WEARABLE TECHNOLOGY

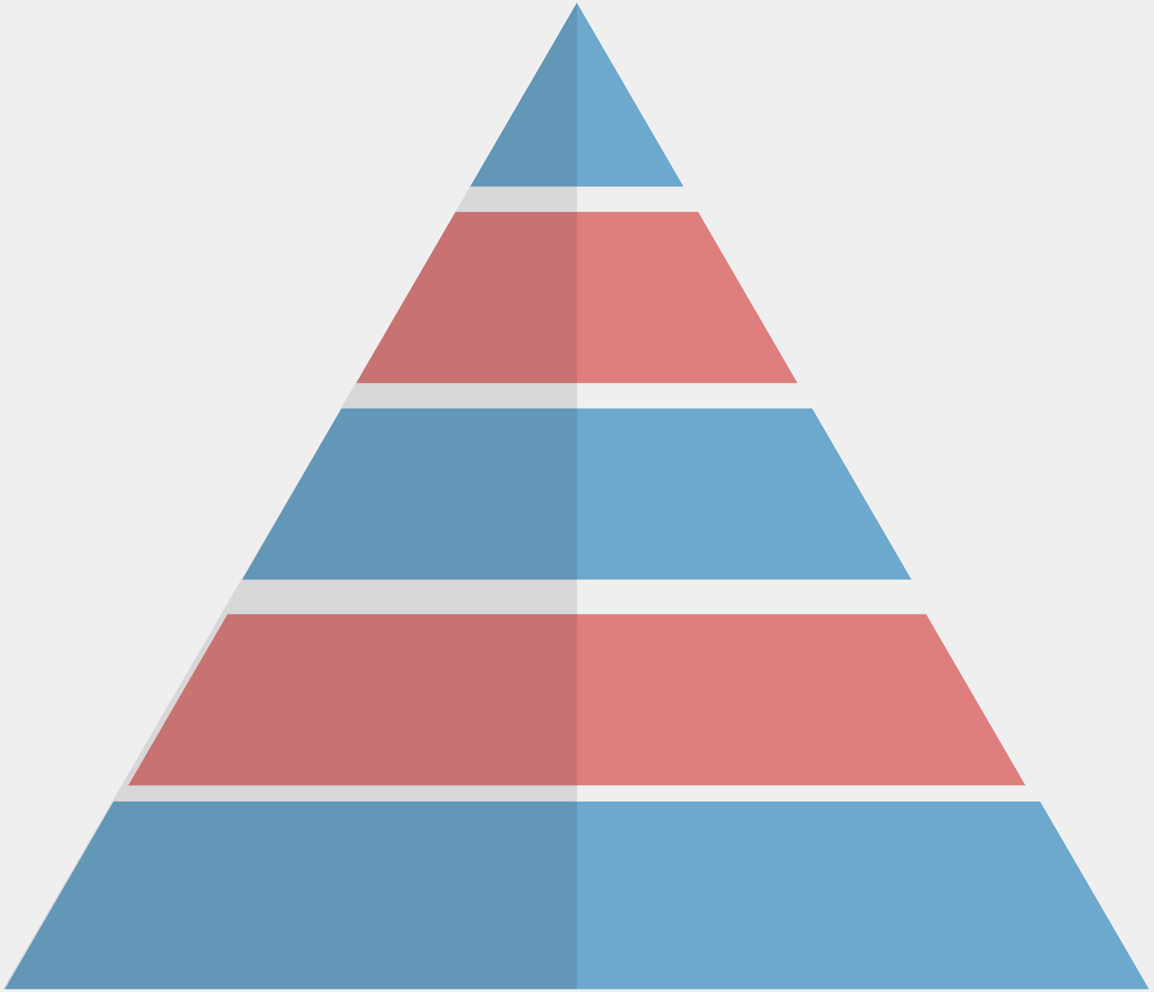
By: Qixiao Zhang, Ketan Vanjani, Jay Patel

Contents

PART 01

Mission Statement

Mission Statement



Goal?

People can control robotic prostheses with their minds.

How to do?

Machine Learning

What to do?

Identify the hand gesture with EMG signal

Why this?

Amputated people need prostheses to make their live more convenient

PART 02

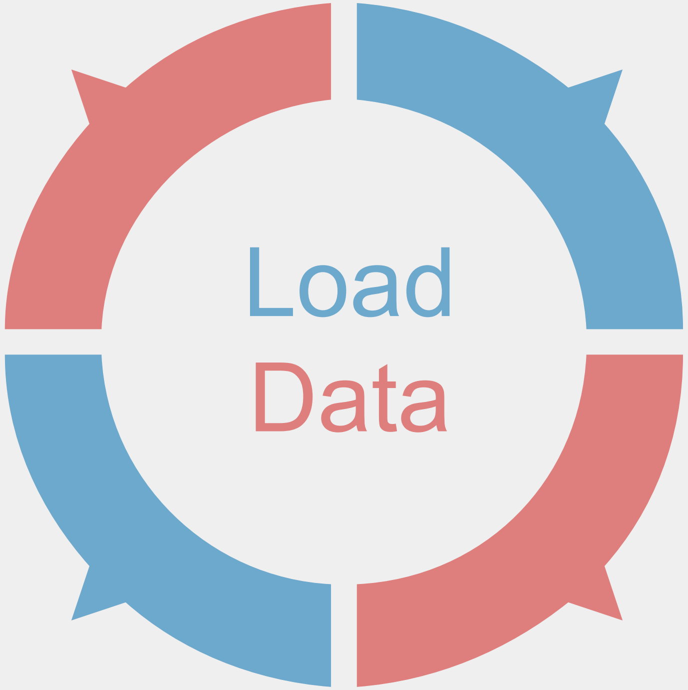
Data Introduction

Data Intro

Ninapro is a publicly available multimodal database to foster research on robotic & prosthetic hands controlled with artificial intelligence.

Ninapro includes electromyography, kinematic, inertial, eye tracking, visual, clinical and neurocognitive data.
Ninapro data are used worldwide by scientific researchers in machine learning, robotics, medical and neurocognitive sciences.

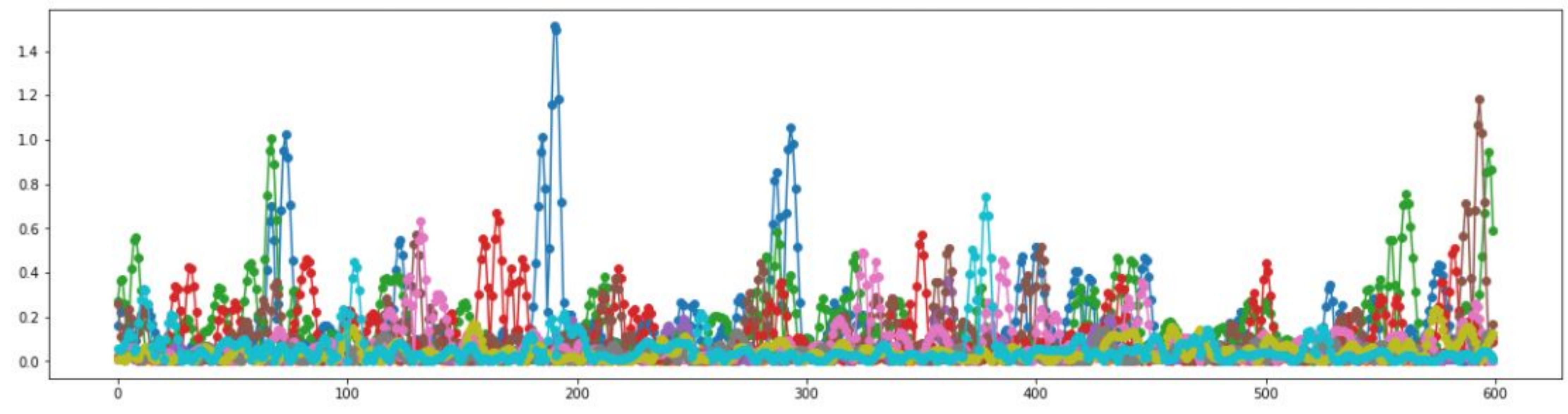
Data Intro




```python
def load_data():

    X_train = np.load("/content/drive/MyDrive/subject 11/subject11_train_matrix.npy.part",mmap_mode='r',allow_pickle=True)
    X_test = np.load("/content/drive/MyDrive/subject 11/subject11_test_matrix.npy.part",mmap_mode='r',allow_pickle=True)
    y_train = np.load("/content/drive/MyDrive/subject 11/subject11_train_labels.npy",mmap_mode='r',allow_pickle=True)
    y_test = np.load("/content/drive/MyDrive/subject 11/subject11_test_labels.npy",mmap_mode='r',allow_pickle=True)

    # Reduce sample size

    # X_train = X_train[np.random.permutation(44000)[:2000],:,:]
    # X_test = X_test[np.random.permutation(22000)[:700],:,:]
    # y_train = y_train[np.random.permutation(44000)[:2000]]
    # y_test = y_test[np.random.permutation(22000)[:700]]

    # if spec == True:
    #     X_train = Win2Spec(X_train)
    #     X_test = Win2Spec(X_test)

    # if GAF == True:
    #     X_train = Win2GAF(X_train)
    #     X_test = Win2GAF(X_test)

    # y_train = pd.get_dummies(y_train)
    # y_test = pd.get_dummies(y_test)
    y_train = get_categorical(y_train)
    y_test = get_categorical(y_test)

    X_train = X_train.astype('float32')
    X_test = X_test.astype('float32')
    y_train = y_train.astype('float32')
    y_test = y_test.astype('float32')

    return X_train, X_test, y_train, y_test
```

Data set has three dimensions:
(samples, windows, channels)

Data Intro

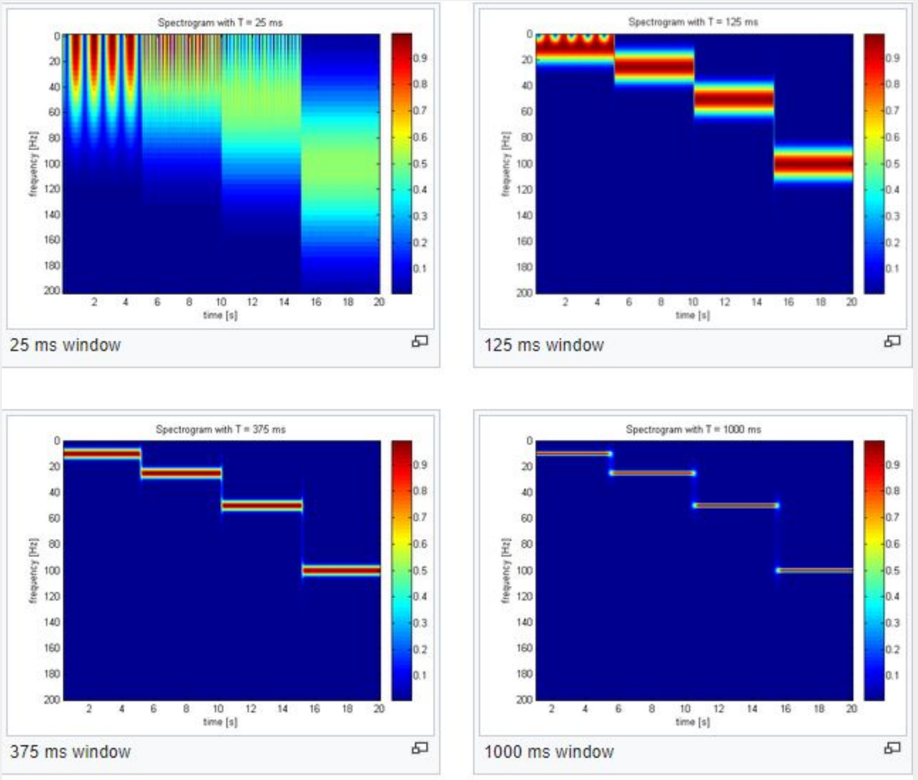


Data Preview

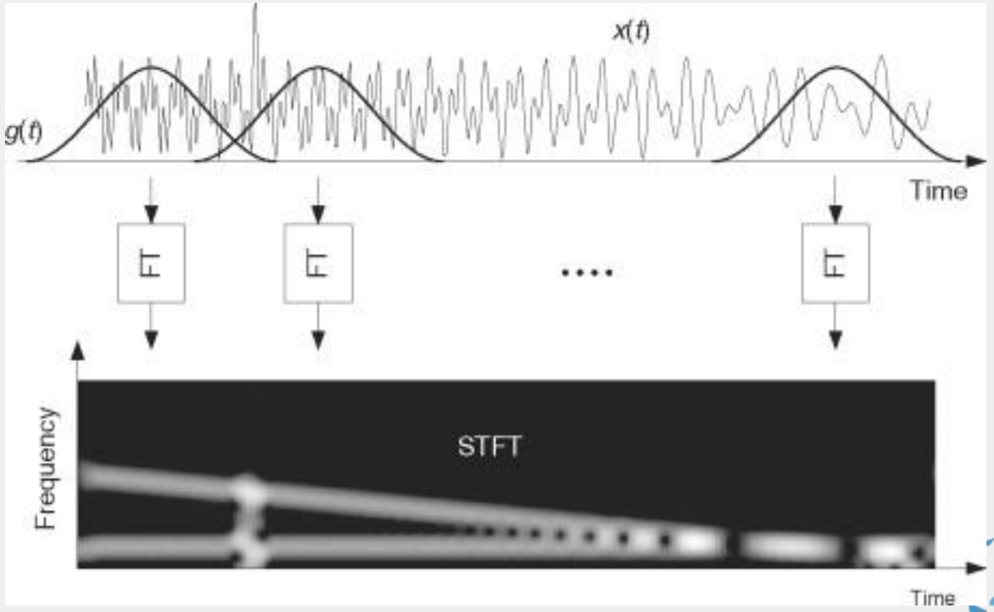
```
plt.figure(figsize=(15,5))
rdm = np.random.randint(0,20000,10)
for i in rdm:
    plt.plot(X_train[i,:,0],'-o')
```

Short Time Fourier Transform (STFT)

The STFT is a Fourier Transform technique used to determine the frequency and phase of certain sections of a given signal.
This gives a Fourier spectrum, which is often then viewed on a spectrogram.

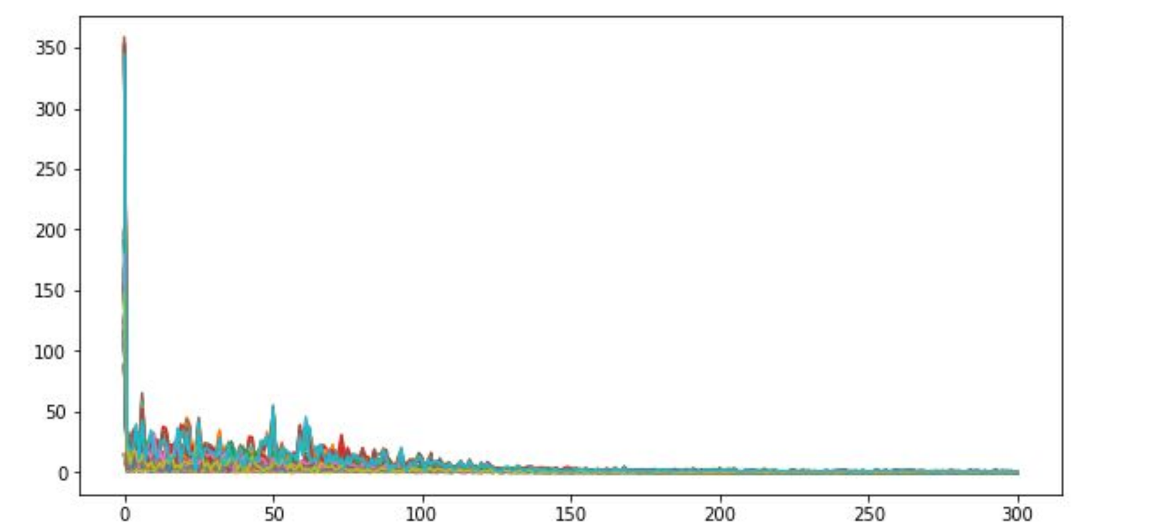


Python Command:
scipy.signal.stft(x,window='hann'...)



Implement FFT on the data

```
plt.figure(figsize=(10,5))
#  plt.plot(X_train[:,0,0])
for  i  in  range(10):
    for  j  in  range(12):
        x_fft  =  np.fft.rfft(X_train[i,:,j])
        plt.plot(abs(x_fft))
```



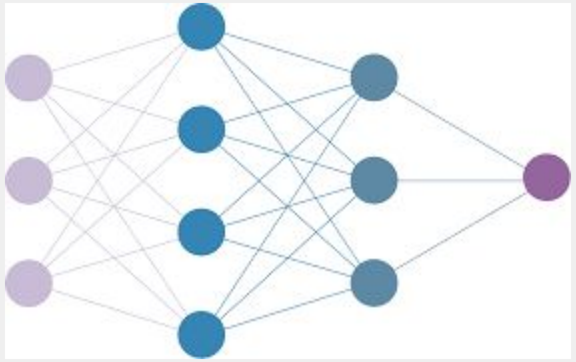
PART 03

Model Construction

Network

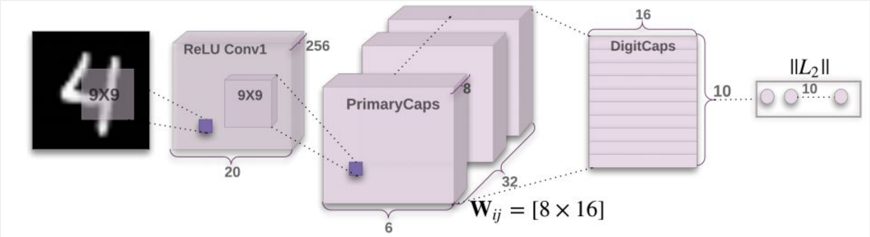
Neural Network

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates.



Convolutional Neural Network

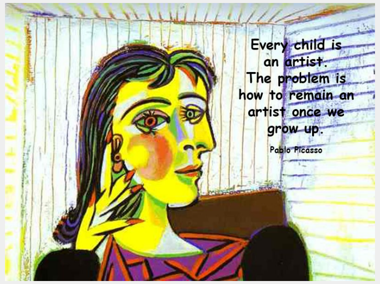
In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of artificial neural network, most applied to analyze visual imagery.

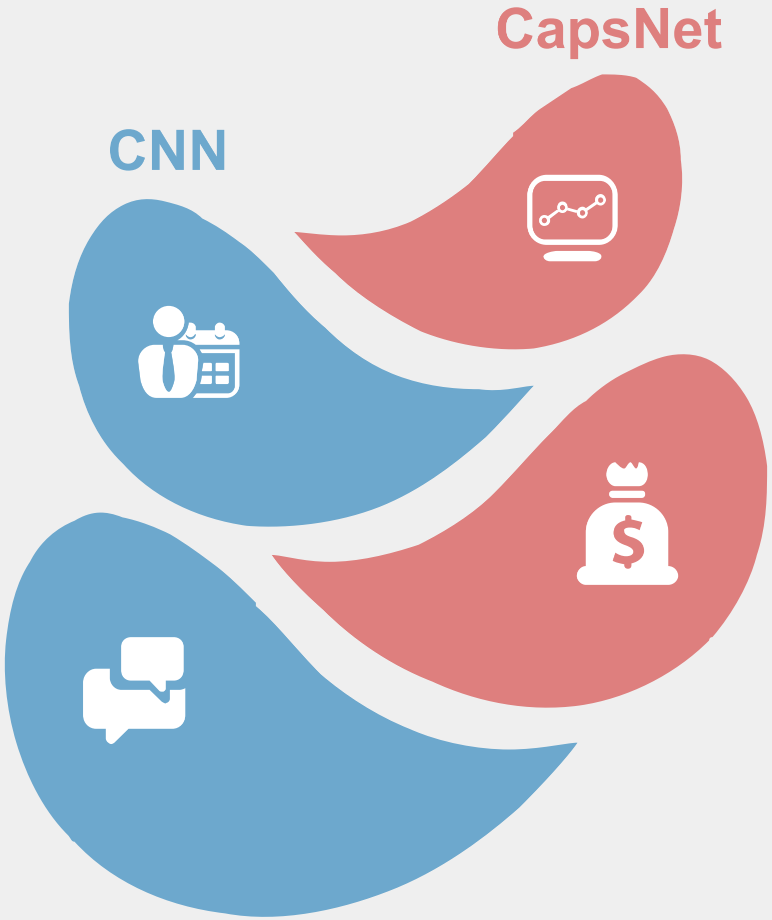


CapsNet

A Capsule Neural Network (CapsNet) is a machine learning system that is a type of artificial neural network (ANN) that can be used to better model hierarchical relationships. The approach is an attempt to more closely mimic biological neural organization.

CNN vs CapsNet





◆ Lack of context

If the beak is located on the tail, for example, the classifier will still consider the image as one of a bird (although, obviously, no bird carries its beak on its tail).

◆ Information loss

by which meaningful pixels (or neurons) are discarded since max pooling only select one neuron or one pixel from a cluster.

◆ Output is a vector

Capsule gives us a vector as an output that has a direction. Overcome all the drawbacks that are present on CNN.

◆ Fewer data, Better results

Need fewer data to train same model

CapsNet Code

```python
from keras import layers, models
from keras import backend as K
from tensorflow.keras.utils import to_categorical

def CapsNet(input_shape, n_class, num_routing):
    """
    :param input_shape: data shape, 4d, [None, width, height, channels]
    :param n_class: number of classes
    :param num_routing: number of routing iterations
    :return: A Keras Model with 2 inputs and 2 outputs
    """
    x = layers.Input(shape=input_shape)
    gru = layers.GRU(32)(x)

    # Layer 1: Just a conventional Conv2D layer
    conv1 = tf.keras.layers.Conv2D(filters=128,kernel_size=3, padding='valid',name='conv1',
                                   kernel_initializer= initializers.glorot_uniform(),activation='relu')(gru)
    bn1 = tf.keras.layers.Dropout(0.5)(conv1)
    bn1 = tf.keras.layers.BatchNormalization()(bn1)

    conv2 = tf.keras.layers.Conv2D(filters=128,kernel_size=3,padding='valid',
                                   kernel_initializer= initializers.glorot_uniform(),activation='relu')(bn1)
    bn2 = tf.keras.layers.Dropout(0.5)(conv2)
    bn2 = tf.keras.layers.BatchNormalization()(bn2)

    conv3 = tf.keras.layers.Conv2D(filters=256,kernel_size=3,padding='valid',
                                   kernel_initializer= initializers.glorot_uniform(),activation='relu')(bn2)
    bn3 = tf.keras.layers.Dropout(0.5)(conv3)
    bn3 = tf.keras.layers.BatchNormalization()(bn3)

    conv4 = tf.keras.layers.Conv2D(filters=256,kernel_size=3,padding='valid',
                                   kernel_initializer= initializers.glorot_uniform(),activation='relu')(bn3)
    bn4 = tf.keras.layers.Dropout(0.5)(conv4)
    bn4 = tf.keras.layers.BatchNormalization()(bn4)

    # Layer 2: Conv2D layer with `squash` activation, then reshape to [None, num_capsule, dim_vector]
    primarycaps = PrimaryCap(bn4, dim_capsule=4, n_channels=4, kernel_size=4, strides=2, padding='valid')

    # Layer 3: Capsule layer. Routing algorithm works here.
    digitcaps = CapsuleLayer(num_capsule=n_class, dim_capsule=8, routings=num_routing, name='digitcaps')(primarycaps)


    digit_probs = tf.keras.layers.Lambda(lambda x: tf.norm(x, axis=-1),
                                         name="digit_probs")(digitcaps)

    model = tf.keras.Model(inputs=x,
                           outputs=digit_probs,
                           name="Efficient-CapsNet")


    return model
```

Conv Layer (ReLU)

Primary Cap

Digit Caps

Digit Probs

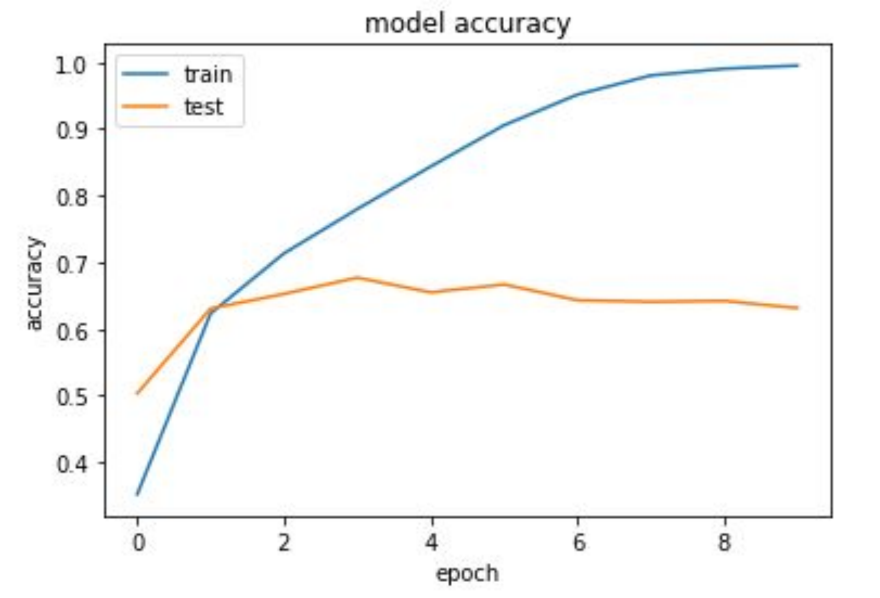
PART 04

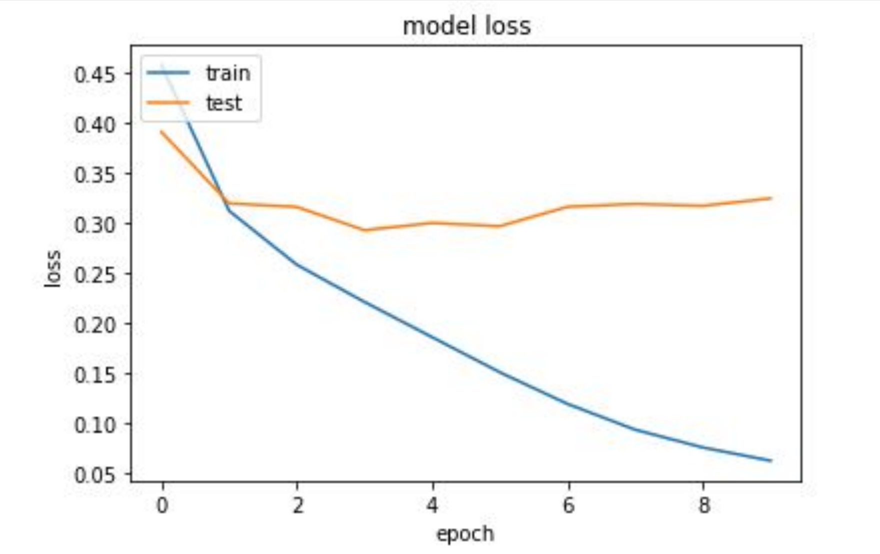
Results and Conclusion

Model training

```
[21] histories, model = train_model(model, X_train, y_train, X_test , y_test, batch_size=batch_size, save_to= 'temp', epochs = epochs)

Epoch 1/10
1377/1377 [==============================] - ETA: 0s - loss: 0.4572 - categorical_accuracy: 0.3522
Epoch 00001: val_categorical_accuracy improved from -inf to 0.50337, saving model to temp_best_model.h5
1377/1377 [==============================] - 223s 157ms/step - loss: 0.4572 - categorical_accuracy: 0.3522 - val_loss: 0.3906 - val_categorical_accuracy: 0.5034
Epoch 2/10
1377/1377 [==============================] - ETA: 0s - loss: 0.3120 - categorical_accuracy: 0.6233
Epoch 00002: val_categorical_accuracy improved from 0.50337 to 0.63058, saving model to temp_best_model.h5
1377/1377 [==============================] - 212s 152ms/step - loss: 0.3120 - categorical_accuracy: 0.6233 - val_loss: 0.3193 - val_categorical_accuracy: 0.6306
Epoch 3/10
1377/1377 [==============================] - ETA: 0s - loss: 0.2583 - categorical_accuracy: 0.7134
Epoch 00003: val_categorical_accuracy improved from 0.63058 to 0.65262, saving model to temp_best_model.h5
1377/1377 [==============================] - 211s 153ms/step - loss: 0.2583 - categorical_accuracy: 0.7134 - val_loss: 0.3158 - val_categorical_accuracy: 0.6526
Epoch 4/10
1377/1377 [==============================] - ETA: 0s - loss: 0.2212 - categorical_accuracy: 0.7799
Epoch 00004: val_categorical_accuracy improved from 0.65262 to 0.67697, saving model to temp_best_model.h5
1377/1377 [==============================] - 210s 152ms/step - loss: 0.2212 - categorical_accuracy: 0.7799 - val_loss: 0.2926 - val_categorical_accuracy: 0.6770
Epoch 5/10
1377/1377 [==============================] - ETA: 0s - loss: 0.1861 - categorical_accuracy: 0.8436
Epoch 00005: val_categorical_accuracy did not improve from 0.67697
1377/1377 [==============================] - 213s 155ms/step - loss: 0.1861 - categorical_accuracy: 0.8436 - val_loss: 0.2998 - val_categorical_accuracy: 0.6549
Epoch 6/10
1377/1377 [==============================] - ETA: 0s - loss: 0.1511 - categorical_accuracy: 0.9056
Epoch 00006: val_categorical_accuracy did not improve from 0.67697
1377/1377 [==============================] - 209s 152ms/step - loss: 0.1511 - categorical_accuracy: 0.9056 - val_loss: 0.2966 - val_categorical_accuracy: 0.6668
Epoch 7/10
1377/1377 [==============================] - ETA: 0s - loss: 0.1196 - categorical_accuracy: 0.9515
Epoch 00007: val_categorical_accuracy did not improve from 0.67697
1377/1377 [==============================] - 210s 153ms/step - loss: 0.1196 - categorical_accuracy: 0.9515 - val_loss: 0.3159 - val_categorical_accuracy: 0.6432
Epoch 8/10
1377/1377 [==============================] - ETA: 0s - loss: 0.0939 - categorical_accuracy: 0.9803
Epoch 00008: val_categorical_accuracy did not improve from 0.67697
1377/1377 [==============================] - 211s 153ms/step - loss: 0.0939 - categorical_accuracy: 0.9803 - val_loss: 0.3188 - val_categorical_accuracy: 0.6409
Epoch 9/10
1377/1377 [==============================] - ETA: 0s - loss: 0.0762 - categorical_accuracy: 0.9904
Epoch 00009: val_categorical_accuracy did not improve from 0.67697
1377/1377 [==============================] - 212s 154ms/step - loss: 0.0762 - categorical_accuracy: 0.9904 - val_loss: 0.3169 - val_categorical_accuracy: 0.6422
Epoch 10/10
1377/1377 [==============================] - ETA: 0s - loss: 0.0630 - categorical_accuracy: 0.9951
Epoch 00010: val_categorical_accuracy did not improve from 0.67697
1377/1377 [==============================] - 215s 156ms/step - loss: 0.0630 - categorical_accuracy: 0.9951 - val_loss: 0.3244 - val_categorical_accuracy: 0.6314
1378/1378 [==============================] - 60s 43ms/step - loss: 0.2119 - categorical_accuracy: 0.8424
691/691 [==============================] - 30s 44ms/step - loss: 0.2926 - categorical_accuracy: 0.6770
Train: 0.842, Test: 0.677
```

Model training






```
# summarize history for accuracy
plt.plot(histories.history['categorical_accuracy'])
plt.plot(histories.history['val_categorical_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
# summarize history for loss
plt.clf
plt.plot(histories.history['loss'])
plt.plot(histories.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

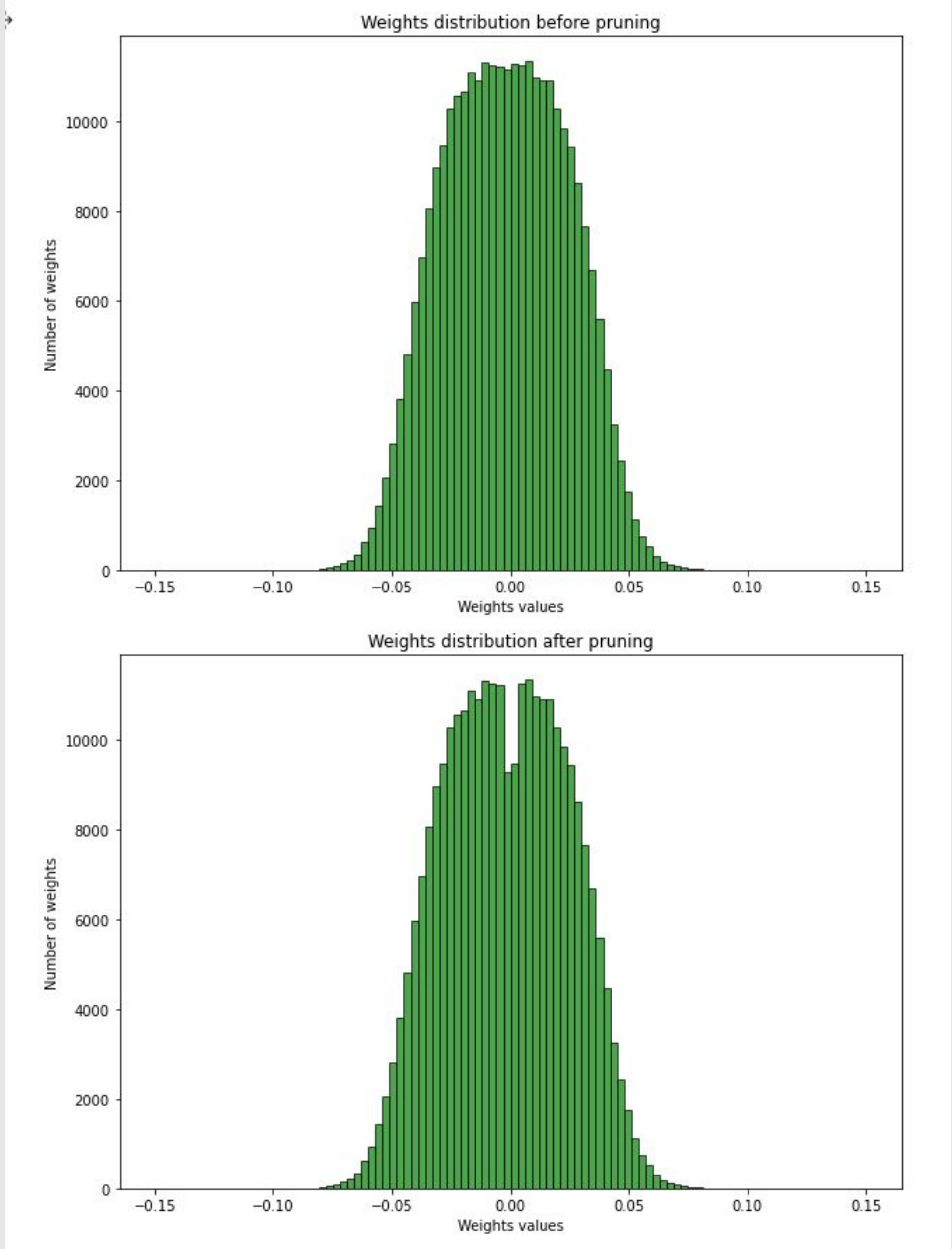
Model Pruning to specific layer

Model Pruning

```python
def plot_histogram(weights_list: list,
                   include_zeros=True,
                   title=''):

    """A function to plot weights distribution"""

    weights = []
    for w in weights_list:
        weights.extend(list(w.ravel()))

    if not include_zeros:
        weights = [w for w in weights if w != 0]

    fig = plt.figure(figsize=(10, 7))
    ax = fig.add_subplot(111)

    ax.hist(weights,
            bins=100,
            facecolor='green',
            edgecolor='black',
            alpha=0.7,
            range=(-0.15, 0.15))

    ax.set_title('Weights distribution {}'.format(title))
    ax.set_xlabel('Weights values')
    ax.set_ylabel('Number of weights')
```

[31]

```python
weights_array_layer7 = model.layers[7].get_weights()[0]
bias_array_layer7 = model.layers[7].get_weights()[1]
# print(model.layers[7].get_weights()[0].shape)
# plt.plot(weights_array_layer7[0,1,:,:])
# np.min(np.abs((model.layers[7].get_weights()[0])))
plot_histogram(weights_array_layer7,include_zeros=False,title='before pruning')
weights_array_layer7_pruned = np.where(np.abs(weights_array_layer7)>5e-4,weights_array_layer7,0)
plot_histogram(weights_array_layer7_pruned,include_zeros=False,title='after pruning')
model.layers[7].set_weights([weights_array_layer7_pruned,bias_array_layer7])
```



Model Pruning to specific layer

Before Pruning:

```
[25] model.fit(X_train,y_train,validation_data=(X_test,y_test))

1378/1378 [==============================] - 212s 153ms/step - loss: 0.1481 - categorical_accuracy: 0.9113 - val_loss: 0.3061 - val_categorical_accuracy: 0.6578
<keras.callbacks.History at 0x7f42c0b55250>
```

After Pruning:

```
model.fit(X_train,y_train,validation_data=(X_test,y_test))

1378/1378 [==============================] - 210s 152ms/step - loss: 0.1171 - categorical_accuracy: 0.9576 - val_loss: 0.2992 - val_categorical_accuracy: 0.6530
<keras.callbacks.History at 0x7f42c0739510>
```

Reference

Big Thanks To:

Professor S. Farokh Atashzar, Werable Technology VIP Softwear Team

"Papers with Code - Ninapro DB2 Dataset." NinaPro DB2 Dataset | Papers With Code, https://paperswithcode.com/dataset/ninapro-db2.

Atzori, M., Gijsberts, A., Castellini, C. et al. Electromyography data for non-invasive naturally-controlled robotic hand prostheses. Sci Data 1, 140053 (2014). https://doi.org/10.1038/sdata.2014.53

Wang, Yiwei et al. "Multitask CapsNet: An Imbalanced Data Deep Learning Method for Predicting Toxicants." ACS omega vol. 6,40 26545-26555. 29 Sep. 2021, doi:10.1021/acsomega.1c03842

Thank you