In [2]:

```python
#Import the Libraries in Python
import numpy as np
import seaborn as sns
import random
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
from keras.preprocessing.image import ImageDataGenerator
from keras.layers.normalization import BatchNormalization

from keras.layers.core import Dense, Dropout, Activation # Types of layers to be used in our model
from keras.utils import np_utils                         # NumPy related tools
```

Data Loading

In [3]:

```python
gen = ImageDataGenerator(rescale = 1./255, rotation_range=8, width_shift_range=0.08, shear_range=0.3,
                         height_shift_range=0.08, zoom_range=0.08, validation_split=0.2)

train_gen = gen.flow_from_directory(r'C:\Users\kerby\Desktop\NTUC Learning Hub\Data Analytics Course\17. Capstone Project 4\natural_images',
                                    target_size = (150, 150),
                                    color_mode = 'rgb',
                                    batch_size = 32,
                                    shuffle = True,
                                    seed = 42,
                                    class_mode = 'categorical',subset='training')
test_gen  = gen.flow_from_directory(r'C:\Users\kerby\Desktop\NTUC Learning Hub\Data Analytics Course\17. Capstone Project 4\natural_images',
                                    target_size = (150, 150),
                                    color_mode = 'rgb',
                                    batch_size = 32,
                                    shuffle = True,
                                    seed = 42,
                                    class_mode = 'categorical',subset='validation')
```

```
Found 5522 images belonging to 8 classes.
Found 1377 images belonging to 8 classes.
```

In [4]:

```python
np.array(train_gen[0][0]).shape
```

Out[4]:

```
(32, 150, 150, 3)
```

In [5]:

```python
np.array(train_gen[0][0][0]).shape
```

Out[5]:

```
(150, 150, 3)
```

In [6]:

```python
np.array(train_gen[0][1][0]).shape
```

Out[6]:

```
(8,)
```

In [8]:

```python
train_gen[0][0][0]
```

Out[8]:

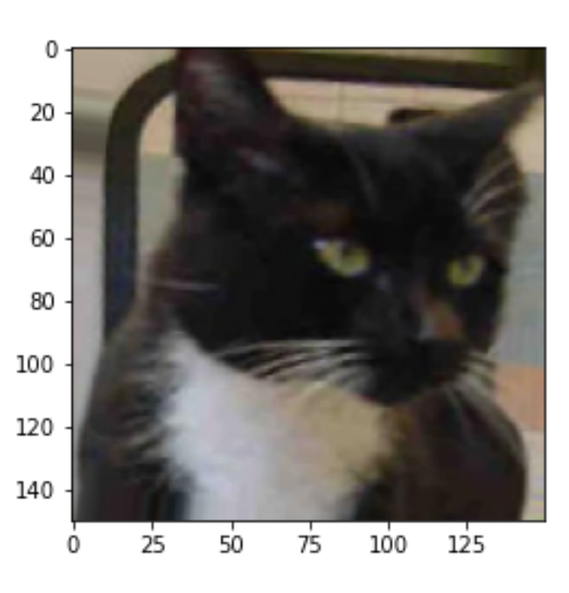
```
array([[[0.7192682 , 0.6604447 , 0.58440125],
        [0.71823764, 0.6594141 , 0.58268356],
        [0.7172071 , 0.6583835 , 0.580966  ],
        ...,
        [0.1913797 , 0.18198219, 0.10740106],
        [0.185123  , 0.16943671, 0.12629946],
        [0.1725775 , 0.15689123, 0.11375397]],

       [[0.7094666 , 0.65064305, 0.57613325],
        [0.7108407 , 0.6520172 , 0.5775074 ],
        [0.71221477, 0.65339124, 0.57888144],
        ...,
        [0.18564197, 0.16450575, 0.11047409],
        [0.17580757, 0.15695736, 0.12963977],
        [0.16948289, 0.1509762 , 0.12424087]],

       [[0.7029861 , 0.64416254, 0.56965274],
        [0.7033296 , 0.6445061 , 0.5699963 ],
        [0.7036732 , 0.64484966, 0.5703398 ],
        ...,
        [0.16163729, 0.1329182 , 0.09929758],
        [0.16005868, 0.14045084, 0.11692141],
        [0.167157  , 0.14754915, 0.12521207]],

       ...,

       [[0.5921569 , 0.5568628 , 0.5294118 ],
        [0.5921569 , 0.5568628 , 0.5294118 ],
        [0.5921569 , 0.5568628 , 0.5294118 ],
        ...,
        [0.69803923, 0.7428136 , 0.6921648 ],
        [0.6981251 , 0.7451153 , 0.6980049 ],
        [0.69984275, 0.7454588 , 0.69731784]],

       [[0.5921569 , 0.5568628 , 0.5294118 ],
        [0.5921569 , 0.5568628 , 0.5294118 ],
        [0.5921569 , 0.5568628 , 0.5294118 ],
        ...,
        [0.6966593 , 0.71390164, 0.62311417],
        [0.69803923, 0.7186719 , 0.6300862 ],
        [0.69803923, 0.72107655, 0.6362696 ]],

       [[0.5921569 , 0.5568628 , 0.5294118 ],
        [0.5921569 , 0.5568628 , 0.5294118 ],
        [0.5921569 , 0.5568628 , 0.5294118 ],
        ...,
        [0.6708599 , 0.64225453, 0.5429419 ],
        [0.67251766, 0.64837414, 0.54724026],
        [0.67492235, 0.6549012 , 0.5547979 ]]], dtype=float32)
```

In [7]:

```python
plt.imshow(train_gen[0][0][0])
```

Out[7]:

```
&lt;matplotlib.image.AxesImage at 0x1315e4b04f0&gt;
```



Building a "Deep" Convolutional Neural Network

In [7]:

```python
from tensorflow.keras import Input, Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dropout, Dense
from tensorflow.keras.callbacks import EarlyStopping

model = Sequential()

# Convolution Layer 1
model.add(Conv2D(32, (3, 3), input_shape=(150,150,3))) # 32 different 3x3 kernels -- so 32 feature maps
model.add(BatchNormalization(axis=-1))               # normalize each feature map before activation
convLayer01 = Activation('relu')                     # activation
model.add(convLayer01)

# Convolution Layer 2
model.add(Conv2D(32, (3, 3)))                        # 32 different 3x3 kernels -- so 32 feature maps
model.add(BatchNormalization(axis=-1))               # normalize each feature map before activation
model.add(Activation('relu'))                        # activation
convLayer02 = MaxPooling2D(pool_size=(2,2))          # Pool the max values over a 2x2 kernel
model.add(convLayer02)

# Convolution Layer 3
model.add(Conv2D(64,(3, 3)))                         # 64 different 3x3 kernels -- so 64 feature maps
model.add(BatchNormalization(axis=-1))               # normalize each feature map before activation
convLayer03 = Activation('relu')                     # activation
model.add(convLayer03)

# Convolution Layer 4
model.add(Conv2D(64, (3, 3)))                        # 64 different 3x3 kernels -- so 64 feature maps
model.add(BatchNormalization(axis=-1))               # normalize each feature map before activation
model.add(Activation('relu'))                        # activation
convLayer04 = MaxPooling2D(pool_size=(2,2))          # Pool the max values over a 2x2 kernel
model.add(convLayer04)
model.add(Flatten())                                 # Flatten final 4x4x64 output matrix into a 1024-length vector

# Fully Connected Layer 5
model.add(Dense(128))                                # 128 FCN nodes
model.add(BatchNormalization())                      # normalization
model.add(Activation('relu'))                        # activation

# Fully Connected Layer 6
model.add(Dropout(0.2))                              # 20% dropout of randomly selected nodes
model.add(Dense(8))                                  # final 8 FCN nodes
model.add(Activation('softmax'))                     # softmax activation

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896       
_________________________________________________________________
batch_normalization (BatchNo (None, 148, 148, 32)      128       
_________________________________________________________________
activation (Activation)      (None, 148, 148, 32)      0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 146, 146, 32)      9248      
_________________________________________________________________
batch_normalization_1 (Batch (None, 146, 146, 32)      128       
_________________________________________________________________
activation_1 (Activation)    (None, 146, 146, 32)      0         
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 73, 73, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 71, 71, 64)        18496     
_________________________________________________________________
batch_normalization_2 (Batch (None, 71, 71, 64)        256       
_________________________________________________________________
activation_2 (Activation)    (None, 71, 71, 64)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 69, 69, 64)        36928     
_________________________________________________________________
batch_normalization_3 (Batch (None, 69, 69, 64)        256       
_________________________________________________________________
activation_3 (Activation)    (None, 69, 69, 64)        0         
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 34, 34, 64)        0         
_________________________________________________________________
flatten (Flatten)            (None, 73984)             0         
_________________________________________________________________
dense (Dense)                (None, 128)               9470080   
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512       
_________________________________________________________________
activation_4 (Activation)    (None, 128)               0         
_________________________________________________________________
dropout (Dropout)            (None, 128)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 1032      
_________________________________________________________________
activation_5 (Activation)    (None, 8)                 0         
=================================================================
Total params: 9,537,960
Trainable params: 9,537,320
Non-trainable params: 640
_________________________________________________________________
```

In [8]:

```python
# we'll use the same optimizer

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [10]:

```python
# We can now train our model which is fed data by our batch loader
# Steps per epoch should always be total size of the set divided by the batch size

# SIGNIFICANT MEMORY SAVINGS (important for larger, deeper networks)

history = model.fit_generator(train_gen,
                              steps_per_epoch=5522//32,
                              epochs=15,
                              verbose=1,
                              validation_data=test_gen,
                              validation_steps=1377//32)
```

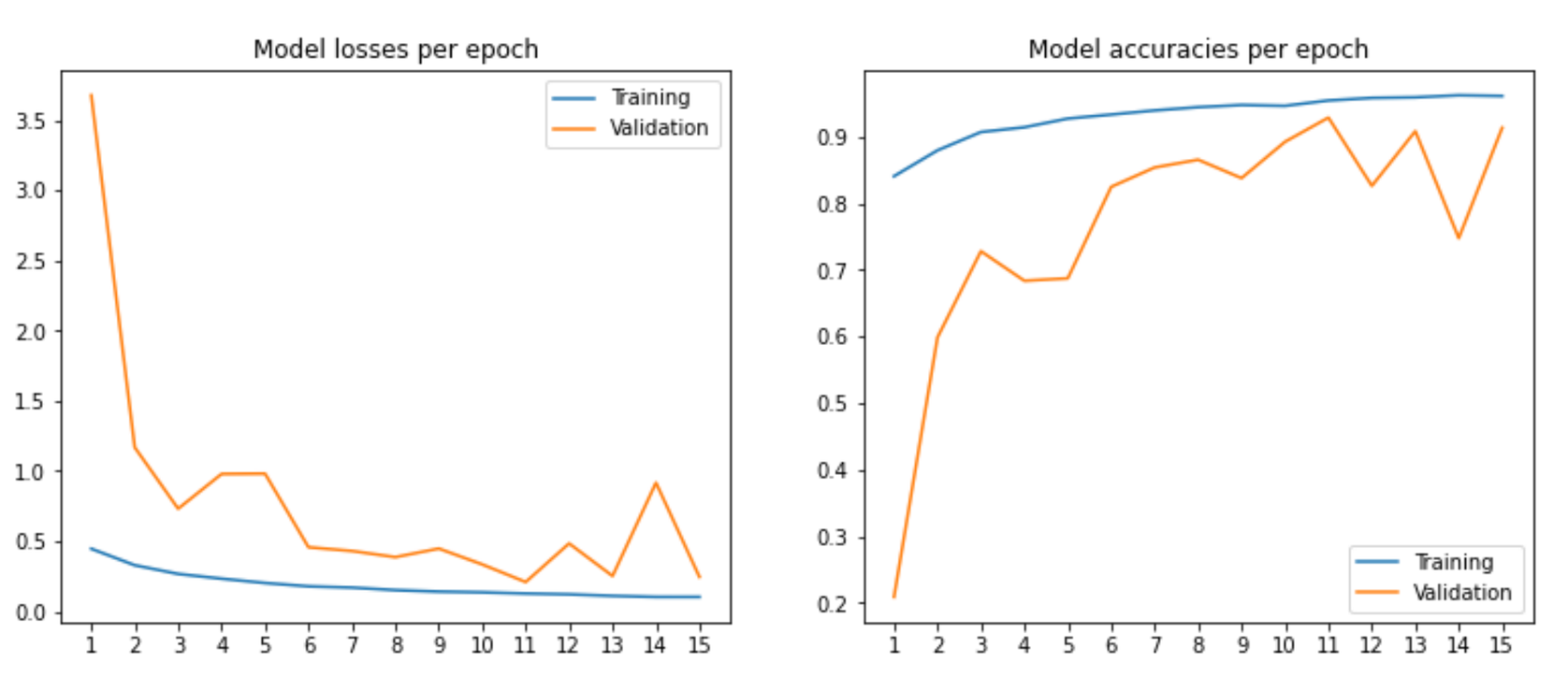
```
Epoch 1/15
172/172 [==============================] - 409s 2s/step - loss: 0.4460 - accuracy: 0.8408 - val_loss: 3.6793 - val_accuracy: 0.2093
Epoch 2/15
172/172 [==============================] - 409s 2s/step - loss: 0.3286 - accuracy: 0.8796 - val_loss: 1.1689 - val_accuracy: 0.5988
Epoch 3/15
172/172 [==============================] - 407s 2s/step - loss: 0.2665 - accuracy: 0.9071 - val_loss: 0.7316 - val_accuracy: 0.7282
Epoch 4/15
172/172 [==============================] - 403s 2s/step - loss: 0.2327 - accuracy: 0.9142 - val_loss: 0.9793 - val_accuracy: 0.6839
Epoch 5/15
172/172 [==============================] - 829s 5s/step - loss: 0.2018 - accuracy: 0.9273 - val_loss: 0.9815 - val_accuracy: 0.6875
Epoch 6/15
172/172 [==============================] - 399s 2s/step - loss: 0.1780 - accuracy: 0.9333 - val_loss: 0.4567 - val_accuracy: 0.8249
Epoch 7/15
172/172 [==============================] - 398s 2s/step - loss: 0.1689 - accuracy: 0.9395 - val_loss: 0.4301 - val_accuracy: 0.8539
Epoch 8/15
172/172 [==============================] - 399s 2s/step - loss: 0.1511 - accuracy: 0.9444 - val_loss: 0.3863 - val_accuracy: 0.8656
Epoch 9/15
172/172 [==============================] - 399s 2s/step - loss: 0.1401 - accuracy: 0.9477 - val_loss: 0.4473 - val_accuracy: 0.8379
Epoch 10/15
172/172 [==============================] - 400s 2s/step - loss: 0.1353 - accuracy: 0.9464 - val_loss: 0.3343 - val_accuracy: 0.8924
Epoch 11/15
172/172 [==============================] - 400s 2s/step - loss: 0.1268 - accuracy: 0.9545 - val_loss: 0.2087 - val_accuracy: 0.9288
Epoch 12/15
172/172 [==============================] - 400s 2s/step - loss: 0.1210 - accuracy: 0.9583 - val_loss: 0.4838 - val_accuracy: 0.8263
Epoch 13/15
172/172 [==============================] - 399s 2s/step - loss: 0.1100 - accuracy: 0.9592 - val_loss: 0.2518 - val_accuracy: 0.9084
Epoch 14/15
172/172 [==============================] - 400s 2s/step - loss: 0.1026 - accuracy: 0.9623 - val_loss: 0.9167 - val_accuracy: 0.7478
Epoch 15/15
172/172 [==============================] - 400s 2s/step - loss: 0.1024 - accuracy: 0.9612 - val_loss: 0.2450 - val_accuracy: 0.9135
```

In [11]:

```python
# Let's plot the model accuracy and loss curves during training
epoch_range = range(1, len(history.history['loss']) + 1)
plt.subplots(figsize=(13,5))
plt.subplot(1,2,1)
plt.plot(epoch_range, history.history['loss'])
plt.plot(epoch_range, history.history['val_loss'])
plt.title("Model losses per epoch")
plt.xticks(epoch_range)
plt.legend(['Training', 'Validation'])
plt.subplot(1,2,2)
plt.plot(epoch_range, history.history['accuracy'])
plt.plot(epoch_range, history.history['val_accuracy'])
plt.title("Model accuracies per epoch")
plt.xticks(epoch_range)
plt.legend(['Training', 'Validation'])
```

Out[11]:

```
&lt;matplotlib.legend.Legend at 0x1a2b78a6520&gt;
```



Test Accuracy

In [12]:

```python
score = model.evaluate(test_gen)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
44/44 [==============================] - 19s 439ms/step - loss: 0.2507 - accuracy: 0.9114
Test score: 0.2506970167160034
Test accuracy: 0.9114015698432922
```

In [13]:

```python
classes = ['airplane','car','cat','dog','flower','fruit','motorbike','person']
```

In [14]:

```python
x_test = []
y_test = []

for i in range(len(test_gen)):
    x_test += list(test_gen[i][0])
    y_test += list(test_gen[i][1])
```

In [15]:

```python
x_test = np.array(x_test)
x_test.shape
```

Out[15]:

```
(1377, 150, 150, 3)
```

In [16]:

```python
y_test = np.array(y_test)
y_test.shape
```

Out[16]:

```
(1377, 8)
```

In [17]:

```python
y_pred = model.predict(x_test)
y_pred_idx = np.argmax(y_pred, axis=-1)
y_test_idx = np.argmax(y_test, axis=1)
```

In [18]:

```python
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_test_idx, y_pred_idx))
```

```
              precision    recall  f1-score   support

           0       0.99      0.80      0.89       145
           1       0.99      0.93      0.96       193
           2       0.79      0.86      0.82       177
           3       0.72      0.79      0.75       140
           4       0.96      0.93      0.95       168
           5       0.96      1.00      0.98       200
           6       0.97      0.98      0.98       157
           7       0.98      0.99      0.98       197

    accuracy                           0.92      1377
   macro avg       0.92      0.91      0.91      1377
weighted avg       0.92      0.92      0.92      1377
```

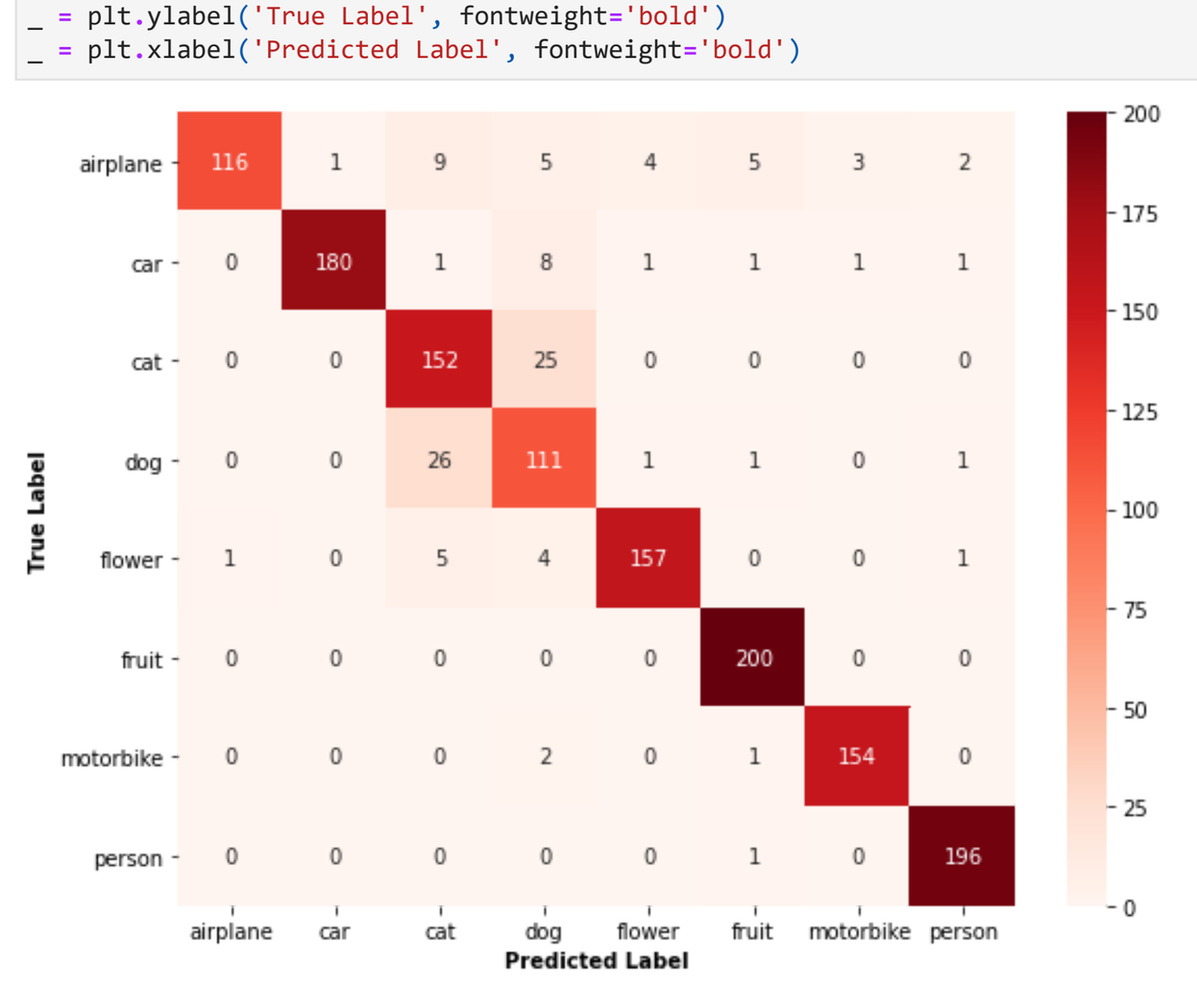
In [19]:

```python
print(confusion_matrix(y_test_idx, y_pred_idx))
```

```
[[116   1   9   5   4   5   3   2]
 [  0 180   1   8   1   1   1   1]
 [  0   0 152  25   0   0   0   0]
 [  0   0  26 111   1   1   0   1]
 [  1   0   5   4 157   0   0   1]
 [  0   0   0   0   0 200   0   0]
 [  0   0   0   2   0   1 154   0]
 [  0   0   0   0   0   1   0 196]]
```

In [20]:

```python
# Plot the confusion matrix using Seaborn library
plt.figure(figsize=(9,7))
_ = sns.heatmap(confusion_matrix(y_test_idx, y_pred_idx),
                annot=True, annot_kws={"size":10}, fmt='d',
                cmap=plt.cm.Reds,
                xticklabels=classes, yticklabels=classes)
_ = plt.ylabel('True Label', fontweight='bold')
_ = plt.xlabel('Predicted Label', fontweight='bold')
```



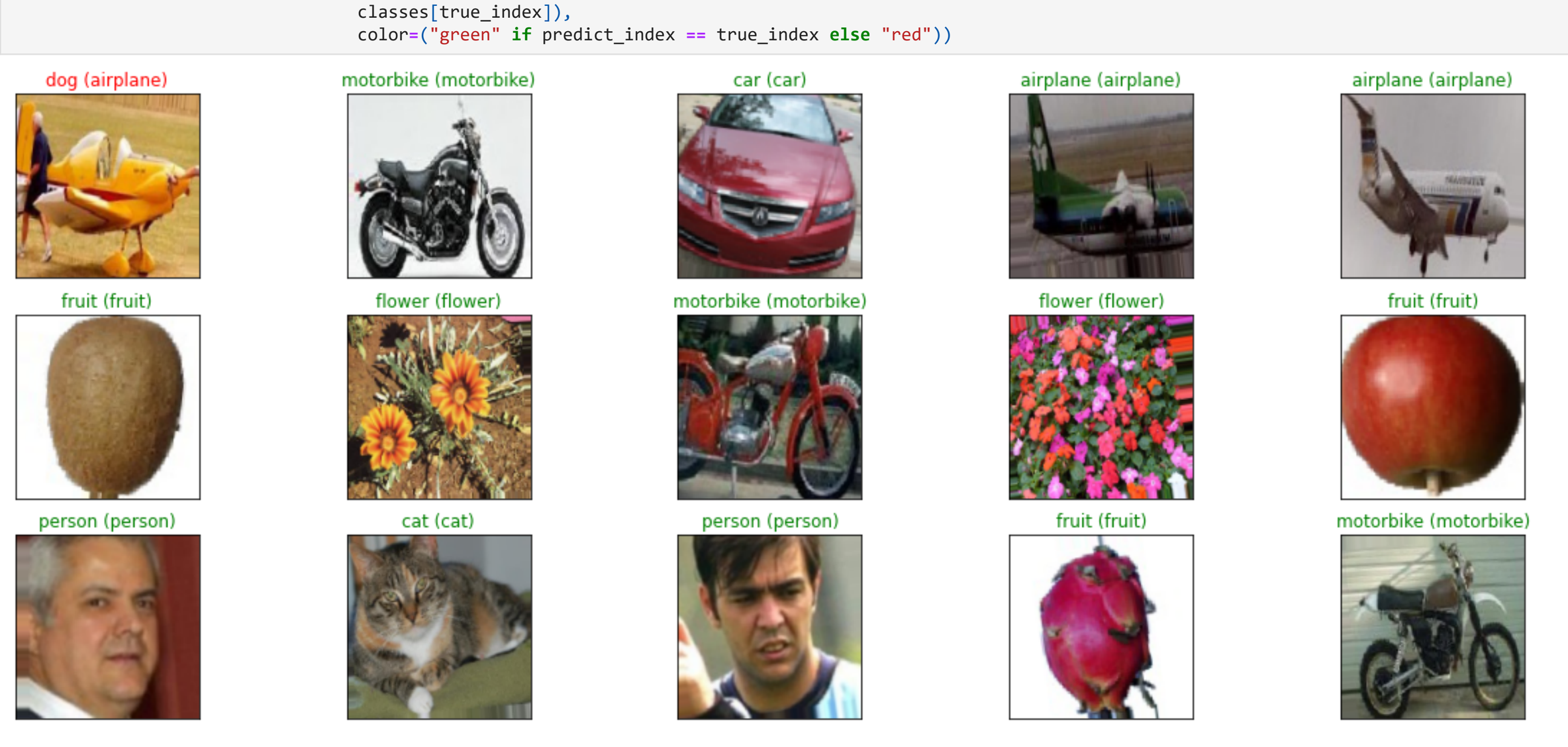
Visualizing the Predictions

In [21]:

```python
classes = ['airplane','car','cat','dog','flower','fruit','motorbike','person']
```

In [22]:

```python
# Plot a random sample of test images, their predicted labels and ground truth
figure = plt.figure(figsize=(20, 8))
for i, index in enumerate(np.random.choice(x_test.shape[0], size=15, replace=False)):
    ax = figure.add_subplot(3, 5, i + 1, xticks=[], yticks=[])
    # Display each image
    ax.imshow(x_test[index])
    predict_index = np.argmax(y_pred[index])
    true_index = np.argmax(y_test[index])
    # Set the title for each image
    ax.set_title("{} ({})".format(classes[predict_index],
                                  classes[true_index]),
                                  color=("green" if predict_index == true_index else "red"))
```



Test using Google Image

In [3]:

```python
from PIL import Image
from numpy import asarray
```

In [4]:

```python
def predict_pic(picture):
    """ takes in a jpg file --> 'pic.jpg' """

    image = Image.open(picture) #Load the image
    image_resize = np.array(image.resize((150,150))) #Resize the image
    plt.imshow(image_resize)
    image_resize_expand = np.expand_dims(image_resize, 0)

    image_pred = model.predict(image_resize_expand) #Calling out model
    image_pred_idx = int(np.argmax(image_pred, axis=-1))
    print('Predicted image is:', classes[image_pred_idx])
```

In [5]:

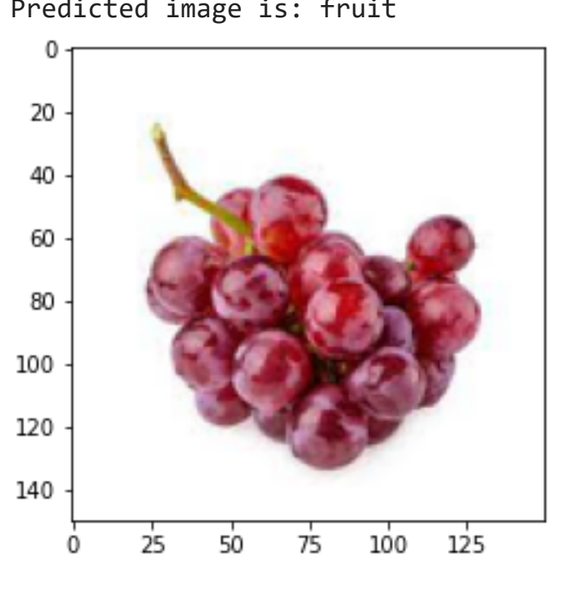
```python
predict_pic('apple.jpg')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-5-2e033c56d903> in <module>
----> 1 predict_pic('apple.jpg')

<ipython-input-4-b2b6356681eb> in predict_pic(picture)
      3 
      4     image = Image.open(picture) #Load the image
----> 5     image_resize = np.array(image.resize((150,150))) #Resize the image
      6     plt.imshow(image_resize)
      7     image_resize_expand = np.expand_dims(image_resize, 0)

NameError: name 'np' is not defined
```

In [39]:

```python
predict_pic('grape.jpg')
```

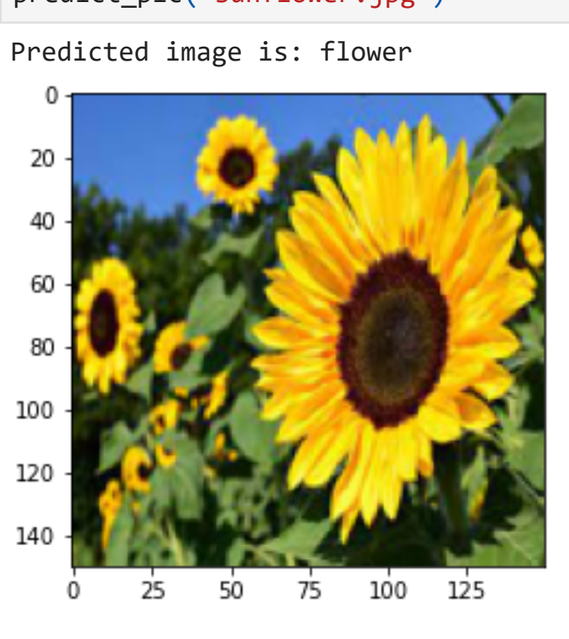
```
Predicted image is: fruit
```



In [27]:

```python
predict_pic('sunflower.jpg')
```

```
Predicted image is: flower
```

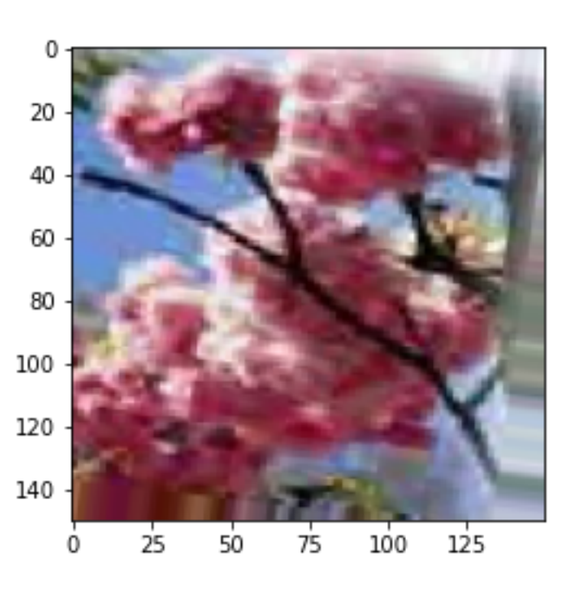


In [29]:

```python
plt.imshow(test_gen[2][0][12])
```

Out[29]:

```
&lt;matplotlib.image.AxesImage at 0x1a2b8feb9d0&gt;
```



In [30]:

```python
image_pred = model.predict(np.expand_dims(test_gen[2][0][12],0))
image_pred_idx = int(np.argmax(image_pred, axis=-1))
print('Predicted image is:', classes[image_pred_idx])
```

```
Predicted image is: flower
```