Deep Learning

SE 4030

Assignment 02

Deep Learning Algorithm - Report

Submitted by:

IT 17012652: K.A.D.K Omalka

IT 17051026: J.M.T Jayasekara

Table of Contents

1. INTRODUCTION

Humans engage in all kinds of different activities while going on with their daily routines. Human activities can be categorized according to basic human needs and wants. Some common activities that can be identified in a person's routine can be listed as eating, drinking, texting, calling, sitting, walking, idling and talking, etc. In some special scenarios, being able to automatically identify these activities separately is of immense importance. When considering situations where a person's activities need to be identified and recorded in a daily or hourly basis, it will be inefficient to have another person to monitor them as it will not guarantee much accuracy or reliability. Several such instances can be considered where a trained model will over rule a human supervisor's abilities in several aspects.

As a solution for this, an activity recognition model has been trained using a supervised deep learning algorithm according to a custom build dataset for some selected activities. This trained model can be utilized in several scenarios where human activity monitoring is required.

In a largely scaled company with several branches in rural areas, it is crucial to monitor and supervise the employees to ensure that the maximum work productivity and efficiency is maintained within the office hours. Although the conventional solution for this is to appoint a branch manager or a supervisor, it will not be the most economical or the most reliable method to monitor the employees' activities. For this, the higher management can easily obtain reports on each and every employee's activities within the branch on a daily basis through analyzing the CCTV camera footages obtained inside the branch with this specially trained activity recognition model.

Monitoring and recording the activities of specially isolated people, dangerous inmates in prisons or mentally ill patients in hospitals who might act in unpredictable or insecure manner is another scenario where a specially trained activity recognition model can be utilized effectively.

The trained activity detection model is currently trained to detect 5 basic activities which are drinking, writing, phone usage, laptop usage and idling. This model has been implemented, trained and tested using Convolutional Neural Networks (CNN2D) with Keras and TensorFlow.

2. METHODOLOGY

2.1 Data Collection

For the training and testing purposes of this activity recognition model, a custom dataset has been created with 5 different classes of activities.

The dataset can be accessed here.

The dataset contains images of several people engaging in the following same 5 activities;

- Drinking
- Writing
- Using a Laptop
- Using a mobile phone
- Idling

Each image class contains around 1000 images and all of these images are of 1280 x 720 pixels dimension with a bit depth of 24.

These images are photographs of office workers that were snapped using mobile phones while being engaged in these activities in their office environment.

As the currently available custom dataset is limited and contains only 5 classes of the very basic human activities, it can be broadened by adding different other activities and by adding more diverse images to each class folder.

The following figure depicts the formation of the custom dataset with the five different classes for training purposes.

Figure 2.1: Training dataset folder

The training dataset folder contains the 5 classed labeled as “d”, “i”, “l”, “p” and “w”, which represents drinking, idling, laptop usage, phone usage and writing respectively.

Some sample data from each of the 5 classes of activities are depicted in the following figures.

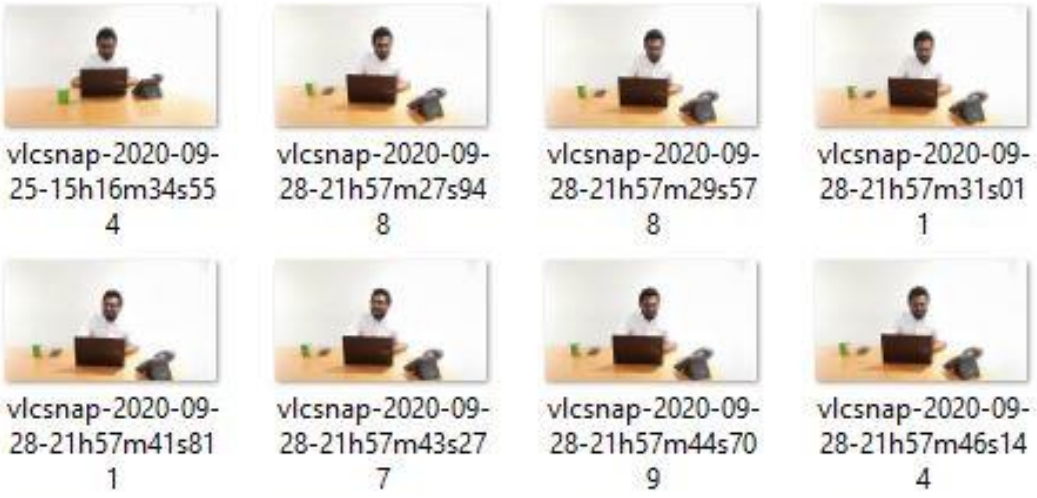


Figure 2.2: "Laptop usage" sample data

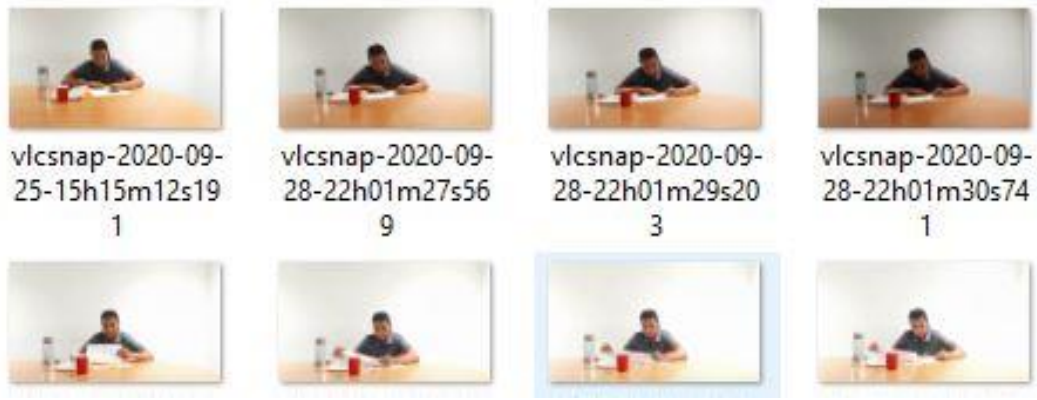


Figure 2.3: "Writing" sample data

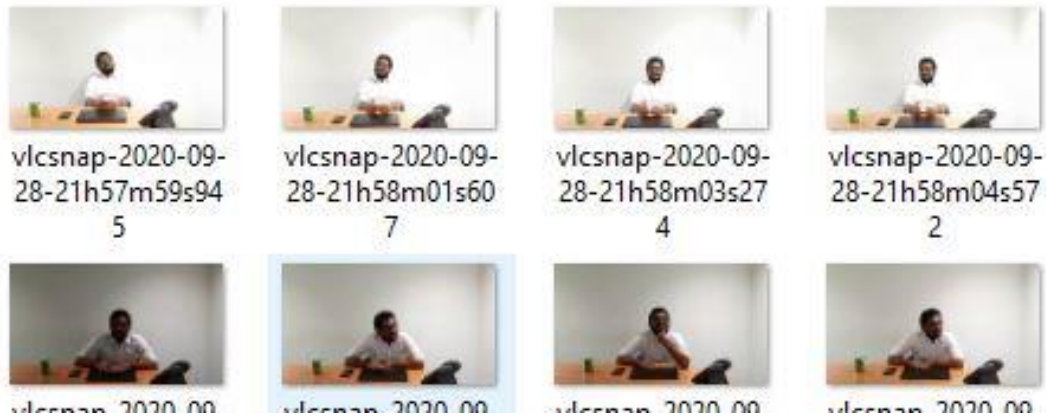


Figure 2.4: "Idling" sample data

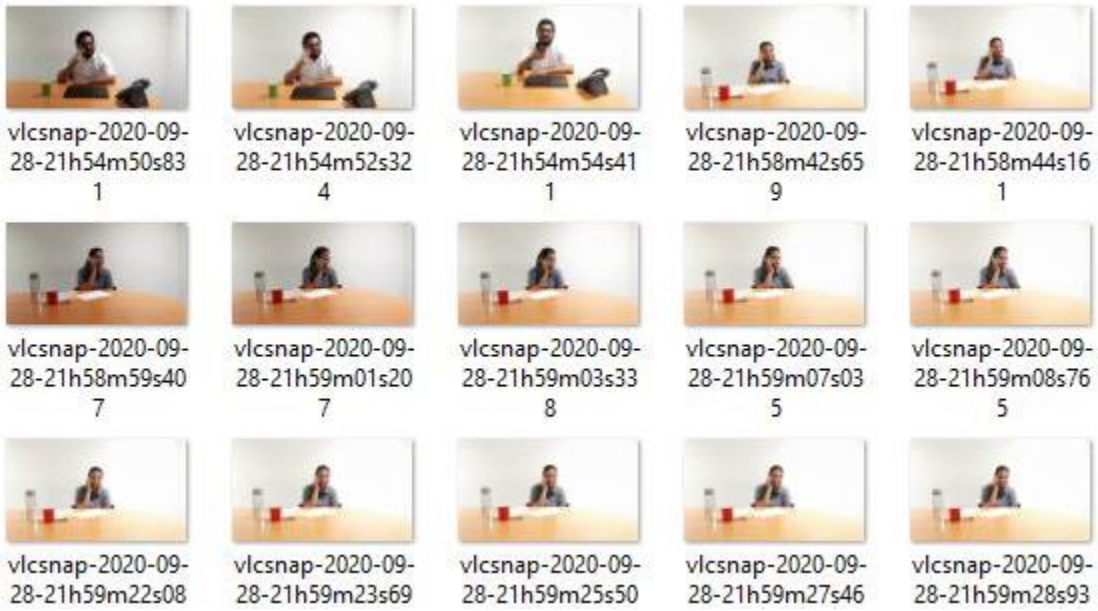


Figure 2.5: "Mobile phone usage" sample data

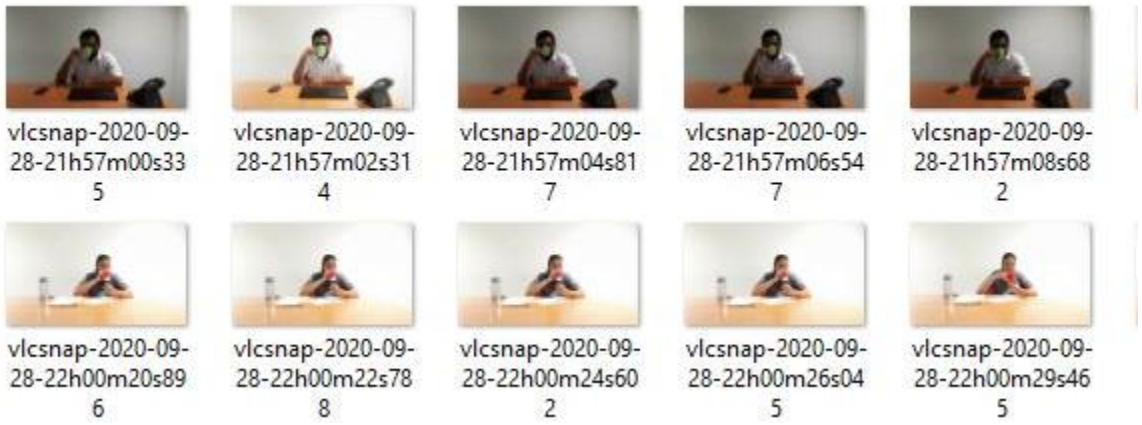


Figure 2.5: "Drinking" sample data

As the model was trained using supervised learning deep learning techniques, the dataset had to be separated in to two sections as the model should have two separate datasets for the training and testing datasets. The training dataset contains of the folders named by the relevant activity containing images of each type of activity in separate folders. The testing dataset contains a lot of images which contains all the activities that the model has been trained for in one single folder where the images are jumbled and are not in any particular order as such.

2.2 Data Pre-Processing

Data pre-processing is a crucial step in training and testing any AI model, but this dataset being an image dataset makes it much more important to be preprocessed in a particular manner which best suits the given scenario before being used for the training and testing purposes. This is because, supervised training heavily depends on the training data and having image data of higher quality will increase the accuracy of the model, therefore this custom dataset has undergone digital image preprocessing regarding several aspects with the usage of several common digital image preprocessing algorithms like image resizing and grayscale conversion.

Grayscale Conversion:

These images in the custom activity dataset contains of people engaging in different activities in an office environment with lot of noise and distractions in the background which requires the images to be turned grayscale for noise reduction and improve the image feature by suppressing unwanted distortions and enhancement of some important image features so that the activity detection model can benefit from this improved data to work on.

In [4]:
```
# display training data
for cls in classes:
    path = os.path.join(training_dataset,cls)

    for img in os.listdir(path):
        imgPath = os.path.join(path,img)
        imgArray = cv2.imread(imgPath,cv2.IMREAD_GRAYSCALE)
        print(imgArray)
        plt.imshow(imgArray, cmap='gray')
        plt.show()
        break
    break
```

```
[[215 215 214 ... 231 231 233]
 [215 215 214 ... 231 231 233]
 [215 215 214 ... 233 233 234]
 ...
 [185 185 185 ... 215 215 215]
 [185 185 185 ... 218 218 218]
 [185 185 185 ... 219 219 219]]
```

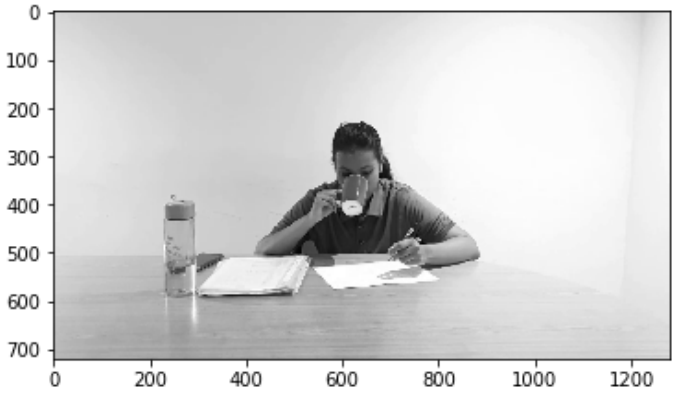


Image Resizing:

One of the preliminary concerns is that the images being of very high resolution and image size which will require a higher CPU capacity and will bring upon memory allocation issues in the training procedure. Therefore, the image data should be resized to a size that is most suitable for the available resources.

Also, when gathering images as data for this custom dataset, it is usually difficult to collect images of the same size and when training a model, it is important to establish a base size for all images fed into our AI model.

The images in the custom activity are of 1280 x 720 pixels dimension with a bit depth of 24 which is of very high resolution and large image size. Therefore, we have resized the images into 240 x 240 pixels dimensions which brough upon a high training accuracy.

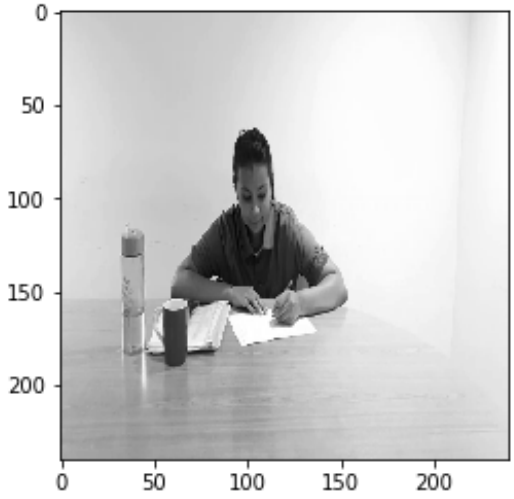
The following figures depicts the process of separately resizing the images in both the training dataset and the testing dataset.

In [6]:
```
# displaying shape of the images.
print(imgArray.shape)
```

```
(720, 1280)
```

In [7]:
```
# defining the image sizes
imgSize1 = 240
imgSize2 = 240
```

In [8]:
```
# resizing the images
newImg = cv2.resize(testArray,(imgSize1,imgSize2))
plt.imshow(newImg,cmap='gray')
plt.show()
```



In [9]:
```
# generating training dataset with new sizes.
training_data = []

def createTrainingDataset():
    for category in classes:
        path = os.path.join(training_dataset,category)
        NoOfClasses = classes.index(category)

        for img in os.listdir(path):
            imgArray = cv2.imread(os.path.join(path,img),cv2.IMREAD_GRAYSCALE)
            newImg = cv2.resize(imgArray,(imgSize1,imgSize2))
            training_data.append([
                newImg,NoOfClasses])
```

In [10]:
```
# generating testing dataset with new sizes.
testing_data = []

def createTestingDataset():
    for img in os.listdir(testing_dataset):
        imgArray = cv2.imread(os.path.join(testing_dataset,img),cv2.IMREAD_GRAYSCALE)
        newImg = cv2.resize(imgArray,(imgSize1,imgSize2))
        testing_data.append([img,
            newImg])
```

In [11]:
```
createTrainingDataset()
```

In [12]:
```
createTestingDataset()
```

2.3 Supervised Learning Approach

There are two main techniques that are used for image classification. One such technique is supervised classification where the image classification is guided by human aide and the other technique is unsupervised where the classification is done via software.

Supervised learning is the process where an algorithm learns and trains from a labeled dataset where the correct label is known and the algorithm goes on making predictions on the giving training dataset and is evaluated accordingly.

Here we have utilized the supervised learning approach as we have the images classified and labeled under different activities for the model to be trained. The supervised deep learning algorithm that is applied in training this activity detection model is Convolutional Neural Networks which depends on this training data which is labeled according to the supervised learning approach.

3. IMPLEMENTATION

3.1 Application of Convolutional Neural Networks (Conv2D)

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. Here we have applied Keras Conv2D layers in creating the activity detection model.

The Keras framework: Conv2D layers

Conv2D layer is one of the most widely used layers within the Keras framework for deep learning. Keras Conv2D is a 2D Convolution Layer, this layer creates a convolution kernel that is wind with layers input which helps produce a tensor of outputs.

Keras Conv2D class has several parameters which control the whole model building process and should be set with much understanding, as these will heavily impact on the final model that is built.

These parameters are ***kernel size, input shape, activation*** and ***padding***.

```
In [23]: model = Sequential()

In [24]: model.add(Conv2D(32,kernel_size=(5,5),activation='relu',input_shape=(240,240,1)))
         model.add(BatchNormalization())
         model.add(Conv2D(32,kernel_size=(5,5),activation='relu',padding='same'))
         model.add(BatchNormalization(axis = 3))
         model.add(MaxPooling2D(pool_size=(2,2),padding='same'))
         model.add(Dropout(0.3))
```

3.2 Usage of Conv2D parameters and other functions

kernel_size:

The kernel size is specified as a tuple which contains the width and height of the window. The kernel_size should always be an odd integer. Normally, values for kernel_size are 1x1, 3x3, 5x5 and 7x7. It is advised to increase the kernel_size from 3x3 to 5x5 if the image size is greater than 128px. Therefore, when implementing this model, the kernel size parameter has been **set to 5x5** as the image size exceeds 128px.

activation:

This is a convenience parameter, which allows to supply a string specifying the name of the activation function that is needed to be applied after performing the convolution. In implementing this model, convolution is performed and then a ReLU activation function is applied.

input shape:

In CNN a 4D array is always given as input. Therefore, as specified here, the input data has the parameters height, width and depth. Here, the input shape's height and width are set to 240 as the images have been resized to **240px** and the bit depth as **1**.

padding:

In Keras Conv2D, the padding parameter can be set as either "valid" or "same". The input volume is not zero-padded with the valid parameter the and the spatial dimensions are allowed to reduce. Although the default Keras Conv2D value is "valid", in this model, it is **set to "same"** for the majority of the layers in the network and then the spatial dimensions are reduced by Max Pooling.

Max Pooling: used to reduce the spatial dimensions of the output volume.

[25]:
```
model.add(Conv2D(64,kernel_size=(5,5),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(64,kernel_size=(5,5),activation='relu',padding='same'))
model.add(BatchNormalization(axis = 3))
model.add(MaxPooling2D(pool_size=(2,2),padding='same'))
model.add(Dropout(0.3))
```

[26]:
```
model.add(Conv2D(128,kernel_size=(5,5),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(128,kernel_size=(5,5),activation='relu',padding='same'))
model.add(BatchNormalization(axis = 3))
model.add(MaxPooling2D(pool_size=(2,2),padding='same'))
model.add(Dropout(0.5))
```

[27]:
```
model.add(Flatten())
model.add(Dense(units = 512,activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(units = 128,activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(5,activation='softmax'))
```

3.3 Specifications of the implemented model

This model was trained with the custom training dataset in batches of 50 and each batch was trained in 20 epochs.
The activity detection model has been trained on 401 images and tested on 802 images.

```
[47]: model.compile(loss='categorical_crossentropy',metrics=['accuracy'],optimizer='adam')

[48]: callbacks = [EarlyStopping(monitor='val_accuracy',patience=5)]

[49]: batch_size = 50
      n_epochs = 20

[50]: results = model.fit(x_train,Y_train,batch_size=batch_size,epochs=n_epochs,verbose=1,va
      llbacks)
```

The process of the activity detection model being trained with the training data in 20 epochs is captured and depicted in the figure below. Here the training dataset is again divided into sections for validations.

```
Train on 140 samples, validate on 61 samples
Epoch 1/20
140/140 [==============================] - 192s 1s/step - loss: 1.9213 - accuracy: 0.4286 - val_loss: 47.6784 -
0.0820
Epoch 2/20
140/140 [==============================] - 127s 904ms/step - loss: 0.5051 - accuracy: 0.8214 - val_loss: 43.8119
y: 0.0820
Epoch 3/20
140/140 [==============================] - 140s 1s/step - loss: 0.3339 - accuracy: 0.8714 - val_loss: 39.0428 -
0.0820
Epoch 4/20
140/140 [==============================] - 148s 1s/step - loss: 0.2365 - accuracy: 0.9429 - val_loss: 34.4172 -
0.0820
Epoch 5/20
140/140 [==============================] - 140s 999ms/step - loss: 0.1793 - accuracy: 0.9357 - val_loss: 27.3881
y: 0.0820
Epoch 6/20
140/140 [==============================] - 120s 855ms/step - loss: 0.1432 - accuracy: 0.9714 - val_loss: 21.6952
y: 0.1311
Epoch 7/20
140/140 [==============================] - 137s 979ms/step - loss: 0.1185 - accuracy: 0.9357 - val_loss: 17.2087
y: 0.2459
Epoch 8/20
140/140 [==============================] - 135s 961ms/step - loss: 0.0628 - accuracy: 0.9786 - val_loss: 13.6525
y: 0.3607
Epoch 9/20
140/140 [==============================] - 117s 834ms/step - loss: 0.0892 - accuracy: 0.9786 - val_loss: 11.0808
y: 0.4098
Epoch 10/20
140/140 [==============================] - 124s 888ms/step - loss: 0.0481 - accuracy: 0.9714 - val_loss: 9.0854
0.4098
Epoch 11/20
140/140 [==============================] - 132s 945ms/step - loss: 0.0263 - accuracy: 0.9929 - val_loss: 7.4321
0.4098
Epoch 12/20
140/140 [==============================] - 110s 786ms/step - loss: 0.0312 - accuracy: 0.9857 - val_loss: 6.1400
```

The summary of the specifications of the successfully trained activity detection model is shown in the table given below.

```
[28]: model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 236, 236, 32)      832
_________________________________________________________________
batch_normalization_1 (Batch (None, 236, 236, 32)      128
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 236, 236, 32)      25632
_________________________________________________________________
batch_normalization_2 (Batch (None, 236, 236, 32)      128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 118, 118, 32)      0
_________________________________________________________________
dropout_1 (Dropout)          (None, 118, 118, 32)      0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 118, 118, 64)      51264
_________________________________________________________________
batch_normalization_3 (Batch (None, 118, 118, 64)      256
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 118, 118, 64)      102464
_________________________________________________________________
batch_normalization_4 (Batch (None, 118, 118, 64)      256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 59, 59, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 59, 59, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 59, 59, 128)       204928
_________________________________________________________________
batch_normalization_5 (Batch (None, 59, 59, 128)       512
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 59, 59, 128)       409728
_________________________________________________________________
batch_normalization_6 (Batch (None, 59, 59, 128)       512
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 30, 30, 128)       0
_________________________________________________________________
dropout_3 (Dropout)          (None, 30, 30, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 115200)            0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               58982912
_________________________________________________________________
```

```
_________________________________________________________________
batch_normalization_7 (Batch (None, 512)               2048
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               65664
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 645
=================================================================
Total params: 59,847,909
Trainable params: 59,845,989
Non-trainable params: 1,920
_________________________________________________________________
```

Finally, the accuracy and loss graphs were plotted for the trained model.

[51]:
```
# model accuracy
plt.plot(results.history['accuracy'])
plt.plot(results.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# model loss
plt.plot(results.history['loss'])
plt.plot(results.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

4. RESULTS AND DISCUSSION

The implemented model was successfully trained on 401 labeled images and wad validated on 802 images in 20 epochs of batches of 50.

The trained model's parameters were identified as follows;

```
Total params: 59,339,493
Trainable params: 59,337,573
Non-trainable params: 1,920
```

The model was able to successfully identify the activities that it was trained for. The following figures display some instances where the model correctly labeled the given images.

The activities are labeled as 0,1,2,3 and 4 which represents the activities drinking, idling, using mobile, using laptop and writing respectively.

```
print('Activity: {}'.format(np.argmax(prediction)))
newImg = cv2.resize(testing_data[300][1],(imgSize1,imgSize2))
plt.imshow(newImg,cmap='gray')
plt.show()
```

```
Activity: 0
```

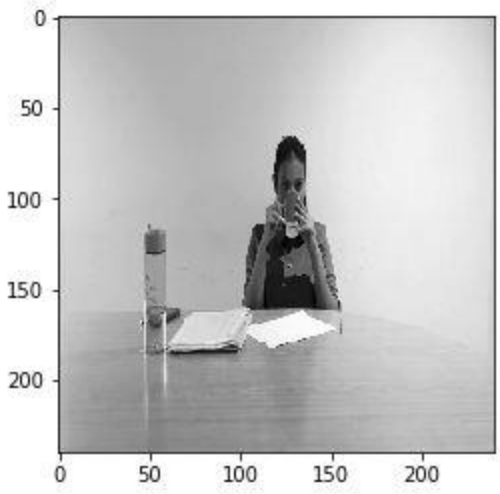


Figure 4.1: Correctly detecting drinking labeled as 0

```
print('Activity: {}'.format(np.argmax(prediction)))
newImg = cv2.resize(testing_data[300][1],(imgSize1,imgSize2))
plt.imshow(newImg,cmap='gray')
plt.show()
```

```
Activity: 4
```

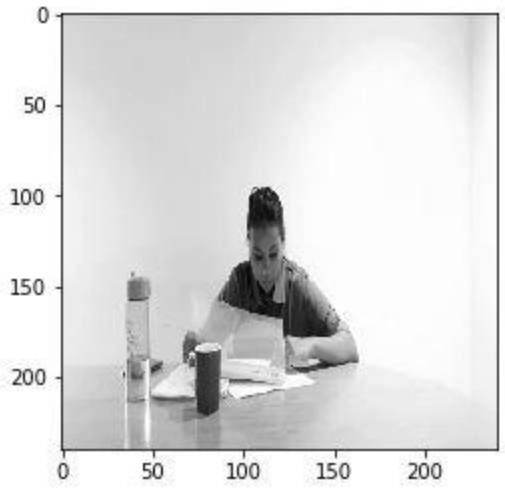


Figure 4.2: Correctly detecting writing labeled as 4

```
print('Activity: {}'.format(np.argmax(prediction)))
newImg = cv2.resize(testing_data[150][1],(imgSize1,imgSize2))
plt.imshow(newImg,cmap='gray')
plt.show()
```

```
Activity: 2
```

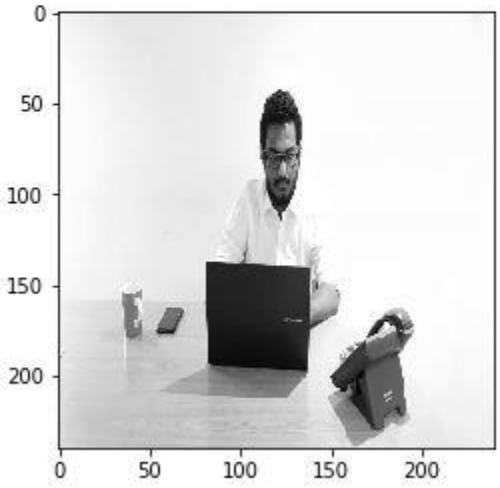
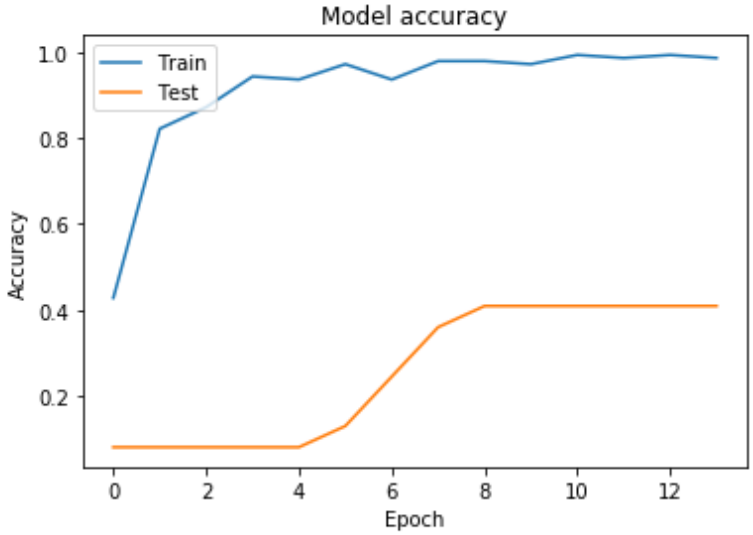
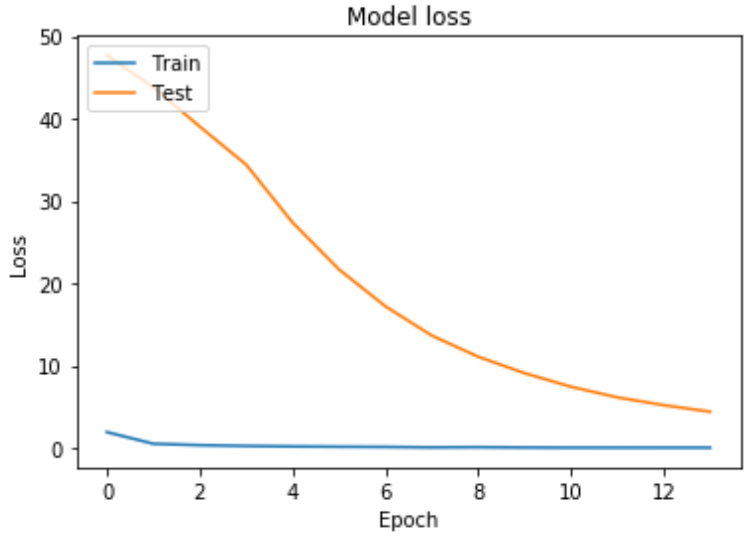


Figure 4.3: Correctly detecting using the laptop labeled as 2

The model accuracy and loss graphs

Here it is observed that the training accuracy is high and the testing accuracy is lower comparatively. This is probably due to the custom dataset not being large enough for each different activity in both the training dataset and the testing dataset.





5. FUTURE WORK

As per further studies, the images per each activity can be increased and the number of different activities too can be increased so that the model is able to detect a variety of different activities other than these basic 5 activities.

Here the activities are some basic activities that are related to office workers, but these activities are not the only activities that are observed among them as humans tend to engage in many different activities throughout the day.

Many other activities can be added as it has a really wide scope even when considering an office environment. But this activity detection model is not only useful to detect the activities of employees, this model can be used in many different scenarios that might require monitoring a lot of other different people in various other scenarios which have been discussed in the introduction section.

This again shows that it has a much larger scope and can be further improved with a large amount of activities and sufficient number of images from each different activity to enable the model to gain a much higher testing accuracy in the future.

6. REFERENCES

[1] https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c

[2] https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/

[3] https://towardsdatascience.com/understanding-input-and-output-shapes-in-convolution-network-keras-f143923d56ca

Link to the dataset:

https://mysliit-my.sharepoint.com/:f:/g/personal/it17051026_my_sliit_lk/EryJH_8ZZ-lCsfwYjxynTpgBF1LrgnpJg37m8DRfoUu9Ig?e=4SlDlB

7. APPENDIX I: The code

```
# defining the path for datasets.

training_dataset = 'training'

testing_dataset = 'testing'

# define classes

# d - drinking activity, i - idling activity, l - laptop using activity, p - phone using activity, w - writing activity

classes = ['d','i','l','p','w']


# display training data

for cls in classes:

    path = os.path.join(training_dataset,cls)


    for img in os.listdir(path):

        imgPath = os.path.join(path,img)

        imgArray = cv2.imread(imgPath,cv2.IMREAD_GRAYSCALE)

        print(imgArray)

        plt.imshow(imgArray, cmap='gray')

        plt.show()

        break

    break


# display testing data

testArray = []

for img in os.listdir(testing_dataset):
```

```
    imgArray = cv2.imread(os.path.join(testing_dataset,img),cv2.IMREAD_GRAYSCALE)
    testArray = imgArray
    plt.imshow(imgArray, cmap='gray')
    plt.show()
    break

# displaying shape of the images.
print(imgArray.shape)

# defining the image sizes
imgSize1 = 240
imgSize2 = 240

# resizing the images
newImg = cv2.resize(testArray,(imgSize1,imgSize2))
plt.imshow(newImg,cmap='gray')
plt.show()

# generating training dataset with new sizes.
training_data = []

def createTrainingDataset():
    for category in classes:
        path = os.path.join(training_dataset,category)
        NoOfClasses = classes.index(category)

        for img in os.listdir(path):
            imgArray = cv2.imread(os.path.join(path,img),cv2.IMREAD_GRAYSCALE)
            newImg = cv2.resize(imgArray,(imgSize1,imgSize2))
```

```
            training_data.append([
                newImg,NoOfClasses])

# generating testing dataset with new sizes.
testing_data = []

def createTestingDataset():
    for img in os.listdir(testing_dataset):
        imgArray = cv2.imread(os.path.join(testing_dataset,img),cv2.IMREAD_GRAYSCALE)
        newImg = cv2.resize(imgArray,(imgSize1,imgSize2))
        testing_data.append([img,
            newImg])

createTrainingDataset()
createTestingDataset()

random.shuffle(training_data)
x = []
y = []

for features, label in training_data:
    x.append(features)
    y.append(label)

X = np.array(x).reshape(-1,imgSize1,imgSize2,1)

x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=50)

Y_train = np_utils.to_categorical(y_train,num_classes=5)
```

```
Y_test = np_utils.to_categorical(y_test,num_classes=5)

model = Sequential()

model.add(Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=(240,240,1)))
model.add(BatchNormalization())
model.add(Conv2D(32,kernel_size=(3,3),activation='relu',padding='same'))
model.add(BatchNormalization(axis = 3))
model.add(MaxPooling2D(pool_size=(2,2),padding='same'))
model.add(Dropout(0.3))

model.add(Conv2D(64,kernel_size=(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(64,kernel_size=(3,3),activation='relu',padding='same'))
model.add(BatchNormalization(axis = 3))
model.add(MaxPooling2D(pool_size=(2,2),padding='same'))
model.add(Dropout(0.3))

model.add(Conv2D(128,kernel_size=(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(128,kernel_size=(3,3),activation='relu',padding='same'))
model.add(BatchNormalization(axis = 3))
model.add(MaxPooling2D(pool_size=(2,2),padding='same'))
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(units = 512,activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
```

```
model.add(Dense(units = 128,activation='relu'))

model.add(Dropout(0.25))

model.add(Dense(5,activation='softmax'))


model.summary()

model.compile(loss='categorical_crossentropy',metrics=['accuracy'],optimizer='adam')

callbacks = [EarlyStopping(monitor='val_accuracy',patience=5)]


batch_size = 50

n_epochs = 20


results =
model.fit(x_train,Y_train,batch_size=batch_size,epochs=n_epochs,verbose=1,validation_data=(x_test,Y_
test),callbacks=callbacks)


# model accuracy

plt.plot(results.history['accuracy'])

plt.plot(results.history['val_accuracy'])

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()


# model loss

plt.plot(results.history['loss'])

plt.plot(results.history['val_loss'])

plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')
```

```
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

predection = model.predict(np.array(testing_data[150][1]).reshape(-1,imgSize1,imgSize2,1))
model.save_weights('./activity-recognition.h5', overwrite=True)
model.save('./activity-recognition.h5')
ActivityRecognitionModel = load_model('activity-recognition.h5')
testImg = np.array(testing_data[150][1]).reshape(-1,imgSize1,imgSize2,1)
predection = ActivityRecognitionModel.predict(testImg)

print('Activity: {}'.format(np.argmax(predection)))
newImg = cv2.resize(testing_data[150][1],(imgSize1,imgSize2))
plt.imshow(newImg,cmap='gray')
plt.show()
```