch 2/20
47/47 [==============================] - 17s 372ms/step - loss: 0.4783 - accuracy: 0.7737 - val_loss: 0.47
64 - val_accuracy: 0.7600
Epoch 3/20
47/47 [==============================] - 17s 373ms/step - loss: 0.3829 - accuracy: 0.8287 - val_loss: 0.39
77 - val_accuracy: 0.8320
Epoch 4/20
47/47 [==============================] - 17s 370ms/step - loss: 0.2978 - accuracy: 0.8797 - val_loss: 0.35
```

.pkl the file

In [51]:
```
# # use the built-in open() function to open a file
# output_file = open("cnn_model_2.pkl", "wb") # "wb" means "write as bytes"

# # dump the variable's contents into the file
# joblib.dump(cnn_model_2, output_file)

# # close the file, ensuring nothing stays in the buffer
# output_file.close()
```

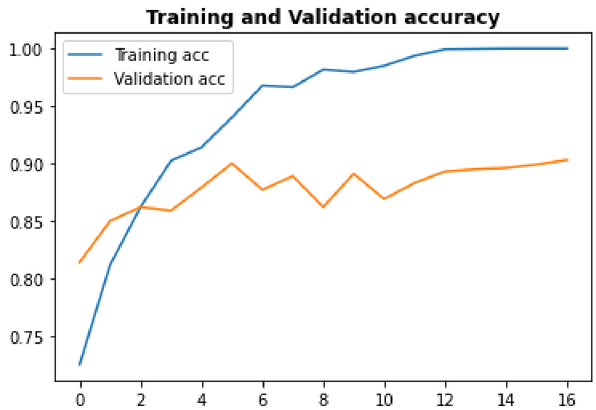
```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op,
_jit_compiled_convolution_op while saving (showing 3 of 3). These functions will not be directly callable
after loading.

INFO:tensorflow:Assets written to: ram://49dd6c0e-afa3-4491-9883-fc0ede85b7fe/assets

INFO:tensorflow:Assets written to: ram://49dd6c0e-afa3-4491-9883-fc0ede85b7fe/assets
```

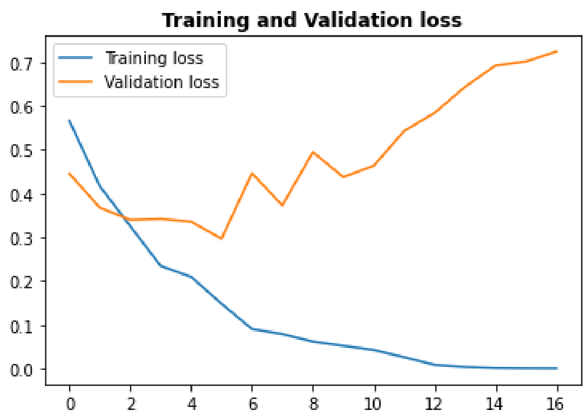
In [52]:
```
# use the built-in open() function again, this time to read
cnn_model_2_file = open("cnn_model_2.pkl", "rb") # "rb" means "read as bytes"
# load the variable's contents from the file into a variable
loaded_cnn_model_2 = joblib.load(cnn_model_2_file)
# close the file
cnn_model_2_file.close()
```

CNN Tuning V2 Results

In [157]:
```
# get model results
model_results(loaded_cnn_model_2, train_images, train_y, test_images, test_y)
```

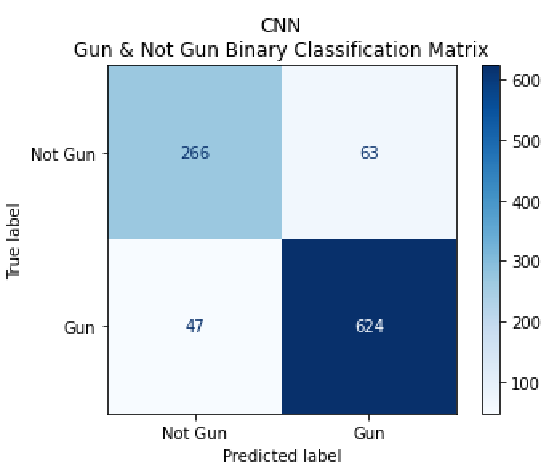
```
94/94 [==============================] - 7s 74ms/step - loss: 4.9062e-04 - accuracy: 1.0000
32/32 [==============================] - 2s 51ms/step - loss: 0.8087 - accuracy: 0.8900
Train Results Loss: 0.00049
Train Results Accuracy: 1.0
--------------------------------------------------
Test Results Loss: 0.8087
Test Results Accuracy: 0.89
```





In [158]:
```
# get confusion matrix and test results for test image set
get_test_results(loaded_cnn_model_2, test_images, test_y)
```

```
32/32 [==============================] - 2s 51ms/step - loss: 0.8087 - accuracy: 0.8900
32/32 [==============================] - 2s 72ms/step
```



There is a bit of overfitting after changing the model to an Adam optimizer. Lets reduce the overfitting with regularization.

CNN Tuning V3 with L1 (Lasso) Regularization

- Added kernal regularizer with L1 regularization at last layer.

In [49]:
```
# establish the regularization strength of lambda
reg_l1 = l1(3e-3) # 1e-5 to .1
```

In [51]:
```
cnn_model_3 = models.Sequential()
cnn_model_3._name = "CNN3RegL1"

cnn_model_3.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(224, 224, 3)))

cnn_model_3.add(layers.MaxPooling2D((2, 2)))

cnn_model_3.add(layers.Conv2D(32, (4, 4), activation='relu'))
cnn_model_3.add(layers.MaxPooling2D((2, 2)))

cnn_model_3.add(layers.Conv2D(64, (3, 3), activation='relu'))
cnn_model_3.add(layers.MaxPooling2D((2, 2)))

cnn_model_3.add(layers.Flatten())
cnn_model_3.add(layers.Dense(64,
                             activation='relu',
                             kernel_regularizer = reg_l1)) # added l1 regularization
cnn_model_3.add(layers.Dense(1, activation='sigmoid'))

cnn_model_3.compile(loss='binary_crossentropy',
              optimizer="adam",
              metrics=['accuracy'])
```

⏰ **Below code runs for about 10 minutes**

In [52]:
```
# create a CNN model 3
cnn_model_3 = cnn_model_3.fit(train_images,
                              train_y,
                              epochs=30, # changed to 30
                              batch_size=64,
                              validation_data=(val_images, val_y),
                              callbacks=[trainCallback])
```

```
Epoch 1/30

2022-08-03 23:10:05.061776: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

47/47 [==============================] - ETA: 0s - loss: 9.4700 - accuracy: 0.6373

2022-08-03 23:10:21.470258: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.
```

.pkl the file

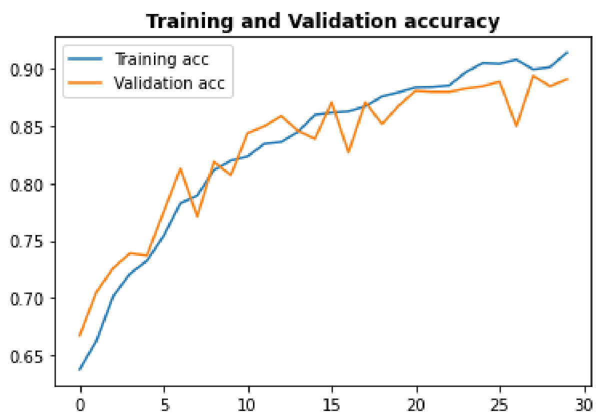
In [54]:
```
# # use the built-in open() function to open a file
# output_file = open("cnn_model_3.pkl", "wb") # "wb" means "write as bytes"

# # dump the variable's contents into the file
# joblib.dump(cnn_model_3, output_file)

# # close the file, ensuring nothing stays in the buffer
# output_file.close()
```

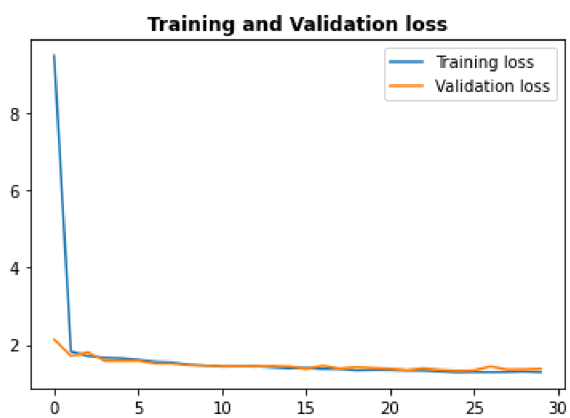
In [55]:
```
# use the built-in open() function again, this time to read
cnn_model_3_file = open("cnn_model_3.pkl", "rb") # "rb" means "read as bytes"
# load the variable's contents from the file into a variable
loaded_cnn_model_3 = joblib.load(cnn_model_3_file)
# close the file
cnn_model_3_file.close()
```

CNN Tuning V3 with L1 (Lasso) Regularization Results

In [56]:
```
# get model results
model_results(loaded_cnn_model_3, train_images, train_y, test_images, test_y)
```

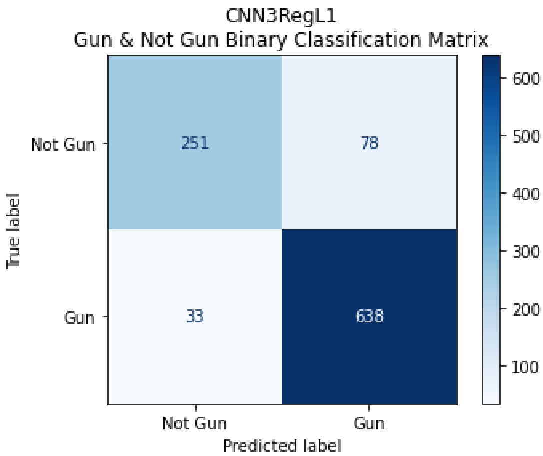
```
2022-08-03 23:19:20.971380: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

94/94 [==============================] - 7s 66ms/step - loss: 1.2748 - accuracy: 0.9260
32/32 [==============================] - 2s 71ms/step - loss: 1.3895 - accuracy: 0.8890
Train Results Loss: 1.27477
Train Results Accuracy: 0.926
--------------------------------------------------
Test Results Loss: 1.38947
Test Results Accuracy: 0.889
```





In [57]:
```
# get confusion matrix and test results for test image set
get_test_results(loaded_cnn_model_3, test_images, test_y)
```

```
32/32 [==============================] - 2s 72ms/step - loss: 1.3895 - accuracy: 0.8890
 2/32 [>.............................] - ETA: 1s

2022-08-03 23:19:41.184930: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

32/32 [==============================] - 2s 71ms/step
```



We fixed the issue of overfitting by simply adding a L1 regularization to the last layer of the previous CNN model.

Also, this is our **best** performing model so far, with a accuracy of 89% on the test set, and recall of 95%.

In this scenario, the recall score is defined as the total amount of correctly predicted 'gun' images (True Positives) over the entirety of True Positives and False Negatives (predicted not gun when actually a gun).

Lets now see what a L2 regularization does and attempt to improve from here.

CNN Tuning V4 with L2 (Ridge) Regularization

- try L2 regularization

In [109]:
```
# establish the regularization strength of lambda
reg_l2 = l2(3e-3) # 1e-5 to .1
```

In [81]:
```
cnn_model_4 = models.Sequential()
cnn_model_4._name = "CNN4RegL2"

cnn_model_4.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(224, 224, 3)))

cnn_model_4.add(layers.MaxPooling2D((2, 2)))

cnn_model_4.add(layers.Conv2D(32, (4, 4), activation='relu'))
cnn_model_4.add(layers.MaxPooling2D((2, 2)))

cnn_model_4.add(layers.Conv2D(64, (3, 3), activation='relu'))
cnn_model_4.add(layers.MaxPooling2D((2, 2)))

cnn_model_4.add(layers.Flatten())
cnn_model_4.add(layers.Dense(64,
                             activation='relu',
                             kernel_regularizer = reg_l2)) # added l2 regularization
cnn_model_4.add(layers.Dense(1, activation='sigmoid'))

cnn_model_4.compile(loss='binary_crossentropy',
                      optimizer="adam",
                      metrics=['accuracy'])
```

⏰ **Below code runs for about 7-8 minutes**

In [82]:
```
# create a CNN model 4
cnn_model_4 = cnn_model_4.fit(train_images,
                              train_y,
                              epochs=20,
                              batch_size=64,
                              validation_data=(val_images, val_y),
                              callbacks=[trainCallback])
```

```
Epoch 1/20

2022-08-03 11:07:40.668604: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

47/47 [==============================] - ETA: 0s - loss: 0.9030 - accuracy: 0.6350

2022-08-03 11:07:57.620826: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.
```

.pkl the file

In [83]:
```
# # use the built-in open() function to open a file
# output_file = open("cnn_model_4.pkl", "wb") # "wb" means "write as bytes"

# # dump the variable's contents into the file
# joblib.dump(cnn_model_4, output_file)

# # close the file, ensuring nothing stays in the buffer
# output_file.close()
```

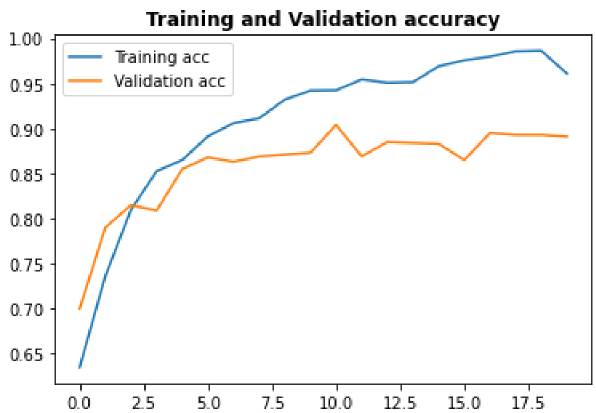
```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op,
_jit_compiled_convolution_op while saving (showing 3 of 3). These functions will not be directly callable
after loading.

INFO:tensorflow:Assets written to: ram://b1672b6a-d536-4556-86b8-2564debc7330/assets

INFO:tensorflow:Assets written to: ram://b1672b6a-d536-4556-86b8-2564debc7330/assets
```

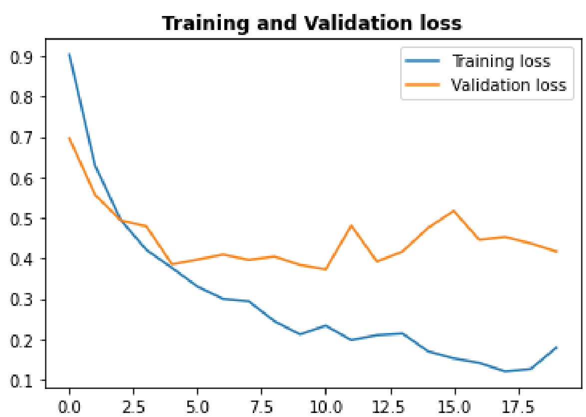
In [84]:
```
# use the built-in open() function again, this time to read
cnn_model_4_file = open("cnn_model_4.pkl", "rb") # "rb" means "read as bytes"
# load the variable's contents from the file into a variable
loaded_cnn_model_4 = joblib.load(cnn_model_4_file)
# close the file
cnn_model_4_file.close()
```

CNN Tuning V4 with L2 (Ridge) Regularization Results

In [85]:
```
# get model results
model_results(loaded_cnn_model_4, train_images, train_y, test_images, test_y)
```

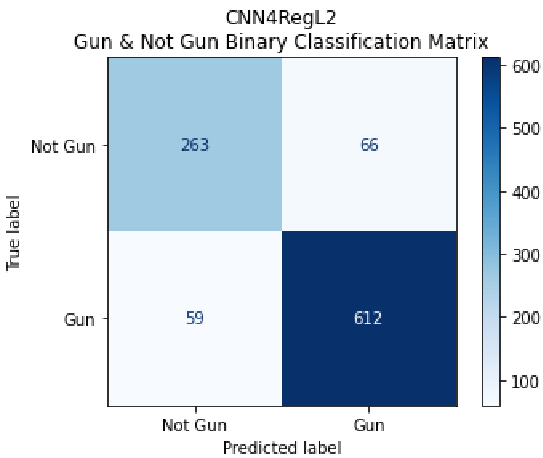
```
2022-08-03 11:14:54.383289: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

94/94 [==============================] - 7s 73ms/step - loss: 0.1659 - accuracy: 0.9780
32/32 [==============================] - 2s 74ms/step - loss: 0.4535 - accuracy: 0.8750
Train Results Loss: 0.16587
Train Results Accuracy: 0.978
--------------------------------------------------
Test Results Loss: 0.45352
Test Results Accuracy: 0.875
```





In [86]:
```
# get confusion matrix and test results for test image set
get_test_results(loaded_cnn_model_4, test_images, test_y)
```

```
32/32 [==============================] - 2s 71ms/step - loss: 0.4535 - accuracy: 0.8750
 2/32 [>.............................] - ETA: 3s

2022-08-03 11:15:12.308268: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

32/32 [==============================] - 2s 75ms/step
```



Changing the regularization layer from L1 (Lasso) to L2 (Ridge) actually does not resolve the overfitting issues. Applying an L1 regularization proves to be the most effective regularizer amongst the two options.

CNN Tuning V5 with Dropout & L1 Regularization

- Add a dropout layer of 20% at the start, keep L1 regularization

In [99]:
```
cnn_model_5 = models.Sequential()
cnn_model_5._name = "CNN5DropoutWithL1"

cnn_model_5.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)))
cnn_model_5.add(Dropout(0.2)) # dropout on previous activations (20% of the 20 nodes prev)

cnn_model_5.add(layers.MaxPooling2D((2, 2)))

cnn_model_5.add(layers.Conv2D(32, (4, 4), activation='relu'))
cnn_model_5.add(layers.MaxPooling2D((2, 2)))

cnn_model_5.add(layers.Conv2D(64, (3, 3), activation='relu'))
cnn_model_5.add(layers.MaxPooling2D((2, 2)))

cnn_model_5.add(layers.Flatten())
cnn_model_5.add(layers.Dense(64, activation='relu',
                             kernel_regularizer = reg_l1)) # l1 regularization
cnn_model_5.add(layers.Dense(1, activation='sigmoid'))

cnn_model_5.compile(loss='binary_crossentropy',
              optimizer="adam",
              metrics=['accuracy'])
```

⏰ **Below code runs for about 9 minutes**

In [100]:
```
# create a CNN model 5
cnn_model_5 = cnn_model_5.fit(train_images,
                              train_y,
                              epochs=20,
                              batch_size=64,
                              validation_data=(val_images, val_y),
                              callbacks=[trainCallback])
```

```
Epoch 1/20

2022-08-03 11:46:15.103440: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

47/47 [==============================] - ETA: 0s - loss: 9.1677 - accuracy: 0.6360

2022-08-03 11:46:38.245533: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

47/47 [==============================] - 26s 528ms/step - loss: 9.1677 - accuracy: 0.6360 - val_loss: 2.11
89 - val_accuracy: 0.6690
Epoch 2/20
47/47 [==============================] - 24s 506ms/step - loss: 1.8233 - accuracy: 0.6820 - val_loss: 1.76
91 - val_accuracy: 0.6980
Epoch 3/20
47/47 [==============================] - 24s 520ms/step - loss: 1.7628 - accuracy: 0.7050 - val_loss: 1.76
41 - val_accuracy: 0.7200
Epoch 4/20
47/47 [==============================] - 26s 536ms/step - loss: 1.6919 - accuracy: 0.7207 - val_loss: 1.69
```

.pkl the file

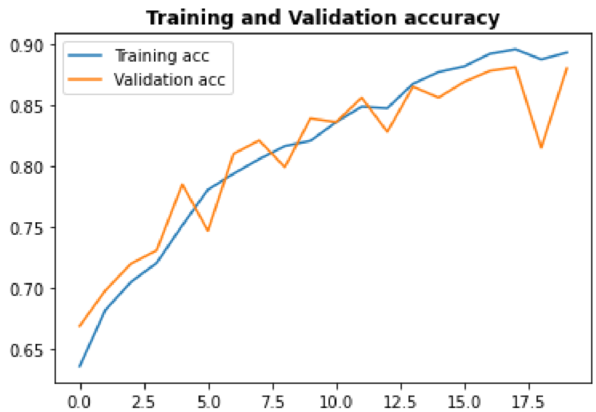
In [102]:
```
# # use the built-in open() function to open a file
# output_file = open("cnn_model_5.pkl", "wb") # "wb" means "write as bytes"

# # dump the variable's contents into the file
# joblib.dump(cnn_model_5, output_file)

# # close the file, ensuring nothing stays in the buffer
# output_file.close()
```

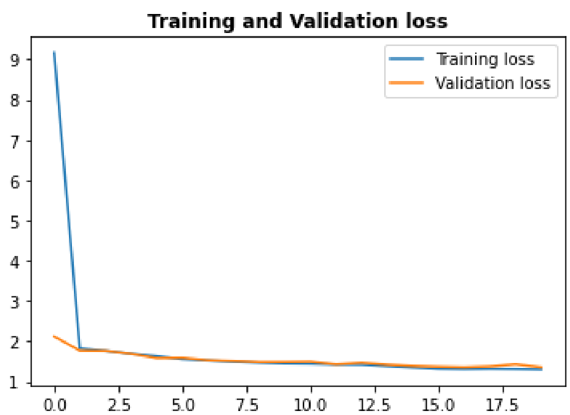
In [103]:
```
# use the built-in open() function again, this time to read
cnn_model_5_file = open("cnn_model_5.pkl", "rb") # "rb" means "read as bytes"
# load the variable's contents from the file into a variable
loaded_cnn_model_5 = joblib.load(cnn_model_5_file)
# close the file
cnn_model_5_file.close()
```

CNN Tuning V5 with Dropout & L1 Results

```
In [104]: # get model results
model_results(loaded_cnn_model_5, train_images, train_y, test_images, test_y)
```

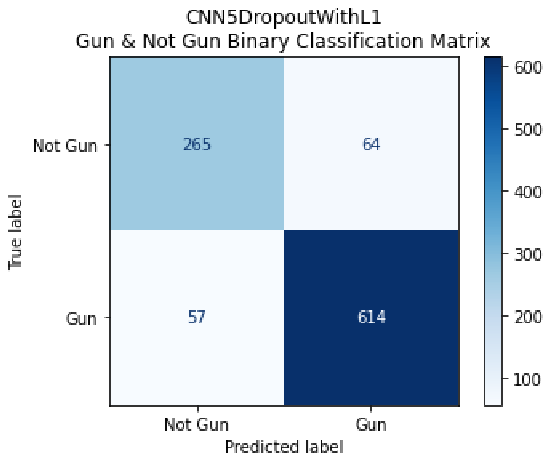
```
2022-08-03 11:58:50.789104: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

94/94 [==============================] - 8s 75ms/step - loss: 1.2803 - accuracy: 0.9197
32/32 [==============================] - 2s 75ms/step - loss: 1.3600 - accuracy: 0.8790
Train Results Loss: 1.28025
Train Results Accuracy: 0.91967
--------------------------------------------------
Test Results Loss: 1.35997
Test Results Accuracy: 0.879
```





In [106]:
```
# get confusion matrix and test results for test image set
get_test_results(loaded_cnn_model_5, test_images, test_y)
```

```
32/32 [==============================] - 2s 70ms/step - loss: 1.3600 - accuracy: 0.8790
32/32 [==============================] - 2s 73ms/step
```



While the accuracy score is better than our CNN V3 model with L1 regularization, the recall score is not as good.

CNN Tuning V6

- Mid layers changed to 64 filters, final layer has 128 filters
- Added another Conv2D layer
- Increased max pooling strides to 2

In [203]:
```
cnn_model_6 = models.Sequential()
cnn_model_6._name = "CNN6"

cnn_model_6.add(layers.Conv2D(32, (3, 3), activation='relu',
                                          input_shape=(224, 224, 3)))

cnn_model_6.add(layers.Conv2D(32, (3,3), activation="relu")) # added another 32 conv. layer

cnn_model_6.add(layers.MaxPooling2D((2, 2), strides=(2,2))) # stride of 2

cnn_model_6.add(layers.Conv2D(64, (4, 4), activation='relu'))  # changed to 64 filters
cnn_model_6.add(layers.Conv2D(64, (3, 3), activation='relu')) # changed to 64 filters

cnn_model_6.add(layers.MaxPooling2D((2, 2), strides=(2,2))) # stride of 2

cnn_model_6.add(layers.Flatten())
cnn_model_6.add(layers.Dense(128,   # changed to 128 filters
                             activation='relu',
                             kernel_regularizer = reg_l1)) # keep l1 regularization

cnn_model_6.add(layers.Dense(1, activation='sigmoid'))

cnn_model_6.compile(loss='binary_crossentropy',
                    optimizer="adam",
                    metrics=['accuracy'])
```

⏰ **Below code runs for about 15 minutes**

In [204]:
```
# create a CNN model 5
cnn_model_6 = cnn_model_6.fit(train_images,
                              train_y,
                              epochs=20,
                              batch_size=64,
                              validation_data=(val_images, val_y),
                              callbacks=[trainCallback])
```

```
Epoch 1/20

2022-08-03 15:48:44.537194: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

47/47 [==============================] - ETA: 0s - loss: 31.3434 - accuracy: 0.6280

2022-08-03 15:49:25.950788: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

47/47 [==============================] - 47s 968ms/step - loss: 31.3434 - accuracy: 0.6280 - val_loss: 10.
2192 - val_accuracy: 0.6670
Epoch 2/20
47/47 [==============================] - 44s 939ms/step - loss: 9.5306 - accuracy: 0.6720 - val_loss: 9.13
84 - val_accuracy: 0.6920
Epoch 3/20
47/47 [==============================] - 44s 940ms/step - loss: 9.0558 - accuracy: 0.6727 - val_loss: 8.79
86 - val_accuracy: 0.6780
Epoch 4/20
47/47 [==============================] - 45s 958ms/step - loss: 8.8902 - accuracy: 0.6863 - val_loss: 8.92
```

.pkl the file

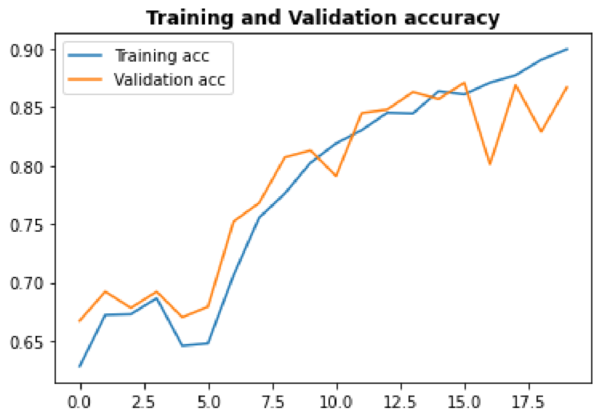
In [209]:
```
# # use the built-in open() function to open a file
# output_file = open("cnn_model_6.pkl", "wb") # "wb" means "write as bytes"

# # dump the variable's contents into the file
# joblib.dump(cnn_model_6, output_file)

# # close the file, ensuring nothing stays in the buffer
# output_file.close()
```

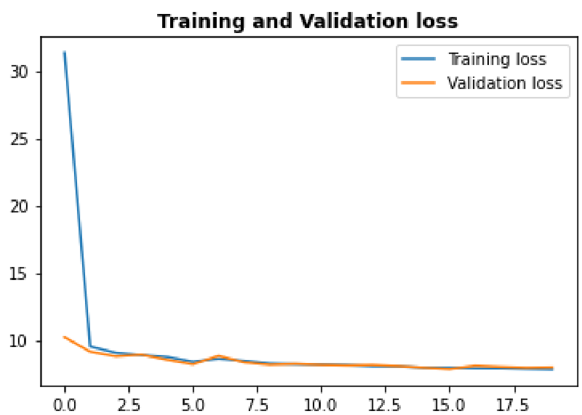
In [206]:
```
# use the built-in open() function again, this time to read
cnn_model_6_file = open("cnn_model_6.pkl", "rb") # "rb" means "read as bytes"
# load the variable's contents from the file into a variable
loaded_cnn_model_6 = joblib.load(cnn_model_6_file)
# close the file
cnn_model_6_file.close()
```

CNN Tuning V6 with Dropout & L1 Results

In [207]:
```
# get model results
model_results(loaded_cnn_model_6, train_images, train_y, test_images, test_y)
```

```
2022-08-03 16:04:02.216901: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

94/94 [==============================] - 18s 184ms/step - loss: 7.8361 - accuracy: 0.9110
32/32 [==============================] - 6s 188ms/step - loss: 7.9410 - accuracy: 0.8750
Train Results Loss: 7.83609
Train Results Accuracy: 0.911
--------------------------------------------------
Test Results Loss: 7.94096
Test Results Accuracy: 0.875
```



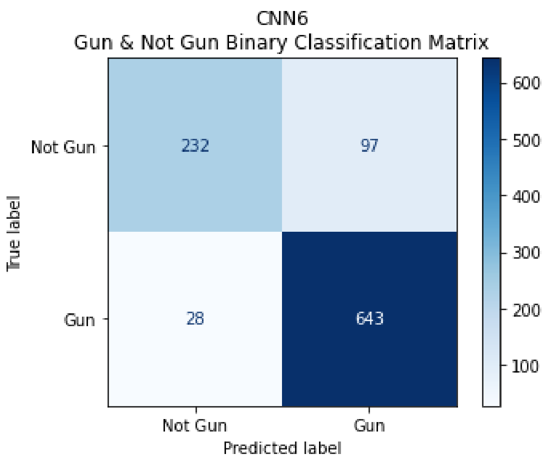


In [208]:
```
# get confusion matrix and test results for test image set
get_test_results(loaded_cnn_model_6, test_images, test_y)
```

```
32/32 [==============================] - 6s 181ms/step - loss: 7.9410 - accuracy: 0.8750

2022-08-03 16:04:38.069868: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

WARNING:tensorflow:6 out of the last 73 calls to <function Model.make_predict_function.<locals>.predict_fu
nction at 0x387206e60> triggered tf.function retracing. Tracing is expensive and the excessive number of t
racings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different
shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of
the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For
(3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing (https://www.tensorfl
ow.org/guide/function#controlling_retracing) and https://www.tensorflow.org/api_docs/python/tf/function (h
ttps://www.tensorflow.org/api_docs/python/tf/function) for  more details.
```


```
WARNING:tensorflow:6 out of the last 73 calls to <function Model.make_predict_function.<locals>.predict_fu
nction at 0x387206e60> triggered tf.function retracing. Tracing is expensive and the excessive number of t
racings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different
shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of
the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For
(3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing (https://www.tensorfl
ow.org/guide/function#controlling_retracing) and https://www.tensorflow.org/api_docs/python/tf/function (h
ttps://www.tensorflow.org/api_docs/python/tf/function) for  more details.
```


```
32/32 [==============================] - 6s 175ms/step
```



We get a better recall score of 96%, however, sacrifice a little bit in the accuracy compared to the previous CNN V5 model.

CNN Tuning V7

Lets use a simpler model and remove some layers to see how the model performs.

- Removed repeating convolutional layers

In [40]:

```
cnn_model_7 = models.Sequential()
cnn_model_7._name = "CNN7"

cnn_model_7.add(layers.Conv2D(32, (3, 3), activation='relu',
                                          input_shape=(224, 224, 3)))

## removed a layer here ##

cnn_model_7.add(layers.MaxPooling2D((2, 2), strides=(2,2))) # keep max pool stride

cnn_model_7.add(layers.Conv2D(64, (4, 4), activation='relu'))  # keep 64 filters

## removed a layer here ##

cnn_model_7.add(layers.MaxPooling2D((2, 2), strides=(2,2))) # keep max pool stride

cnn_model_7.add(layers.Flatten())
cnn_model_7.add(layers.Dense(128,    # keep to 128 filters
                             activation='relu',
                             kernel_regularizer = reg_l1)) # keep l1 regularization

cnn_model_7.add(layers.Dense(1, activation='sigmoid'))

cnn_model_7.compile(loss='binary_crossentropy',
                    optimizer="adam",
                    metrics=['accuracy'])
```

⏰ **Below code runs for about 15 minutes**

In [41]:

```
# create a CNN model 5
cnn_model_7 = cnn_model_7.fit(train_images,
                              train_y,
                              epochs=20,
                              batch_size=64,
                              validation_data=(val_images, val_y),
                              callbacks=[trainCallback])
```

```
47/47 [==============================] - 28s 598ms/step - loss: 9.5858 - accuracy: 0.6473 - val_loss: 9.45
22 - val_accuracy: 0.6670
Epoch 5/20
47/47 [==============================] - 27s 583ms/step - loss: 9.6522 - accuracy: 0.6473 - val_loss: 9.67
47 - val_accuracy: 0.6850
Epoch 6/20
47/47 [==============================] - 27s 582ms/step - loss: 9.5298 - accuracy: 0.6787 - val_loss: 9.29
10 - val_accuracy: 0.7120
Epoch 7/20
47/47 [==============================] - 27s 584ms/step - loss: 9.4764 - accuracy: 0.6997 - val_loss: 9.53
18 - val_accuracy: 0.7250
Epoch 8/20
47/47 [==============================] - 27s 587ms/step - loss: 9.4275 - accuracy: 0.7120 - val_loss: 9.47
57 - val_accuracy: 0.7080
Epoch 9/20
47/47 [==============================] - 27s 586ms/step - loss: 9.4018 - accuracy: 0.7340 - val_loss: 9.20
83 - val_accuracy: 0.7580
Epoch 10/20
47/47 [==============================] - 27s 585ms/step - loss: 9.3815 - accuracy: 0.7623 - val_loss: 9.42
36 - val_accuracy: 0.7760
```

.pkl the file

In [112]:

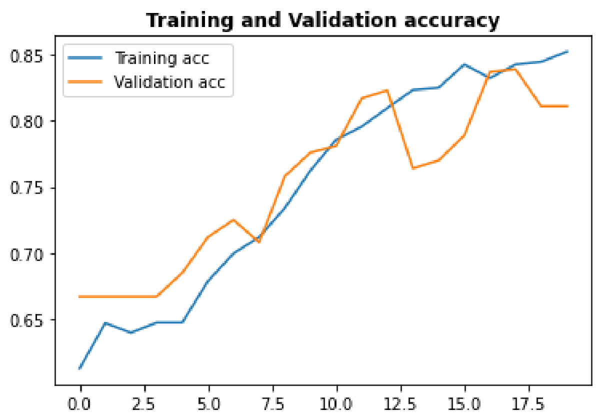
```
# # use the built-in open() function to open a file
# output_file = open("cnn_model_7.pkl", "wb") # "wb" means "write as bytes"

# # dump the variable's contents into the file
# joblib.dump(cnn_model_7, output_file)

# # close the file, ensuring nothing stays in the buffer
# output_file.close()
```

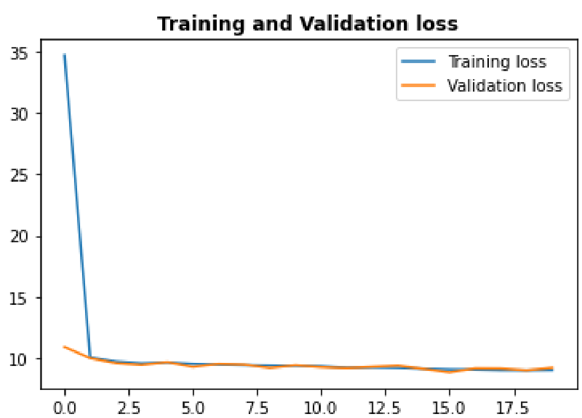
In [43]:
```
# use the built-in open() function again, this time to read
cnn_model_7_file = open("cnn_model_7.pkl", "rb") # "rb" means "read as bytes"
# load the variable's contents from the file into a variable
loaded_cnn_model_7 = joblib.load(cnn_model_7_file)
# close the file
cnn_model_7_file.close()
```

CNN Tuning V7 Results

In [47]:
```
# get model results
model_results(loaded_cnn_model_7, train_images, train_y, test_images, test_y)
```

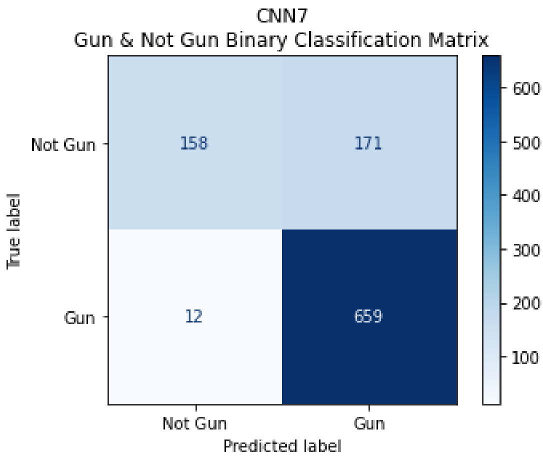
```
2022-08-03 23:03:20.329988: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

94/94 [==============================] - 11s 118ms/step - loss: 9.1449 - accuracy: 0.8320
32/32 [==============================] - 4s 132ms/step - loss: 9.1904 - accuracy: 0.8170
Train Results Loss: 9.14487
Train Results Accuracy: 0.832
--------------------------------------------------
Test Results Loss: 9.19037
Test Results Accuracy: 0.817
```





In [48]:
```
# get confusion matrix and test results for test image set
get_test_results(loaded_cnn_model_7, test_images, test_y)
```

```
32/32 [==============================] - 4s 132ms/step - loss: 9.1904 - accuracy: 0.8170
 1/32 [..............................] - ETA: 6s

2022-08-03 23:03:41.794065: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

32/32 [==============================] - 4s 129ms/step
```



So we drastically improved on the recall score, up to 98%!

However, as observed on the previous models, we lose even more on accuracy, down to 82%.

CNN Tuning V8

- Added dropouts, increased max pooling layer to 3x3

In [110]:
```
cnn_model_8 = models.Sequential()
cnn_model_8._name = "CNN8MaxPool3x3"

cnn_model_8.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(224, 224, 3)))
cnn_model_8.add(layers.MaxPooling2D((3, 3))) # changed to 3x3

cnn_model_8.add(layers.Conv2D(32, (4, 4), activation='relu'))
cnn_model_8.add(layers.MaxPooling2D((3, 3))) # changed to 3x3
cnn_model_8.add(layers.Dropout(0.3)) # added dropouts

cnn_model_8.add(layers.Conv2D(32, (4, 4), activation='relu'))
cnn_model_8.add(layers.MaxPooling2D((3, 3))) # changed to 3x3
cnn_model_8.add(layers.Dropout(0.3)) # added dropouts

cnn_model_8.add(layers.Conv2D(64, (3, 3), activation='relu', kernel_regularizer=reg_l2))
cnn_model_8.add(layers.MaxPooling2D((3, 3))) # changed to 3x3

cnn_model_8.add(layers.Flatten())
cnn_model_8.add(layers.Dense(64, activation='relu'))
cnn_model_8.add(layers.Dense(1, activation='sigmoid'))

cnn_model_8.compile(loss='binary_crossentropy',
              optimizer="adam",
              metrics=['accuracy'])
```

In [111]:
```python
cnn_model_8 = cnn_model_8.fit(train_images,
                              train_y,
                              epochs=20,
                              batch_size=64,
                              validation_data=(val_images, val_y),
                              callbacks=[trainCallback])
```

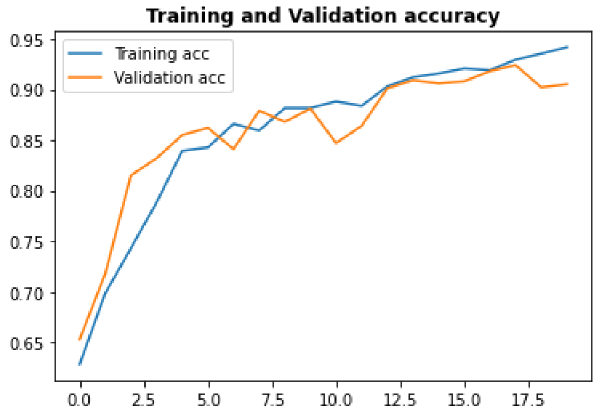
```
74 - val_accuracy: 0.9090
Epoch 15/20
47/47 [==============================] - 15s 326ms/step - loss: 0.2317 - accuracy: 0.9157 - val_loss: 0.27
95 - val_accuracy: 0.9060
Epoch 16/20
47/47 [==============================] - 15s 325ms/step - loss: 0.2307 - accuracy: 0.9207 - val_loss: 0.27
76 - val_accuracy: 0.9080
Epoch 17/20
47/47 [==============================] - 15s 325ms/step - loss: 0.2132 - accuracy: 0.9190 - val_loss: 0.25
34 - val_accuracy: 0.9180
Epoch 18/20
47/47 [==============================] - 15s 323ms/step - loss: 0.2074 - accuracy: 0.9293 - val_loss: 0.24
33 - val_accuracy: 0.9240
Epoch 19/20
47/47 [==============================] - 15s 324ms/step - loss: 0.1864 - accuracy: 0.9353 - val_loss: 0.28
43 - val_accuracy: 0.9020
Epoch 20/20
47/47 [==============================] - 15s 329ms/step - loss: 0.1857 - accuracy: 0.9417 - val_loss: 0.27
39 - val_accuracy: 0.9050
```

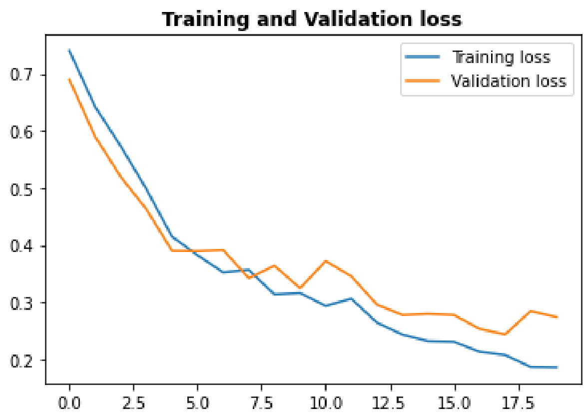
In [ ]:
```python
# # use the built-in open() function to open a file
# output_file = open("cnn_model_8.pkl", "wb") # "wb" means "write as bytes"

# # dump the variable's contents into the file
# joblib.dump(cnn_model_8, output_file)

# # close the file, ensuring nothing stays in the buffer
# output_file.close()
```

In [ ]:
```python
# # use the built-in open() function again, this time to read
# cnn_model_8_file = open("cnn_model_8.pkl", "rb") # "rb" means "read as bytes"
# # load the variable's contents from the file into a variable
# loaded_cnn_model_8 = joblib.load(cnn_model_8_file)
# # close the file
# cnn_model_8_file.close()
```

In [113]:
```
model_results(cnn_model_8, train_images, train_y, test_images, test_y)
```

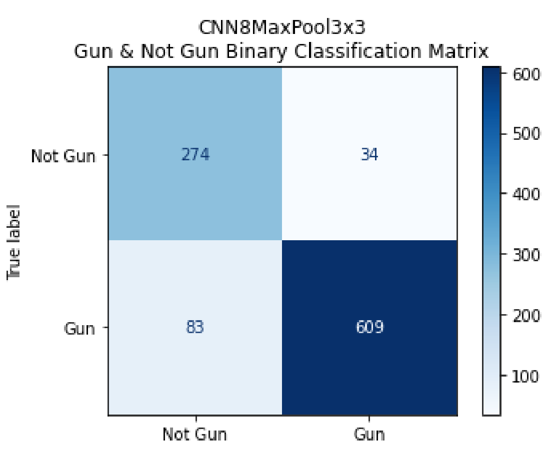
```
94/94 [==============================] - 6s 63ms/step - loss: 0.1916 - accuracy: 0.9443
32/32 [==============================] - 2s 61ms/step - loss: 0.2991 - accuracy: 0.8830
Train Results Loss: 0.19157
Train Results Accuracy: 0.94433
--------------------------------------------------
Test Results Loss: 0.2991
Test Results Accuracy: 0.883
```





In [115]:
```
get_test_results(cnn_model_8, test_images, test_y)
```

```
32/32 [==============================] - 2s 59ms/step - loss: 0.2991 - accuracy: 0.8830
 3/32 [=>............................] - ETA: 1s

2022-08-05 16:36:53.326416: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

32/32 [==============================] - 2s 55ms/step
```



At this point, we are trying to find a nice balance between accuracy and recall. However, the two measurements are inter-related and will have a trade-off in values. In other words, increasing the accuracy will decrease the recall and vice-versa. We'll stop the modeling process here and come to some conclusions.

Conclusions & Recommendations

Model Performance

Ultimately, we found that through the modeling process, even our best performing machine learning model performs exceptionally well when it comes to classification of gun and not gun images.

Specifically, we determined that our best model, the CNN- V6 had a 88% accuracy and a 96% recall. As discussed, there is a trade-off when it comes to improving upon accuracy and recall. As one metric increases, the other effectively decreases and a balance must be found between the two.

Ultimately, the model that strikes the best balance was the CNN-V6 model.

Model Value & Limitations

While we were able to effectively prove that machine learning CNN models can perform exceptionally well at distinguishing between images of guns and not guns, there are still difficulties in implementing this technology at a larger scale.

For example, there are data privacy and ethical concerns related to the usage of images on social media platforms such as Instagram or Facebook. Further it is important to note that even if an image is classified as a gun, there needs to be evidence to suggest that the social media post is inherently violent in nature. There is potential to explore whether a social media post is violent with other machine learning techniques such as using natural language processing (NLP).

Recommendations & Next Steps

- Secure partnerships with social media platforms such as Instagram and Facebook to garner implementation at a larger scale. Discuss the ethical situations surrounding data usage and privacy.
- Secure more data related to social media posts. Posts with guns alone are not necessarily violent in nature. Explore NLP for texts associated with images of guns flagging potentially dangerous individuals.
- Expand to the public sector (ie. security systems and cameras). Potential for privacy issues in public. However, recognition of guns in public can trigger faster response times to active shooter situations.

Predicting Images with MobileNet (Additional)

For fun, below is code for executing image classification with the MobileNet pretrained model. The MobileNet model is pre-trained with weights of images of multiple classes and can provide percentage class classification of the image.

In [244]:
```
from IPython.display import Image
mobile = keras.applications.mobilenet.MobileNet()
```

In [1]:
```
# specify location of images
data_gun_dir = 'image_data/gun/'
data_not_gun_dir = 'image_data/not_gun/'
```

In [246]:
```
# function that prepares images for mobile net classification
def prepare_image(file):
    img_path = 'image_data/not_gun/' # location of images, change to image_data/not_gun to classify not guns
    img = load_img(img_path + file, target_size = (224,224))
    img_array = img_to_array(img)
    img_array_expanded_dims = np.expand_dims(img_array, axis = 0)
    return keras.applications.mobilenet.preprocess_input(img_array_expanded_dims)
```

In [247]:
```
# select a random image from folder
image_name = str(np.random.choice(imgs_not_gun))  # change to imgs_not_gun to classify not guns

# define filename
filename = data_not_gun_dir + image_name
filename
```

Out[247]:
```
'image_data/not_gun/0-1.jpg'
```

In [248]:
```
# what image was selected?
Image(filename = filename, width = 300, height = 200)
```

Out[248]:

In [249]:
```
# process the image, return a prediction of the image
preprocessed_image = prepare_image(image_name)
predictions = mobile.predict(preprocessed_image)
results = tf.keras.applications.imagenet_utils.decode_predictions(predictions)
results
```

```
2022-08-03 17:14:37.860564: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

1/1 [==============================] - 0s 350ms/step
```

Out[249]:
```
[[('n03041632', 'cleaver', 0.6477464),
  ('n04141327', 'scabbard', 0.23451719),
  ('n04074963', 'remote_control', 0.042253114),
  ('n04086273', 'revolver', 0.015863894),
  ('n02951585', 'can_opener', 0.014446885)]]
```