CIFAR-100 Image Classification

Installing the Libraries

In [1]:

```
# !pip install tensorflow
# !pip install keras
# !pip install h5py
# !pip install pandas
# !pip install numpy
# !pip install pickle
```

Importing the Libraries

In [2]:

```
import pickle
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pylab import rcParams
import tensorflow as tf
import keras
%matplotlib inline
from keras.models import Sequential, load_model
from keras.layers import Conv2D, MaxPool2D, Dropout, Flatten, Dense
from keras.layers.normalization import BatchNormalization
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import Callback, EarlyStopping, ModelCheckpoint
from sklearn.metrics import confusion_matrix, classification_report
from skimage.transform import resize
import seaborn as sns
import cv2
```

```
Using TensorFlow backend.
```

Loading the CIFAR-100 Dataset

In [3]:

```
#function to open the files in the Python version of the dataset
def unpickle(file):
    with open(file, 'rb') as fo:
        myDict = pickle.load(fo, encoding='latin1')
    return myDict
```

In [4]:

```
trainData = unpickle('train')

#type of items in each file
for item in trainData:
    print(item, type(trainData[item]))
```

```
filenames <class 'list'>
batch_label <class 'str'>
fine_labels <class 'list'>
coarse_labels <class 'list'>
data <class 'numpy.ndarray'>
```

In [5]:

```
print(len(trainData['data']))
print(len(trainData['data'][0]))
```

```
50000
3072
```

There are 50000 images in the training dataset and each image is a 3 channel 32 *32 pixel image (32* 32 * 3 = 3072).

In [6]:

```
print(np.unique(trainData['fine_labels']))
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
 96 97 98 99]
```

There are 100 different fine labels for the images (0 to 99).

In [7]:

```
print(np.unique(trainData['coarse_labels']))
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
```

There are 10 different coarse labels for the images (0 to 9).

In [8]:

```
print(trainData['batch_label'])
```

```
training batch 1 of 1
```

In [9]:

```
print(len(trainData['filenames']))
```

```
50000
```

In [10]:

```
testData = unpickle('test')
#testData
```

In [11]:

```
metaData = unpickle('meta')
#metaData
```

Meta file has a dictionary of fine labels and coarse labels.

In [12]:

```
#storing coarse labels along with its number code in a dataframe
category = pd.DataFrame(metaData['coarse_label_names'], columns=['SuperClass'])
category
```

Out[12]:

	SuperClass
0	aquatic_mammals
1	fish
2	flowers
3	food_containers
4	fruit_and_vegetables
5	household_electrical_devices
6	household_furniture
7	insects
8	large_carnivores
9	large_man-made_outdoor_things
10	large_natural_outdoor_scenes
11	large_omnivores_and_herbivores
12	medium_mammals
13	non-insect_invertebrates
14	people
15	reptiles
16	small_mammals
17	trees
18	vehicles_1
19	vehicles_2

The above list shows coarse label number and name, which we are denoting as categories.

In [13]:

```
#storing fine labels along with its number code in a dataframe
subCategory = pd.DataFrame(metaData['fine_label_names'], columns=['SubClass'])
subCategory
```

Out[13]:

	SubClass
0	apple
1	aquarium_fish
2	baby
3	bear
4	beaver
...	...
95	whale
96	willow_tree
97	wolf
98	woman
99	worm

100 rows × 1 columns

The above list shows fine label number and name, which we are denoting as subcategories.

In [14]:

```
X_train = trainData['data']
X_train
```

Out[14]:

```
array([[255, 255, 255, ...,  10,  59,  79],
       [255, 253, 253, ..., 253, 253, 255],
       [250, 248, 247, ..., 194, 207, 228],
       ...,
       [248, 240, 236, ..., 180, 174, 205],
       [156, 151, 151, ..., 114, 107, 126],
       [ 31,  30,  31, ...,  72,  69,  67]], dtype=uint8)
```

Image Transformation for Tensorflow (Keras) and Convolutional Neural Networks

In [15]:

```
#4D array input for building the CNN model using Keras
X_train = X_train.reshape(len(X_train),3,32,32).transpose(0,2,3,1)
#X_train
```

Exploring the Images in the Dataset

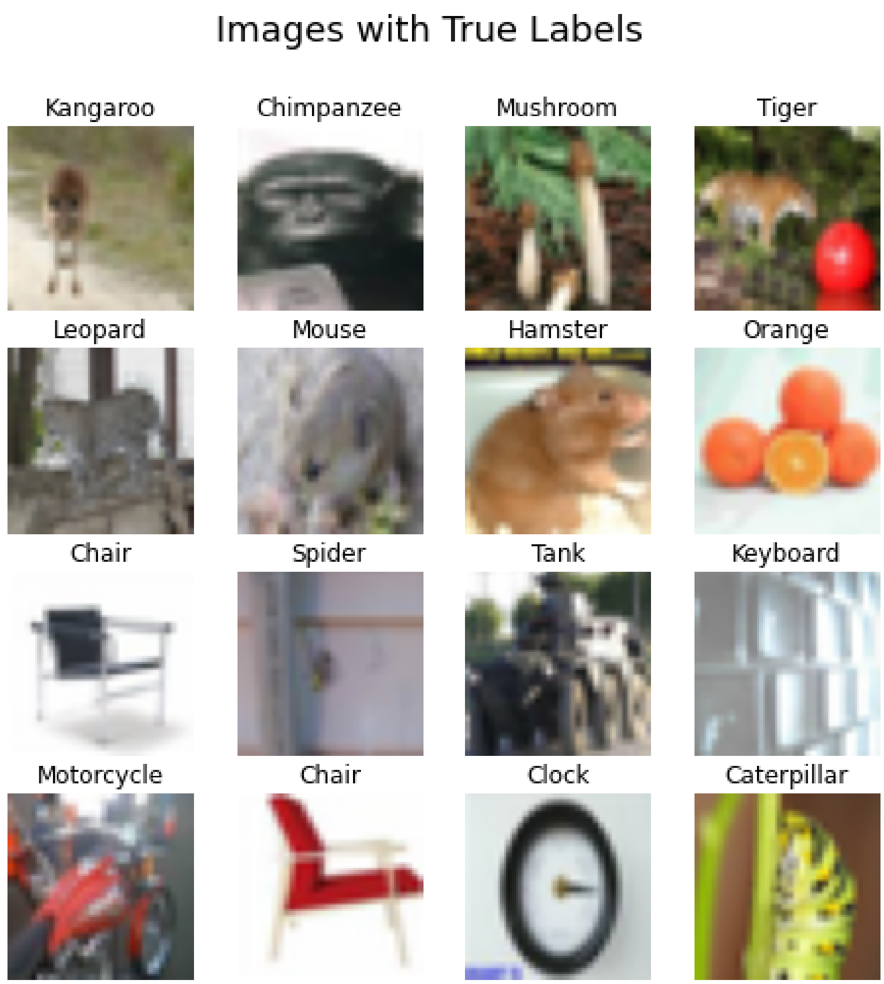
In [16]:

```
#generating a random number to display a random image from the dataset along with the label's number and name

rcParams['figure.figsize'] = 2,2

imageId = np.random.randint(0, len(X_train))

plt.imshow(X_train[imageId])

plt.axis('off')

print("Image number selected : {}".format(imageId))
print("Shape of image : {}".format(X_train[imageId].shape))
print("Image category number: {}".format(trainData['coarse_labels'][imageId]))
print("Image category name: {}".format(category.iloc[trainData['coarse_labels'][imageId]][0].capitalize()))
print("Image subcategory number: {}".format(trainData['fine_labels'][imageId]))
print("Image subcategory name: {}".format(subCategory.iloc[trainData['fine_labels'][imageId]][0].capitalize()))
```

```
Image number selected : 16699
Shape of image : (32, 32, 3)
Image category number: 2
Image category name: Flowers
Image subcategory number: 70
Image subcategory name: Rose
```

In [17]:

```
#16 random images to display at a time along with their true labels
rcParams['figure.figsize'] = 8,8

num_row = 4
num_col = 4

#to get 4 * 4 = 16 images together
imageId = np.random.randint(0, len(X_train), num_row * num_col)
#imageId

fig, axes = plt.subplots(num_row, num_col)
plt.suptitle('Images with True Labels', fontsize=18)

for i in range(0, num_row):
    for j in range(0, num_col):
        k = (i*num_col)+j
        axes[i,j].imshow(X_train[imageId[k]])
        axes[i,j].set_title(subCategory.iloc[trainData['fine_labels'][imageId[k]]][0]
.capitalize())
        axes[i,j].axis('off')
```



Data Pre-processing

In [18]:

```
#transforming the testing dataset
X_test = testData['data']
X_test = X_test.reshape(len(X_test),3,32,32).transpose(0,2,3,1)
X_test.shape
```

Out[18]:

```
(10000, 32, 32, 3)
```

In [19]:

```
y_train = trainData['fine_labels']
#y_train

y_test = testData['fine_labels']
#y_test
```

Converting class vectors to binary class matrices

In [20]:

```
num_class = 100

y_train = keras.utils.to_categorical(y_train, num_class)
#y_train

y_test = keras.utils.to_categorical(y_test, num_class)
#y_test
```

Rescaling by dividing every image pixel by 255

In [21]:

```
X_train = X_train / 255.
#X_train

X_test = X_test / 255.
#X_test
```

Building Convolutional Neural Network

In [22]:

```
#initializing CNN model
model = Sequential()

#Stack 1
#convolution
model.add(Conv2D(filters=128, kernel_size=3, padding="same", activation="relu", input
_shape=X_train.shape[1:]))
model.add(Conv2D(filters=128, kernel_size=3, padding="same", activation="relu"))
#pooling
model.add(MaxPool2D(pool_size=2, strides=2))
#dropout
model.add(Dropout(0.2))

#Stack 2
#convolution
model.add(Conv2D(filters=256, kernel_size=3, padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=3, padding="same", activation="relu"))
#pooling
model.add(MaxPool2D(pool_size=2, strides=2))
#dropout
model.add(Dropout(0.5))

#Stack 3
#convolution
model.add(Conv2D(filters=512, kernel_size=3, padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=3, padding="same", activation="relu"))
#pooling
model.add(MaxPool2D(pool_size=2, strides=2))
#dropout
model.add(Dropout(0.5))

#flattening
model.add(Flatten())

#full connection
model.add(Dense(units=1000, activation="relu"))
#dropout
model.add(Dropout(0.5))

#full connection
model.add(Dense(units=1000, activation="relu"))
#dropout
model.add(Dropout(0.5))

#output layer
model.add(Dense(units=num_class, activation="softmax"))
```

In [23]:

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 128)       3584      
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 128)       147584    
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 128)       0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 128)       0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 256)       295168    
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 256)       590080    
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 256)         0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 256)         0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 512)         1180160   
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 512)         2359808   
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 512)         0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 4, 512)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 1000)              8193000   
_________________________________________________________________
dropout_4 (Dropout)          (None, 1000)              0         
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              1001000   
_________________________________________________________________
dropout_5 (Dropout)          (None, 1000)              0         
_________________________________________________________________
dense_3 (Dense)              (None, 100)               100100    
=================================================================
Total params: 13,870,484
Trainable params: 13,870,484
Non-trainable params: 0
_________________________________________________________________
```

Training Convolutional Neural Network

In [24]:

```
epochs = 100
batch_size = 64
```

In [25]:

```
optimizer = keras.optimizers.Adam(lr=0.0001)

#model compiling
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accurac
y'])
```

In [26]:

```
#early stopping to monitor the validation loss and avoid overfitting
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=20)

#saving the model checkpoint for the best model
model_checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss', mode='min', s
ave_best_only=True, verbose=1)
```

In [27]:

```
#image augmentation to expand the training dataset
#validation split to test the model
data_gen = ImageDataGenerator(
            shear_range=0.2,
            zoom_range=0.2,
            horizontal_flip=True,
            featurewise_center=True,
            width_shift_range=0.1,
            validation_split=0.2)

data_gen.fit(X_train, seed=123)

train_data_gen = data_gen.flow(X_train, y_train,
                               batch_size=batch_size,
                               subset="training", seed=123)

valid_data_gen = data_gen.flow(X_train, y_train,
                               batch_size=batch_size,
                               subset="validation", seed=123)
```

In [28]:

```
model_history = model.fit_generator(train_data_gen,
                          steps_per_epoch=40000//batch_size,
                          validation_data=valid_data_gen,
                          validation_steps=10000//batch_size,
                          callbacks=[early_stop, model_checkpoint],
                          verbose=1,
                          epochs=epochs)
```

```
Epoch 1/100
625/625 [==============================] - 77s 123ms/step - loss: 4.4891
- accuracy: 0.0225 - val_loss: 4.1244 - val_accuracy: 0.0422

Epoch 00001: val_loss improved from inf to 4.12444, saving model to best_
model.h5
Epoch 2/100
625/625 [==============================] - 62s 99ms/step - loss: 4.1185 -
accuracy: 0.0613 - val_loss: 3.9124 - val_accuracy: 0.0987

Epoch 00002: val_loss improved from 4.12444 to 3.91238, saving model to b
est_model.h5
Epoch 3/100
625/625 [==============================] - 62s 100ms/step - loss: 3.8198
- accuracy: 0.1062 - val_loss: 3.5346 - val_accuracy: 0.1519

Epoch 00003: val_loss improved from 3.91238 to 3.53457, saving model to b
est_model.h5
Epoch 4/100
625/625 [==============================] - 62s 100ms/step - loss: 3.6061
- accuracy: 0.1402 - val_loss: 3.3968 - val_accuracy: 0.1896

Epoch 00004: val_loss improved from 3.53457 to 3.39685, saving model to b
est_model.h5
Epoch 5/100
625/625 [==============================] - 62s 99ms/step - loss: 3.4123 -
accuracy: 0.1716 - val_loss: 3.2359 - val_accuracy: 0.2170

Epoch 00005: val_loss improved from 3.39685 to 3.23591, saving model to b
est_model.h5
Epoch 6/100
625/625 [==============================] - 62s 99ms/step - loss: 3.2499 -
accuracy: 0.1996 - val_loss: 3.0751 - val_accuracy: 0.2503

Epoch 00006: val_loss improved from 3.23591 to 3.07513, saving model to b
est_model.h5
Epoch 7/100
625/625 [==============================] - 62s 99ms/step - loss: 3.1088 -
accuracy: 0.2277 - val_loss: 2.9113 - val_accuracy: 0.2729

Epoch 00007: val_loss improved from 3.07513 to 2.91127, saving model to b
est_model.h5
Epoch 8/100
625/625 [==============================] - 62s 99ms/step - loss: 2.9730 -
accuracy: 0.2531 - val_loss: 2.9122 - val_accuracy: 0.3055

Epoch 00008: val_loss did not improve from 2.91127
Epoch 9/100
625/625 [==============================] - 62s 100ms/step - loss: 2.8634
- accuracy: 0.2762 - val_loss: 2.4022 - val_accuracy: 0.3212

Epoch 00009: val_loss improved from 2.91127 to 2.40219, saving model to b
est_model.h5
Epoch 10/100
625/625 [==============================] - 62s 99ms/step - loss: 2.7357 -
accuracy: 0.3020 - val_loss: 2.3201 - val_accuracy: 0.3569

Epoch 00010: val_loss improved from 2.40219 to 2.32013, saving model to b
```

```
est_model.h5
Epoch 11/100
625/625 [==============================] - 62s 99ms/step - loss: 2.6381 -
accuracy: 0.3220 - val_loss: 2.4147 - val_accuracy: 0.3591

Epoch 00011: val_loss did not improve from 2.32013
Epoch 12/100
625/625 [==============================] - 62s 99ms/step - loss: 2.5316 -
accuracy: 0.3428 - val_loss: 2.8137 - val_accuracy: 0.3879

Epoch 00012: val_loss did not improve from 2.32013
Epoch 13/100
625/625 [==============================] - 62s 99ms/step - loss: 2.4537 -
accuracy: 0.3599 - val_loss: 2.0775 - val_accuracy: 0.4067

Epoch 00013: val_loss improved from 2.32013 to 2.07748, saving model to b
est_model.h5
Epoch 14/100
625/625 [==============================] - 62s 99ms/step - loss: 2.3710 -
accuracy: 0.3763 - val_loss: 1.9521 - val_accuracy: 0.4131

Epoch 00014: val_loss improved from 2.07748 to 1.95212, saving model to b
est_model.h5
Epoch 15/100
625/625 [==============================] - 62s 99ms/step - loss: 2.3059 -
accuracy: 0.3904 - val_loss: 1.9901 - val_accuracy: 0.4329

Epoch 00015: val_loss did not improve from 1.95212
Epoch 16/100
625/625 [==============================] - 62s 99ms/step - loss: 2.2357 -
accuracy: 0.4078 - val_loss: 2.2303 - val_accuracy: 0.4473

Epoch 00016: val_loss did not improve from 1.95212
Epoch 17/100
625/625 [==============================] - 62s 99ms/step - loss: 2.1795 -
accuracy: 0.4183 - val_loss: 2.2215 - val_accuracy: 0.4495

Epoch 00017: val_loss did not improve from 1.95212
Epoch 18/100
625/625 [==============================] - 62s 99ms/step - loss: 2.1217 -
accuracy: 0.4331 - val_loss: 2.3314 - val_accuracy: 0.4598

Epoch 00018: val_loss did not improve from 1.95212
Epoch 19/100
625/625 [==============================] - 62s 99ms/step - loss: 2.0694 -
accuracy: 0.4441 - val_loss: 2.0617 - val_accuracy: 0.4695

Epoch 00019: val_loss did not improve from 1.95212
Epoch 20/100
625/625 [==============================] - 62s 99ms/step - loss: 2.0202 -
accuracy: 0.4569 - val_loss: 1.9479 - val_accuracy: 0.4819

Epoch 00020: val_loss improved from 1.95212 to 1.94786, saving model to b
est_model.h5
Epoch 21/100
625/625 [==============================] - 62s 99ms/step - loss: 1.9734 -
accuracy: 0.4679 - val_loss: 1.7877 - val_accuracy: 0.4767
```

```
Epoch 00021: val_loss improved from 1.94786 to 1.78774, saving model to b
est_model.h5
Epoch 22/100
625/625 [==============================] - 62s 99ms/step - loss: 1.9263 -
accuracy: 0.4798 - val_loss: 1.8871 - val_accuracy: 0.4880

Epoch 00022: val_loss did not improve from 1.78774
Epoch 23/100
625/625 [==============================] - 62s 99ms/step - loss: 1.8884 -
accuracy: 0.4848 - val_loss: 1.5224 - val_accuracy: 0.4904

Epoch 00023: val_loss improved from 1.78774 to 1.52237, saving model to b
est_model.h5
Epoch 24/100
625/625 [==============================] - 62s 99ms/step - loss: 1.8536 -
accuracy: 0.4964 - val_loss: 1.7345 - val_accuracy: 0.5079

Epoch 00024: val_loss did not improve from 1.52237
Epoch 25/100
625/625 [==============================] - 62s 99ms/step - loss: 1.8101 -
accuracy: 0.5041 - val_loss: 1.9266 - val_accuracy: 0.5103

Epoch 00025: val_loss did not improve from 1.52237
Epoch 26/100
625/625 [==============================] - 62s 99ms/step - loss: 1.7770 -
accuracy: 0.5099 - val_loss: 1.7127 - val_accuracy: 0.5074

Epoch 00026: val_loss did not improve from 1.52237
Epoch 27/100
625/625 [==============================] - 62s 99ms/step - loss: 1.7369 -
accuracy: 0.5195 - val_loss: 1.8954 - val_accuracy: 0.5264

Epoch 00027: val_loss did not improve from 1.52237
Epoch 28/100
625/625 [==============================] - 62s 99ms/step - loss: 1.7037 -
accuracy: 0.5314 - val_loss: 1.4269 - val_accuracy: 0.5231

Epoch 00028: val_loss improved from 1.52237 to 1.42691, saving model to b
est_model.h5
Epoch 29/100
625/625 [==============================] - 62s 99ms/step - loss: 1.6765 -
accuracy: 0.5353 - val_loss: 1.1089 - val_accuracy: 0.5318

Epoch 00029: val_loss improved from 1.42691 to 1.10886, saving model to b
est_model.h5
Epoch 30/100
625/625 [==============================] - 62s 98ms/step - loss: 1.6344 -
accuracy: 0.5448 - val_loss: 1.6630 - val_accuracy: 0.5352

Epoch 00030: val_loss did not improve from 1.10886
Epoch 31/100
625/625 [==============================] - 62s 99ms/step - loss: 1.6083 -
accuracy: 0.5523 - val_loss: 1.6792 - val_accuracy: 0.5399

Epoch 00031: val_loss did not improve from 1.10886
Epoch 32/100
625/625 [==============================] - 62s 99ms/step - loss: 1.5860 -
accuracy: 0.5574 - val_loss: 1.8428 - val_accuracy: 0.5421
```

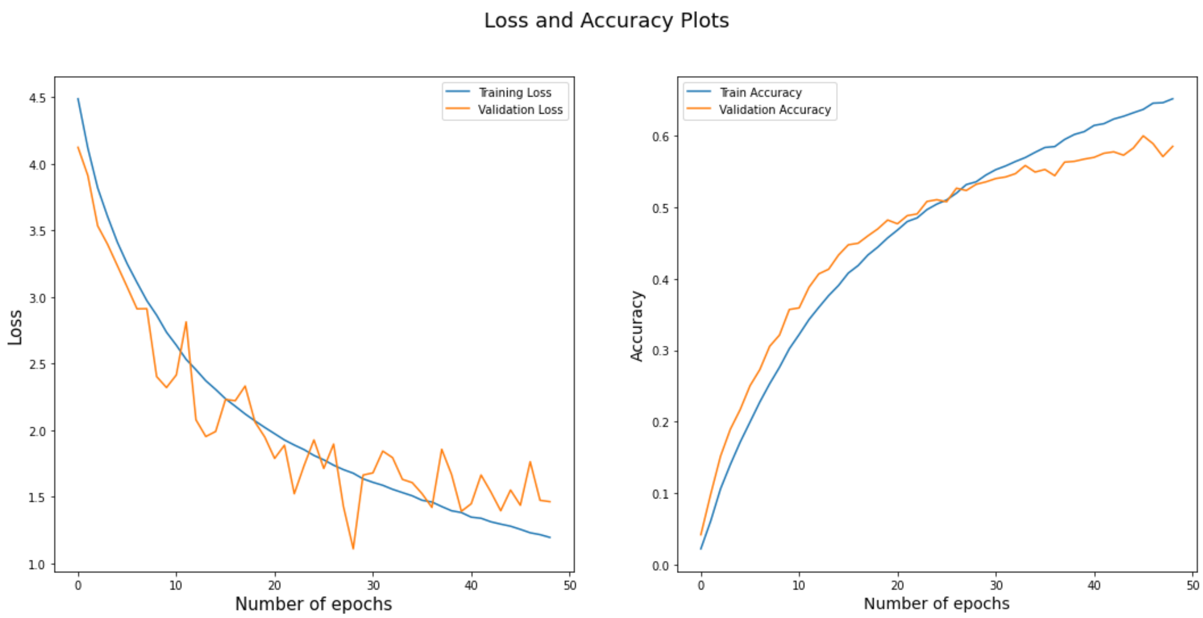
```
Epoch 00032: val_loss did not improve from 1.10886
Epoch 33/100
625/625 [==============================] - 62s 99ms/step - loss: 1.5558 -
accuracy: 0.5637 - val_loss: 1.7934 - val_accuracy: 0.5467

Epoch 00033: val_loss did not improve from 1.10886
Epoch 34/100
625/625 [==============================] - 62s 99ms/step - loss: 1.5311 -
accuracy: 0.5692 - val_loss: 1.6305 - val_accuracy: 0.5581

Epoch 00034: val_loss did not improve from 1.10886
Epoch 35/100
625/625 [==============================] - 62s 99ms/step - loss: 1.5070 -
accuracy: 0.5765 - val_loss: 1.6059 - val_accuracy: 0.5489

Epoch 00035: val_loss did not improve from 1.10886
Epoch 36/100
625/625 [==============================] - 62s 99ms/step - loss: 1.4734 -
accuracy: 0.5833 - val_loss: 1.5226 - val_accuracy: 0.5525

Epoch 00036: val_loss did not improve from 1.10886
Epoch 37/100
625/625 [==============================] - 62s 99ms/step - loss: 1.4605 -
accuracy: 0.5845 - val_loss: 1.4200 - val_accuracy: 0.5439

Epoch 00037: val_loss did not improve from 1.10886
Epoch 38/100
625/625 [==============================] - 62s 99ms/step - loss: 1.4264 -
accuracy: 0.5944 - val_loss: 1.8564 - val_accuracy: 0.5628

Epoch 00038: val_loss did not improve from 1.10886
Epoch 39/100
625/625 [==============================] - 62s 99ms/step - loss: 1.3946 -
accuracy: 0.6015 - val_loss: 1.6670 - val_accuracy: 0.5639

Epoch 00039: val_loss did not improve from 1.10886
Epoch 40/100
625/625 [==============================] - 62s 99ms/step - loss: 1.3813 -
accuracy: 0.6054 - val_loss: 1.3906 - val_accuracy: 0.5670

Epoch 00040: val_loss did not improve from 1.10886
Epoch 41/100
625/625 [==============================] - 62s 99ms/step - loss: 1.3466 -
accuracy: 0.6141 - val_loss: 1.4473 - val_accuracy: 0.5694

Epoch 00041: val_loss did not improve from 1.10886
Epoch 42/100
625/625 [==============================] - 62s 99ms/step - loss: 1.3383 -
accuracy: 0.6165 - val_loss: 1.6631 - val_accuracy: 0.5752

Epoch 00042: val_loss did not improve from 1.10886
Epoch 43/100
625/625 [==============================] - 62s 99ms/step - loss: 1.3118 -
accuracy: 0.6230 - val_loss: 1.5356 - val_accuracy: 0.5772

Epoch 00043: val_loss did not improve from 1.10886
Epoch 44/100
```

```
625/625 [==============================] - 62s 99ms/step - loss: 1.2944 -
accuracy: 0.6269 - val_loss: 1.3946 - val_accuracy: 0.5725

Epoch 00044: val_loss did not improve from 1.10886
Epoch 45/100
625/625 [==============================] - 62s 99ms/step - loss: 1.2787 -
accuracy: 0.6318 - val_loss: 1.5505 - val_accuracy: 0.5825

Epoch 00045: val_loss did not improve from 1.10886
Epoch 46/100
625/625 [==============================] - 62s 100ms/step - loss: 1.2550
- accuracy: 0.6365 - val_loss: 1.4359 - val_accuracy: 0.5995

Epoch 00046: val_loss did not improve from 1.10886
Epoch 47/100
625/625 [==============================] - 62s 100ms/step - loss: 1.2293
- accuracy: 0.6452 - val_loss: 1.7633 - val_accuracy: 0.5886

Epoch 00047: val_loss did not improve from 1.10886
Epoch 48/100
625/625 [==============================] - 62s 99ms/step - loss: 1.2153 -
accuracy: 0.6458 - val_loss: 1.4732 - val_accuracy: 0.5706

Epoch 00048: val_loss did not improve from 1.10886
Epoch 49/100
625/625 [==============================] - 62s 99ms/step - loss: 1.1946 -
accuracy: 0.6513 - val_loss: 1.4631 - val_accuracy: 0.5848

Epoch 00049: val_loss did not improve from 1.10886
Epoch 00049: early stopping
```

Visualizing the Loss and Accuracy

In [29]:

```
#plot to visualize the loss and accuracy against number of epochs
plt.figure(figsize=(18,8))

plt.suptitle('Loss and Accuracy Plots', fontsize=18)

plt.subplot(1,2,1)
plt.plot(model_history.history['loss'], label='Training Loss')
plt.plot(model_history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.xlabel('Number of epochs', fontsize=15)
plt.ylabel('Loss', fontsize=15)

plt.subplot(1,2,2)
plt.plot(model_history.history['accuracy'], label='Train Accuracy')
plt.plot(model_history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.xlabel('Number of epochs', fontsize=14)
plt.ylabel('Accuracy', fontsize=14)
plt.show()
```



Model Evaluation

In [57]:

```
#train_loss, train_accuracy = model.evaluate_generator(generator=train_data_gen, step
s=40000//batch_size)
valid_loss, valid_accuracy = model.evaluate_generator(generator=valid_data_gen, steps
=10000//batch_size)
test_loss, test_accuracy = model.evaluate_generator(data_gen.flow(X_test, y_test, see
d=123), steps=len(X_test)//batch_size)

print('Validation Accuracy: ', round((valid_accuracy * 100), 2), "%")
print('Test Accuracy: ', round((test_accuracy * 100), 2), "%")
print(" ")
print('Validation Loss: ', round(valid_loss, 2))
print('Test Loss: ', round(test_loss, 2))
```

```
Validation Accuracy:  58.81 %
Test Accuracy:  59.17 %

Validation Loss:  1.48
Test Loss:  1.47
```

Confusion Matrix

In [31]:

```
y_pred = model.predict(X_test)

cm = confusion_matrix(np.argmax(y_test, axis=1), np.argmax(y_pred, axis=1))
print(cm)
```

```
[[64  1  0 ...  0  0  0]
 [ 0 58  0 ...  1  0  1]
 [ 0  0 22 ...  1 25  0]
 ...
 [ 0  0  0 ... 47  0  0]
 [ 0  0  2 ...  0 50  2]
 [ 0  0  0 ...  0  0 63]]
```

In [32]:

```
#report to see which category has been predicted incorectly and which has been predic
ted correctly
target = ["Category {}".format(i) for i in range(num_class)]
print(classification_report(np.argmax(y_test, axis=1), np.argmax(y_pred, axis=1), tar
get_names=target))
```

	precision	recall	f1-score	support
Category 0	0.96	0.64	0.77	100
Category 1	0.78	0.58	0.67	100
Category 2	0.45	0.22	0.30	100
Category 3	0.50	0.09	0.15	100
Category 4	0.39	0.17	0.24	100
Category 5	0.32	0.35	0.33	100
Category 6	0.66	0.43	0.52	100
Category 7	0.45	0.32	0.37	100
Category 8	0.53	0.52	0.52	100
Category 9	0.73	0.60	0.66	100
Category 10	0.42	0.22	0.29	100
Category 11	0.29	0.22	0.25	100
Category 12	0.72	0.41	0.52	100
Category 13	0.66	0.31	0.42	100
Category 14	0.64	0.38	0.48	100
Category 15	0.35	0.50	0.41	100
Category 16	0.57	0.42	0.48	100
Category 17	0.56	0.68	0.61	100
Category 18	0.56	0.31	0.40	100
Category 19	0.59	0.26	0.36	100
Category 20	0.30	0.90	0.45	100
Category 21	0.96	0.24	0.38	100
Category 22	0.14	0.62	0.23	100
Category 23	0.24	0.52	0.32	100
Category 24	0.38	0.80	0.51	100
Category 25	0.37	0.35	0.36	100
Category 26	0.25	0.40	0.31	100
Category 27	0.29	0.11	0.16	100
Category 28	0.72	0.50	0.59	100
Category 29	0.28	0.52	0.37	100
Category 30	0.49	0.40	0.44	100
Category 31	0.60	0.37	0.46	100
Category 32	0.46	0.28	0.35	100
Category 33	0.71	0.25	0.37	100
Category 34	0.35	0.15	0.21	100
Category 35	0.24	0.36	0.29	100
Category 36	0.38	0.39	0.38	100
Category 37	0.34	0.63	0.44	100
Category 38	0.19	0.33	0.24	100
Category 39	0.09	0.88	0.17	100
Category 40	0.48	0.37	0.42	100
Category 41	0.57	0.69	0.63	100
Category 42	0.35	0.23	0.28	100
Category 43	0.29	0.47	0.36	100
Category 44	0.19	0.05	0.08	100
Category 45	0.38	0.32	0.35	100
Category 46	0.67	0.20	0.31	100
Category 47	0.67	0.42	0.52	100
Category 48	0.85	0.62	0.72	100
Category 49	0.53	0.46	0.49	100
Category 50	0.23	0.21	0.22	100
Category 51	0.66	0.29	0.40	100
Category 52	0.61	0.55	0.58	100
Category 53	0.76	0.83	0.79	100
Category 54	0.84	0.47	0.60	100
Category 55	0.18	0.03	0.05	100

```
 Category 56       0.81      0.51      0.63       100
 Category 57       0.56      0.57      0.57       100
 Category 58       0.86      0.42      0.56       100
 Category 59       0.64      0.39      0.48       100
 Category 60       0.61      0.74      0.67       100
 Category 61       0.26      0.36      0.30       100
 Category 62       0.48      0.63      0.55       100
 Category 63       0.55      0.24      0.33       100
 Category 64       0.75      0.06      0.11       100
 Category 65       0.54      0.13      0.21       100
 Category 66       0.48      0.25      0.33       100
 Category 67       0.47      0.17      0.25       100
 Category 68       0.88      0.63      0.73       100
 Category 69       0.35      0.64      0.45       100
 Category 70       0.49      0.69      0.58       100
 Category 71       0.79      0.55      0.65       100
 Category 72       0.29      0.15      0.20       100
 Category 73       0.46      0.06      0.11       100
 Category 74       0.40      0.18      0.25       100
 Category 75       0.79      0.63      0.70       100
 Category 76       0.97      0.33      0.49       100
 Category 77       0.67      0.12      0.20       100
 Category 78       0.16      0.16      0.16       100
 Category 79       0.61      0.19      0.29       100
 Category 80       0.14      0.02      0.04       100
 Category 81       0.63      0.61      0.62       100
 Category 82       0.96      0.65      0.77       100
 Category 83       0.59      0.44      0.51       100
 Category 84       0.53      0.34      0.41       100
 Category 85       0.53      0.69      0.60       100
 Category 86       0.16      0.73      0.26       100
 Category 87       0.73      0.43      0.54       100
 Category 88       0.46      0.24      0.32       100
 Category 89       0.69      0.36      0.47       100
 Category 90       0.93      0.38      0.54       100
 Category 91       0.48      0.61      0.54       100
 Category 92       0.68      0.26      0.38       100
 Category 93       0.62      0.10      0.17       100
 Category 94       0.93      0.76      0.84       100
 Category 95       0.55      0.38      0.45       100
 Category 96       0.38      0.38      0.38       100
 Category 97       0.23      0.47      0.31       100
 Category 98       0.22      0.50      0.31       100
 Category 99       0.36      0.63      0.46       100

    accuracy                           0.41     10000
   macro avg       0.52      0.41      0.41     10000
weighted avg       0.52      0.41      0.41     10000
```

Visualizing the Predictions

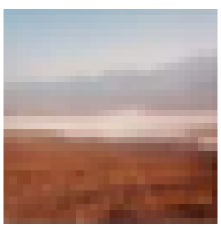
In [33]:

```
#dataframe of predictions
prediction = np.argmax(y_pred, axis=1)
prediction = pd.DataFrame(prediction)
#prediction
```

In [34]:

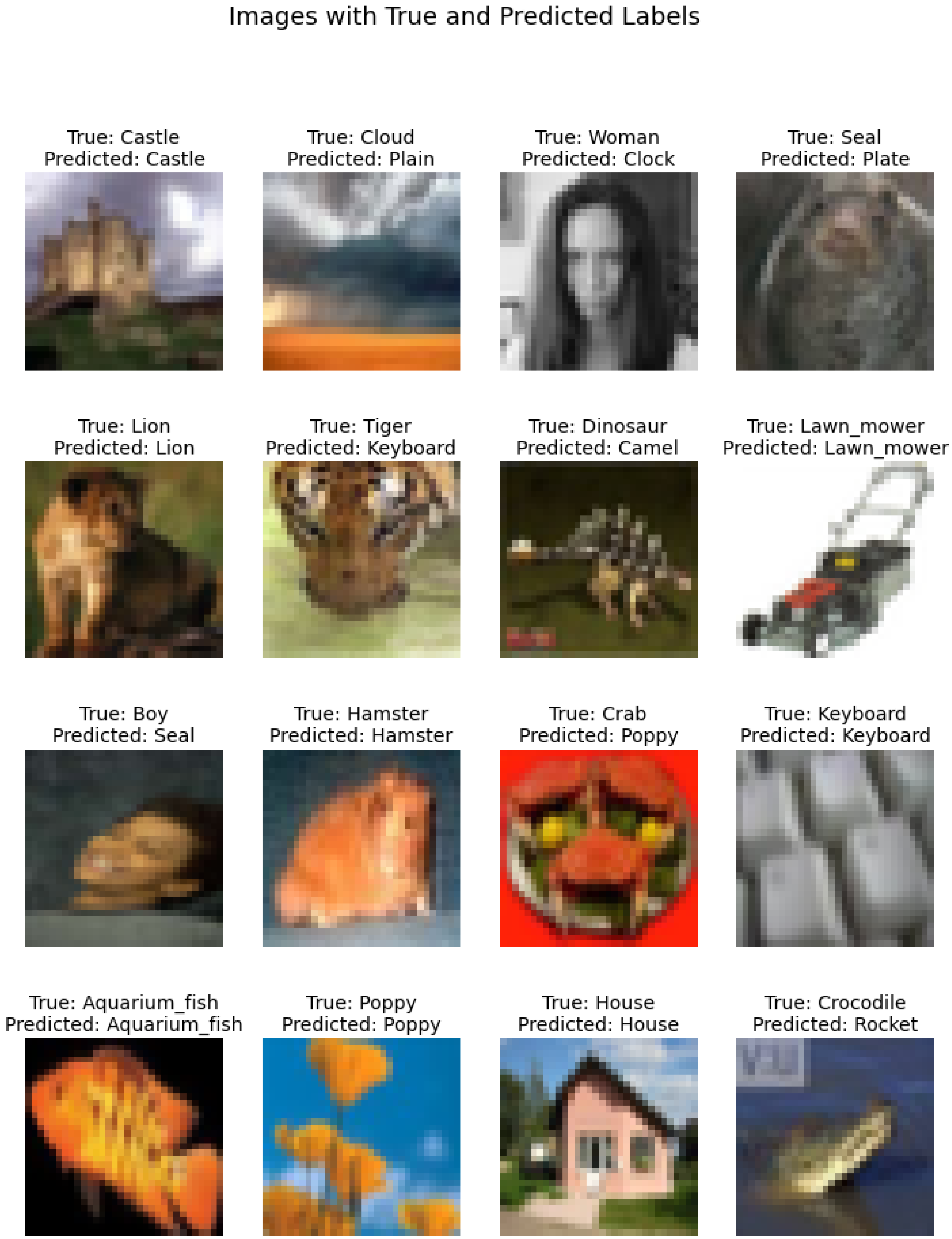
```
#generating a random number to display a random image from the dataset along with the
true and predicted label
imageId = np.random.randint(0, len(X_test))

rcParams['figure.figsize'] = 2,2

plt.imshow(X_test[imageId])

plt.axis('off')

print("True Label: " + str(subCategory.iloc[testData['fine_labels'][imageId]][0].capi
talize()))
print("Predicted Label: " + str(subCategory.iloc[prediction.iloc[imageId]]).split()[2
].capitalize())
```

```
True Label: Plain
Predicted Label: Cloud
```



In [35]:

```
#16 random images to display at a time along with their true and random labels
rcParams['figure.figsize'] = 12,15

num_row = 4
num_col = 4

imageId = np.random.randint(0, len(X_test), num_row * num_col)

fig, axes = plt.subplots(num_row, num_col)

for i in range(0, num_row):
    for j in range(0, num_col):
        k = (i*num_col)+j
        axes[i,j].imshow(X_test[imageId[k]])
        axes[i,j].set_title("True: " + str(subCategory.iloc[testData['fine_labels'][i
mageId[k]]][0]).capitalize()
                             + "\nPredicted: " + str(subCategory.iloc[prediction.iloc
[imageId[k]]]).split()[2].capitalize(),
                            fontsize=14)
        axes[i,j].axis('off')
        fig.suptitle("Images with True and Predicted Labels", fontsize=18)

plt.show()
```



Testing the Model

In [36]:

```
#function to resize the image
def resize_test_image(test_img):

    img = cv2.imread(test_img)
    #plt.imshow(img)
    img_RGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    #plt.imshow(img_RGB)
    resized_img = cv2.resize(img_RGB, (32, 32))
    #plt.imshow(resized_img)
    resized_img = resized_img / 255.
    #plt.imshow(resized_img)
    return resized_img

#resize_test_image('orange.jpeg')
```

In [37]:

```
#function to get prediction for test image from the model
def predict_test_image(test_img):

    resized_img = resize_test_image(test_img)
    prediction = model.predict(np.array([resized_img]))

    return prediction

#predict_test_image('orange.jpeg')
```

In [38]:

```
#function to get the sorted prediction
def sort_prediction_test_image(test_img):

    prediction = predict_test_image(test_img)

    index = np.arange(0,100)

    for i in range(100):
        for j in range(100):
            if prediction[0][index[i]] > prediction[0][index[j]]:
                temp = index[i]
                index[i] = index[j]
                index[j] = temp

    return index

#sort_prediction_test_image('orange.jpeg')
```

In [39]:

```
#function to get the dataframe for top 5 predictions
def df_top5_prediction_test_image(test_img):

    sorted_index = sort_prediction_test_image(test_img)
    prediction = predict_test_image(test_img)

    subCategory_name = []
    prediction_score = []

    k = sorted_index[:6]

    for i in range(len(k)):
        subCategory_name.append(subCategory.iloc[k[i]][0])
        prediction_score.append(round(prediction[0][k[i]], 2))

    df = pd.DataFrame(list(zip(subCategory_name, prediction_score)), columns=['Label'
, 'Probability'])

    return df

df_top5_prediction_test_image('orange.jpeg')
```

Out[39]:

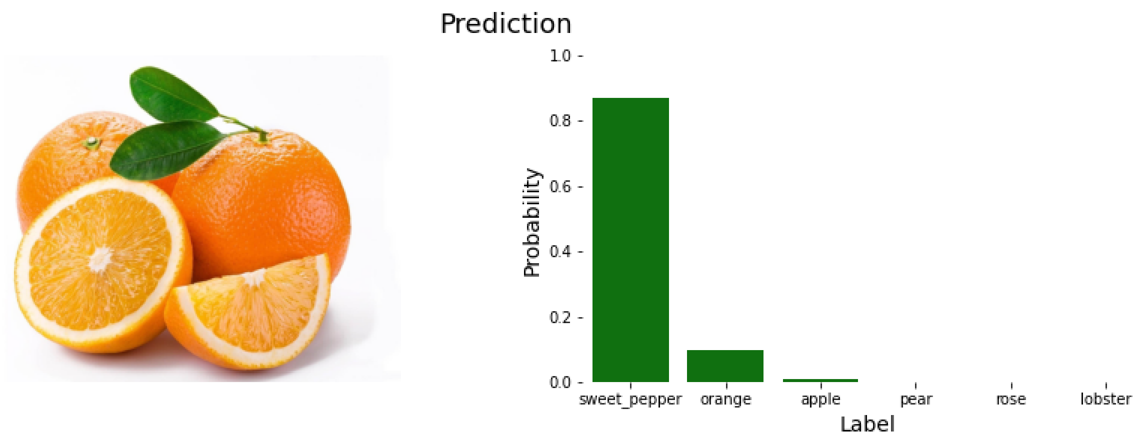
	Label	Probability
0	sweet_pepper	0.87
1	orange	0.10
2	apple	0.01
3	pear	0.00
4	rose	0.00
5	lobster	0.00

In [40]:

```
#function to get the plot for top 5 predictions
def plot_top5_prediction_test_image(test_img):

    fig, axes = plt.subplots(1, 2, figsize=(15,4))
    fig.suptitle("Prediction", fontsize=18)

    new_img = plt.imread(test_img)
    axes[0].imshow(new_img)
    axes[0].axis('off')

    data = df_top5_prediction_test_image(test_img)
    x=df_top5_prediction_test_image(test_img)['Label']
    y=df_top5_prediction_test_image(test_img)['Probability']

    axes[1] = sns.barplot(x=x, y=y, data=data, color="green")

    plt.xlabel('Label', fontsize=14)
    plt.ylabel('Probability', fontsize=14)

    plt.ylim(0,1.0)

    axes[1].grid(False)
    axes[1].spines["top"].set_visible(False)
    axes[1].spines["right"].set_visible(False)
    axes[1].spines["bottom"].set_visible(False)
    axes[1].spines["left"].set_visible(False)

    plt.show()
```

In [41]:

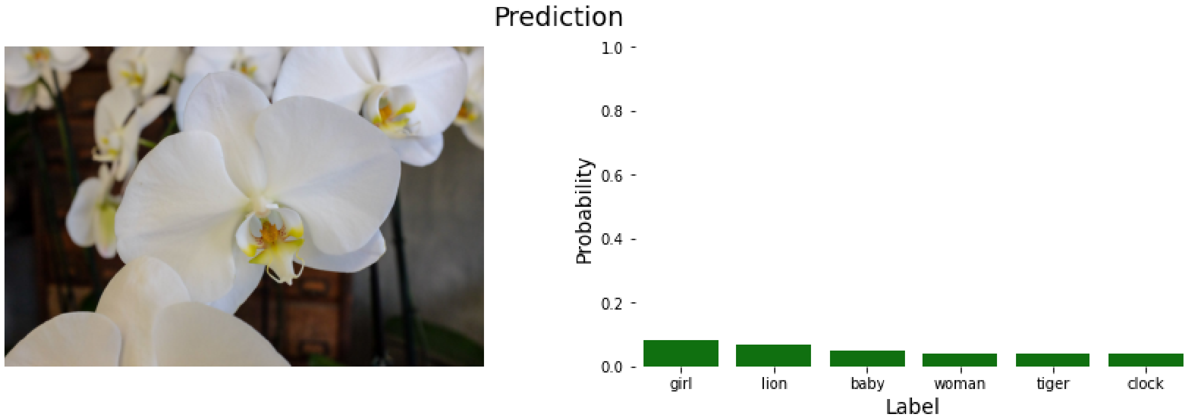
```
plot_top5_prediction_test_image('orange.jpeg')
```



The model predicted orange incorrectly.

In [42]:

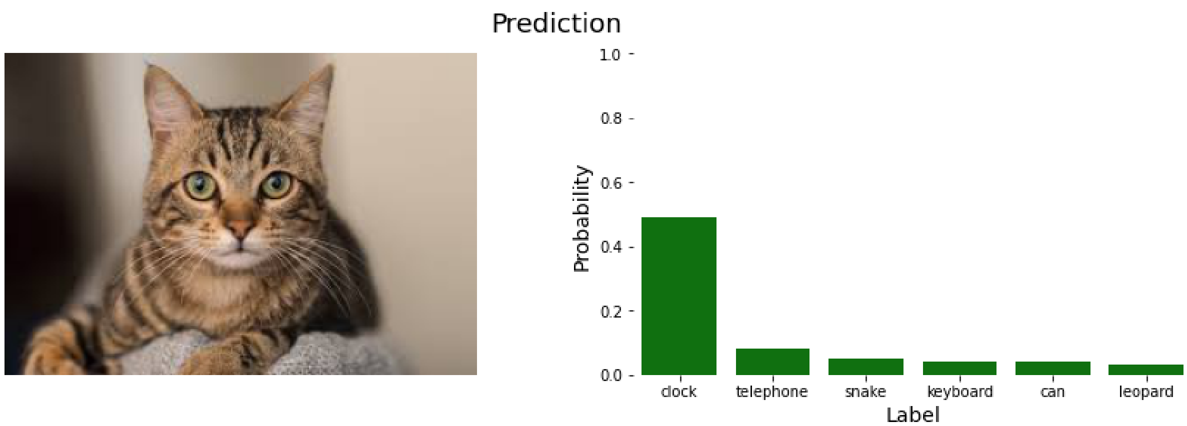
```
plot_top5_prediction_test_image('Orchid.jpg')
```



The model predicted orchid incorrectly.

In [43]:

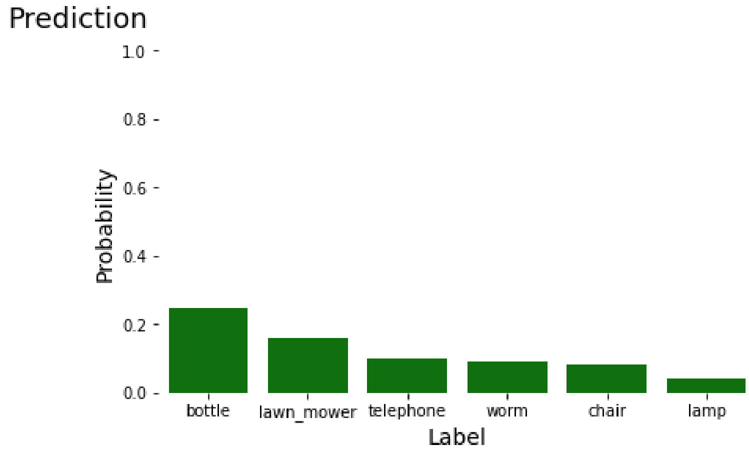
```
plot_top5_prediction_test_image('cat.jpeg')
```



The model predicted clock incorrectly.

In [44]:

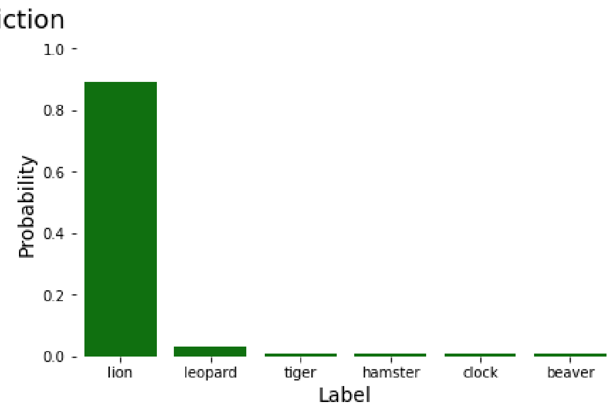
```
plot_top5_prediction_test_image('bottle.jpeg')
```



The model predicted bottle correctly.

In [45]:

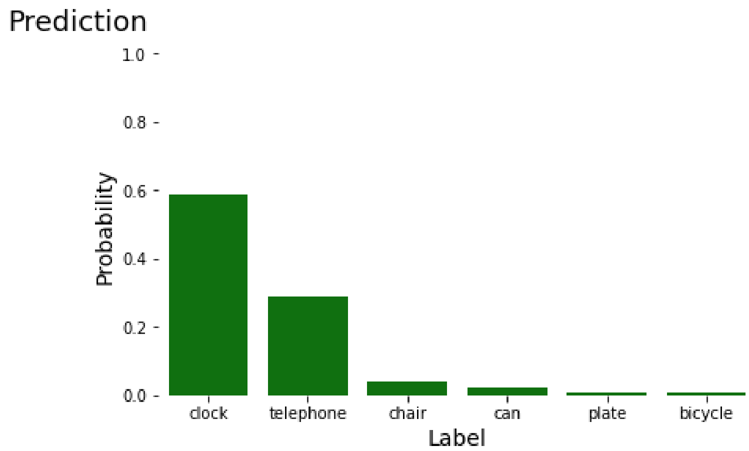
```
plot_top5_prediction_test_image('lion.jpg')
```



The model predicted lion correctly.

In [46]:

```
plot_top5_prediction_test_image('clock.jpg')
```



The model predicted clock incorrectly.

In [47]:

```
#saving the trained model as data file in .h5 format
model.save('model10.h5')
```

In [48]:

```
#storing the data file to Google Cloud Storage
from tensorflow.python.lib.io import file_io
with file_io.FileIO('model10.h5', mode='rb') as input_file:
    with file_io.FileIO('model10.h5', mode='w') as output_file:
        output_file.write(input_file.read())
        print("Model has been successfully stored to Google Cloud Storage.")
```

```
Model has been successfully stored to Google Cloud Storage.
```