NAME-ISTIAQUE MANNAFEE SHAIKAT
MATRICULATION NUMBER-303527

Part 1: Research Paper (5 Points)

Architecture design:

In this paper the authors divided multivariate time series to uni variate ones and performed feature learning on each uni-variate series individually. Then they concatenate normal MLP at the end of feature learning. In this paper,3 channel inputs and each length is 256 which is fed into 2 stages of feature extractor and learns hierarchical features through filter, activation and pooling layers. For the MC-DNCC model simple back-propagation algorithm is used to train the model and SGD is used to update the parameters.
The loss function is (-tf.reduce_sum(Y * tf.log(y_conv))). To optimize the loss function the author used gradient based optimization.
For parameter updating they used three phases they are feed forward, back propagation pass the gradients applied.
Feed forward pass:

- Determine the predicted output of MC-DCNN on input vectors
- computes features maps from layer to layer and stage to stage until the output is obtained.

Back propagation Pass:

- After predicted y is calculated, the predicted error E can be calculated using loss function
- Chain-rule of derivative is used to back propagate each parameter layers one by one to get the derivatives

Once the derivative of parameters is obtained gradients is applied to to converge fast and the weight is updated in MLP.

Part 2: Research Paper Implementation (15 points)

Explanation of the code:

1. Importing the data and dropped the NA values

```
#*****importing all the data and merged into one dataframe except subject 108 and subject 109**********
path = r'E:\Documents\University of Hildesheim\Distributed data analytics\lab6\PAMAP2_Dataset\Protocol'
all_files = glob.glob(path + "/*.dat")
data = []
X=[]
for filename in all_files:
    df=pd.read_csv(filename,header=None,index_col=None,sep=r"\s+")
    df=df.convert_objects(convert_numeric=True)
    df=df.dropna()
    data.append(df)
```

2. Selecting specific activity ID

   The author selected specific activity ID to compress the data or it will months to train the model.

```
#****************selecting specific ID******************
act_id=[3,4,12,13]
data_four_act=[]
for x in range (len(data)):
    data_2=data[x]
    for i in range (len(act_id)):
#         print(i)
        data_four_act.append(data_2[data_2[1] ==act_id[i]])
    X.append(pd.concat(data_four_act, axis=0, ignore_index=True))
```

3. Function Declaration

   The author normalized the feature using mean and standard deviation.

```
# FUNCTION DECLARATION
def feature_normalize(dataset):
    mu = np.mean(dataset,axis = 0)
    sigma = np.std(dataset,axis = 0)
    return (dataset - mu)/sigma
```

For tensor board summary and sliding window algorithm the following code is used

```
def window(data, size):
    start = 0
    while start < len(data):
        yield start, start + size
        start += int(size / 2)
def slide_window(x_train,y_train,window_size):
    segments = np.zeros(((len(x_train)//(window_size//2))-1,window_size,3))
    labels = np.zeros(((len(y_train)//(window_size//2))-1))
    i_segment = 0
    i_label = 0

    for (start,end) in window(x_train,window_size):
#         print(start,end)
        if(len(x_train[start:end]) == window_size):
            m = stats.mode(y_train[start:end])
            segments[i_segment] = x_train[start:end]
            labels[i_label] = m[0]
            i_label+=1
            i_segment+=1
    return segments, labels
```

Weight,bias conv and max_pool functions are created for the ease of coding

```
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev = 0.1)
    return tf.Variable(initial)

def bias_variable(shape):
    initial = tf.constant(0.1, shape = shape)
    return tf.Variable(initial)

def depth_conv2d(x, W):
  return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='VALID')

def max_pool(x,kernel_size,stride_size):
  return tf.nn.max_pool(x, ksize=[1, 1, kernel_size, 1],
                        strides=[1, 1, stride_size, 1], padding='VALID')
```

4. Leave one out Cross validation ,sliding window and one hot encoding of label
   - The data is split into train and test by leave one out cross validation which take 6 subjects as train data and 1 as test data.
   - Then the data is normalized for the IMU Hand 3D column.
   - Then comes the sliding windows with size of 256 and step size of 128,64,32,16,8 which is varied to get the accuracy
   - The label is one hot encoded to get better accuracy which was not done by the author.

```
loo = LeaveOneOut()
for train_index, test_index in loo.split(X):
   train_X=[]

   print("TRAIN:", train_index, "TEST:", test_index)
   for i in range (len(train_index)):
       train_2=[]
       train_2.append(X[train_index[i]])
   train_X=pd.concat(train_2, axis=0, ignore_index=True)
   trainx=(feature_normalize(train_X.iloc[:, np.r_[4:7]]))##**************3D column normalization***************
   test_X=X[test_index[0]]
   testx=(feature_normalize(test_X.iloc[:, np.r_[4:7]]))##**************3D column normalization***************
   trainy=train_X[1].values
   testy=test_X[1].values

   window_size=256
   print ("sliding window........")
   train_x, train_y = slide_window(trainx,trainy,window_size)
   test_x, test_y = slide_window(testx,testy,window_size)

   train_1 = pd.get_dummies(train_y)
   test = pd.get_dummies(test_y)
   train_1, test = train_1.align(test, join='inner', axis=1)
   train_y = np.asarray(train_1)
   test_y = np.asarray(test)
```

5.Convolution Neural Network

In this part the CNN code is written which has two hidden layer.After getting the final pool2 we take the shape of the last layer and then flatten it out to findout y_conv

```
input_height = 1
window_size = window_size
num_labels = 4
num_channels = 3
batch_size = 128
stride_size = 2
kernel_size_1 = 7
kernel_size_2 = 3
kernel_size_3 = 1
depth_1 = 256
depth_2 = 256
depth_3 = 256
num_hidden = 512 # neurons in the fully connected layer
dropout_1 = tf.placeholder(tf.float32)
dropout_2 = tf.placeholder(tf.float32)
dropout_3 = tf.placeholder(tf.float32)

learning_rate = 0.0005
training_epochs = 10
total_batches = train_x.shape[0] // batch_size
train_x = train_x.reshape(len(train_x),1, window_size,num_channels) # opp
test_x = test_x.reshape(len(test_x),1, window_size,num_channels) # opport

X = tf.placeholder(tf.float32, shape=[None,input_height,window_size,num_c
Y = tf.placeholder(tf.float32, shape=[None,num_labels])

# hidden layer 1
W_conv1 = weight_variable([1, kernel_size_1, num_channels, depth_1])
b_conv1 = bias_variable([depth_1])
h_conv1 = tf.nn.relu(depth_conv2d(X, W_conv1) + b_conv1)
# h_conv1 = tf.nn.dropout(tf.identity(h_conv1), dropout_1)
h_conv1 = tf.nn.dropout(h_conv1, dropout_1)
h_pool1 = max_pool(h_conv1,kernel_size_1,stride_size)

# hidden layer 2
W_conv2 = weight_variable([1, kernel_size_2, depth_1, depth_2])
b_conv2 = bias_variable([depth_2])
h_conv2 = tf.nn.relu(depth_conv2d(h_pool1, W_conv2) + b_conv2)
h_conv2 = tf.nn.dropout(h_conv2, dropout_2)
h_pool2 = max_pool(h_conv2,kernel_size_2,stride_size)

# fully connected layer

 #first we get the shape of the last layer and flatten it out
shape = h_pool2.get_shape().as_list()
 print ("shape's shape:", shape)

W_fc1 = weight_variable([shape[1] * shape[2] * shape[3],num_hidden])
b_fc1 = bias_variable([num_hidden])

h_pool3_flat = tf.reshape(h_pool2, [-1, shape[1] * shape[2] * shape[3]])
 #print "c_flat shape =",h_pool3_flat.shape
h_fc1 = tf.nn.relu(tf.matmul(h_pool3_flat,W_fc1) + b_fc1)
h_fc1 = tf.nn.dropout(h_fc1, dropout_3)

 #readout layer.
W_fc2 = weight_variable([num_hidden,num_labels])
b_fc2 = bias_variable([num_labels])
y_conv = tf.matmul(h_fc1,W_fc2) + b_fc2
```

For the loss function the author used cross entropy and optimizer SGD then correct_pred and accuracy is calculated

```
 # COST FUNCTION
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=Y, logits=y_conv))
 loss=tf.reduce_mean(tf.sqrt(tf.square(Y - y_conv)))
optimizer = tf.train.GradientDescentOptimizer(learning_rate = learning_rate).minimize(loss)
 optimizer = tf.train.AdamOptimizer(learning_rate = learning_rate).minimize(loss)
correct_prediction = tf.equal(tf.argmax(y_conv,1), tf.argmax(Y,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

Then the data are made into batch to be fed into the optimizer for minimize the loss

```
with tf.Session() as session:

    # merged_summary_op = tf.summary.merge_all()
    # summary_writer = tf.summary.FileWriter("./", session.graph)
    tf.initialize_all_variables().run()

    for epoch in range(training_epochs):

        cost_history = np.empty(shape=[0],dtype=float)
        for b in range(total_batches):
            offset = (b * batch_size) % (train_y.shape[0] - batch_size)
            batch_x = train_x[offset:(offset + batch_size), :, :, :]
            batch_y = train_y[offset:(offset + batch_size), :]

            # print "batch_x shape =",batch_x.shape
            # print "batch_y shape =",batch_y.shape
            # _, c, summary= session.run([optimizer, loss,merged_summary_op],feed_dic
            # cost_history = np.append(cost_history,c)
            _, c= session.run([optimizer, loss],feed_dict={X: batch_x, Y : batch_y})
            cost_history = np.append(cost_history,c)
```

RESULT

STEP	MC-DCNN(1)	MC-DCNN(2)
128	88.13	88.31
64	90.22	84.00
32	89.34	82.32
16	90.12	88.77
8	90.33	93.32

REASON FOR NOT ABLE TO REPRODUCE THE SAME RESULT

I made few changes while implementing the paper like

- I choose 3D acceleration for IMU Hand as in the paper it was not written clearly which IMU they used for feature learning.
- One-hot encoding of label data to get better accuracy
- Use of different architecture – instead passing each channel in feature extraction I passed all the channels in the together.