MVP

As MVP, I focused starting the project from scratch until minimum viable, trying to achieve most of the main subjects, And make it the base work of my project.

Steps of work:

1. Preparing the environment

. Install Python on my machine.
. Install Anaconda.
. Install Jupyter Notebook.
. Install libraries (Numpy, Tensorflow, Pickle... etc..).

2. Import libraries

3. Download/Load dataset

I used CIFAR 10 dataset, it has consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

I trained 50,000 images, and tested 10,000 images.

I displayed images by using Matplotlib.

4. Build a Convolution Neural Network (CNN)

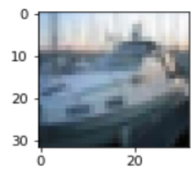
Since its best deep learning model for image classification, so I use it as mean model.

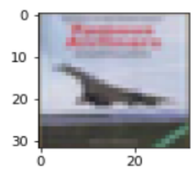
After I trained CNN model, the performance was **70%** as accuracy. Then I predicted **3** images from test dataset as shown in figure (1)

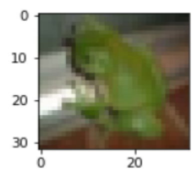
7. Time to predict

I am going to predict elements with indexes (2, 3, 5)

In [17]:
```
# Let's see the elements before ask prediction from the model
plot_image(X_test, y_test, 2)
plot_image(X_test, y_test, 3)
plot_image(X_test, y_test, 7)
```







In [18]:
```
# Start prediction
predict = CNN.predict(X_test)
```

In [19]:
```
# Now Let's ask model
predict_1 = np.argmax(predict[2])
predict_2 = np.argmax(predict[3])
predict_3 = np.argmax(predict[7])

print(predict_1)
print(predict_2)
print(predict_3)
```

```
8
0
2
```

In [20]:
```
# Results
result_1 = classes[predict_1]
result_2 = classes[predict_2]
result_3 = classes[predict_3]

print(result_1)
print(result_2)
print(result_3)
```

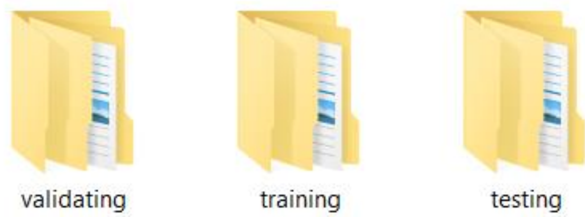
```
ship
airplane
bird
```

Figure (1)

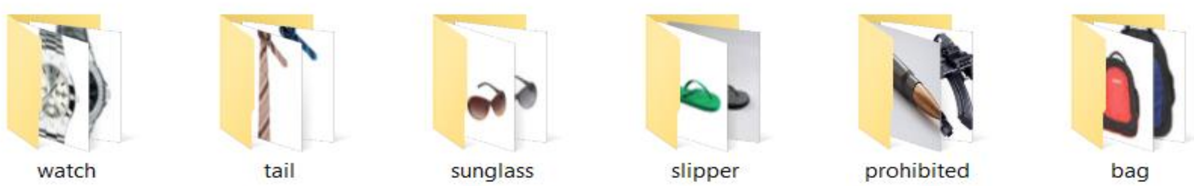
5. Collect/Create my own dataset

One of my goals in this project, is to learn how I can create a local dataset with classifications, and I did, but with small number of images (~ 600 images) and (60*80) image size for each. And the reason behind these small numbers is because the limitation of my laptop, so it's still a good starter for huge concept.

Here is a quick look of my dataset:



Main Folders



Each folder has ~ 100 images

Category Folders

Some of Images in Prohibited folder

5. Tensorflow

The core open source library to help you develop and train ML models.

6. Build a Convolution Neural Network (CNN) to apply on my own dataset

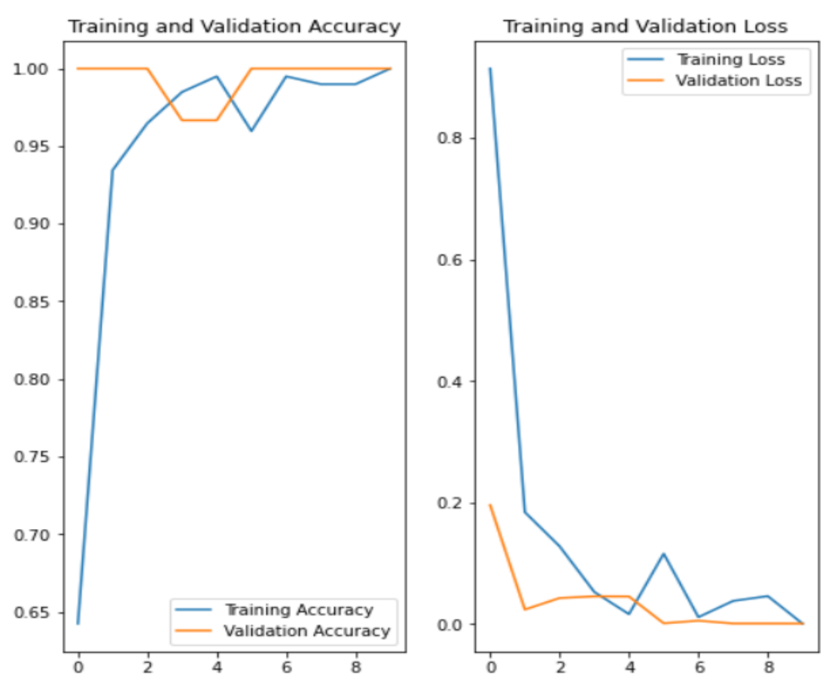
7. First train results

Train the model

In [35]:

```
# Handle train()
history = model.fit(train_ds, validation_data=val_ds, epochs=epochs) # epochs ==> 10
```

```
Epoch 1/10
397/397 [==============================] - 3s 7ms/step - loss: 0.9144 - accuracy: 0.6423 - val_loss: 0.1956 - val_accuracy: 1.0000
Epoch 2/10
397/397 [==============================] - 3s 6ms/step - loss: 0.1841 - accuracy: 0.9345 - val_loss: 0.0237 - val_accuracy: 1.0000
Epoch 3/10
397/397 [==============================] - 2s 6ms/step - loss: 0.1279 - accuracy: 0.9647 - val_loss: 0.0427 - val_accuracy: 1.0000
Epoch 4/10
397/397 [==============================] - 2s 6ms/step - loss: 0.0527 - accuracy: 0.9849 - val_loss: 0.0455 - val_accuracy: 0.9667
Epoch 5/10
397/397 [==============================] - 2s 6ms/step - loss: 0.0161 - accuracy: 0.9950 - val_loss: 0.0453 - val_accuracy: 0.9667
Epoch 6/10
397/397 [==============================] - 2s 6ms/step - loss: 0.1158 - accuracy: 0.9597 - val_loss: 0.0011 - val_accuracy: 1.0000
Epoch 7/10
397/397 [==============================] - 2s 6ms/step - loss: 0.0112 - accuracy: 0.9950 - val_loss: 0.0054 - val_accuracy: 1.0000
Epoch 8/10
397/397 [==============================] - 2s 6ms/step - loss: 0.0379 - accuracy: 0.9899 - val_loss: 7.9636e-04 - val_accuracy: 1.0000
Epoch 9/10
397/397 [==============================] - 3s 6ms/step - loss: 0.0459 - accuracy: 0.9899 - val_loss: 8.0147e-04 - val_accuracy: 1.0000
Epoch 10/10
```



8. Second train results (with some improvements)

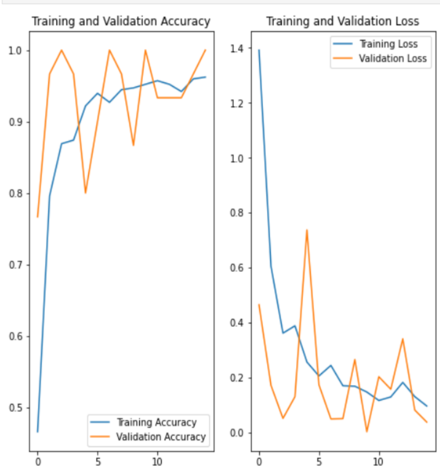
Compile and train the model (Again!)

In [40]:
```
model.compile(optimizer='adam', loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])
```

In [41]:
```
# Reduce epochs value to 15
epochs = 15

history = model.fit(train_ds, validation_data=val_ds, epochs=epochs)
```

```
Epoch 1/15
397/397 [==============================] - 4s 8ms/step - loss: 1.3917 - accuracy: 0.4660 - val_loss: 0.4654 - val_accuracy: 0.7667
Epoch 2/15
397/397 [==============================] - 3s 7ms/step - loss: 0.6050 - accuracy: 0.7960 - val_loss: 0.1719 - val_accuracy: 0.9667
Epoch 3/15
397/397 [==============================] - 3s 7ms/step - loss: 0.3620 - accuracy: 0.8690 - val_loss: 0.0520 - val_accuracy: 1.0000
Epoch 4/15
397/397 [==============================] - 3s 7ms/step - loss: 0.3887 - accuracy: 0.8741 - val_loss: 0.1308 - val_accuracy: 0.9667
Epoch 5/15
397/397 [==============================] - 3s 7ms/step - loss: 0.2569 - accuracy: 0.9219 - val_loss: 0.7373 - val_accuracy: 0.8000
Epoch 6/15
397/397 [==============================] - 3s 7ms/step - loss: 0.2063 - accuracy: 0.9395 - val_loss: 0.1742 - val_accuracy: 0.9000
Epoch 7/15
397/397 [==============================] - 3s 7ms/step - loss: 0.2446 - accuracy: 0.9270 - val_loss: 0.0500 - val_accuracy: 1.0000
Epoch 8/15
397/397 [==============================] - 3s 7ms/step - loss: 0.1707 - accuracy: 0.9446 - val_loss: 0.0510 - val_accuracy: 0.9667
Epoch 9/15
397/397 [==============================] - 3s 8ms/step - loss: 0.1689 - accuracy: 0.9471 - val_loss: 0.2661 - val_accuracy: 0.8667
Epoch 10/15
397/397 [==============================] - 3s 8ms/step - loss: 0.1478 - accuracy: 0.9521 - val_loss: 0.0034 - val_accuracy: 1.0000
Epoch 11/15
397/397 [==============================] - 3s 8ms/step - loss: 0.1171 - accuracy: 0.9572 - val_loss: 0.2034 - val_accuracy: 0.9333
Epoch 12/15
397/397 [==============================] - 3s 8ms/step - loss: 0.1300 - accuracy: 0.9521 - val_loss: 0.1581 - val_accuracy: 0.9333
Epoch 13/15
397/397 [==============================] - 3s 8ms/step - loss: 0.1829 - accuracy: 0.9421 - val_loss: 0.3414 - val_accuracy: 0.9333
Epoch 14/15
397/397 [==============================] - 3s 9ms/step - loss: 0.1312 - accuracy: 0.9597 - val_loss: 0.0823 - val_accuracy: 0.9667
Epoch 15/15
397/397 [==============================] - 4s 11ms/step - loss: 0.0970 - accuracy: 0.9622 - val_loss: 0.0385 - val_accuracy: 1.0000
```

In [42]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

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



I think it's better :)

9. Predict and test

Predict

Let's try prediction on one of images in test folder

In [43]:

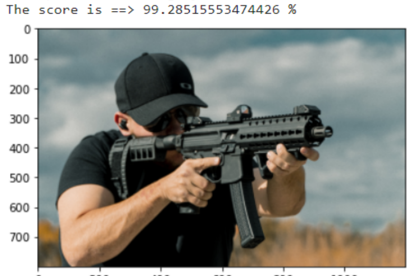
```
# Create image URL variable
image_url =  "dataset/testing/bag/1.jpg"

img = tf.keras.utils.load_img(image_url, target_size=(img_height, img_width)) # Load image and resize it with (60*80)
img_array = tf.keras.utils.img_to_array(img) # Convert to array
img_array = tf.expand_dims(img_array, 0) # Create a batch

# Start predict
predictions = model.predict(img_array)

# Get the score of prediction
score = tf.nn.softmax(predictions[0])

# Display the image
plt.imshow(image.load_img(image_url))

# Show the prediction result with score
print(f"This prediction category is ==> {class_names[np.argmax(score)]}")
print(f"The score is ==> {100 * np.max(score)} %")
```

```
This prediction category is ==> bag
The score is ==> 99.91505146026611 %
```

In [44]:

```
# One more prediction
image_url =  "dataset/testing/prohibited/3.jpg"

img = tf.keras.utils.load_img(image_url, target_size=(img_height, img_width)) # Load image and resize it with (60*80)
img_array = tf.keras.utils.img_to_array(img) # Convert to array
img_array = tf.expand_dims(img_array, 0) # Create a batch

# Start predict
predictions = model.predict(img_array)

# Get the score of prediction
score = tf.nn.softmax(predictions[0])

# Display the image
plt.imshow(image.load_img(image_url))

# Show the prediction result with score
print(f"This prediction category is ==> {class_names[np.argmax(score)]}")
print(f"The score is ==> {100 * np.max(score)} %")
```

```
This prediction category is ==> prohibited
The score is ==> 99.28515553474426 %
```



Nice job until now!

And that`s it for now!