Project Report

Topic:

The project assigned to me is **Heart Attack Detection**. This project presents a proposal to identify people with an apparent heart attack by detecting characteristic postures of heart attack. The method of identifying infarcts (in this case, possible heart attack) makes use of convolutional neural networks. I have built a CNN model and trained it on a GPU-enabled server (Google Colab) to recognize heart-attack from a video. I have trained the model with a specially prepared set of images from the internet and self-made that contain people simulating a heart attack. The promising results in the classification of infarcts show 91.75% accuracy and 92.85% sensitivity.

Dataset:

I created my own dataset by taking images of myself and my friends and also many from the internet. All the pictures contain only one person. In images labelled as an infarct, all images show a person's posture in which they have one or both hands on the chest. As regards to no infarct situations, images where people are performing daily activities were used. The initial image data set consisted of a total of 1520 images, 760 images of class “Infarct” and 760 images of class “No Infarct”. For better learning and extracting of features, the number of images in the data set is increased using data augmentation. The following steps were performed:

- Each image was scaled to a maximum size of 256×256 pixels, maintaining the original proportion, both for “Infarct” and “No Infarct” images. This is in order to reduce the amount of data to be processed during the augmented data technique.
- After that, the images were classified into two categories, “Infarct” and “No Infarct”. Furthermore, each category was split into the three subcategories of training, validation, and testing, as shown in Table 1.
- As the CNNs only have to infer a possible heart attack, people were extracted from the background of the image by reducing the noise caused by the variation of the background in order to improve the training set.
- People were automatically located in each image. For this purpose, I performed the instance segmentation and background removal and replaced the pixels with magenta color so as to be a contrast to the person. This is shown in Figure 1.
- Data augmentation is a process for generating new samples by transforming training data. In this case, each original picture generated 20 more different images. For this, six transformations were combined and applied to each image

(rotation, increase/decrease in width or height, zoom, horizontal flip, and brightness change). This is shown in Figure 2.

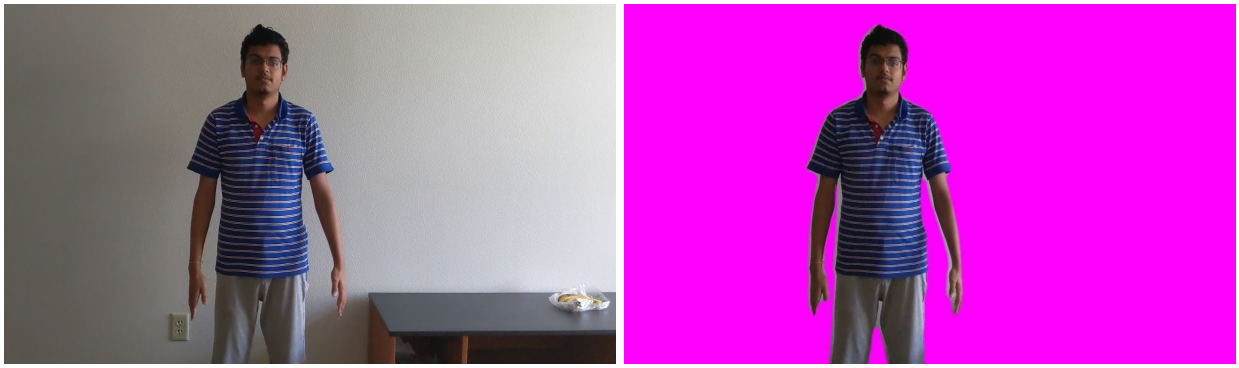
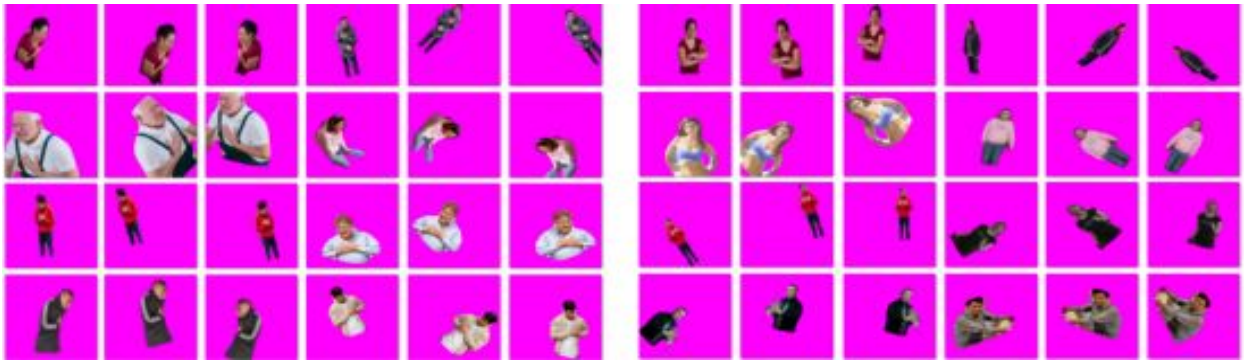


Figure 1: Instance segmentation and background removal



`

Figure 2: Data Augmentation

CNN Model:

Architecture:

At the beginning of the network, the architecture has five convolutional blocks, where each block is firstly composed of a convolution layer to highlight the general features in the image. Then, a max pooling layer is provided to keep the number of variables of the network low, in this way maintaining a size easy to compute.
In the middle of the network, just after the convolution blocks, there is a dropout layer that prevents the generated model from presenting an envelope training, mainly due to the limited amount of data. After this, a flatten layer allows changing the 2D design of the convolutional layers to a vectorial one so that the values generated in the previous layers are passed to the traditional neuron layers.
At the end of the network, ten layers composed of traditional neurons are arranged, each with 128 neurons, which deliver the result of forward propagation to a softmax function

with two outputs. These will classify whether there is or is not a person with a heart attack in the image.
A snippet of code of constructing the model/network is shown as follow:

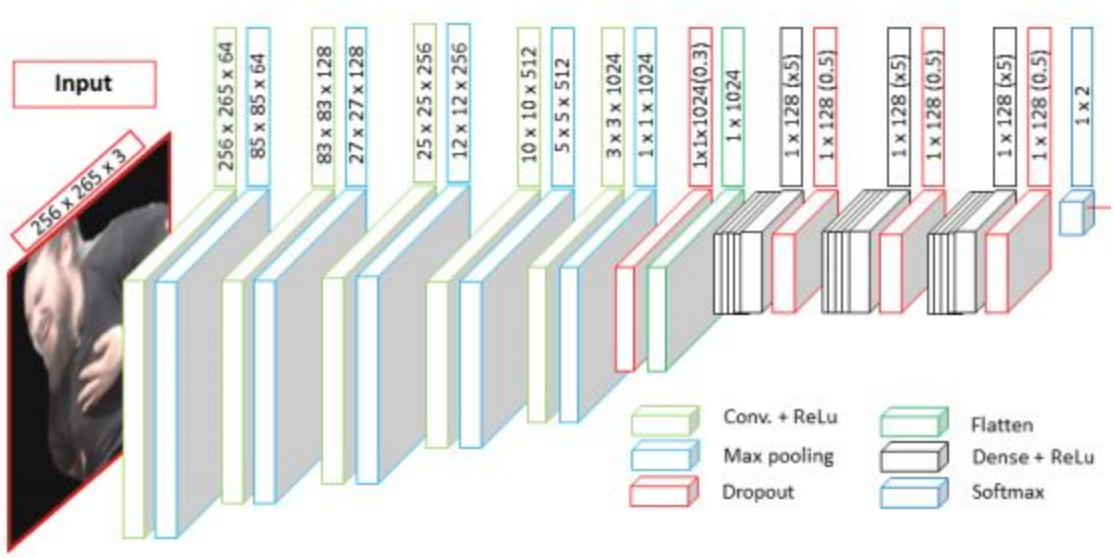


FIgure 3: Neural Network Architecture

```
NET = Sequential()
NET.add(Convolution2D(64, kernel_size=(3 ,3), padding ="same", input_shape=(256, 256, 3), activation='relu'))
NET.add(MaxPooling2D((3,3), strides=(3,3)))
NET.add(Convolution2D(128, kernel_size=(3, 3), activation='relu'))
NET.add(MaxPooling2D((3,3), strides=(3,3)))
NET.add(Convolution2D(256, kernel_size=(3, 3), activation='relu'))
NET.add(MaxPooling2D((2,2), strides=(2,2)))
NET.add(Convolution2D(512, kernel_size=(3, 3), activation='relu'))
NET.add(MaxPooling2D((2,2), strides=(2,2)))
NET.add(Convolution2D(1024, kernel_size=(3, 3), activation='relu'))
NET.add(MaxPooling2D((2,2), strides=(2,2)))
NET.add(Dropout(0.3))
NET.add(Flatten())

for _ in range(5):
    NET.add(Dense(128, activation='relu'))
NET.add(Dropout(0.5))

for _ in range(5):
    NET.add(Dense(128, activation='relu'))
NET.add(Dropout(0.5))

for _ in range(5):
    NET.add(Dense(128, activation='relu'))
NET.add(Dropout(0.5))

NET.add(Dense(CLASSES, activation='softmax'))

sgd = SGD(lr=LR, decay=1e-4, momentum=0.9, nesterov=True)

NET.compile(optimizer=sgd,
              loss='binary_crossentropy',
              metrics=['acc', 'mse'])
```

Input:

I chose the 70%–15%–15% for training, validation, and testing, which is a typical configuration in many other applications based on neural networks. The number of images and the distribution is given in Table 1. Once the training images were selected, the model was built using the Tensorflow framework. A precision of 99% was achieved during training, which allowed 91.75% accuracy and 92.85% sensitivity in the test set, defining a learning rate of 0.003 and using the gradient descent optimiser.

Class	Initial Training	Training Augmented	Initial Validation	Validation Augmented	Initial Testing	Testing Augmented	Final
Infarct	532	10,640	114	2280	114	2280	15,960
No Infarct	532	10,640	114	2280	114	2280	15,960
Total	1064	21,280	228	4,560	228	4560	31,920

Table 1 : Data distribution and total images after augmentation

The output for each hidden layers and the final output shape is described in Figure 4.

Hyperparameters:

In this project, there are four hyperparameters: batch size; epochs, dropout and learning rate.

As of now, tried Batch Sizes are 32 and 48,
Epochs 10,
Steps per epoch 500,
Dropout 0.5
Learning Rate 0.3.

More fine-tuning has to be done on these hyper-parameters in order to reach more efficient results.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 256, 256, 64)      1792
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 85, 85, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 83, 83, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 27, 27, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 25, 25, 256)       295168
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 256)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 10, 10, 512)       1180160
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 5, 5, 512)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 3, 3, 1024)        4719616
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 1, 1, 1024)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 1, 1024)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               131200
_________________________________________________________________
dense_2 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_3 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_4 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_5 (Dense)              (None, 128)               16512
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_7 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_8 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_9 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_10 (Dense)             (None, 128)               16512
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_11 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_12 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_13 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_14 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_15 (Dense)             (None, 128)               16512
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_16 (Dense)             (None, 2)                 258
=================================================================
Total params: 6,633,218
Trainable params: 6,633,218
Non-trainable params: 0
```

Figure 4: Network Summary

Instruction on how to test the trained CNN:

The whole workflow i.e. codes, videos used, generated graphs and json files can be found at the Github repo.

For training the mode, you should run *train.py*. The model will be saved as *model.h5* in the */model* directory.

For directly testing the code on the test dataset using the pre-trained model, you can directly run *test.py*.

There are three steps in the whole process:

1. Getting frames from the video. For this, you have to run *frame_generator.py.* For now, we have to manually give the video filename inside the code but will be changed to taking argument from the terminal in the next submission.

2. Second step is the instance segmentation and background removal from each of the extracted frames. For this, we will run *seg_backrem.py.* In this case also, for now, we have to manually give the frames path inside the code but will be changed to taking argument from the terminal in the next submission.

3. The final step is testing each of the background removed frames by running *test_modified.py.* This will also generate the required plot that at a specific time, what is the predicted probability of heart attack and also creates the json file.

Future Work for upcoming submissions:

1. Taking the video filename as an argument and automating the extracting frames, instance segmentation and background removal processes.
2. More work in extracting facial expressions and combined testing with the hand posture model.
3. Working on reducing the number of false positives.