```
      client.publish("eFishFarm/oxygenPump", oxygenPumpState_buff);
      client.publish("eFishFarm/heatPump", heatPumpState_buff);
      client.publish("eFishFarm/colderPump", colderPumpState_buff);
      client.publish("eFishFarm/feederPump",feederpumpState_buff);
    } else {
     Serial.print("failed, rc=");
     Serial.print(client.state());
     Serial.println(" try again in 5 seconds");
     // Wait 5 seconds before retrying
     //delay(50000);
    }
  }
}
```

C) Convolutional Neural Network for Fish Diseases Classification

```
import glob
import pandas as pd
import numpy as np
import cv2
from keras.utils import to_categorical
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.image as mpimg
from sklearn.model_selection import train_test_split
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

classes = ['Epizootic ulcerative syndrome (EUS)', 'Ichthyophthirius multifiliis
(Ich)']
num_classes = len(classes)

#train_path='training_data'

# validation split
validation_size = 0.2
test_size=0.2
# batch size
batch_size =4

img_size=32
num_channels=3

num_iteration=70

#data = dataset.read_train_sets(train_path, img_size, classes,
validation_size=validation_size)
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.2989, 0.5870, 0.1140])

imgs = []
imgs1 = []
label = []
data = glob.glob('train_for_2_Classes32x32/*')
print("Train DataSet Size:",len(data))
labels_main = pd.read_csv('trainLabelsFor2Classes32x32.csv')
print(labels_main.head(150)) #print the first 5 data labels
labels = labels_main.iloc[:,1].tolist() # convert labels arrays to list

#Finally, you will convert these integer values into one-hot encoding values
# using the to_categorical function.
#conversion = {'Epizootic ulcerative syndrome (EUS)':0,'Ichthyophthirius multifiliis
(Ich)':1,'Columnaris':2}

conversion = {'Epizootic ulcerative syndrome (EUS)':0,'Ichthyophthirius multifiliis
(Ich)':1}
num_labels = []
```

```
num_labels.append([conversion[item] for item in labels])
num_labels = np.array(num_labels)
#print(num_labels)

label_one = to_categorical(num_labels)
label_one = label_one.reshape(-1,2)
print(label_one.shape) # (600, 2)

#Now you will read the images from the train folder by looping one-by-one using OpenCV
# and store them in a list, and finally, you will convert that list into a NumPy
array.
# The shape of your final output should be (1000, 32, 32, 1).
for i in data:
    img = cv2.imread(i)
    if img is not None:
        imgs.append(img)
        img1= rgb2gray(img) #convert into grayscale
        imgs1.append(img1)

train_imgs = np.array(imgs)
train_imgs1 = np.array(imgs1)

#print("Training dataset size: ", train_imgs1.shape)# (1000, 32, 32, 1)

'''
plt.figure(figsize=[10,10])


# Display the first image in training data
plt.subplot(121)
curr_img = np.reshape(train_imgs[25], (32,32,3))
curr_lbl = labels_main.iloc[0,1]
plt.imshow(curr_img)
plt.title("(Label: " + str(curr_lbl) + ")")

# Display the second image in training data
plt.subplot(122)
curr_img = np.reshape(train_imgs[1], (32,32,3))
curr_lbl = labels_main.iloc[1,1]
plt.imshow(curr_img)
plt.title("(Label: " + str(curr_lbl) + ")")
'''
# Normalize your grayscale images between 0 and 1 before you feed them into the
convolution neural network.
train_images = train_imgs / np.max(train_imgs)
print(np.max(train_images), np.min(train_images))  # (1.0, 0.0)

# Split the 600 images into training & testing images with a 20% split,
#  which means the model will be trained on 800 images and tested on 200 images.
X_train, X_test, y_train, y_test = train_test_split(train_images, label_one,
test_size=0.20, random_state=42)

#X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.20,
random_state=42)

train_X = X_train.reshape(-1, 32, 32, 3)
test_X = X_test.reshape(-1, 32, 32, 3)
#val_X = X_val.reshape(-1, 32, 32, 3)

        #
*********************************************************************************
print("Training Data set shape:",train_X.shape)
print("Testing Data set shape:", test_X.shape)
#print("Validation Data set shape:", val_X.shape)

print("y_train:\n",y_train)

train_y = y_train
test_y = y_test
```

```
#val_y=y_val

#print("Train Data set label shape:", train_y.shape)
#print("Test Data set label shape:", test_y.shape)
#print(train_y)
#print("-----------------------------------------------")
#print(test_y)

def create_weights(shape):
    return tf.Variable(tf.truncated_normal(shape, stddev=0.05))

def create_biases(size):
    return tf.Variable(tf.constant(0.05, shape=[size]))

def create_convolutional_layer(input,
               num_input_channels,
               conv_filter_size,
               num_filters):

    ## We shall define the weights that will be trained using create_weights function.
    weights = create_weights(shape=[conv_filter_size, conv_filter_size,
num_input_channels, num_filters])
    ## We create biases using the create_biases function. These are also trained.
    biases = create_biases(num_filters)

    ## Creating the convolutional layer
    layer = tf.nn.conv2d(input=input,
                     filter=weights,
                     strides=[1, 1, 1, 1],
                     padding='SAME')

    layer += biases

    ## We shall be using max-pooling.
    layer = tf.nn.max_pool(value=layer,
                            ksize=[1, 2, 2, 1],
                            strides=[1, 2, 2, 1],
                            padding='SAME')
    ## Output of pooling is fed to Relu which is the activation function for us.
    layer = tf.nn.relu(layer)

    return layer


def create_flatten_layer(layer):
    layer_shape = layer.get_shape()
    num_features = layer_shape[1:4].num_elements()
    layer = tf.reshape(layer, [-1, num_features])

    return layer



def create_fc_layer(input,
             num_inputs,
             num_outputs,
             use_relu=True):

    #Let's define trainable weights and biases.
    weights = create_weights(shape=[num_inputs, num_outputs])
    biases = create_biases(num_outputs)

    layer = tf.matmul(input, weights) + biases
    if use_relu:
        layer = tf.nn.relu(layer)

    return layer
```

```
# Placeholders and input
x = tf.placeholder(tf.float32, shape=[None, img_size,img_size,num_channels], name='x')

y_true = tf.placeholder(tf.float32, shape=[None, num_classes], name='y_true')
y_true_cls = tf.argmax(y_true, dimension=1)

filter_size_conv1=3
num_filters_conv1=32

filter_size_conv2=3
num_filters_conv2=64

filter_size_conv3=3
num_filters_conv3=128

fc_layer_size=128

#Network Design
layer_conv1 = create_convolutional_layer(input=x,
               num_input_channels=num_channels,
               conv_filter_size=filter_size_conv1,
               num_filters=num_filters_conv1)

layer_conv2 = create_convolutional_layer(input=layer_conv1,
               num_input_channels=num_filters_conv1,
               conv_filter_size=filter_size_conv2,
               num_filters=num_filters_conv2)

layer_conv3= create_convolutional_layer(input=layer_conv2,
               num_input_channels=num_filters_conv2,
               conv_filter_size=filter_size_conv3,
               num_filters=num_filters_conv3)

layer_flat = create_flatten_layer(layer_conv3)

layer_fc1 = create_fc_layer(input=layer_flat,
                     num_inputs=layer_flat.get_shape()[1:4].num_elements(),
                     num_outputs=fc_layer_size,
                     use_relu=True)

layer_fc2 = create_fc_layer(input=layer_fc1,
                     num_inputs=fc_layer_size,
                     num_outputs=num_classes,
                     use_relu=False)

y_pred = tf.nn.softmax(layer_fc2,name="y_pred") # Prediction
y_pred_cls = tf.argmax(y_pred, dimension=1) # The class having higher probability is 
the prediction of the network

'''
Now, let’s define the cost that will be minimized to reach the optimum value of 
weights. We will use a simple cost that will be calculated using a Tensorflow function 
softmax_cross_entropy_with_logits which takes the output of last fully connected layer 
and actual labels to calculate cross_entropy whose average will give us the cost.
'''
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=y_pred,labels=y_true)
cost = tf.reduce_mean(cross_entropy)

'''
We shall use AdamOptimizer for gradient calculation and weight optimization. We shall 
specify that we are trying to minimise cost with a learning rate of 0.0001
'''
optimizer = tf.train.AdamOptimizer(learning_rate=1e-4).minimize(cost)

correct_prediction = tf.equal(y_pred_cls, y_true_cls)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
'''
val_acc = session.run(accuracy,feed_dict=feed_dict_validate)
acc = session.run(accuracy, feed_dict=feed_dict_train)
```

```
'''

# Training function
# Initializing the variables
init = tf.global_variables_initializer()

#def train(num_iteration):
with tf.Session() as sess:
    sess.run(init)
    train_loss = []
    test_loss = []
    #val_loss=[]
    train_accuracy = []
    test_accuracy = []
    #val_accuracy=[]
    summary_writer = tf.summary.FileWriter('./Output', sess.graph)
    #global total_iterations

    for i in range(num_iteration):
        for batch in range(len(train_X)//batch_size):
            #batch1=batch

            batch_x = train_X[batch*batch_size:min((batch+1)*batch_size,len(train_X))]
            batch_y = train_y[batch*batch_size:min((batch+1)*batch_size,len(train_y))]

            #batch_val =
batch_x[batch1*batch_size:min((batch1+1)*batch_size,len(batch_x))]
            #batch_val_y =
batch_y[batch1*batch_size:min((batch1+1)*batch_size,len(batch_y))]

            feed_dict_tr = {x: batch_x,
                            y_true: batch_y}

            opt = sess.run(optimizer, feed_dict= feed_dict_tr)
            '''
            if i % int(data.train.num_examples/batch_size) == 0:
                val_loss = session.run(cost, feed_dict=feed_dict_val)
                epoch = int(i / int(data.train.num_examples/batch_size))

                show_progress(epoch, feed_dict_tr, feed_dict_val, val_loss)
                saver.save(session, 'dogs-cats-model')
            '''
            loss, acc = sess.run([cost, accuracy], feed_dict=feed_dict_tr)

            # Calculate accuracy for all  validation images
            #feed_dict_val = {x: batch_val,  y_true: batch_val_y}
            #opt_val = sess.run(optimizer, feed_dict= feed_dict_val)
            #valid_loss1,val_acc= sess.run([cost,accuracy], feed_dict=feed_dict_val)

        print("Iter " + str(i) + ", Loss= " + \
                      "{:.6f}".format(loss) + ", Training Accuracy= " + \
                      "{:.5f}".format(acc))


        '''

        train_loss.append(loss)
        val_loss.append(valid_loss1)
        val_accuracy.append(val_acc)
        print("Validation Accuracy:","{:.5f}".format(val_acc))
        '''
        print("Optimization Finished!")

        # Calculate accuracy for all  test images
        test_acc,valid_loss = sess.run([accuracy,cost], feed_dict={x: test_X,y_true :
test_y})
        train_loss.append(loss)
        test_loss.append(valid_loss)
        train_accuracy.append(acc)
```

```
            test_accuracy.append(test_acc)
            print("Testing Accuracy:","{:.5f}".format(test_acc))

    #train(num_iteration)
    #Create a saver object which will save all the variables
    saver = tf.train.Saver()
    saver.save(sess, 'Output/the_model')
    # loss plots between training and validation data
    plt.plot(range(len(train_loss)), train_loss, 'b', label='Training loss')
    plt.plot(range(len(train_loss)), test_loss, 'r', label='Test loss')
    plt.title('Training and Test loss')
    plt.xlabel('Epochs ',fontsize=16)
    plt.ylabel('Loss',fontsize=16)
    plt.legend()
    plt.figure()
    plt.show()

    # plot the accuracy between training and validation data
    plt.plot(range(len(train_loss)), train_accuracy, 'b', label='Training Accuracy')
    plt.plot(range(len(train_loss)), test_accuracy, 'r', label='Test Accuracy')
    plt.title('Training and Test Accuracy')
    plt.xlabel('Epochs ',fontsize=16)
    plt.ylabel('Loss',fontsize=16)
    plt.legend()
    plt.figure()
    plt.show()

saver = tf.train.import_meta_graph('Output/the_model.meta')
#saver.restore(sess, tf.train.latest_checkpoint('./'))

'''
 Use an image for prediction
'''
images = []
image =cv2.imread("train_for_2_Classes32x32/54.jpeg")
# Resizing the image to our desired size and
# preprocessing will be done exactly as done during training
image = cv2.resize(image, (img_size, img_size), cv2.INTER_LINEAR)
images.append(image)
images = np.array(images, dtype=np.uint8)
images = np.array(images,dtype=np.float32)
images = np.multiply(images, 1.0/255.0)
#The input to the network is of shape [None image_size image_size num_channels]. Hence
we reshape.
x_batch = images.reshape(1, img_size,img_size,num_channels)


graph = tf.get_default_graph()

y_pred = graph.get_tensor_by_name("y_pred:0")

## Let's feed the images to the input placeholders
x= graph.get_tensor_by_name("x:0")
y_true = graph.get_tensor_by_name("y_true:0")
y_test_images = np.zeros((1, 2))

feed_dict_testing = {x: x_batch, y_true: y_test_images}

init = tf.global_variables_initializer()

#def train(num_iteration):
with tf.Session() as sess:
    sess.run(init)
    result=sess.run(y_pred, feed_dict=feed_dict_testing)
    print("Prediction Result:",result)
```