Modeling - Multiclass

Note: See Binary Image Classification notebook for modeling with gun & not gun images.

In [2]:

```
import joblib
import splitfolders
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)
```

Establish Image Directories

In [3]:

```
# location of all image datasets to be split
input_folder = 'image_datasets/'
```

Split the images into train, test, and validation sets.

- 60% Train
- 20% Validation
- 20% Test

In [4]:

```
# takes in all the image datasets to be split
# puts split images into datasets_split with 60% in train, 20% val, 20% test
splitfolders.ratio(input_folder, output = 'datasets_split',
                   seed = 42, ratio = (0.6, 0.2, 0.2))
```

```
Copying files: 6641 files [00:03, 1915.58 files/s]
```

Define location paths of each split image set.

In [5]:

```
X_train = os.path.join('datasets_split/train/')
val_set = os.path.join('datasets_split/val/')
X_test = os.path.join('datasets_split/test/')
```

In [6]:

```
# check location
X_test
```

Out[6]:

```
'datasets_split/test/'
```

Preprocessing Images

- Normalize
- Set up labels for each class

In [7]:

```
train_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
                                                     X_train,
                                                     target_size=(224, 224),
                                                     batch_size=3500) # total in train


val_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
                                                     val_set,
                                                     target_size=(224, 224),
                                                     batch_size=1200) # total in val

test_generator = ImageDataGenerator(rescale=1./255).flow_from_directory(
                                                     X_test,
                                                     target_size=(224, 224),
                                                     batch_size=1200) # total in test
```

```
Found 3983 images belonging to 7 classes.
Found 1326 images belonging to 7 classes.
Found 1332 images belonging to 7 classes.
```

Create the labels:

In [8]:

```
# create the data sets and label the image classes
train_images, train_labels = next(train_generator)
val_images, val_labels = next(val_generator)
test_images, test_labels = next(test_generator)
```

In [9]:

```
# check shape of images in train set
train_images.shape
```

Out[9]:

```
(3500, 224, 224, 3)
```

In [10]:

```
# check labels for train
test_labels
```

Out[10]:

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

In [11]:

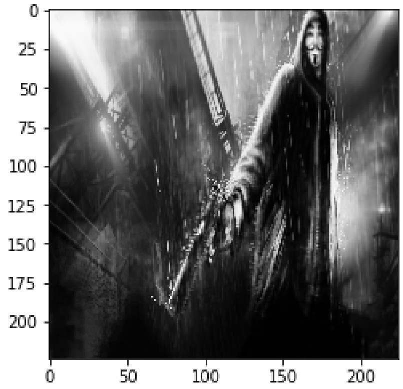
```
# check what each image is classified as
class_labels = train_generator.class_indices
class_labels
```

Out[11]:

```
{'baseball_bats': 0,
 'faces': 1,
 'guitar': 2,
 'gun': 3,
 'knives': 4,
 'people': 5,
 'phone': 6}
```

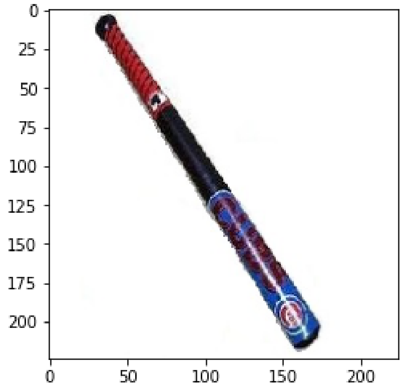
In [12]:
```
# check an example gun image
sample_train_image = train_images[5]
sample_train_label = train_labels[5]
display(plt.imshow(sample_train_image))
print('Label: {}'.format(sample_train_label))
```

```
<matplotlib.image.AxesImage at 0x1780e5ed0>

Label: [0. 0. 0. 1. 0. 0. 0.]
```



In [13]:
```
# check an example not gun image
sample_train_image = train_images[52]
sample_train_label = train_labels[52]
display(plt.imshow(sample_train_image))
print('Label: {}'.format(sample_train_label))
```

```
<matplotlib.image.AxesImage at 0x1781a7550>

Label: [1. 0. 0. 0. 0. 0. 0.]
```



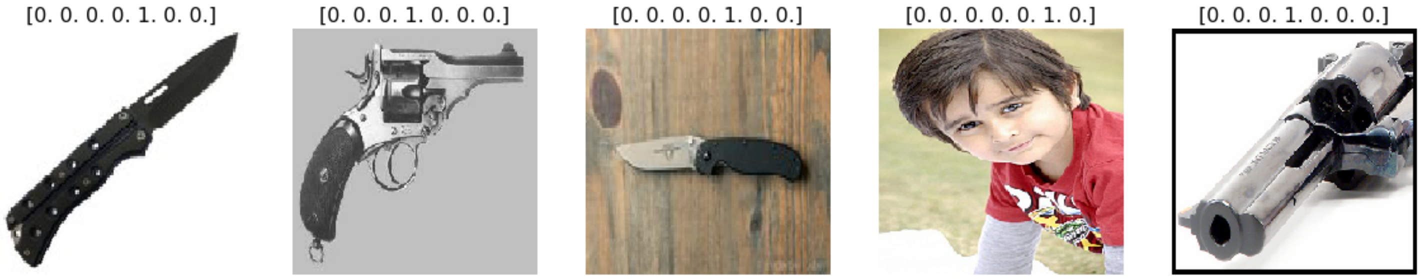
Visualize the Image Dataset

In [14]:

```
# 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')
```

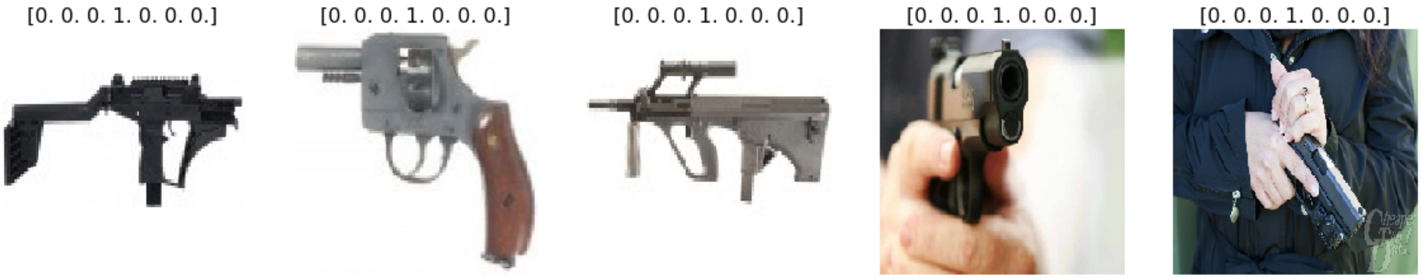
In [15]:

```
# peek at 5 images in the train set
plots(train_images[10:15], titles = train_labels[10:15])
```



In [16]:

```
# peek at 5 images in the test set
plots(test_images[10:15], titles = test_labels[10:15])
```



In [17]:

```
# 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: 3500
Number of testing samples: 1200
Number of validation samples: 1200
----------------------------------------
train_images shape: (3500, 224, 224, 3)
train_labels shape: (3500, 7)
----------------------------------------
test_images shape: (1200, 224, 224, 3)
test_labels shape: (1200, 7)
----------------------------------------
val_images shape: (1200, 224, 224, 3)
val_labels shape: (1200, 7)
```

Reshaping the images in each set based on number of pixels

In [18]:

```
# reshapes the images to (num of images in set, num of pixels ie. 224 x 224 x 3 = 150528)
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_img.shape)
print(test_img.shape)
print(val_img.shape)
```

```
(3500, 150528)
(1200, 150528)
(1200, 150528)
```

Refine the labels as the y train, validation, and test sets.

In [19]:

```
train_labels
```

Out[19]:

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

In [20]:

```
# # reshape the target
# y_train = np.reshape(train_labels[:,0], (3983,1))
# y_val = np.reshape(val_labels[:,0], (1326,1))
# y_test = np.reshape(test_labels[:,0], (1332,1))

y_train = train_labels
y_val = val_labels
y_test = test_labels
```

In [21]:

```
# check test_y
y_test[0:15]
```

Out[21]:

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

In [22]:

```
y_test.shape
```

Out[22]:

```
(1200, 7)
```

Build Baseline Dense Network

In [23]:

```
# Build a baseline fully connected model
np.random.seed(42)

baseline_model = models.Sequential()

baseline_model.add(layers.Dense(128, activation='relu', input_shape=(150528,)))

# 2 hidden layers
baseline_model.add(layers.Dense(64, activation='relu'))
baseline_model.add(layers.Dense(32, activation='relu'))
baseline_model.add(layers.Dense(7, activation='softmax')) # 7 different classes; ie. multiclass
```

```
Metal device set to: Apple M1

2022-08-04 11:55:03.042978: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:
305] Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel may not have been bui
lt with NUMA support.
2022-08-04 11:55:03.043609: I tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:
271] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0 MB memory) -> physical
PluggableDevice (device: 0, name: METAL, pci bus id: <undefined>)
```

In [25]:

```
baseline_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               19267712

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 32)                2080

 dense_3 (Dense)             (None, 7)                 231

=================================================================
Total params: 19,278,279
Trainable params: 19,278,279
Non-trainable params: 0
_________________________________________________________________
```

In [26]:

```
# terminate training if doesnt improve on specified min_delta for 5 epochs
trainCallback = EarlyStopping(monitor='val_loss', min_delta = 1e-5, patience = 5)
```

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

In [51]:

```
baseline_model.compile(optimizer='Adam',
              loss='categorical_crossentropy', # for multiclass classification
              metrics=['accuracy'])

baseline_model = baseline_model.fit(train_img,
                                    y_train,
                                    epochs=30,
                                    batch_size=64,
                                    validation_data=(val_img, y_val),
                                    callbacks=[trainCallback])
```

```
2022-08-03 23:34:48.878649: W tensorflow/core/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU f
requency: 0 Hz

Epoch 1/30

2022-08-03 23:34:49.658502: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

55/55 [==============================] - ETA: 0s - loss: 22.4937 - accuracy: 0.4349

2022-08-03 23:34:55.420853: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

55/55 [==============================] - 7s 122ms/step - loss: 22.4937 - accuracy: 0.4349 - val_loss: 9.03
52 - val_accuracy: 0.3017
Epoch 2/30
55/55 [==============================] - 6s 103ms/step - loss: 5.6566 - accuracy: 0.5103 - val_loss: 17.47
25 - val_accuracy: 0.6008
Epoch 3/30
55/55 [==============================] - 6s 102ms/step - loss: 6.3493 - accuracy: 0.5080 - val_loss: 1.424
2 - val_accuracy: 0.5817
```

Save the model as a .pkl file

In [53]:

```
# # use the built-in open() function to open a file
# output_file = open("baseline_model_multiclass.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()
```

```
INFO:tensorflow:Assets written to: ram://48b4e156-e1ff-4c1c-94c6-b4897e6ff69d/assets
```

In [54]:

```
# use the built-in open() function again, this time to read
model_file = open("baseline_model_multiclass.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 [28]:

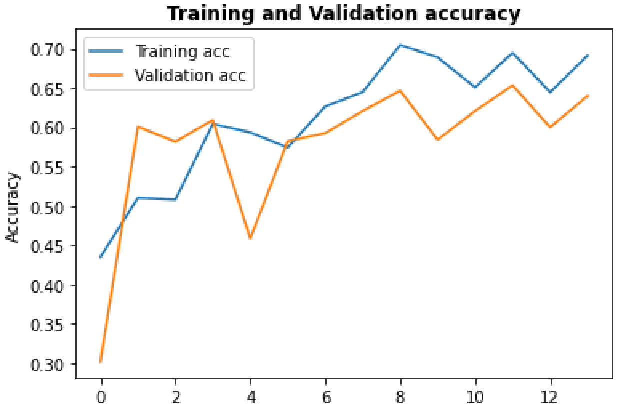
```
# create a helper function that returns loss and accuracy results from model
# also plots the loss and accuracy

def model_results(mod, train_img, y_train, test_img, y_test):
    """ 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, y_train)
    results_test = mod.model.evaluate(test_img, y_test)

    # 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.ylabel('Accuracy')
    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.ylabel('Categorical Crossentropy Loss')
    plt.legend()
    plt.show()
```

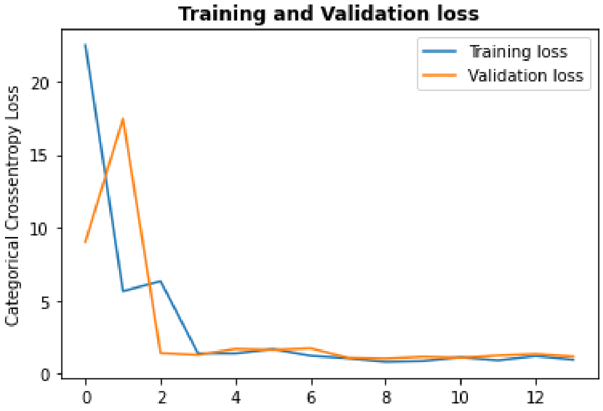
Baseline Model Results

In [56]:

```
# get baseline model results
model_results(loaded_baseline_model, train_img, y_train, test_img, y_test)
```

```
2022-08-03 23:37:21.591833: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

110/110 [==============================] - 6s 48ms/step - loss: 0.8263 - accuracy: 0.7337
38/38 [==============================] - 2s 50ms/step - loss: 1.2207 - accuracy: 0.6450
Train Results Loss: 0.82631
Train Results Accuracy: 0.73371
--------------------------------------------------
Test Results Loss: 1.22067
Test Results Accuracy: 0.645
```


Training and Validation accuracy


Training and Validation loss

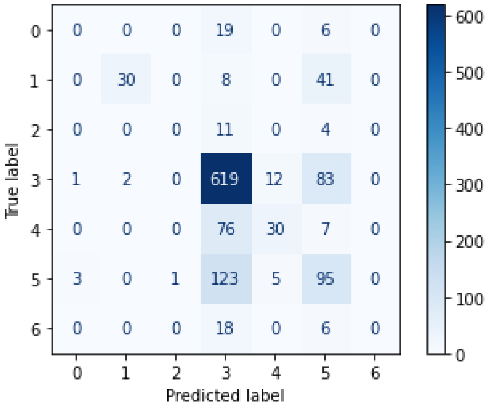
In [29]:

```
# 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 = np.argmax(y_proba, axis= -1)
    predicted

    # need to assign the class label to the actual y_test
    y_test_labeled = np.argmax(y_test, axis = -1)
    y_test_labeled

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

    disp = ConfusionMatrixDisplay(
        confusion_matrix=cm)

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

    # get class labels for reference
    print(class_labels)
```

In [58]:

```
# get confusion matrix and test results for test image set
get_test_results(loaded_baseline_model, test_img, y_test)
```

```
38/38 [==============================] - 2s 46ms/step - loss: 1.2207 - accuracy: 0.6450
 3/38 [=>............................] - ETA: 1s

2022-08-03 23:37:39.602741: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

38/38 [==============================] - 2s 53ms/step
```



```
{'baseball_bats': 0, 'faces': 1, 'guitar': 2, 'gun': 3, 'knives': 4, 'people': 5, 'phone': 6}
```

Applying CNN with L1 (Lasso) Regularization

(aka. CNN Tuning V3 From Binary Classification Notebook)

Modifications performed:

- input shape must be in (224, 224, 3) form (ie. use `train_images` and NOT `train_img`
- changed to 7 outputs for last layer
- changed loss to categorical crossentropy

In [30]:

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

In [84]:

```
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(7, activation='softmax'))   # for multiclass classification

cnn_model_3.compile(loss='categorical_crossentropy', # for multiclass classification
              optimizer="adam",
              metrics=['accuracy'])
```

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

In [85]:

```
# create a CNN model for multiclass
cnn_model_multiclass = cnn_model_3.fit(train_images,
                                        y_train,
                                        epochs=30,
                                        batch_size=64,
                                        validation_data=(val_images, y_val),
                                        callbacks=[trainCallback])
```

```
Epoch 1/30

2022-08-03 23:49:30.542992: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

55/55 [==============================] - ETA: 0s - loss: 9.4841 - accuracy: 0.5834

2022-08-03 23:49:50.621448: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.
```

.pkl the file

In [87]:

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

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

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

In [88]:

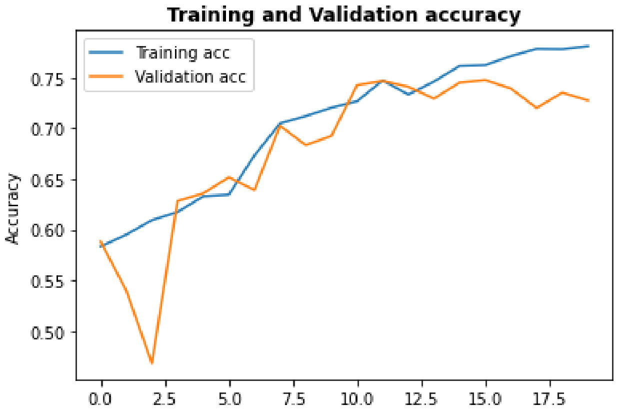
```
# use the built-in open() function again, this time to read
cnn_model_multiclass_file = open("cnn_model_multiclass.pkl", "rb") # "rb" means "read as bytes"
# load the variable's contents from the file into a variable
loaded_cnn_model_multiclass = joblib.load(cnn_model_multiclass_file)
# close the file
cnn_model_multiclass_file.close()
```

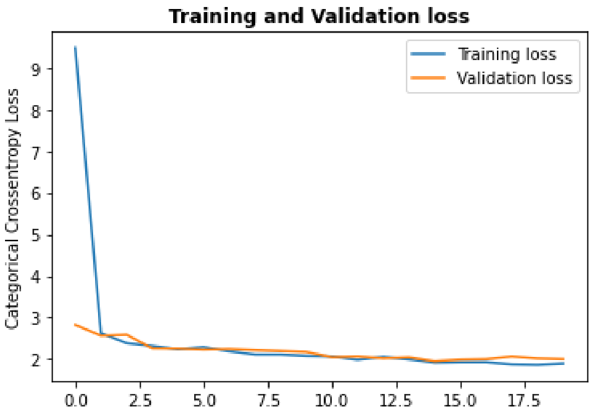
CNN Tuning V3 with L1 (Lasso) Regularization Results

In [90]:

```
# get model results
model_results(loaded_cnn_model_multiclass, train_images, y_train, test_images, y_test)
```

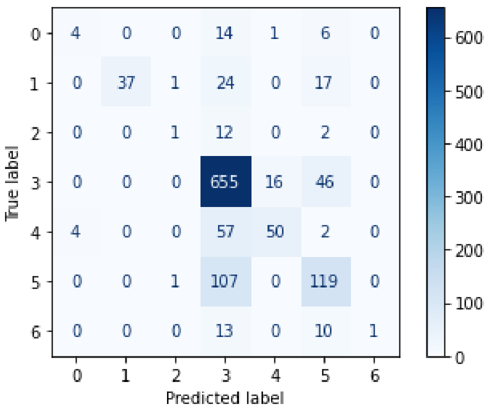
```
2022-08-03 23:58:25.914326: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

110/110 [==============================] - 9s 72ms/step - loss: 1.8555 - accuracy: 0.7871
38/38 [==============================] - 3s 72ms/step - loss: 2.0575 - accuracy: 0.7225
Train Results Loss: 1.85552
Train Results Accuracy: 0.78714
--------------------------------------------------
Test Results Loss: 2.05753
Test Results Accuracy: 0.7225
```





```
In [92]: # get confusion matrix and test results for test image set
         get_test_results(loaded_cnn_model_multiclass, test_images, y_test)
```

```
38/38 [==============================] - 3s 71ms/step - loss: 2.0575 - accuracy: 0.7225
 1/38 [..............................] - ETA: 4s

2022-08-03 23:58:50.874676: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

38/38 [==============================] - 3s 76ms/step
```



True label \ Predicted label	0	1	2	3	4	5	6
0	4	0	0	14	1	6	0
1	0	37	1	24	0	17	0
2	0	0	1	12	0	2	0
3	0	0	0	655	16	46	0
4	4	0	0	57	50	2	0
5	0	0	1	107	0	119	0
6	0	0	0	13	0	10	1

```
{'baseball_bats': 0, 'faces': 1, 'guitar': 2, 'gun': 3, 'knives': 4, 'people': 5, 'phone': 6}
```

The testing score is up to 73% which is an improvement! Lets build off of this score moving forward.

CNN V4

- Added additional convolutional layer
- Added dropout

```
In [31]: cnn_model_4 = models.Sequential()
         cnn_model_4._name = "CNN4"

         cnn_model_4.add(layers.Conv2D(32, (3, 3), activation='relu',input_shape=(224, 224, 3)))
         cnn_model_4.add(Dropout(0.2)) # dropout on previous activations (20% of the 20 nodes prev)

         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.Conv2D(64, (3, 3), activation='relu')) # added another 64 layer
         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_l1)) # added l1 regularization
         cnn_model_4.add(layers.Dense(7, activation='softmax'))   # for multiclass classification

         cnn_model_4.compile(loss='categorical_crossentropy', # for multiclass classification
                             optimizer="adam",
                             metrics=['accuracy'])
```

In [32]:

```
# create a CNN model for multiclass
cnn_model_multiclass_2 = cnn_model_4.fit(train_images,
                                         y_train,
                                         epochs=30,
                                         batch_size=64,
                                         validation_data=(val_images, y_val),
                                         callbacks=[trainCallback])
```

```
2022-08-04 12:02:35.380263: W tensorflow/core/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU f
requency: 0 Hz

Epoch 1/30

2022-08-04 12:02:36.045370: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

55/55 [==============================] - ETA: 0s - loss: 6.5986 - accuracy: 0.5814

2022-08-04 12:03:01.444900: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

55/55 [==============================] - 28s 492ms/step - loss: 6.5986 - accuracy: 0.5814 - val_loss: 1.95
07 - val_accuracy: 0.6233
Epoch 2/30
55/55 [==============================] - 27s 487ms/step - loss: 1.5748 - accuracy: 0.6049 - val_loss: 1.59
67 - val_accuracy: 0.6208
Epoch 3/30
55/55 [==============================] - 27s 487ms/step - loss: 1.4203 - accuracy: 0.6149 - val_loss: 1.36
39 - val_accuracy: 0.6617
```

.pkl the file

In [35]:

```
# # use the built-in open() function to open a file
# output_file = open("cnn_model_multiclass_2.pkl", "wb") # "wb" means "write as bytes"

# # dump the variable's contents into the file
# joblib.dump(cnn_model_multiclass_2, output_file)

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

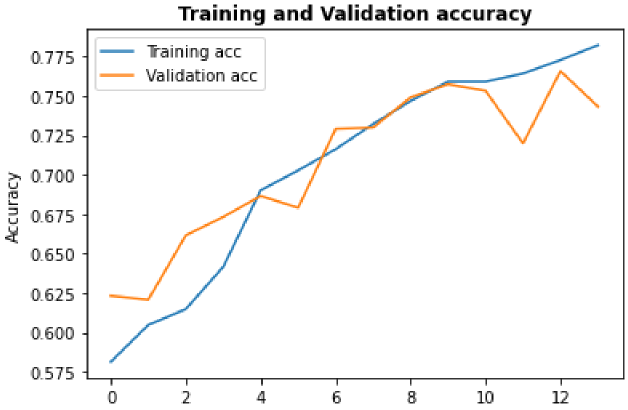
In [36]:

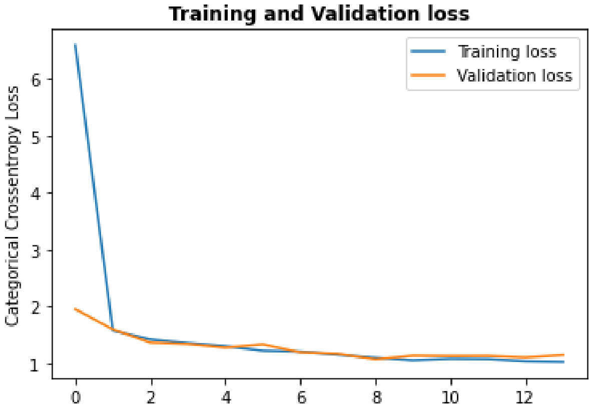
```
# use the built-in open() function again, this time to read
cnn_model_multiclass_2_file = open("cnn_model_multiclass_2.pkl", "rb") # "rb" means "read as bytes"
# load the variable's contents from the file into a variable
loaded_cnn_model_multiclass_2 = joblib.load(cnn_model_multiclass_2_file)
# close the file
cnn_model_multiclass_2_file.close()
```

CNN Tuning V4 Results

```
In [37]: # get model results
         model_results(loaded_cnn_model_multiclass_2, train_images, y_train, test_images, y_test)
```

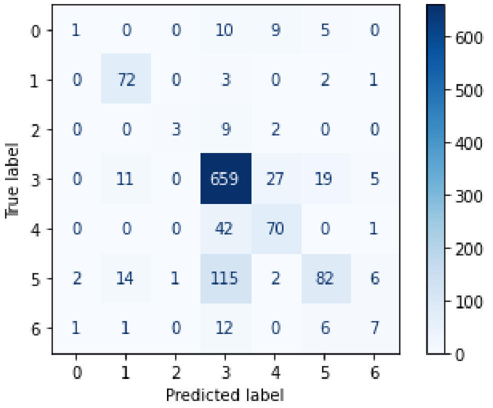
```
2022-08-04 12:11:03.678045: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

110/110 [==============================] - 6s 47ms/step - loss: 1.0299 - accuracy: 0.8023
38/38 [==============================] - 3s 69ms/step - loss: 1.1482 - accuracy: 0.7450
Train Results Loss: 1.02987
Train Results Accuracy: 0.80229
--------------------------------------------------
Test Results Loss: 1.14816
Test Results Accuracy: 0.745
```





```
In [39]: # get confusion matrix and test results for test image set
         get_test_results(loaded_cnn_model_multiclass_2, test_images, y_test)
```

```
38/38 [==============================] - 2s 45ms/step - loss: 1.1482 - accuracy: 0.7450
 3/38 [=>............................] - ETA: 1s

2022-08-04 12:11:23.092254: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

38/38 [==============================] - 2s 47ms/step
```



```
{'baseball_bats': 0, 'faces': 1, 'guitar': 2, 'gun': 3, 'knives': 4, 'people': 5, 'phone': 6}
```

Slightly better accuracy on the testing score; up to 75%.

CNN V5

- Changed pooling layers size to 3 x 3

```
In [40]: cnn_model_5 = models.Sequential()
         cnn_model_5._name = "CNN5"

         cnn_model_5.add(layers.Conv2D(32, (3, 3), activation='relu',input_shape=(224, 224, 3)))
         cnn_model_5.add(Dropout(0.2))
         cnn_model_5.add(layers.MaxPooling2D((3, 3))) # changed to 3x3 max pooling

         cnn_model_5.add(layers.Conv2D(32, (4, 4), activation='relu'))
         cnn_model_5.add(layers.MaxPooling2D((3, 3)))

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

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

         cnn_model_5.add(layers.Flatten())
         cnn_model_5.add(layers.Dense(64,
                                      activation='relu',
                                      kernel_regularizer = reg_l1)) # added l1 regularization
         cnn_model_5.add(layers.Dense(7, activation='softmax'))   # for multiclass classification

         cnn_model_5.compile(loss='categorical_crossentropy', # for multiclass classification
                              optimizer="adam",
                              metrics=['accuracy'])
```

In [41]:

```
# create a CNN model for multiclass
cnn_model_multiclass_3 = cnn_model_5.fit(train_images,
                                         y_train,
                                         epochs=30,
                                         batch_size=64,
                                         validation_data=(val_images, y_val),
                                         callbacks=[trainCallback])
```

```
Epoch 1/30

2022-08-04 13:21:25.392658: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

55/55 [==============================] - ETA: 0s - loss: 2.4041 - accuracy: 0.5929

2022-08-04 13:21:50.638756: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

55/55 [==============================] - 28s 481ms/step - loss: 2.4041 - accuracy: 0.5929 - val_loss: 2.09
39 - val_accuracy: 0.6067
Epoch 2/30
55/55 [==============================] - 25s 463ms/step - loss: 1.8822 - accuracy: 0.6060 - val_loss: 1.61
67 - val_accuracy: 0.6358
Epoch 3/30
55/55 [==============================] - 25s 461ms/step - loss: 1.4548 - accuracy: 0.6486 - val_loss: 1.30
50 - val_accuracy: 0.6575
Epoch 4/30
55/55 [==============================] - 25s 462ms/step - loss: 1.2302 - accuracy: 0.6649 - val_loss: 1.16
```

.pkl the file

In [43]:

```
# # use the built-in open() function to open a file
# output_file = open("cnn_model_multiclass_3.pkl", "wb") # "wb" means "write as bytes"

# # dump the variable's contents into the file
# joblib.dump(cnn_model_multiclass_3, output_file)

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

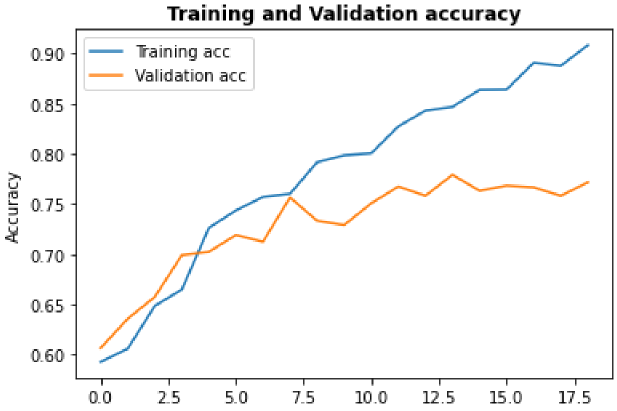
In [44]:

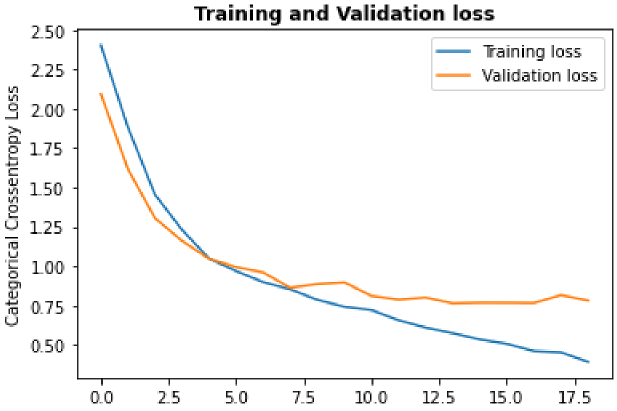
```
# use the built-in open() function again, this time to read
cnn_model_multiclass_3_file = open("cnn_model_multiclass_3.pkl", "rb") # "rb" means "read as bytes"
# load the variable's contents from the file into a variable
loaded_cnn_model_multiclass_3 = joblib.load(cnn_model_multiclass_3_file)
# close the file
cnn_model_multiclass_3_file.close()
```

CNN Tuning V4 Results

```
In [45]: # get model results
         model_results(loaded_cnn_model_multiclass_3, train_images, y_train, test_images, y_test)
```

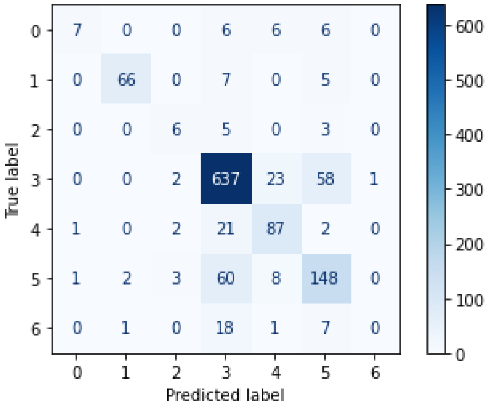
```
2022-08-04 13:30:21.104932: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

110/110 [==============================] - 5s 37ms/step - loss: 0.3806 - accuracy: 0.9240
38/38 [==============================] - 2s 60ms/step - loss: 0.7569 - accuracy: 0.7925
Train Results Loss: 0.38057
Train Results Accuracy: 0.924
--------------------------------------------------
Test Results Loss: 0.75688
Test Results Accuracy: 0.7925
```





```
In [46]: # get confusion matrix and test results for test image set
         get_test_results(loaded_cnn_model_multiclass_3, test_images, y_test)
```

```
38/38 [==============================] - 2s 58ms/step - loss: 0.7569 - accuracy: 0.7925
 2/38 [>.............................] - ETA: 3s

2022-08-04 13:30:31.971341: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

38/38 [==============================] - 2s 59ms/step
```



	0	1	2	3	4	5	6
0	7	0	0	6	6	6	0
1	0	66	0	7	0	5	0
2	0	0	6	5	0	3	0
3	0	0	2	637	23	58	1
4	1	0	2	21	87	2	0
5	1	2	3	60	8	148	0
6	0	1	0	18	1	7	0

```
{'baseball_bats': 0, 'faces': 1, 'guitar': 2, 'gun': 3, 'knives': 4, 'people': 5, 'phone': 6}
```

Slightly overfitting, but accuracy is up to 80%!

Clearly there is room for improvement, for the purposes of this project, we will cease the modeling process. Ultimately, we were able to prove that there is capability of expanding to multiple classes other than guns and not gun images.