Import libraries

In [11]:
```
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

# 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
```

Loading and Processing of Naive Images

First step of this project is to load, transform and understand the images of the naive honey bees and naive bumble bees. These images are naive because, they are the images of the naive honey and bumble bees instead of the experienced bees.

More details on the naive and experienced bees can be found in the article below

Naive&ExperiencedBees

Navigate to Project Folder

In [2]:
```
# from google.colab import drive
# drive.mount('/content/drive')
# %cd 'drive/Shareddrives/ML PROJECT'
# %ls
%cd ..
%ls
```

```
G:\Shared drives\ML PROJECT\FP_Nehal_Anthony_Alex
 Volume in drive G is Google Drive
 Volume Serial Number is 1983-1116

 Directory of G:\Shared drives\ML PROJECT\FP_Nehal_Anthony_Alex

2022-04-26  03:33 PM    <DIR>          .
2022-04-26  03:26 PM    <DIR>          ..
2022-04-26  03:13 PM    <DIR>          EDA
2022-04-26  03:33 PM    <DIR>          Original_data
2022-04-26  03:35 PM    <DIR>          pseudo_resnet20
2022-04-26  03:31 PM                 0 Readme.txt
2022-04-26  03:14 PM    <DIR>          ResNet20
2022-04-26  03:18 PM    <DIR>          Resnet50
2022-04-26  03:30 PM    <DIR>          SVM
2022-04-26  03:18 PM    <DIR>          VanillaCNN
               1 File(s)              0 bytes
               9 Dir(s)  20,366,364,672 bytes free
```

In [7]:
```
# Function for natural sort to make things chronological
# https://stackoverflow.com/questions/4836710/is-there-a-built-in-function-for-string-natural-sort
import re
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 image files we use
img_files = os.listdir('Original_data/dataset_alternate/dataset_1/')
print('# of images: ', len(img_files))
```

```
labels_ = pd.read_csv('Original_data/dataset_alternate/labels_1.csv').sort_values(by='id')
labels_ = labels_.astype(int, copy=True, errors='raise')
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]
img_files2 = natural_sort(img_files)
```

```
# of images:  1654
```

Making the Pipeline to create black and white images

In [11]:

```
# 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/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 = bw.rotate(180).crop([20, 20, 80, 80]).resize((100, 100))
    rcz.save(rcz_path)

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

In [8]:

```
# Sort bw and rcz images after they are created above

# bw_images = natural_sort(os.listdir('Augmented_data/bw/'))
# rcz_images = natural_sort(os.listdir('Augmented_data/rcz/'))

# print(rcz_images[0:10])
# len(rcz_images)
```

Deep Learning Approach to Solving this Problem

Import Bee Images, Train, test, split

In [10]:

```
# 4. Importing the image data
# create empty list
image_list_bw = []
image_list_rcz = []
image_list = []
bw = False
rcz = False
for i in range(labels.shape[0]):
    # load image
    if bw == True and rcz==True:
        img_bw = io.imread('Augmented_data/bw/'+bw_images[i].format(i)).astype(np.float32)
        img_rcz = io.imread('Augmented_data/rcz/'+rcz_images[i].format(i)).astype(np.float32)
        image_list_bw.append(img_bw)
        image_list_rcz.append(img_rcz)
    else:
        img = io.imread('Original_data/dataset_alternate/dataset_1/'+img_files2[i].format(i)).astype(np.uint8)
        image_list.append(img)


if bw == 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)

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 (1654, 50, 50, 3)
(array([0, 1]), array([827, 827], dtype=int64))
x_train shape: (1058, 50, 50, 3)
1058 train samples
265 test samples
331 eval samples
```

Show some Bees

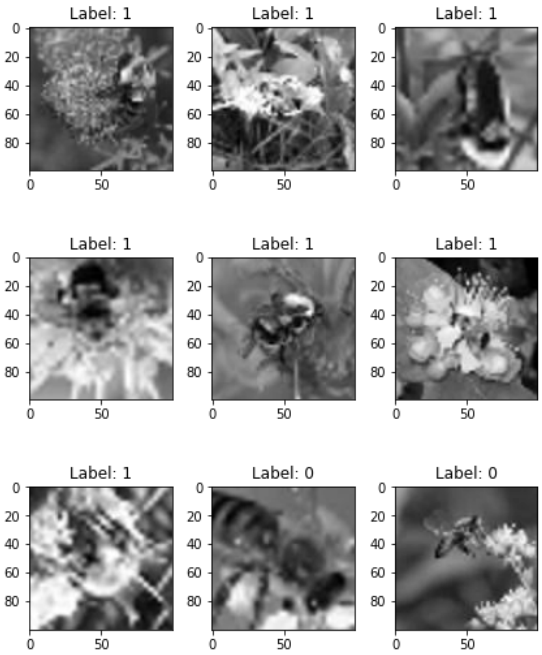
In [16]:

```
# 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][:,:,0], cmap='gray')
    ax.set_title('Label: {}'.format(labels_[i]))
plt.tight_layout()
plt.show()
```



Resnet-style *model*

1. Import TensorFlow to start working

In [17]:

```
import tensorflow as tf
print("TensorFlow version:", tf.__version__)
import keras
import numpy as np
from keras.layers import Input
from keras import layers
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Activation
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import GlobalMaxPooling2D
from keras.layers import ZeroPadding2D
from keras.layers import AveragePooling2D
from keras.layers import GlobalAveragePooling2D
from keras.layers import BatchNormalization
# from tensorflow.keras import Model
from keras.models import Model
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions
# Load the TensorBoard notebook extension
%load_ext tensorboard
import datetime, os
```

```
TensorFlow version: 2.8.0
```

In [18]:

```
# Define Model
f1, f2, f3 = [16, 32, 64]
kernel_ = (3,3)
class Bee_clf(Model):
    def __init__(self):
        super(Bee_clf, self).__init__()
        self.conv0 = Conv2D(3, (1,1), strides=(1, 1), padding='same', name='3_chan')
        self.preprocess = preprocess_input
        self.ZeroPadding0 = ZeroPadding2D((1, 1))
        self.conv1 = Conv2D(f1, kernel_, strides=(1, 1), name='conv1')
        self.bn1 = BatchNormalization(axis=-1)
        self.Activation1 = Activation('relu')
        self.ZeroPadding1 = ZeroPadding2D((1, 1))
        self.MaxPooling = AveragePooling2D()

        # filters f1
        self.shortcut1 = Conv2D(f1, (1,1), strides=(1, 1), padding='same', name='short1')
        self.bns1 = BatchNormalization(axis=1)
        self.Activations1 = Activation('relu')

        self.conv2 = Conv2D(f1, kernel_,strides=(1, 1), padding='same', name='conv2')
        self.bn2 = BatchNormalization(axis=-1)
        self.Activation2 = Activation('relu')
        self.conv3 = Conv2D(f1, kernel_,strides=(1, 1), padding='same', name='conv3')
        self.bn3 = BatchNormalization(axis=-1)
        self.Activation3 = Activation('relu')

        self.shortcut2 = Conv2D(f1, (1,1), strides=(1, 1), padding='same', name='short2')
        self.bns2 = BatchNormalization(axis=-1)
        self.Activations2 = Activation('relu')

        self.conv4 = Conv2D(f1, kernel_,strides=(1, 1), padding='same', name='conv4')
        self.bn4 = BatchNormalization(axis=-1)
        self.Activation4 = Activation('relu')
        self.conv5 = Conv2D(f1, kernel_,strides=(1, 1), padding='same', name='conv5')
        self.bn5 = BatchNormalization(axis=-1)
        self.Activation5 = Activation('relu')

        self.shortcut2a = Conv2D(f1, (1,1), strides=(1, 1), padding='same', name='short2a')
        self.bns2a = BatchNormalization(axis=-1)
        self.Activations2a = Activation('relu')

        self.conv4a = Conv2D(f1, kernel_,strides=(1, 1), padding='same', name='conv4a')
        self.bn4a = BatchNormalization(axis=-1)
        self.Activation4a = Activation('relu')
        self.conv5a = Conv2D(f1, kernel_,strides=(1, 1), padding='same', name='conv5a')
        self.bn5a = BatchNormalization(axis=-1)
        self.Activation5a = Activation('relu')

        # filters f2
        self.shortcut3 = Conv2D(f2, (1,1), strides=(1, 1), padding='same', name='short3')
        self.bns3 = BatchNormalization(axis=3)
        self.Activations3 = Activation('relu')

        self.conv6 = Conv2D(f2, kernel_,strides=(1, 1), padding='same', name='conv6')
        self.bn6 = BatchNormalization(axis=-1)
        self.Activation6 = Activation('relu')
        self.conv7 = Conv2D(f2, kernel_,strides=(1, 1), padding='same', name='conv7')
        self.bn7 = BatchNormalization(axis=3)
        self.Activation7 = Activation('relu')

        self.shortcut4 = Conv2D(f2, (1,1), strides=(1, 1), padding='same', name='short4')
        self.bns4 = BatchNormalization(axis=-1)
        self.Activations4 = Activation('relu')

        self.conv8 = Conv2D(f2, kernel_,strides=(1, 1), padding='same', name='conv8')
        self.bn8 = BatchNormalization(axis=-1)
        self.Activation8 = Activation('relu')
        self.conv9 = Conv2D(f2, kernel_,strides=(1, 1), padding='same', name='conv9')
        self.bn9 = BatchNormalization(axis=-1)
        self.Activation9 = Activation('relu')

        self.shortcut4a = Conv2D(f2, (1,1), strides=(1, 1), padding='same', name='short4a')
        self.bns4a = BatchNormalization(axis=-1)
        self.Activations4a = Activation('relu')

        self.conv8a = Conv2D(f2, kernel_,strides=(1, 1), padding='same', name='conv8a')
        self.bn8a = BatchNormalization(axis=-1)
        self.Activation8a = Activation('relu')
        self.conv9a = Conv2D(f2, kernel_,strides=(1, 1), padding='same', name='conv9a')
        self.bn9a = BatchNormalization(axis=-1)
        self.Activation9a = Activation('relu')

        # filters f3
        self.shortcut5 = Conv2D(f3, (1,1), strides=(1, 1), padding='same', name='short5')
        self.bns5 = BatchNormalization(axis=-1)
        self.Activations5 = Activation('relu')

        self.conv10 = Conv2D(f3, kernel_,strides=(1, 1), padding='same', name='conv10')
        self.bn10 = BatchNormalization(axis=-1)
        self.Activation10 = Activation('relu')
        self.conv11 = Conv2D(f3, kernel_,strides=(1, 1), padding='same', name='conv11')
        self.bn11 = BatchNormalization(axis=-1)
        self.Activation11 = Activation('relu')

        self.shortcut6 = Conv2D(f3, (1,1), strides=(1, 1), padding='same', name='short6')
        self.bns6 = BatchNormalization(axis=3)
        self.Activations6 = Activation('relu')
```

```
        self.conv12 = Conv2D(f3, kernel_,strides=(1, 1), padding='same', name='conv12')
        self.bn12 = BatchNormalization(axis=-1)
        self.Activation12 = Activation('relu')
        self.conv13= Conv2D(f3, kernel_,strides=(1, 1), padding='same', name='conv13')
        self.bn13 = BatchNormalization(axis=3)
        self.Activation13 = Activation('relu')

        self.shortcut7 = Conv2D(f3, (1,1), strides=(1, 1), padding='same', name='short7')
        self.bns7 = BatchNormalization(axis=-1)
        self.Activations7 = Activation('relu')

        self.conv12a = Conv2D(f3, kernel_,strides=(1, 1), padding='same', name='conv12a')
        self.bn12a = BatchNormalization(axis=-1)
        self.Activation12a = Activation('relu')
        self.conv13a= Conv2D(f3, kernel_,strides=(1, 1), padding='same', name='conv13a')
        self.bn13a = BatchNormalization(axis=-1)
        self.Activation13a = Activation('relu')

        # self.Pooling = MaxPooling2D((2,2), name='pool')
        self.Pooling = GlobalAveragePooling2D( name='pool')
        self.flatten = Flatten()
        self.d1 = Dense(32, activation='relu')
        self.drop = Dropout(0.25)
        self.d2 = Dense(1, activation='sigmoid')

    def call(self, x):
        # Beginning
        x = self.conv0(x)
        # x = tf.cast(x,'float32')
        x = self.preprocess(x)
        x = self.ZeroPadding0(x)
        x = self.conv1(x) #1
        x = self.bn1(x)
        x = self.Activation1(x)
        x = self.MaxPooling(x)
        x = self.ZeroPadding1(x)
        # store input tensor to the next layer
        shortcut1 = self.shortcut1(x)
        shortcut1 = self.bns1(shortcut1)
        # shortcut1 = self.Activations1(shortcut1)

        # Filters F1
        x = self.conv2(x) #1
        x = self.bn2(x)
        x = self.Activation2(x)
        x = self.conv3(x) #2
        x = self.bn3(x)
        # input tensor to the next layer
        x = layers.add([x, shortcut1])
        x = self.Activation3(x)
        # store the input tennsor for the next layer
        shortcut2 = self.shortcut2(x)
        shortcut2 = self.bns2(shortcut2)
        # shortcut2 = self.Activations2(shortcut2)


        x = self.conv4(x) #3
        x = self.bn4(x)
        x = self.Activation4(x)
        x = self.conv5(x) #4
        x = self.bn5(x)
        x = layers.add([x, shortcut2])
        x = self.Activation5(x)
        shortcut2a = self.shortcut2a(x)
        shortcut2a = self.bns2a(shortcut2a)
        # shortcut2a = self.Activations2a(shortcut2a)


        x = self.conv4a(x) #5
        x = self.bn4a(x)
        x = self.Activation4a(x)
        x = self.conv5a(x) #6
        x = self.bn5a(x)
        x = layers.add([x, shortcut2a])
        x = self.Activation5a(x)

        # Filters F2
        shortcut3 = self.shortcut3(x)
        shortcut3 = self.bns3(shortcut3)
        # shortcut3 = self.Activations3(shortcut3)

        x = self.conv6(x) #1
        x = self.bn6(x)
        x = self.Activation6(x)
        x = self.conv7(x) #2
        x = self.bn7(x)
        x = layers.add([x, shortcut3])
        x = self.Activation7(x)

        shortcut4 = self.shortcut4(x)
        shortcut4 = self.bns4(shortcut4)
        # shortcut4 = self.Activations4(shortcut4)


        x = self.conv8(x) #3
        x = self.bn8(x)
        x = self.Activation8(x)
        x = self.conv9(x) #4
```

```
        x = self.bn9(x)
        x = layers.add([x, shortcut4])
        x = self.Activation9(x)

        shortcut4a = self.shortcut4a(x)
        shortcut4a = self.bns4a(shortcut4a)
        # shortcut4a = self.Activations4a(shortcut4a)


        x = self.conv8a(x) #5
        x = self.bn8a(x)
        x = self.Activation8a(x)
        x = self.conv9a(x) #6
        x = self.bn9a(x)
        x = layers.add([x, shortcut4a])
        x = self.Activation9a(x)

        # filters f3
        shortcut5 = self.shortcut5(x)
        shortcut5 = self.bns5(shortcut5)
        # shortcut5 = self.Activations5(shortcut5)


        x = self.conv10(x) #1
        x = self.bn10(x)
        x = self.Activation10(x)
        x = self.conv11(x) #2
        x = self.bn11(x)
        x = layers.add([x, shortcut5])
        x = self.Activation11(x)

        shortcut6 = self.shortcut6(x)
        shortcut6 = self.bns6(shortcut6)
        # shortcut6 = self.Activations6(shortcut6)


        x = self.conv12(x) #3
        x = self.bn12(x)
        x = self.Activation12(x)
        x = self.conv13(x) #4
        x = self.bn13(x)
        x = layers.add([x, shortcut6])
        x = self.Activation13(x)

        shortcut7 = self.shortcut7(x)
        shortcut7 = self.bns7(shortcut7)
        # shortcut7 = self.Activations7(shortcut7)


        x = self.conv12a(x) #5
        x = self.bn12a(x)
        x = self.Activation12a(x)
        x = self.conv13a(x) #6
        x = self.bn13a(x)
        x = self.Activation13a(x)
        x = layers.add([x, shortcut7])


        x = self.Pooling(x)
        x = self.flatten(x)
        # x = self.d1(x)
        # x = self.drop(x)
        return self.d2(x) # Layer count: 1 + 6 + 6 + 6 + 1

    def model(self):
        x = Input(shape=(100, 100, 1))
        return Model(inputs=[x], outputs=self.call(x))

if __name__ == '__main__':
    sub = Bee_clf()
    sub.model().summary()
```

```
Model: "model"
_________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 100, 100, 1  0           []
                                )]

 3_chan (Conv2D)                (None, 100, 100, 3)  6           ['input_1[0][0]']

 tf.__operators__.getitem (Slic  (None, 100, 100, 3)  0          ['3_chan[0][0]']
 ingOpLambda)

 tf.nn.bias_add (TFOpLambda)    (None, 100, 100, 3)  0           ['tf.__operators__.getitem[0][0]'
                                                                 ]

 zero_padding2d (ZeroPadding2D)  (None, 102, 102, 3)  0          ['tf.nn.bias_add[0][0]']

 conv1 (Conv2D)                 (None, 100, 100, 16  448         ['zero_padding2d[0][0]']
                                )

 batch_normalization (BatchNorm  (None, 100, 100, 16  64         ['conv1[0][0]']
 alization)                     )

 activation (Activation)        (None, 100, 100, 16  0           ['batch_normalization[0][0]']
                                )

 average_pooling2d (AveragePool  (None, 50, 50, 16)  0           ['activation[0][0]']
```

```
ing2D)

zero_padding2d_1 (ZeroPadding2  (None, 52, 52, 16)  0           ['average_pooling2d[0][0]']
D)

conv2 (Conv2D)                 (None, 52, 52, 16)   2320        ['zero_padding2d_1[0][0]']

batch_normalization_2 (BatchNo  (None, 52, 52, 16)  64          ['conv2[0][0]']
rmalization)

activation_2 (Activation)      (None, 52, 52, 16)   0           ['batch_normalization_2[0][0]']

conv3 (Conv2D)                 (None, 52, 52, 16)   2320        ['activation_2[0][0]']

short1 (Conv2D)                (None, 52, 52, 16)   272         ['zero_padding2d_1[0][0]']

batch_normalization_3 (BatchNo  (None, 52, 52, 16)  64          ['conv3[0][0]']
rmalization)

batch_normalization_1 (BatchNo  (None, 52, 52, 16)  208         ['short1[0][0]']
rmalization)

add (Add)                      (None, 52, 52, 16)   0           ['batch_normalization_3[0][0]',
                                                                 'batch_normalization_1[0][0]']

activation_3 (Activation)      (None, 52, 52, 16)   0           ['add[0][0]']

conv4 (Conv2D)                 (None, 52, 52, 16)   2320        ['activation_3[0][0]']

batch_normalization_5 (BatchNo  (None, 52, 52, 16)  64          ['conv4[0][0]']
rmalization)

activation_5 (Activation)      (None, 52, 52, 16)   0           ['batch_normalization_5[0][0]']

conv5 (Conv2D)                 (None, 52, 52, 16)   2320        ['activation_5[0][0]']

short2 (Conv2D)                (None, 52, 52, 16)   272         ['activation_3[0][0]']

batch_normalization_6 (BatchNo  (None, 52, 52, 16)  64          ['conv5[0][0]']
rmalization)

batch_normalization_4 (BatchNo  (None, 52, 52, 16)  64          ['short2[0][0]']
rmalization)

add_1 (Add)                    (None, 52, 52, 16)   0           ['batch_normalization_6[0][0]',
                                                                 'batch_normalization_4[0][0]']

activation_6 (Activation)      (None, 52, 52, 16)   0           ['add_1[0][0]']

conv4a (Conv2D)                (None, 52, 52, 16)   2320        ['activation_6[0][0]']

batch_normalization_8 (BatchNo  (None, 52, 52, 16)  64          ['conv4a[0][0]']
rmalization)

activation_8 (Activation)      (None, 52, 52, 16)   0           ['batch_normalization_8[0][0]']

conv5a (Conv2D)                (None, 52, 52, 16)   2320        ['activation_8[0][0]']

short2a (Conv2D)               (None, 52, 52, 16)   272         ['activation_6[0][0]']

batch_normalization_9 (BatchNo  (None, 52, 52, 16)  64          ['conv5a[0][0]']
rmalization)

batch_normalization_7 (BatchNo  (None, 52, 52, 16)  64          ['short2a[0][0]']
rmalization)

add_2 (Add)                    (None, 52, 52, 16)   0           ['batch_normalization_9[0][0]',
                                                                 'batch_normalization_7[0][0]']

activation_9 (Activation)      (None, 52, 52, 16)   0           ['add_2[0][0]']

conv6 (Conv2D)                 (None, 52, 52, 32)   4640        ['activation_9[0][0]']

batch_normalization_11 (BatchN  (None, 52, 52, 32)  128         ['conv6[0][0]']
ormalization)

activation_11 (Activation)     (None, 52, 52, 32)   0           ['batch_normalization_11[0][0]']

conv7 (Conv2D)                 (None, 52, 52, 32)   9248        ['activation_11[0][0]']

short3 (Conv2D)                (None, 52, 52, 32)   544         ['activation_9[0][0]']

batch_normalization_12 (BatchN  (None, 52, 52, 32)  128         ['conv7[0][0]']
ormalization)

batch_normalization_10 (BatchN  (None, 52, 52, 32)  128         ['short3[0][0]']
ormalization)

add_3 (Add)                    (None, 52, 52, 32)   0           ['batch_normalization_12[0][0]',
                                                                 'batch_normalization_10[0][0]']

activation_12 (Activation)     (None, 52, 52, 32)   0           ['add_3[0][0]']

conv8 (Conv2D)                 (None, 52, 52, 32)   9248        ['activation_12[0][0]']

batch_normalization_14 (BatchN  (None, 52, 52, 32)  128         ['conv8[0][0]']
ormalization)

activation_14 (Activation)     (None, 52, 52, 32)   0           ['batch_normalization_14[0][0]']
```

```
conv9 (Conv2D)                 (None, 52, 52, 32)   9248        ['activation_14[0][0]']

short4 (Conv2D)                (None, 52, 52, 32)   1056        ['activation_12[0][0]']

batch_normalization_15 (BatchN  (None, 52, 52, 32)  128         ['conv9[0][0]']
ormalization)

batch_normalization_13 (BatchN  (None, 52, 52, 32)  128         ['short4[0][0]']
ormalization)

add_4 (Add)                    (None, 52, 52, 32)   0           ['batch_normalization_15[0][0]',
                                                                 'batch_normalization_13[0][0]']

activation_15 (Activation)     (None, 52, 52, 32)   0           ['add_4[0][0]']

conv8a (Conv2D)                (None, 52, 52, 32)   9248        ['activation_15[0][0]']

batch_normalization_17 (BatchN  (None, 52, 52, 32)  128         ['conv8a[0][0]']
ormalization)

activation_17 (Activation)     (None, 52, 52, 32)   0           ['batch_normalization_17[0][0]']

conv9a (Conv2D)                (None, 52, 52, 32)   9248        ['activation_17[0][0]']

short4a (Conv2D)               (None, 52, 52, 32)   1056        ['activation_15[0][0]']

batch_normalization_18 (BatchN  (None, 52, 52, 32)  128         ['conv9a[0][0]']
ormalization)

batch_normalization_16 (BatchN  (None, 52, 52, 32)  128         ['short4a[0][0]']
ormalization)

add_5 (Add)                    (None, 52, 52, 32)   0           ['batch_normalization_18[0][0]',
                                                                 'batch_normalization_16[0][0]']

activation_18 (Activation)     (None, 52, 52, 32)   0           ['add_5[0][0]']

conv10 (Conv2D)                (None, 52, 52, 64)   18496       ['activation_18[0][0]']

batch_normalization_20 (BatchN  (None, 52, 52, 64)  256         ['conv10[0][0]']
ormalization)

activation_20 (Activation)     (None, 52, 52, 64)   0           ['batch_normalization_20[0][0]']

conv11 (Conv2D)                (None, 52, 52, 64)   36928       ['activation_20[0][0]']

short5 (Conv2D)                (None, 52, 52, 64)   2112        ['activation_18[0][0]']

batch_normalization_21 (BatchN  (None, 52, 52, 64)  256         ['conv11[0][0]']
ormalization)

batch_normalization_19 (BatchN  (None, 52, 52, 64)  256         ['short5[0][0]']
ormalization)

add_6 (Add)                    (None, 52, 52, 64)   0           ['batch_normalization_21[0][0]',
                                                                 'batch_normalization_19[0][0]']

activation_21 (Activation)     (None, 52, 52, 64)   0           ['add_6[0][0]']

conv12 (Conv2D)                (None, 52, 52, 64)   36928       ['activation_21[0][0]']

batch_normalization_23 (BatchN  (None, 52, 52, 64)  256         ['conv12[0][0]']
ormalization)

activation_23 (Activation)     (None, 52, 52, 64)   0           ['batch_normalization_23[0][0]']

conv13 (Conv2D)                (None, 52, 52, 64)   36928       ['activation_23[0][0]']

short6 (Conv2D)                (None, 52, 52, 64)   4160        ['activation_21[0][0]']

batch_normalization_24 (BatchN  (None, 52, 52, 64)  256         ['conv13[0][0]']
ormalization)

batch_normalization_22 (BatchN  (None, 52, 52, 64)  256         ['short6[0][0]']
ormalization)

add_7 (Add)                    (None, 52, 52, 64)   0           ['batch_normalization_24[0][0]',
                                                                 'batch_normalization_22[0][0]']

activation_24 (Activation)     (None, 52, 52, 64)   0           ['add_7[0][0]']

conv12a (Conv2D)               (None, 52, 52, 64)   36928       ['activation_24[0][0]']

batch_normalization_26 (BatchN  (None, 52, 52, 64)  256         ['conv12a[0][0]']
ormalization)

activation_26 (Activation)     (None, 52, 52, 64)   0           ['batch_normalization_26[0][0]']

conv13a (Conv2D)               (None, 52, 52, 64)   36928       ['activation_26[0][0]']

batch_normalization_27 (BatchN  (None, 52, 52, 64)  256         ['conv13a[0][0]']
ormalization)

short7 (Conv2D)                (None, 52, 52, 64)   4160        ['activation_24[0][0]']

activation_27 (Activation)     (None, 52, 52, 64)   0           ['batch_normalization_27[0][0]']

batch_normalization_25 (BatchN  (None, 52, 52, 64)  256         ['short7[0][0]']
```

```
ormalization)

add_8 (Add)                    (None, 52, 52, 64)   0           ['activation_27[0][0]',
                                                                 'batch_normalization_25[0][0]']

pool (GlobalAveragePooling2D)  (None, 64)           0           ['add_8[0][0]']

flatten (Flatten)              (None, 64)           0           ['pool[0][0]']

dense_1 (Dense)                (None, 1)            65          ['flatten[0][0]']

==================================================================================================
Total params: 286,599
Trainable params: 284,479
Non-trainable params: 2,120
__________________________________________________________________________________________________
```

3. Batch and Shuffle the dataset: (`tf.data`)

In [19]:
```
np.unique(y_train, return_counts = True)
```

Out[19]:
```
(array([0, 1]), array([1047, 1069]))
```

In [20]:
```
Batch_Size = 64
# Suffle = 10000

train_ds = tf.data.Dataset.from_tensor_slices(
    (x_train, y_train)).shuffle(1000).batch(Batch_Size).prefetch(64)

test_ds = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(Batch_Size)
```

5. Choose an optimizer and loss function for training

In [21]:
```
# Loss: Categorical/Binary Cross Entropy, Mean Squared Error, Focal, etc.
# Optimizer: Adam, SGD, RMSProp, etc.

loss_object = tf.keras.losses.BinaryCrossentropy(from_logits=False)
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=1e-2,
    decay_steps=50000/Batch_Size,
    decay_rate=0.9)
optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)
# optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule)
```

6. Select Metrics

Now, we select metrics to measure the loss and the accuracy of the model. These metrics accumulate the values over epochs and then print the overall result.

In [22]:
```
train_loss = tf.keras.metrics.Mean(name='train_loss')
train_accuracy = tf.keras.metrics.BinaryAccuracy(name='train_accuracy')

test_loss = tf.keras.metrics.Mean(name='test_loss')
test_accuracy = tf.keras.metrics.BinaryAccuracy(name='test_accuracy')
```

7. Train & Test Function (`tf.GradientTape`)

In [23]:
```
# Tensorboard data
current_time = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
train_log_dir = 'logs/' + current_time + '/train'
test_log_dir = 'logs/' + current_time + '/test'
train_summary_writer = tf.summary.create_file_writer(train_log_dir)
test_summary_writer = tf.summary.create_file_writer(test_log_dir)
```

In [24]:
```
@tf.function
def train_step(images, labels):
  with tf.GradientTape() as tape:
    # training=True is only needed if there are layers with different
    # behavior during training versus inference (e.g. Dropout).
    predictions = model(images, training=True)
    loss = loss_object(labels, predictions)

  gradients = tape.gradient(loss, model.trainable_variables)
  optimizer.apply_gradients(zip(gradients, model.trainable_variables))

  train_loss(loss)
  train_accuracy(labels, predictions)
```

In [25]:
```
@tf.function
def test_step(images, labels):
  # training=False is only needed if there are layers with different
  # behavior during training versus inference (e.g. Dropout).
  predictions = model(images, training=False)
  t_loss = loss_object(labels, predictions)

  test_loss(t_loss)
  test_accuracy(labels, predictions)
```

8. Perform Training & Testing of the Model

In [26]:

```
model = Bee_clf() # to reset our model
# model_path = 'models/
# model = tf.keras.models.load_model(model_path) # to load a model
model_start = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
trainloss = []
testloss = []
trainacc = []
testacc = []
EPOCHS = 200  # Number of Epochs
for epoch in range(EPOCHS):
    # Reset the metrics at the start of the next epoch
    train_loss.reset_states()
    train_accuracy.reset_states()
    test_loss.reset_states()
    test_accuracy.reset_states()

    for images, labels in train_ds:
        train_step(images, labels)
    with train_summary_writer.as_default():
        tf.summary.scalar('loss', train_loss.result(), step=epoch)
        tf.summary.scalar('accuracy', train_accuracy.result(), step=epoch)

    for test_images, test_labels in test_ds:
        test_step(test_images, test_labels)
    with test_summary_writer.as_default():
        tf.summary.scalar('loss', test_loss.result(), step=epoch)
        tf.summary.scalar('accuracy', test_accuracy.result(), step=epoch)

    trainloss.append(train_loss.result())
    testloss.append(test_loss.result())
    trainacc.append(train_accuracy.result())
    testacc.append(test_accuracy.result())
    print(
        f'Epoch {epoch + 1}: '
        f'Train Loss: {train_loss.result():0.2f}, '
        f'Train Accuracy: {train_accuracy.result() * 100:0.2f}, '
        f'Test Loss: {test_loss.result():0.2f}, '
        f'Test Accuracy: {test_accuracy.result() * 100:0.2f}\n'
    )
# Model weights
# model_path = 'models/' + model_start
# model.save(model_path)


print('Done!')
```

```
Epoch 1: Train Loss: 0.68, Train Accuracy: 56.85, Test Loss: 0.69, Test Accuracy: 52.28

Epoch 2: Train Loss: 0.63, Train Accuracy: 62.68, Test Loss: 0.77, Test Accuracy: 49.92

Epoch 3: Train Loss: 0.61, Train Accuracy: 65.81, Test Loss: 1.01, Test Accuracy: 49.92

Epoch 4: Train Loss: 0.58, Train Accuracy: 69.16, Test Loss: 1.01, Test Accuracy: 49.92

Epoch 5: Train Loss: 0.56, Train Accuracy: 70.36, Test Loss: 1.21, Test Accuracy: 49.92

Epoch 6: Train Loss: 0.56, Train Accuracy: 71.78, Test Loss: 1.23, Test Accuracy: 49.92

Epoch 7: Train Loss: 0.52, Train Accuracy: 74.49, Test Loss: 1.54, Test Accuracy: 49.92

Epoch 8: Train Loss: 0.52, Train Accuracy: 75.28, Test Loss: 2.79, Test Accuracy: 49.92

Epoch 9: Train Loss: 0.48, Train Accuracy: 77.48, Test Loss: 1.57, Test Accuracy: 50.10

Epoch 10: Train Loss: 0.47, Train Accuracy: 79.50, Test Loss: 0.70, Test Accuracy: 58.80

Epoch 11: Train Loss: 0.47, Train Accuracy: 78.03, Test Loss: 3.05, Test Accuracy: 50.10

Epoch 12: Train Loss: 0.40, Train Accuracy: 82.77, Test Loss: 1.43, Test Accuracy: 51.14

Epoch 13: Train Loss: 0.37, Train Accuracy: 85.66, Test Loss: 1.26, Test Accuracy: 53.14

Epoch 14: Train Loss: 0.34, Train Accuracy: 86.90, Test Loss: 0.69, Test Accuracy: 64.78

Epoch 15: Train Loss: 0.31, Train Accuracy: 88.33, Test Loss: 1.45, Test Accuracy: 52.45

Epoch 16: Train Loss: 0.29, Train Accuracy: 89.11, Test Loss: 1.84, Test Accuracy: 52.10

Epoch 17: Train Loss: 0.25, Train Accuracy: 92.42, Test Loss: 2.58, Test Accuracy: 50.62

Epoch 18: Train Loss: 0.25, Train Accuracy: 91.87, Test Loss: 2.69, Test Accuracy: 50.44

Epoch 19: Train Loss: 0.20, Train Accuracy: 94.30, Test Loss: 0.94, Test Accuracy: 60.69

Epoch 20: Train Loss: 0.19, Train Accuracy: 94.72, Test Loss: 3.62, Test Accuracy: 50.27

Epoch 21: Train Loss: 0.18, Train Accuracy: 95.08, Test Loss: 1.48, Test Accuracy: 58.18

Epoch 22: Train Loss: 0.18, Train Accuracy: 94.85, Test Loss: 0.91, Test Accuracy: 63.12

Epoch 23: Train Loss: 0.17, Train Accuracy: 95.86, Test Loss: 1.09, Test Accuracy: 58.70

Epoch 24: Train Loss: 0.15, Train Accuracy: 96.00, Test Loss: 1.16, Test Accuracy: 60.53

Epoch 25: Train Loss: 0.14, Train Accuracy: 95.96, Test Loss: 1.85, Test Accuracy: 52.80
```

Epoch 26: Train Loss: 0.11, Train Accuracy: 98.39, Test Loss: 1.49, Test Accuracy: 54.32

Epoch 27: Train Loss: 0.09, Train Accuracy: 98.35, Test Loss: 1.95, Test Accuracy: 53.14

Epoch 28: Train Loss: 0.11, Train Accuracy: 97.98, Test Loss: 1.98, Test Accuracy: 52.62

Epoch 29: Train Loss: 0.11, Train Accuracy: 98.12, Test Loss: 1.44, Test Accuracy: 57.31

Epoch 30: Train Loss: 0.10, Train Accuracy: 97.52, Test Loss: 3.95, Test Accuracy: 50.27

Epoch 31: Train Loss: 0.17, Train Accuracy: 94.72, Test Loss: 1.30, Test Accuracy: 60.24

Epoch 32: Train Loss: 0.09, Train Accuracy: 98.58, Test Loss: 3.13, Test Accuracy: 50.62

Epoch 33: Train Loss: 0.12, Train Accuracy: 96.78, Test Loss: 2.03, Test Accuracy: 55.98

Epoch 34: Train Loss: 0.08, Train Accuracy: 99.08, Test Loss: 1.18, Test Accuracy: 62.62

Epoch 35: Train Loss: 0.08, Train Accuracy: 97.98, Test Loss: 2.18, Test Accuracy: 55.32

Epoch 36: Train Loss: 0.10, Train Accuracy: 97.38, Test Loss: 1.00, Test Accuracy: 60.34

Epoch 37: Train Loss: 0.07, Train Accuracy: 99.63, Test Loss: 1.55, Test Accuracy: 56.94

Epoch 38: Train Loss: 0.06, Train Accuracy: 98.71, Test Loss: 1.56, Test Accuracy: 56.85

Epoch 39: Train Loss: 0.09, Train Accuracy: 97.93, Test Loss: 2.97, Test Accuracy: 50.95

Epoch 40: Train Loss: 0.06, Train Accuracy: 99.17, Test Loss: 1.83, Test Accuracy: 59.82

Epoch 41: Train Loss: 0.04, Train Accuracy: 99.95, Test Loss: 1.85, Test Accuracy: 56.62

Epoch 42: Train Loss: 0.05, Train Accuracy: 99.49, Test Loss: 1.44, Test Accuracy: 55.23

Epoch 43: Train Loss: 0.04, Train Accuracy: 99.72, Test Loss: 1.42, Test Accuracy: 57.14

Epoch 44: Train Loss: 0.04, Train Accuracy: 99.17, Test Loss: 3.85, Test Accuracy: 50.27

Epoch 45: Train Loss: 0.11, Train Accuracy: 96.32, Test Loss: 1.96, Test Accuracy: 58.35

Epoch 46: Train Loss: 0.04, Train Accuracy: 99.82, Test Loss: 1.03, Test Accuracy: 62.79

Epoch 47: Train Loss: 0.04, Train Accuracy: 99.22, Test Loss: 0.97, Test Accuracy: 60.17

Epoch 48: Train Loss: 0.05, Train Accuracy: 98.85, Test Loss: 2.02, Test Accuracy: 58.01

Epoch 49: Train Loss: 0.06, Train Accuracy: 99.03, Test Loss: 4.76, Test Accuracy: 50.10

Epoch 50: Train Loss: 0.03, Train Accuracy: 99.68, Test Loss: 1.06, Test Accuracy: 60.84

Epoch 51: Train Loss: 0.05, Train Accuracy: 97.79, Test Loss: 1.07, Test Accuracy: 60.78

Epoch 52: Train Loss: 0.05, Train Accuracy: 99.31, Test Loss: 1.37, Test Accuracy: 60.09

Epoch 53: Train Loss: 0.03, Train Accuracy: 99.22, Test Loss: 1.06, Test Accuracy: 61.65

Epoch 54: Train Loss: 0.04, Train Accuracy: 99.63, Test Loss: 1.06, Test Accuracy: 64.78

Epoch 55: Train Loss: 0.04, Train Accuracy: 99.13, Test Loss: 3.41, Test Accuracy: 51.49

Epoch 56: Train Loss: 0.05, Train Accuracy: 98.81, Test Loss: 5.95, Test Accuracy: 49.92

Epoch 57: Train Loss: 0.04, Train Accuracy: 99.13, Test Loss: 1.12, Test Accuracy: 64.27

Epoch 58: Train Loss: 0.06, Train Accuracy: 98.39, Test Loss: 1.57, Test Accuracy: 60.26

Epoch 59: Train Loss: 0.10, Train Accuracy: 97.15, Test Loss: 1.83, Test Accuracy: 58.80

Epoch 60: Train Loss: 0.03, Train Accuracy: 99.68, Test Loss: 4.31, Test Accuracy: 50.79

Epoch 61: Train Loss: 0.02, Train Accuracy: 100.00, Test Loss: 1.34, Test Accuracy: 60.01

Epoch 62: Train Loss: 0.06, Train Accuracy: 98.48, Test Loss: 3.94, Test Accuracy: 51.12

Epoch 63: Train Loss: 0.13, Train Accuracy: 95.96, Test Loss: 5.41, Test Accuracy: 50.60

Epoch 64: Train Loss: 0.05, Train Accuracy: 98.94, Test Loss: 1.83, Test Accuracy: 57.14

Epoch 65: Train Loss: 0.02, Train Accuracy: 99.95, Test Loss: 1.50, Test Accuracy: 62.96

Epoch 66: Train Loss: 0.02, Train Accuracy: 100.00, Test Loss: 1.12, Test Accuracy: 59.55

Epoch 67: Train Loss: 0.06, Train Accuracy: 98.39, Test Loss: 1.64, Test Accuracy: 57.58

Epoch 68: Train Loss: 0.17, Train Accuracy: 93.43, Test Loss: 5.64, Test Accuracy: 50.95

Epoch 69: Train Loss: 0.08, Train Accuracy: 97.47, Test Loss: 2.40, Test Accuracy: 56.33

Epoch 70: Train Loss: 0.07, Train Accuracy: 97.56, Test Loss: 5.86, Test Accuracy: 50.44

Epoch 71: Train Loss: 0.22, Train Accuracy: 91.77, Test Loss: 1.85, Test Accuracy: 63.73

Epoch 72: Train Loss: 0.06, Train Accuracy: 98.85, Test Loss: 3.21, Test Accuracy: 53.82

Epoch 73: Train Loss: 0.03, Train Accuracy: 99.91, Test Loss: 1.02, Test Accuracy: 63.93

Epoch 74: Train Loss: 0.03, Train Accuracy: 99.26, Test Loss: 1.06, Test Accuracy: 62.87

Epoch 75: Train Loss: 0.02, Train Accuracy: 99.86, Test Loss: 0.97, Test Accuracy: 63.91

Epoch 76: Train Loss: 0.03, Train Accuracy: 99.26, Test Loss: 1.93, Test Accuracy: 57.91

Epoch 77: Train Loss: 0.05, Train Accuracy: 98.71, Test Loss: 6.46, Test Accuracy: 50.62

Epoch 78: Train Loss: 0.02, Train Accuracy: 99.91, Test Loss: 1.26, Test Accuracy: 61.40

Epoch 79: Train Loss: 0.04, Train Accuracy: 99.26, Test Loss: 1.08, Test Accuracy: 62.25

Epoch 80: Train Loss: 0.07, Train Accuracy: 97.20, Test Loss: 1.70, Test Accuracy: 58.85

Epoch 81: Train Loss: 0.06, Train Accuracy: 98.90, Test Loss: 1.55, Test Accuracy: 60.78

Epoch 82: Train Loss: 0.03, Train Accuracy: 99.45, Test Loss: 5.16, Test Accuracy: 49.92

Epoch 83: Train Loss: 0.02, Train Accuracy: 99.77, Test Loss: 3.04, Test Accuracy: 51.14

Epoch 84: Train Loss: 0.06, Train Accuracy: 98.48, Test Loss: 2.26, Test Accuracy: 58.62

Epoch 85: Train Loss: 0.05, Train Accuracy: 98.48, Test Loss: 2.16, Test Accuracy: 61.75

Epoch 86: Train Loss: 0.02, Train Accuracy: 99.77, Test Loss: 1.19, Test Accuracy: 63.48

Epoch 87: Train Loss: 0.01, Train Accuracy: 100.00, Test Loss: 1.10, Test Accuracy: 62.00

Epoch 88: Train Loss: 0.02, Train Accuracy: 99.91, Test Loss: 1.11, Test Accuracy: 62.79

Epoch 89: Train Loss: 0.02, Train Accuracy: 99.17, Test Loss: 1.14, Test Accuracy: 64.45

Epoch 90: Train Loss: 0.04, Train Accuracy: 98.99, Test Loss: 3.00, Test Accuracy: 55.73

Epoch 91: Train Loss: 0.03, Train Accuracy: 99.59, Test Loss: 2.05, Test Accuracy: 59.65

Epoch 92: Train Loss: 0.02, Train Accuracy: 99.63, Test Loss: 2.35, Test Accuracy: 52.80

Epoch 93: Train Loss: 0.01, Train Accuracy: 100.00, Test Loss: 2.35, Test Accuracy: 53.67

Epoch 94: Train Loss: 0.02, Train Accuracy: 99.22, Test Loss: 4.80, Test Accuracy: 50.10

Epoch 95: Train Loss: 0.03, Train Accuracy: 99.91, Test Loss: 5.50, Test Accuracy: 50.10

Epoch 96: Train Loss: 0.05, Train Accuracy: 99.08, Test Loss: 4.21, Test Accuracy: 52.45

Epoch 97: Train Loss: 0.04, Train Accuracy: 99.17, Test Loss: 1.45, Test Accuracy: 59.12

Epoch 98: Train Loss: 0.08, Train Accuracy: 97.98, Test Loss: 6.17, Test Accuracy: 50.27

Epoch 99: Train Loss: 0.13, Train Accuracy: 95.27, Test Loss: 5.46, Test Accuracy: 51.31

Epoch 100: Train Loss: 0.14, Train Accuracy: 94.67, Test Loss: 1.82, Test Accuracy: 61.38

Epoch 101: Train Loss: 0.08, Train Accuracy: 97.75, Test Loss: 1.75, Test Accuracy: 59.28

Epoch 102: Train Loss: 0.07, Train Accuracy: 98.21, Test Loss: 1.58, Test Accuracy: 62.87

Epoch 103: Train Loss: 0.07, Train Accuracy: 97.79, Test Loss: 6.94, Test Accuracy: 50.08

Epoch 104: Train Loss: 0.03, Train Accuracy: 98.81, Test Loss: 1.97, Test Accuracy: 60.26

Epoch 105: Train Loss: 0.03, Train Accuracy: 99.08, Test Loss: 2.45, Test Accuracy: 55.98

Epoch 106: Train Loss: 0.03, Train Accuracy: 99.72, Test Loss: 1.08, Test Accuracy: 61.90

Epoch 107: Train Loss: 0.02, Train Accuracy: 99.77, Test Loss: 2.29, Test Accuracy: 57.81

Epoch 108: Train Loss: 0.01, Train Accuracy: 100.00, Test Loss: 1.25, Test Accuracy: 61.36

Epoch 109: Train Loss: 0.02, Train Accuracy: 99.26, Test Loss: 2.62, Test Accuracy: 54.94

Epoch 110: Train Loss: 0.06, Train Accuracy: 98.35, Test Loss: 2.21, Test Accuracy: 57.81

Epoch 111: Train Loss: 0.08, Train Accuracy: 97.33, Test Loss: 1.44, Test Accuracy: 62.42

Epoch 112: Train Loss: 0.06, Train Accuracy: 98.48, Test Loss: 5.19, Test Accuracy: 51.29

Epoch 113: Train Loss: 0.06, Train Accuracy: 98.30, Test Loss: 2.38, Test Accuracy: 58.87

Epoch 114: Train Loss: 0.07, Train Accuracy: 97.47, Test Loss: 1.52, Test Accuracy: 62.25

Epoch 115: Train Loss: 0.04, Train Accuracy: 99.63, Test Loss: 4.96, Test Accuracy: 50.79

Epoch 116: Train Loss: 0.02, Train Accuracy: 99.91, Test Loss: 1.44, Test Accuracy: 61.03

Epoch 117: Train Loss: 0.01, Train Accuracy: 99.95, Test Loss: 1.27, Test Accuracy: 60.86

Epoch 118: Train Loss: 0.01, Train Accuracy: 99.26, Test Loss: 2.81, Test Accuracy: 53.14

Epoch 119: Train Loss: 0.08, Train Accuracy: 98.21, Test Loss: 2.60, Test Accuracy: 52.97

Epoch 120: Train Loss: 0.05, Train Accuracy: 98.90, Test Loss: 3.92, Test Accuracy: 52.51

Epoch 121: Train Loss: 0.05, Train Accuracy: 99.03, Test Loss: 3.64, Test Accuracy: 53.72

Epoch 122: Train Loss: 0.08, Train Accuracy: 97.33, Test Loss: 3.90, Test Accuracy: 49.75

Epoch 123: Train Loss: 0.03, Train Accuracy: 98.44, Test Loss: 1.50, Test Accuracy: 58.01

Epoch 124: Train Loss: 0.04, Train Accuracy: 98.99, Test Loss: 2.78, Test Accuracy: 55.23

Epoch 125: Train Loss: 0.02, Train Accuracy: 99.95, Test Loss: 4.09, Test Accuracy: 50.62

Epoch 126: Train Loss: 0.02, Train Accuracy: 99.91, Test Loss: 2.97, Test Accuracy: 52.80

Epoch 127: Train Loss: 0.02, Train Accuracy: 99.77, Test Loss: 1.59, Test Accuracy: 62.50

Epoch 128: Train Loss: 0.01, Train Accuracy: 100.00, Test Loss: 1.27, Test Accuracy: 63.21

Epoch 129: Train Loss: 0.01, Train Accuracy: 99.95, Test Loss: 1.71, Test Accuracy: 56.79

Epoch 130: Train Loss: 0.01, Train Accuracy: 100.00, Test Loss: 1.14, Test Accuracy: 60.94

Epoch 131: Train Loss: 0.01, Train Accuracy: 99.95, Test Loss: 1.48, Test Accuracy: 61.55

Epoch 132: Train Loss: 0.01, Train Accuracy: 100.00, Test Loss: 1.37, Test Accuracy: 58.70

Epoch 133: Train Loss: 0.04, Train Accuracy: 99.22, Test Loss: 1.39, Test Accuracy: 63.73

Epoch 134: Train Loss: 0.04, Train Accuracy: 98.67, Test Loss: 1.68, Test Accuracy: 60.09

Epoch 135: Train Loss: 0.07, Train Accuracy: 97.61, Test Loss: 1.33, Test Accuracy: 61.88

Epoch 136: Train Loss: 0.03, Train Accuracy: 99.22, Test Loss: 2.15, Test Accuracy: 60.49

Epoch 137: Train Loss: 0.03, Train Accuracy: 98.35, Test Loss: 2.39, Test Accuracy: 57.58

Epoch 138: Train Loss: 0.06, Train Accuracy: 98.35, Test Loss: 3.05, Test Accuracy: 57.23

Epoch 139: Train Loss: 0.05, Train Accuracy: 98.16, Test Loss: 1.56, Test Accuracy: 57.72

Epoch 140: Train Loss: 0.12, Train Accuracy: 97.52, Test Loss: 3.96, Test Accuracy: 52.35

Epoch 141: Train Loss: 0.04, Train Accuracy: 98.62, Test Loss: 2.88, Test Accuracy: 53.32

Epoch 142: Train Loss: 0.08, Train Accuracy: 98.21, Test Loss: 5.28, Test Accuracy: 49.92

Epoch 143: Train Loss: 0.15, Train Accuracy: 94.99, Test Loss: 5.71, Test Accuracy: 50.95

Epoch 144: Train Loss: 0.04, Train Accuracy: 98.67, Test Loss: 4.83, Test Accuracy: 51.47

Epoch 145: Train Loss: 0.03, Train Accuracy: 99.63, Test Loss: 1.87, Test Accuracy: 60.07

Epoch 146: Train Loss: 0.05, Train Accuracy: 98.44, Test Loss: 2.02, Test Accuracy: 57.21

Epoch 147: Train Loss: 0.04, Train Accuracy: 99.22, Test Loss: 2.91, Test Accuracy: 53.90

Epoch 148: Train Loss: 0.05, Train Accuracy: 98.16, Test Loss: 1.84, Test Accuracy: 60.69

Epoch 149: Train Loss: 0.04, Train Accuracy: 98.99, Test Loss: 7.26, Test Accuracy: 50.25

Epoch 150: Train Loss: 0.01, Train Accuracy: 99.86, Test Loss: 1.62, Test Accuracy: 61.73

Epoch 151: Train Loss: 0.01, Train Accuracy: 99.95, Test Loss: 1.40, Test Accuracy: 60.17

Epoch 152: Train Loss: 0.06, Train Accuracy: 97.79, Test Loss: 5.89, Test Accuracy: 50.25

Epoch 153: Train Loss: 0.16, Train Accuracy: 95.17, Test Loss: 2.15, Test Accuracy: 63.66

Epoch 154: Train Loss: 0.03, Train Accuracy: 99.59, Test Loss: 3.27, Test Accuracy: 55.05

Epoch 155: Train Loss: 0.08, Train Accuracy: 99.17, Test Loss: 1.20, Test Accuracy: 62.85

Epoch 156: Train Loss: 0.04, Train Accuracy: 99.13, Test Loss: 3.20, Test Accuracy: 52.16

Epoch 157: Train Loss: 0.02, Train Accuracy: 99.63, Test Loss: 1.20, Test Accuracy: 61.88

Epoch 158: Train Loss: 0.04, Train Accuracy: 98.53, Test Loss: 1.17, Test Accuracy: 60.49

Epoch 159: Train Loss: 0.04, Train Accuracy: 98.71, Test Loss: 1.76, Test Accuracy: 60.07

Epoch 160: Train Loss: 0.02, Train Accuracy: 99.72, Test Loss: 1.14, Test Accuracy: 62.15

Epoch 161: Train Loss: 0.01, Train Accuracy: 99.95, Test Loss: 2.23, Test Accuracy: 55.57

Epoch 162: Train Loss: 0.03, Train Accuracy: 99.13, Test Loss: 1.43, Test Accuracy: 61.30

Epoch 163: Train Loss: 0.02, Train Accuracy: 99.91, Test Loss: 3.69, Test Accuracy: 53.03

Epoch 164: Train Loss: 0.01, Train Accuracy: 100.00, Test Loss: 1.35, Test Accuracy: 62.60

Epoch 165: Train Loss: 0.01, Train Accuracy: 100.00, Test Loss: 1.79, Test Accuracy: 60.94

Epoch 166: Train Loss: 0.03, Train Accuracy: 99.26, Test Loss: 3.50, Test Accuracy: 54.42

Epoch 167: Train Loss: 0.03, Train Accuracy: 99.22, Test Loss: 3.21, Test Accuracy: 56.94

Epoch 168: Train Loss: 0.06, Train Accuracy: 99.26, Test Loss: 1.40, Test Accuracy: 62.77

Epoch 169: Train Loss: 0.02, Train Accuracy: 99.54, Test Loss: 1.58, Test Accuracy: 61.03

Epoch 170: Train Loss: 0.04, Train Accuracy: 98.44, Test Loss: 2.83, Test Accuracy: 58.51

Epoch 171: Train Loss: 0.05, Train Accuracy: 97.98, Test Loss: 3.85, Test Accuracy: 54.24

Epoch 172: Train Loss: 0.04, Train Accuracy: 98.76, Test Loss: 3.53, Test Accuracy: 53.07

Epoch 173: Train Loss: 0.02, Train Accuracy: 99.72, Test Loss: 2.02, Test Accuracy: 56.44

```
Epoch 174: Train Loss: 0.03, Train Accuracy: 98.39, Test Loss: 1.40, Test Accuracy: 60.32

Epoch 175: Train Loss: 0.04, Train Accuracy: 98.53, Test Loss: 3.26, Test Accuracy: 55.02

Epoch 176: Train Loss: 0.03, Train Accuracy: 99.36, Test Loss: 2.54, Test Accuracy: 60.07

Epoch 177: Train Loss: 0.01, Train Accuracy: 100.00, Test Loss: 2.02, Test Accuracy: 61.05

Epoch 178: Train Loss: 0.01, Train Accuracy: 100.00, Test Loss: 1.93, Test Accuracy: 59.14

Epoch 179: Train Loss: 0.01, Train Accuracy: 99.91, Test Loss: 1.27, Test Accuracy: 63.29

Epoch 180: Train Loss: 0.05, Train Accuracy: 98.53, Test Loss: 2.78, Test Accuracy: 56.50

Epoch 181: Train Loss: 0.03, Train Accuracy: 98.12, Test Loss: 3.52, Test Accuracy: 53.76

Epoch 182: Train Loss: 0.03, Train Accuracy: 99.49, Test Loss: 5.78, Test Accuracy: 51.12

Epoch 183: Train Loss: 0.01, Train Accuracy: 99.95, Test Loss: 2.50, Test Accuracy: 57.75

Epoch 184: Train Loss: 0.02, Train Accuracy: 98.53, Test Loss: 4.30, Test Accuracy: 52.97

Epoch 185: Train Loss: 0.03, Train Accuracy: 99.45, Test Loss: 4.05, Test Accuracy: 53.24

Epoch 186: Train Loss: 0.01, Train Accuracy: 99.91, Test Loss: 2.49, Test Accuracy: 55.32

Epoch 187: Train Loss: 0.01, Train Accuracy: 100.00, Test Loss: 1.38, Test Accuracy: 61.30

Epoch 188: Train Loss: 0.01, Train Accuracy: 99.95, Test Loss: 2.26, Test Accuracy: 57.93

Epoch 189: Train Loss: 0.02, Train Accuracy: 99.26, Test Loss: 1.20, Test Accuracy: 63.19

Epoch 190: Train Loss: 0.05, Train Accuracy: 97.89, Test Loss: 1.94, Test Accuracy: 61.63

Epoch 191: Train Loss: 0.02, Train Accuracy: 99.72, Test Loss: 2.32, Test Accuracy: 60.19

Epoch 192: Train Loss: 0.01, Train Accuracy: 99.82, Test Loss: 1.83, Test Accuracy: 61.86

Epoch 193: Train Loss: 0.01, Train Accuracy: 100.00, Test Loss: 1.35, Test Accuracy: 64.06

Epoch 194: Train Loss: 0.01, Train Accuracy: 99.95, Test Loss: 2.26, Test Accuracy: 57.70

Epoch 195: Train Loss: 0.01, Train Accuracy: 100.00, Test Loss: 1.47, Test Accuracy: 64.58

Epoch 196: Train Loss: 0.05, Train Accuracy: 97.75, Test Loss: 2.66, Test Accuracy: 57.02

Epoch 197: Train Loss: 0.05, Train Accuracy: 99.03, Test Loss: 4.45, Test Accuracy: 51.41

Epoch 198: Train Loss: 0.05, Train Accuracy: 98.35, Test Loss: 4.85, Test Accuracy: 52.72

Epoch 199: Train Loss: 0.07, Train Accuracy: 98.02, Test Loss: 2.52, Test Accuracy: 59.12

Epoch 200: Train Loss: 0.08, Train Accuracy: 97.75, Test Loss: 1.87, Test Accuracy: 59.05

Done!
```

In [27]:

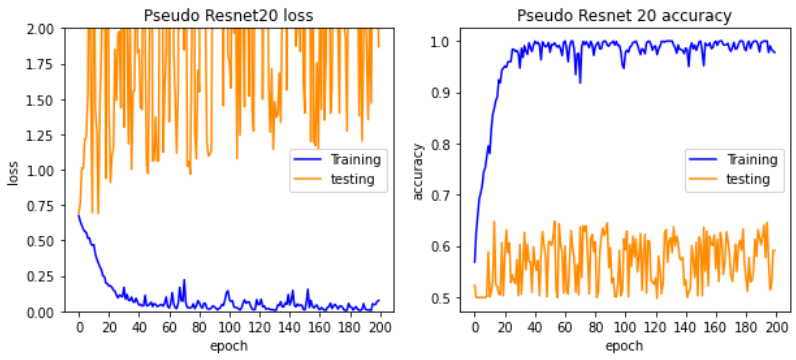
```
a = test_step(x_eval, y_eval)
f'Validation Accuracy: {test_accuracy.result() * 100:0.2f}\n'
```

Out[27]:

```
'Validation Accuracy: 59.05\n'
```

In [28]:

```
fig, ax = plt.subplots(1,2, figsize = (10,4))
ax[0].plot(trainloss, label = "Training", color='blue')
ax[1].plot(trainacc, label = "Training", color='blue')
ax[0].plot(testloss, label = "testing", color='darkorange')
ax[1].plot(testacc, label = "testing", c='darkorange')
ax[1].set_xticks(np.arange(0,201,20))
ax[0].set_xticks(np.arange(0,201,20))
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('Pseudo Resnet20 loss')
ax[1].set_title('Pseudo Resnet 20 accuracy')
# plt.suptitle('Resnet50 pretrained model with additional trained layers to classify the bee images')
ax[0].legend()
ax[1].legend();
```



In [29]:
```
%tensorboard --logdir logs
```