Import libraries

In [1]:

```
import os
import pandas as pd
import numpy as np
import matplotlib as mpl
%matplotlib inline
import matplotlib.pyplot as plt
from IPython.display import display
import re

# Unused so far
import pickle
from pathlib import Path
from skimage import io

# import Image from PIL
from PIL import Image

from skimage.feature import hog
from skimage.color import rgb2gray

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# import train_test_split from sklearn's model selection module
from sklearn.model_selection import train_test_split

# import SVC from sklearn's svm module
from sklearn.svm import SVC

# import accuracy_score from sklearn's metrics module
from sklearn.metrics import roc_curve, auc, accuracy_score, classification_report, confusion_matrix

# Deep learning libraries
import tensorflow as tf
# import keras library
import keras

# import Sequential from the keras models module
from keras.models import Sequential

# import Dense, Dropout, Flatten, Conv2D, MaxPooling2D from the keras layers module
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D, ZeroPadding2D
```

Navigate to Project Folder

In [25]:

```
from google.colab import drive
drive.mount('/content/drive')
%cd 'drive/Shareddrives/ML PROJECT'
%ls
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
[Errno 2] No such file or directory: 'drive/Shareddrives/ML PROJECT'
/content/drive/Shareddrives/ML PROJECT
 Augmented_data/              Models/
 Bee_SVM.ipynb                Nehal_ML.ipynb
 dataset_image_processing/   'Netflix Recommender System.pdf'
'datasets (1).zip'            Notebooks/
 FinalProject_ML.ipynb        Original_data/
 logs/                        Original_Large-Data/
 ML_Project_Proposal.pdf      ReportImages/
 models/
```

In [36]:

```
# Function for natural sort to make things chronological
# https://stackoverflow.com/questions/4836710/is-there-a-built-in-function-for-string-natural-sort
def natural_sort(l):
    convert = lambda text: int(text) if text.isdigit() else text.lower()
    alphanum_key = lambda key: [convert(c) for c in re.split('([0-9]+)', key)]
    return sorted(l, key=alphanum_key)

## The original image files we use (1654 images)
img_files = natural_sort(os.listdir('Original_data/dataset_alternate/dataset_1/'))
print('# of original images: ', len(img_files))
labels_ = pd.read_csv('Original_data/dataset_alternate/labels_1.csv').sort_values(by='id')
labels_ = labels_.astype(int)
labels = labels_['genus'].values
# Show the output of the natural sort so the plots will be chronological
# file_list2 = [f for f in listdir() if isfile(join(f)) if '.csv' in f]
## Augmented images
# bw_images = natural_sort(os.listdir('Augmented_data/bw/'))
# rcz_images = natural_sort(os.listdir('Augmented_data/rcz/'))
# rcz_color = natural_sort(os.listdir('Augmented_data/rcz_color/'))

## The high resolution images
hr_images = natural_sort(os.listdir('Original_Large-Data/images/'))
labels_high = pd.read_csv('Original_Large-Data/images_labels.csv').sort_values(by='id')
labels_high_ = labels_high['genus'].values.astype(int)
print(labels_high[0:10])
print(hr_images[0:10])
print(labels_high_[0:10])
print(len(labels_high_))
print(len(hr_images))
```

```
# of original images:  1654
      id  genus
2680   1      1
1056   2      1
2744   3      1
2233   4      0
2644   5      0
1344   6      1
3568   8      1
1533   9      1
1974  12      0
2979  14      1
['1.jpg', '2.jpg', '3.jpg', '4.jpg', '5.jpg', '6.jpg', '8.jpg', '9.jpg', '12.jpg', '14.jpg']
[1 1 1 0 0 1 1 1 0 1]
3968
3968
```

Making the Pipeline to create black and white images

In [4]:

```
# image_paths = ['Original_data/dataset_alternate/dataset_1/' + im for im in img_files2]

def image_processing(path):
    img = Image.open(path)

    # create the paths to save files to
    bw_path = "Augmented_data/bw/bw_{}.jpg".format(path.stem)
    rcz_path = "Augmented_data/rcz_color/rcz_{}.jpg".format(path.stem)

    # print("Creating grayscale version of {} and saving to {}.".format(path, bw_path))
    # bw = img.convert("L").resize((100, 100))
    # bw.save(bw_path)

    # print("Creating rotated, cropped, and zoomed version of {} and saving to {}.".format(path, rcz_path))
    rcz = img.rotate(180).crop([10, 10, 40, 40]).resize((50, 50))
    rcz.save(rcz_path)

# # # for loop over the image paths
# for img_path in image_paths:
#     image_processing(Path(img_path))
```

Deep Learning Approach to Solving this Problem

Import Bee Images, Train, test, split

In [37]:

```
# 4. Importing the image data
# create empty list
image_list_bw = []
image_list_rcz = []
image_list_hr = []
image_list = []
bw = False
rcz = False
color = False
high_res = True
# for i in range(labels.shape[0]):
for i in range(labels_high_.shape[0]):
    # load image
    if bw == True:
        img_bw = io.imread('Augmented_data/bw/'+bw_images[i].format(i)).astype(np.uint8)
    if rcz == True and bw == True:
        print('Augmented_data/rcz/'+rcz_images[i])
        img_rcz = io.imread('Augmented_data/rcz/'+rcz_images[i].format(i)).astype(np.uint8)
    if color==True:
        img_rcz = io.imread('Augmented_data/rcz_color/'+rcz_color[i].format(i)).astype(np.uint8)
        img = io.imread('Original_data/dataset_alternate/dataset_1/'+img_files[i].format(i)).astype(np.uint8)
    if high_res == True:
        img_hi = io.imread('Original_Large-Data/images/'+hr_images[i].format(i))
    # append to list of all images
    if rcz == True and bw == True:
        image_list_rcz.append(img_rcz)
        image_list_bw.append(img_bw)
    if color == True:
        image_list_rcz.append(img_rcz)
        image_list.append(img)
    if high_res == True:
        image_list_hr.append(img_hi)





# convert image list to single array
if bw == False and rcz == False:
    X = np.array(image_list)
    # X = X_/255
    # assign the genus label values to y
    y = labels

# Only for augmentation
if bw == True and rcz == True:
    X = np.concatenate([image_list_bw, image_list_rcz])
    y = np.concatenate([labels,labels])
    # X = X/255
    X = np.expand_dims(X, axis=-1)
```

```python
# Only for augmentation
if color == True:
    X = np.concatenate([image_list, image_list_rcz])
    y = np.concatenate([labels,labels])
    # X = X/255

if high_res == True:
    X = np.array(image_list_hr)
    y = labels_high_
    # y = np.delete(y, 1597)
    # X = X/255
print("Full X shape", X.shape)

# print value counts for genus
print(np.unique(y, return_counts=True))

# 5 SPLITTING
# split out evaluation sets (x_eval and y_eval)
x_interim, x_eval, y_interim, y_eval = train_test_split(X,y,
                                                        test_size=0.2,
                                                        random_state=52)

# split remaining data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x_interim,
                                                    y_interim,
                                                    test_size=0.2,
                                                    random_state=52)

# examine number of samples in train, test, and validation sets
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
print(x_eval.shape[0], 'eval samples')
```

```
Full X shape (3968, 200, 200, 3)
(array([0, 1]), array([ 826, 3142]))
x_train shape: (2539, 200, 200, 3)
2539 train samples
635 test samples
794 eval samples
```

Standardize Images

In [7]:

```python
# 6 , Normalize the image data
# initialize standard scaler
# ss = StandardScaler()

# def scale_features(train_features, test_features):
#     for image in train_features:
#         # for each channel, apply standard scaler's fit_transform method
#         for channel in range(image.shape[2]):
#             image[:, :, channel] = ss.fit_transform(image[:, :, channel])
#     for image in test_features:
#         # for each channel, apply standard scaler's transform method
#         for channel in range(image.shape[2]):
#             image[:, :, channel] = ss.transform(image[:, :, channel])

# # apply scale_features to four sets of features
# scale_features(x_interim, x_eval)
# scale_features(x_train, x_test)
```

Show some Bees

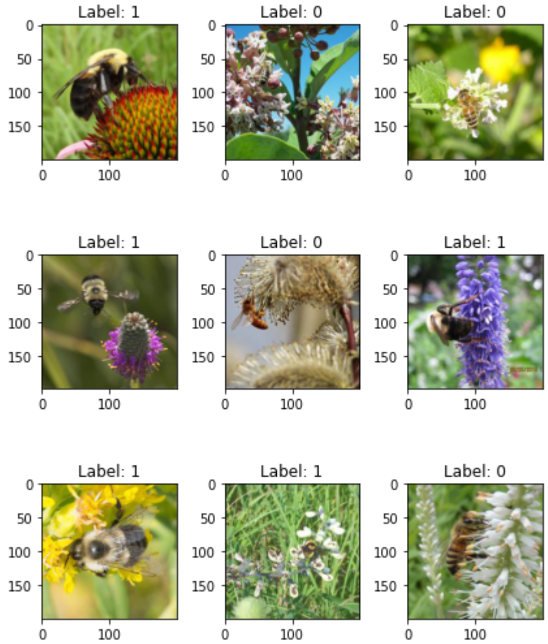
In [38]:

```python
# Visualize

num = 9  # number of images to visualize
images_ = x_train[:num]
labels_ = y_train[:num]

num_row = 3
num_col = 3

fig, axes = plt.subplots(num_row, num_col, figsize=(2*num_col,2.5*num_row))
for i in range(num):
    ax = axes[i//num_col, i%num_col]
    ax.imshow(images_[i][:,:,:], cmap='gray')
    ax.set_title('Label: {}'.format(labels_[i]))
plt.tight_layout()
plt.show()
```



Transfer learning

Although I was not able to find the architecture that yielded high accuracy, with the transfer learning approach...

In [39]:
```
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions

ResNet50_model = ResNet50(weights='imagenet')
```

In [11]:
```
# ResNet50_model.summary()
```

In [40]:
```
'''
Feature Extraction is performed by ResNet50 pretrained on imagenet weights.
Input size is 224 x 224.
'''
def feature_extractor(inputs):
#   input_tensor = Input(shape=(IMG_SIZE,IMG_SIZE,1) )
#   input1 = Conv2D(3,(1,1),padding='same')(inputs)

  input2 = preprocess_input(inputs)
  feature_extractor = tf.keras.applications.resnet.ResNet50(input_shape=(200, 200, 3),
                                               include_top=False,
                                               weights='imagenet')(input2)
  return feature_extractor


'''
Defines final dense layers and subsequent softmax layer for classification.
'''
def classifier(inputs):
    x = tf.keras.layers.GlobalAveragePooling2D()(inputs)
    x = tf.keras.layers.Flatten()(x)
    # x = tf.keras.layers.Dense(128, activation="relu")(x)
    # x = tf.keras.layers.Dense(64, activation="relu")(x)
    # x = tf.keras.layers.Dropout(0.25)(x)
    x = tf.keras.layers.Dense(1, activation = 'sigmoid', name="classification")(x)
    return x

'''
Since input image size is (50 x 50), first upsample the image by factor of (5x5) to transform it to (250 x 250)
Connect the feature extraction and "classifier" layers to build the model.
'''
def final_model(inputs):

    resize = tf.keras.layers.UpSampling2D(size=(1,1))(inputs)
    # resize1 = Conv2D(3,(1,1),padding='same')(resize)

    resnet_feature_extractor = feature_extractor(resize)
    classification_output = classifier(resnet_feature_extractor)

    return classification_output

'''
Define the model and compile it.
Use Stochastic Gradient Descent as the optimizer.
Use Sparse Categorical CrossEntropy as the loss function.
'''
def define_compile_model():

  inputs = tf.keras.layers.Input(shape=(200,200,3))
```

```python
    classification_output = final_model(inputs)
    model = tf.keras.Model(inputs=inputs, outputs = classification_output)
    opt = tf.keras.optimizers.Adamax(learning_rate=0.001)
    model.compile(optimizer=opt,
                  loss=tf.keras.losses.BinaryCrossentropy(from_logits=False),
                  metrics = [tf.keras.metrics.BinaryAccuracy()])

    return model


model = define_compile_model()

model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_2 (InputLayer)        [(None, 200, 200, 3)]     0         
                                                                 
 up_sampling2d (UpSampling2D  (None, 200, 200, 3)      0         
 )                                                               
                                                                 
 tf.__operators__.getitem (S  (None, 200, 200, 3)      0         
 licingOpLambda)                                                 
                                                                 
 tf.nn.bias_add (TFOpLambda)  (None, 200, 200, 3)      0         
                                                                 
 resnet50 (Functional)       (None, 7, 7, 2048)        23587712  
                                                                 
 global_average_pooling2d (G  (None, 2048)             0         
 lobalAveragePooling2D)                                          
                                                                 
 flatten (Flatten)           (None, 2048)              0         
                                                                 
 classification (Dense)      (None, 1)                 2049      
                                                                 
=================================================================
Total params: 23,589,761
Trainable params: 23,536,641
Non-trainable params: 53,120
_________________________________________________________________
```

In [41]:

```python
EPOCHS = 30
history = model.fit(x_train, y_train, epochs=EPOCHS, validation_data = (x_test, y_test), batch_size=64)
```

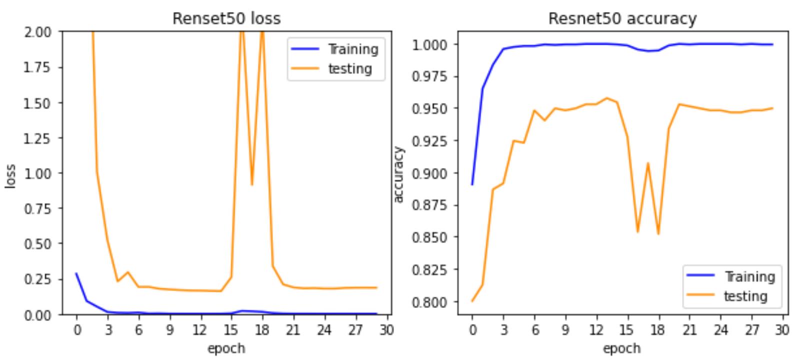
```
Epoch 1/30
40/40 [==============================] - 22s 356ms/step - loss: 0.2832 - binary_accuracy: 0.8905 - val_loss: 569.1245 - val_binary_accuracy: 0.8000
Epoch 2/30
40/40 [==============================] - 12s 308ms/step - loss: 0.0910 - binary_accuracy: 0.9649 - val_loss: 3.5437 - val_binary_accuracy: 0.8126
Epoch 3/30
40/40 [==============================] - 12s 308ms/step - loss: 0.0509 - binary_accuracy: 0.9835 - val_loss: 1.0049 - val_binary_accuracy: 0.8866
Epoch 4/30
40/40 [==============================] - 12s 308ms/step - loss: 0.0133 - binary_accuracy: 0.9957 - val_loss: 0.5238 - val_binary_accuracy: 0.8913
Epoch 5/30
40/40 [==============================] - 12s 308ms/step - loss: 0.0075 - binary_accuracy: 0.9972 - val_loss: 0.2290 - val_binary_accuracy: 0.9244
Epoch 6/30
40/40 [==============================] - 12s 308ms/step - loss: 0.0061 - binary_accuracy: 0.9980 - val_loss: 0.2944 - val_binary_accuracy: 0.9228
Epoch 7/30
40/40 [==============================] - 12s 308ms/step - loss: 0.0091 - binary_accuracy: 0.9980 - val_loss: 0.1904 - val_binary_accuracy: 0.9480
Epoch 8/30
40/40 [==============================] - 12s 308ms/step - loss: 0.0019 - binary_accuracy: 0.9992 - val_loss: 0.1905 - val_binary_accuracy: 0.9402
Epoch 9/30
40/40 [==============================] - 12s 308ms/step - loss: 0.0029 - binary_accuracy: 0.9988 - val_loss: 0.1777 - val_binary_accuracy: 0.9496
Epoch 10/30
40/40 [==============================] - 12s 308ms/step - loss: 0.0010 - binary_accuracy: 0.9992 - val_loss: 0.1724 - val_binary_accuracy: 0.9480
Epoch 11/30
40/40 [==============================] - 12s 308ms/step - loss: 8.2699e-04 - binary_accuracy: 0.9992 - val_loss: 0.1682 - val_binary_accuracy: 0.9496
Epoch 12/30
40/40 [==============================] - 12s 308ms/step - loss: 7.3923e-04 - binary_accuracy: 0.9996 - val_loss: 0.1652 - val_binary_accuracy: 0.9528
Epoch 13/30
40/40 [==============================] - 12s 308ms/step - loss: 7.3091e-04 - binary_accuracy: 0.9996 - val_loss: 0.1643 - val_binary_accuracy: 0.9528
Epoch 14/30
40/40 [==============================] - 12s 308ms/step - loss: 7.2133e-04 - binary_accuracy: 0.9996 - val_loss: 0.1624 - val_binary_accuracy: 0.9575
Epoch 15/30
40/40 [==============================] - 12s 308ms/step - loss: 9.1976e-04 - binary_accuracy: 0.9992 - val_loss: 0.1606 - val_binary_accuracy: 0.9543
Epoch 16/30
40/40 [==============================] - 12s 308ms/step - loss: 0.0031 - binary_accuracy: 0.9984 - val_loss: 0.2599 - val_binary_accuracy: 0.9276
Epoch 17/30
40/40 [==============================] - 12s 308ms/step - loss: 0.0205 - binary_accuracy: 0.9953 - val_loss: 2.1856 - val_binary_accuracy: 0.8535
Epoch 18/30
40/40 [==============================] - 12s 308ms/step - loss: 0.0179 - binary_accuracy: 0.9941 - val_loss: 0.9119 - val_binary_accuracy: 0.9071
Epoch 19/30
40/40 [==============================] - 12s 308ms/step - loss: 0.0139 - binary_accuracy: 0.9945 - val_loss: 2.1092 - val_binary_accuracy: 0.8520
Epoch 20/30
40/40 [==============================] - 12s 308ms/step - loss: 0.0054 - binary_accuracy: 0.9984 - val_loss: 0.3367 - val_binary_accuracy: 0.9339
Epoch 21/30
40/40 [==============================] - 12s 308ms/step - loss: 0.0019 - binary_accuracy: 0.9996 - val_loss: 0.2081 - val_binary_accuracy: 0.9528
Epoch 22/30
40/40 [==============================] - 12s 308ms/step - loss: 9.5905e-04 - binary_accuracy: 0.9992 - val_loss: 0.1868 - val_binary_accuracy: 0.9512
Epoch 23/30
40/40 [==============================] - 12s 308ms/step - loss: 9.7186e-04 - binary_accuracy: 0.9996 - val_loss: 0.1804 - val_binary_accuracy: 0.9496
Epoch 24/30
40/40 [==============================] - 12s 308ms/step - loss: 8.2602e-04 - binary_accuracy: 0.9996 - val_loss: 0.1818 - val_binary_accuracy: 0.9480
Epoch 25/30
40/40 [==============================] - 12s 308ms/step - loss: 6.4320e-04 - binary_accuracy: 0.9996 - val_loss: 0.1786 - val_binary_accuracy: 0.9480
Epoch 26/30
40/40 [==============================] - 12s 308ms/step - loss: 6.9051e-04 - binary_accuracy: 0.9996 - val_loss: 0.1787 - val_binary_accuracy: 0.9465
Epoch 27/30
```

```
40/40 [==============================] - 12s 308ms/step - loss: 7.1058e-04 - binary_accuracy: 0.9992 - val_loss: 0.1835 - val_binary_accuracy: 0.9465
Epoch 28/30
40/40 [==============================] - 12s 308ms/step - loss: 6.5708e-04 - binary_accuracy: 0.9996 - val_loss: 0.1849 - val_binary_accuracy: 0.9480
Epoch 29/30
40/40 [==============================] - 12s 308ms/step - loss: 6.4219e-04 - binary_accuracy: 0.9992 - val_loss: 0.1851 - val_binary_accuracy: 0.9480
Epoch 30/30
40/40 [==============================] - 12s 308ms/step - loss: 6.4572e-04 - binary_accuracy: 0.9992 - val_loss: 0.1848 - val_binary_accuracy: 0.9496
```

In [44]:

```python
train_loss = history.history['loss']
train_acc = history.history['binary_accuracy']
test_loss = history.history['val_loss']
test_acc = history.history['val_binary_accuracy']

fig, ax = plt.subplots(1,2, figsize = (10,4))
ax[0].plot(train_loss, label = "Training", color='blue')
ax[1].plot(train_acc, label = "Training", color='blue')
ax[0].plot(test_loss, label = "testing", color='darkorange')
ax[1].plot(test_acc, label = "testing", c='darkorange')
ax[1].set_xticks(np.arange(0,31,3))
ax[0].set_xticks(np.arange(0,31,3))
# ax[0].set_xlim(0,35)
# ax[1].set_xlim(0,35)
ax[0].set_ylim(0,2)
ax[0].set_xlabel('epoch')
ax[1].set_xlabel('epoch')
ax[0].set_ylabel('loss')
ax[1].set_ylabel('accuracy')
ax[0].set_title('Renset50 loss')
ax[1].set_title('Resnet50 accuracy')
# plt.suptitle('Resnet50 pretrained model with additional trained layers to classify the bee images')
ax[0].legend()
ax[1].legend();
```



In [43]:

```python
# evaluate model on test set
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

print("")

# evaluate model on holdout set
eval_score = model.evaluate(x_eval, y_eval, verbose=0)
# print loss score
print('Eval loss:', eval_score[0])
# print accuracy score
print('Eval accuracy:', eval_score[1])
```

```
Test loss: 0.18482273817062378
Test accuracy: 0.9496062994003296

Eval loss: 0.25692009925842285
Eval accuracy: 0.9458438158035278
```