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.

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.

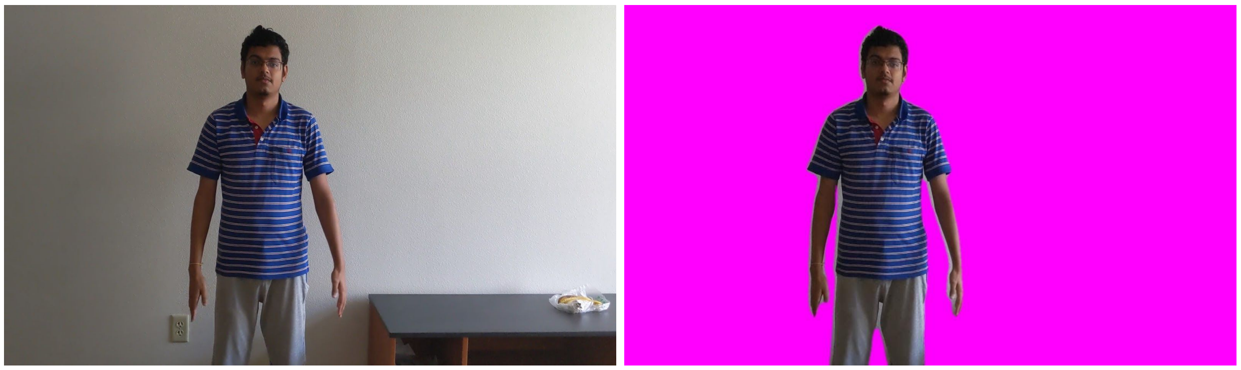
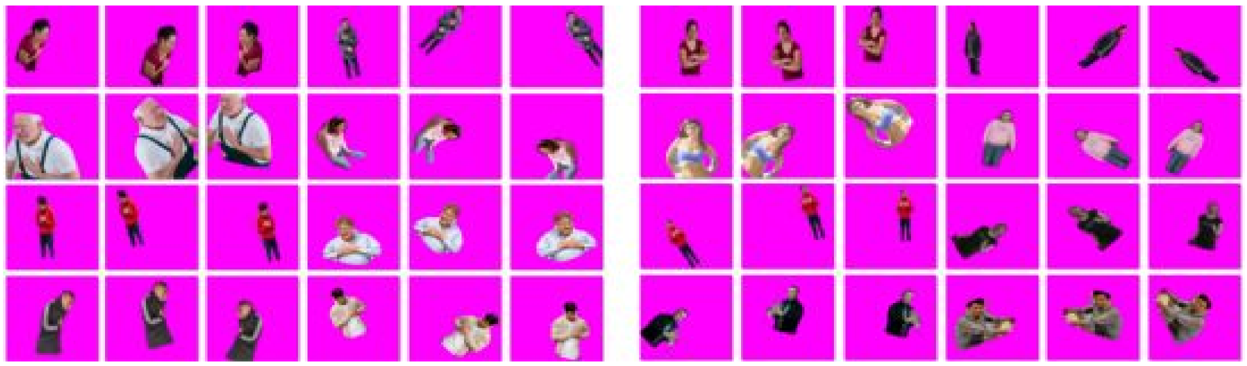


Figure 1: Instance segmentation and background removal


` **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:

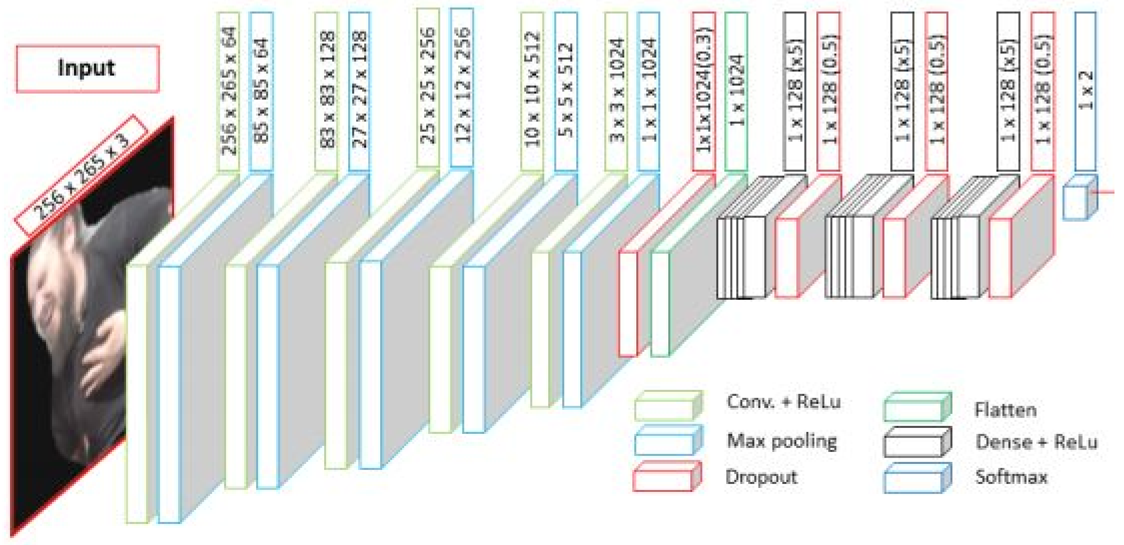
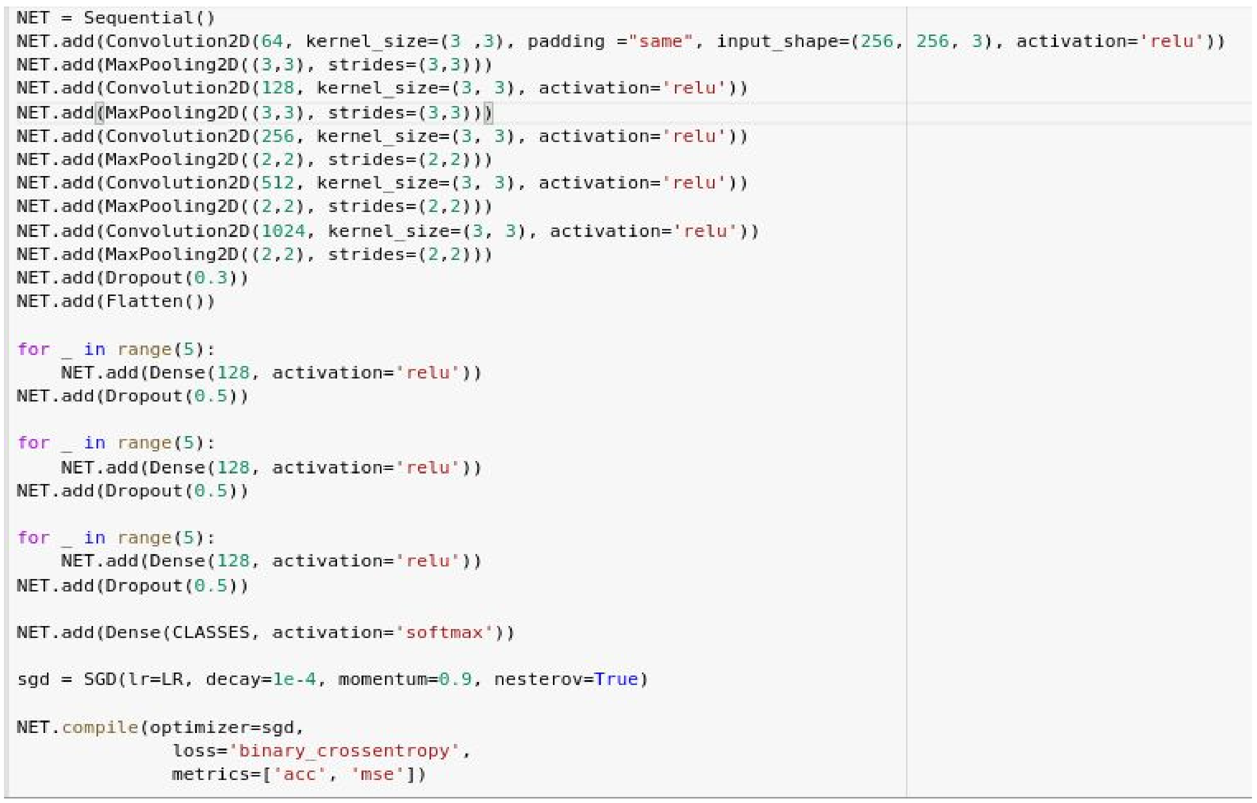


FIgure 3: Neural Network Architecture*

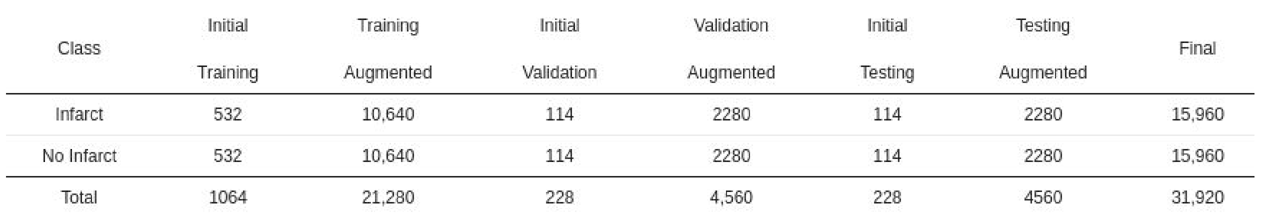
* network figure just for reference, not the exact model



```
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

Output:

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

Hyperparameters:

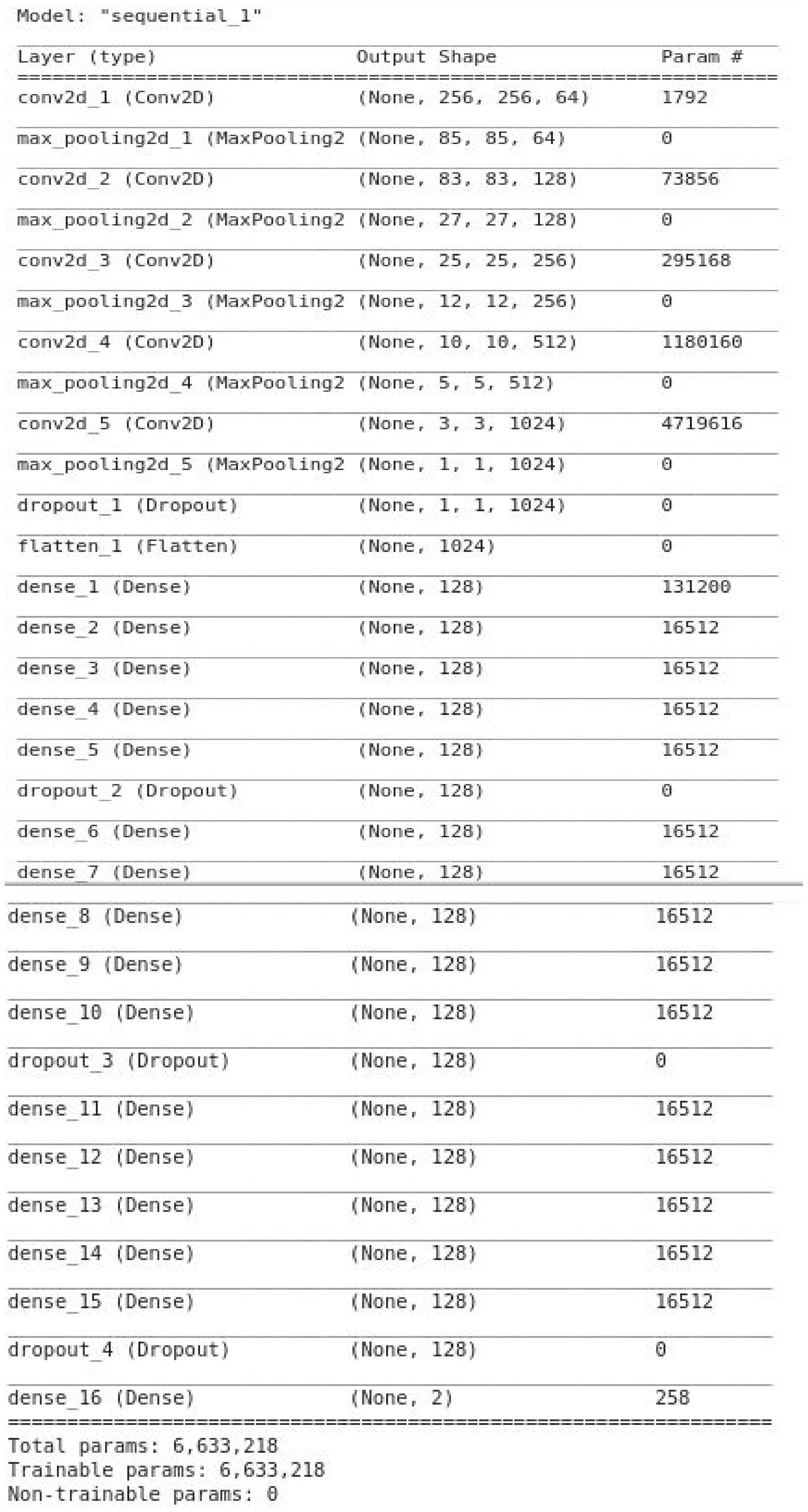
In this model, there are many hyperparameters: batch size, epochs, steps in training and validation, dropout and learning rate.

Hyper-parameters	Batch Size	Epochs	Steps/epoch training	Steps/epoch validation	Dropout	Learning rate
Experimentation	32, 48, 64	5, 10,20	100, 200, 500	20, 50, 100	0.3, 0.5	0.001, 0.003
Optimal	48	10	500	100	0.5	0.003

Table 2 : Range of Hyperparameters tried and optimal values

Annotated code:

The codes for training and testing can be found at the Github repo.
https://github.com/Kartikvenkat98/Heart-Attack-Detection



```
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 showing output shape of tensors after each layer

Training Performance:

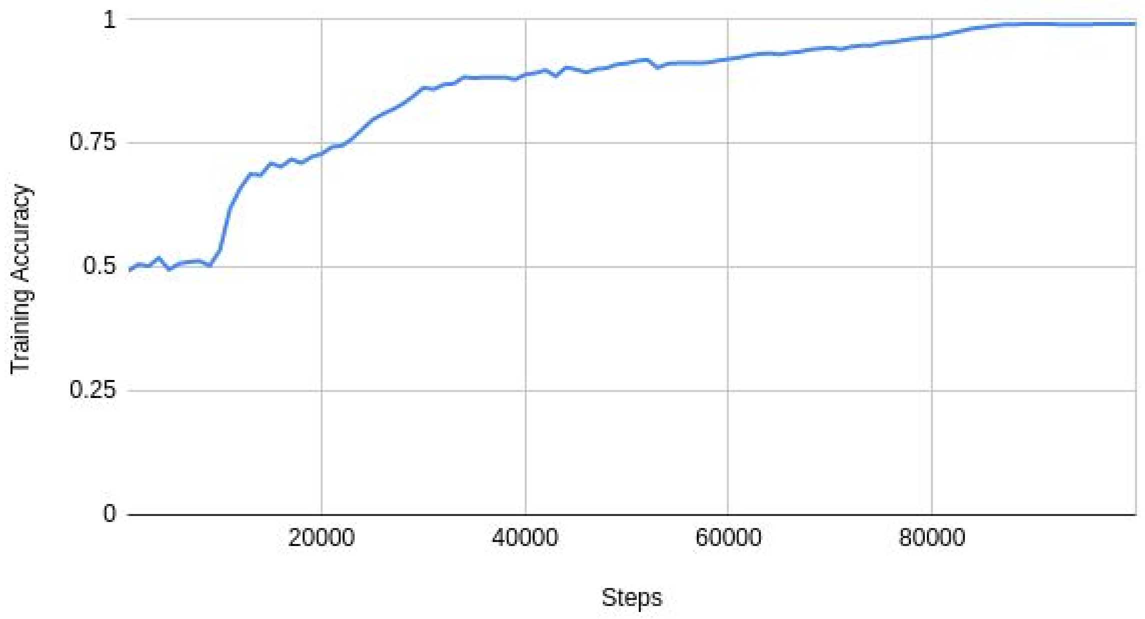


Figure 5: Training Accuracy vs. Steps

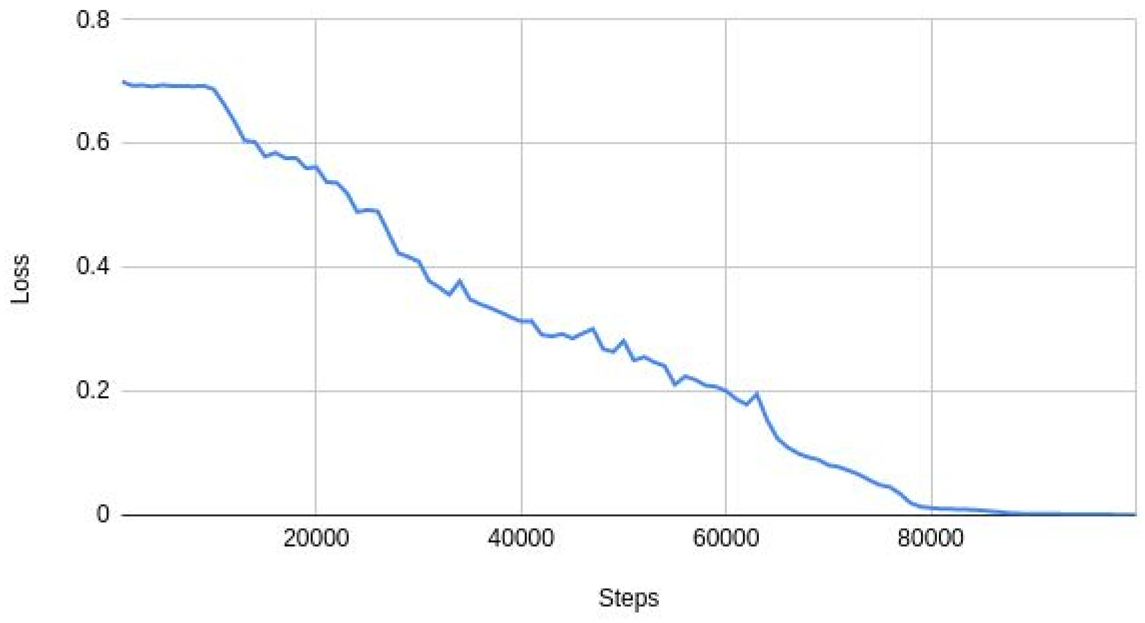


Figure 6: Training Loss vs. Steps

Testing Performance:

```
======= ======== ========
Found 4821 images belonging to 2 classes.
loss: 0.3153, acc: 0.9175, mse: 0.0764
======= ======== ========
```

Drawbacks of the previous model:

1. There are quite a number of false positives as seen in the testing of the videos. This is because when the hand or some part of the hand comes in front of the chest, as there is no 3D depth available, it sees the image as 2D and assumes the hand is on the chest and detects it as positive. But when the side angle is seen, the network detects correctly. So, such images should be added to the training dataset has to be increased so that the network can learn to detect correctly.

2. In this model, I haven't taken into account the facial expressions at this moment. So, it will classify a video of a person laughing with hands on or near the chest as a positive case of a heart attack. This is also a case of false positive. Work is needed in detecting facial expressions and combining the testing of that model with this existing model.

3. In this model, I'm detecting the presence of infarct in each frame of the video. The sample videos I have taken for testing are around 20-25 seconds long. The model takes around 10-15 minutes in total for the whole video. But if we pass just the frames as arguments to the python script, it will take around 15-20 seconds for each frame which is a decent enough time for real-time application.

Improvements from the previous model:

Increase the dataset size by adding images with hands just randomly in front of the chest classified as non-infarct and experimented more with hyperparameters and trained the model again. The accuracy on the test dataset increased by 0.05%. But no significant difference was found in the plots and json files for videos.

As it was proving to be difficult to get a better understanding of the images to an extent, I decided to use OpenPose to generate body landmarks.

OpenPose:

It is a tool for generating facial/body landmarks for humans in video. It is a realtime multi-person 2D pose estimation is a key component in enabling machines to have an understanding of people in images and videos. The method uses a nonparametric representation, referred to as Part Affinity Fields (PAFs), to learn to associate body parts with individuals in the image. This bottom-up system achieves high accuracy and real time performance.

I installed OpenPose in Google Colab and used the BODY_25 model as it is best suited for this project and gives the best results. I ran the OpenPose model on my whole dataset (training, validation and testing images) and trained my CNN model again on these rendered images.

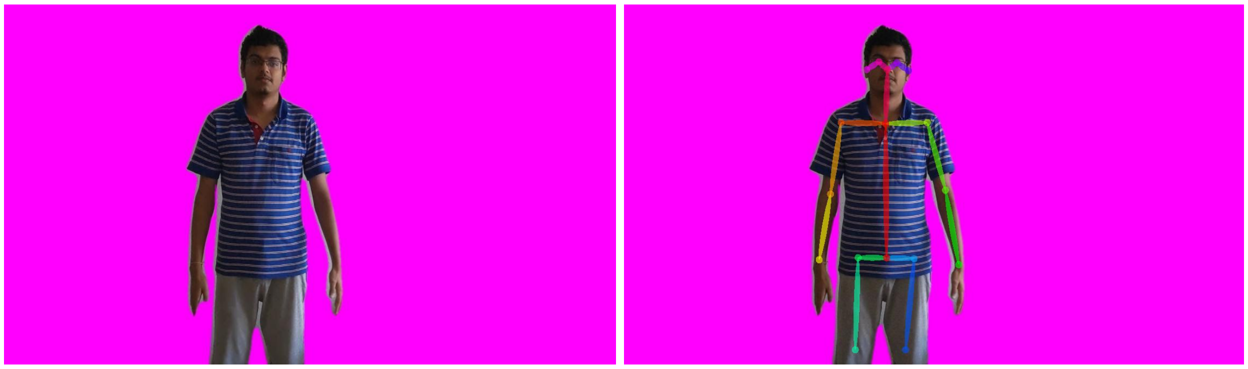
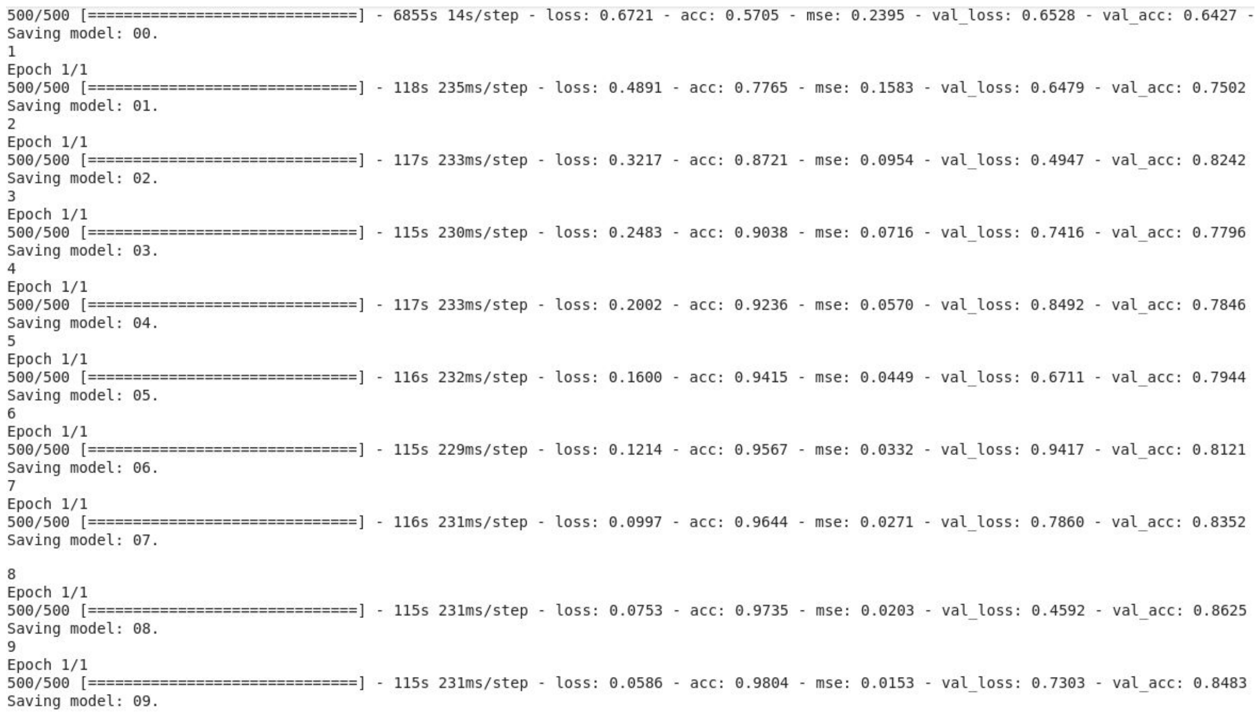


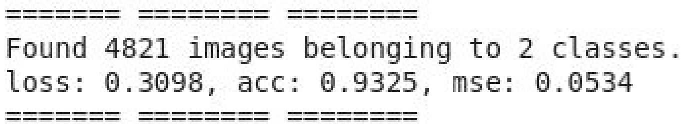
Figure 7 : OpenPose body landmarks generation

Training Performance on body-landmarks generated dataset:



```
500/500 [==============================] - 6855s 14s/step - loss: 0.6721 - acc: 0.5705 - mse: 0.2395 - val_loss: 0.6528 - val_acc: 0.6427 -
Saving model: 00.
1
Epoch 1/1
500/500 [==============================] - 118s 235ms/step - loss: 0.4891 - acc: 0.7765 - mse: 0.1583 - val_loss: 0.6479 - val_acc: 0.7502
Saving model: 01.
2
Epoch 1/1
500/500 [==============================] - 117s 233ms/step - loss: 0.3217 - acc: 0.8721 - mse: 0.0954 - val_loss: 0.4947 - val_acc: 0.8242
Saving model: 02.
3
Epoch 1/1
500/500 [==============================] - 115s 230ms/step - loss: 0.2483 - acc: 0.9038 - mse: 0.0716 - val_loss: 0.7416 - val_acc: 0.7796
Saving model: 03.
4
Epoch 1/1
500/500 [==============================] - 117s 233ms/step - loss: 0.2002 - acc: 0.9236 - mse: 0.0570 - val_loss: 0.8492 - val_acc: 0.7846
Saving model: 04.
5
Epoch 1/1
500/500 [==============================] - 116s 232ms/step - loss: 0.1600 - acc: 0.9415 - mse: 0.0449 - val_loss: 0.6711 - val_acc: 0.7944
Saving model: 05.
6
Epoch 1/1
500/500 [==============================] - 115s 229ms/step - loss: 0.1214 - acc: 0.9567 - mse: 0.0332 - val_loss: 0.9417 - val_acc: 0.8121
Saving model: 06.
7
Epoch 1/1
500/500 [==============================] - 116s 231ms/step - loss: 0.0997 - acc: 0.9644 - mse: 0.0271 - val_loss: 0.7860 - val_acc: 0.8352
Saving model: 07.

8
Epoch 1/1
500/500 [==============================] - 115s 231ms/step - loss: 0.0753 - acc: 0.9735 - mse: 0.0203 - val_loss: 0.4592 - val_acc: 0.8625
Saving model: 08.
9
Epoch 1/1
500/500 [==============================] - 115s 231ms/step - loss: 0.0586 - acc: 0.9804 - mse: 0.0153 - val_loss: 0.7303 - val_acc: 0.8483
Saving model: 09.
```

Testing Performance:



```
======= ======== ========
Found 4821 images belonging to 2 classes.
loss: 0.3098, acc: 0.9325, mse: 0.0534
======= ======== ========
```

The new model showed significant improvement in the accuracy and loss.

Facial Expression Recognition:

In the above model implemented, we can see that I haven't taken facial expressions into account. So, it will classify a video of a person laughing with hands on or near the chest as a positive case of a heart attack. This is a case of false positive. So, I worked on detecting facial expressions and then combined the testing of the facial expression recognition model with the existing model.

Dataset:

Obviously, I couldn't use the same dataset for facial expression recognition that I created previously as this dataset needs to be more focused on the facial pixels. And there was not much time to create a new dataset. So, I used an existing dataset. It is known as the FER-2013 dataset. The dataset can be found at Challenges in Representation Learning: Facial Expression Recognition Challenge.

- The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centered and occupies about the same amount of space in each image. The emotion shown in the facial expression is in one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral).
- train.csv contains two columns, "emotion" and "pixels". The "emotion" column contains a numeric code ranging from 0 to 6, inclusive, for the emotion that is present in the image. The "pixels" column contains a string surrounded in quotes for each image. The contents of this string are space-separated pixel values in row major order. test.csv contains only the "pixels" column and your task is to predict the emotion column.
- The training set consists of 28,709 examples. The public test set used for the leaderboard consists of 3,589 examples. The final test set, which was used to determine the winner of the competition, consists of another 3,589 examples.

CNN Model:

Architecture:

At the beginning of the network, the architecture has three convolutional blocks, where each block is firstly composed of a convolution layer to highlight the general features in the image. Then, an average 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, two layers composed of traditional neurons are arranged, each with 1024 neurons, which deliver the result of forward propagation to a softmax function with seven outputs. These will classify the emotion based on the facial expression. The loss is cross-entropy as it is a multi-class classification.

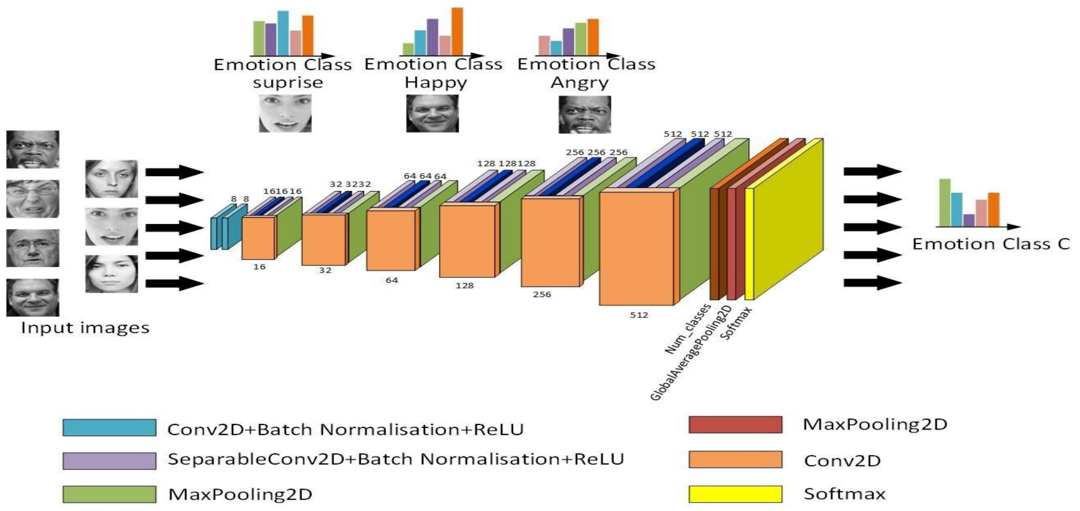
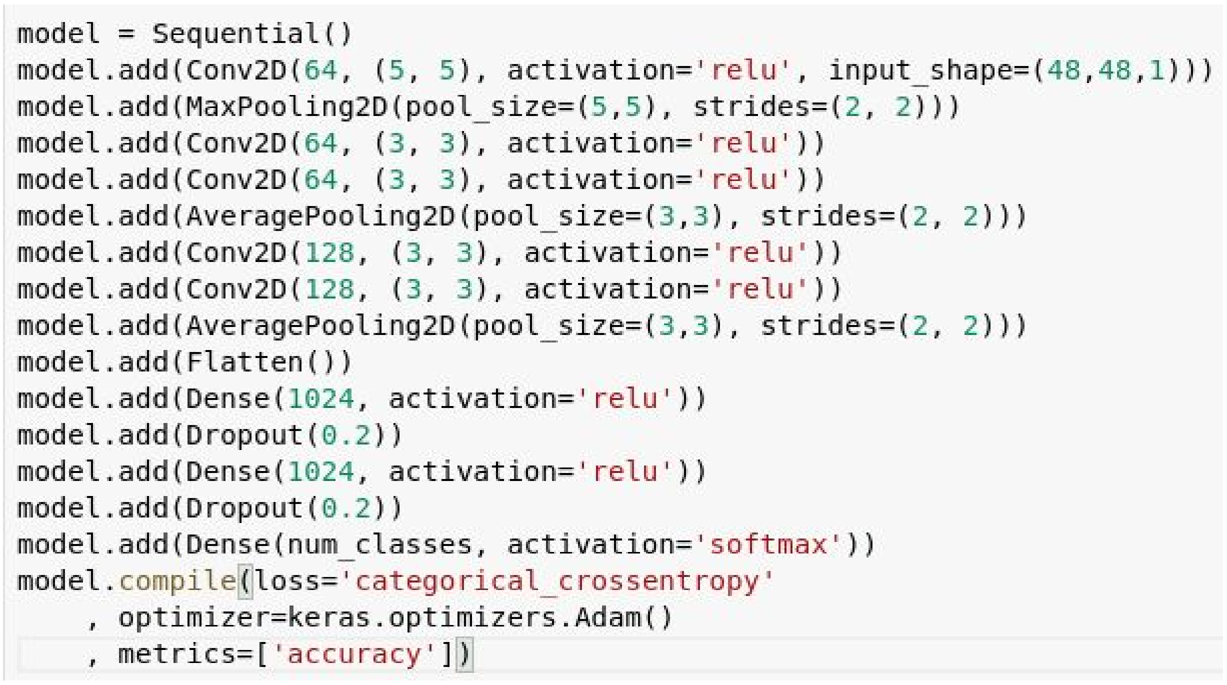


Figure 7: Neural Network Architecture for face expression recognition*

* network figure just for reference, not the exact model



```
model = Sequential()
model.add(Conv2D(64, (5, 5), activation='relu', input_shape=(48,48,1)))
model.add(MaxPooling2D(pool_size=(5,5), strides=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(AveragePooling2D(pool_size=(3,3), strides=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(AveragePooling2D(pool_size=(3,3), strides=(2, 2)))
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy'
    , optimizer=keras.optimizers.Adam()
    , metrics=['accuracy'])
```

Input:

The training set consists of 28,709 examples. The public test set used for the leaderboard consists of 3,589 examples. The final test set, which was used to determine the winner of the competition, consists of another 3,589 examples. So, the distribution chosen here is the 80%–20% for training and testing.

Output:

The output for each of the hidden layers and the final output shape is described in Fig. 8.

Hyperparameters:

For this model, hyperparameters I experimented are: batch size, epochs and dropout.

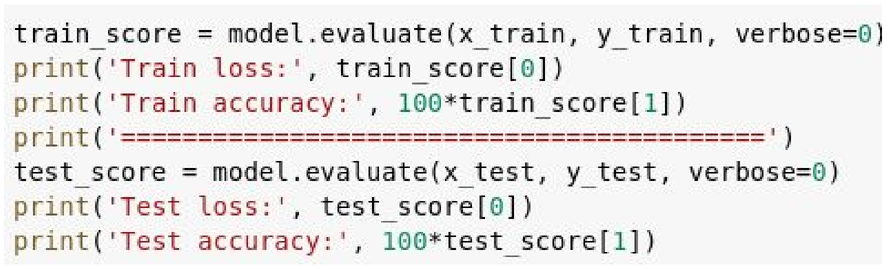
Hyperparameters	Batch Size	Epochs	Dropout
Experimentation	128, 256	2, 3, 5	0.1, 0.2
Optimal	256	5	0.2

Table 3 : Range of Hyperparameters tried and optimal values

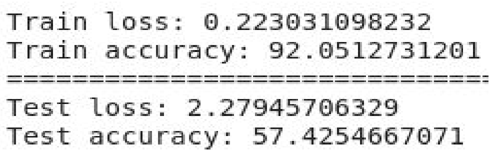
Annotated code:

The codes for training and testing can be found at the Github repo.
https://github.com/Kartikvenkat98/Heart-Attack-Detection

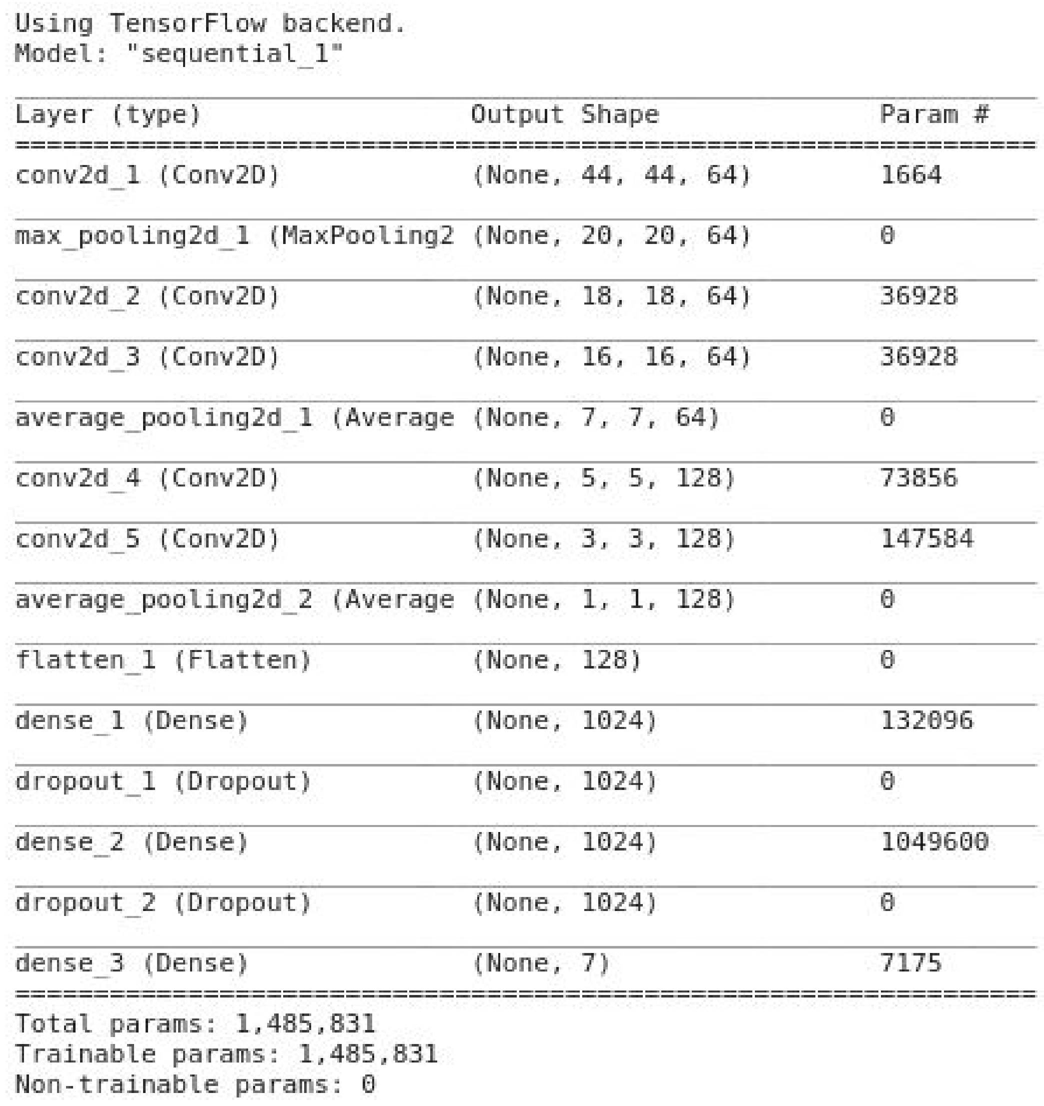
Training and Testing Performance:



```
train_score = model.evaluate(x_train, y_train, verbose=0)
print('Train loss:', train_score[0])
print('Train accuracy:', 100*train_score[1])
print('==========================================')
test_score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', test_score[0])
print('Test accuracy:', 100*test_score[1])
```



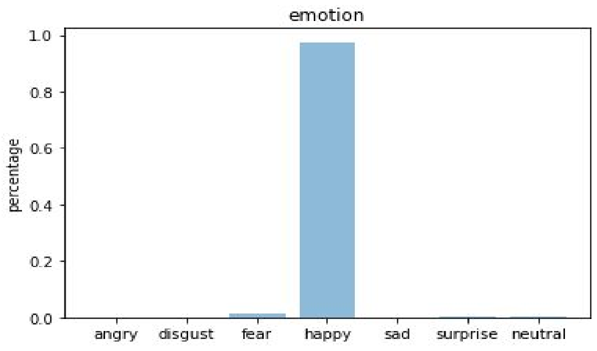
```
Train loss: 0.223031098232
Train accuracy: 92.0512731201
==============================
Test loss: 2.27945706329
Test accuracy: 57.4254667071
```

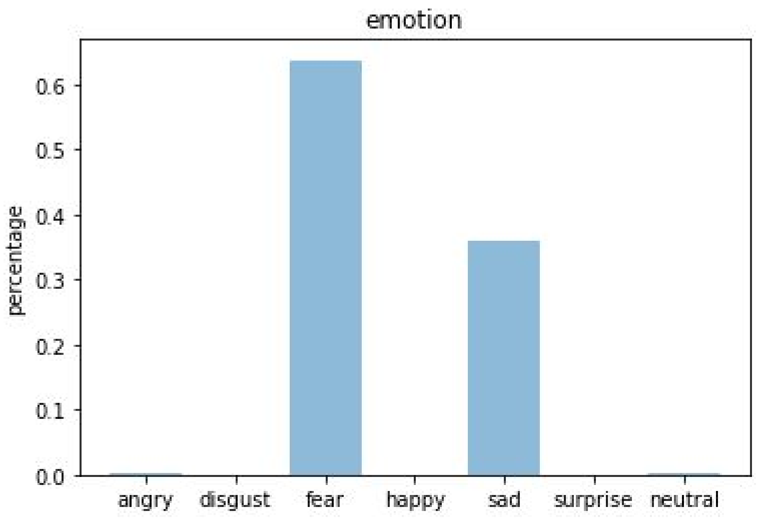


```
Using TensorFlow backend.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 44, 44, 64)        1664
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 20, 20, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 18, 18, 64)        36928
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
average_pooling2d_1 (Average (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 5, 5, 128)         73856
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 3, 3, 128)         147584
_________________________________________________________________
average_pooling2d_2 (Average (None, 1, 1, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              132096
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 7175
=================================================================
Total params: 1,485,831
Trainable params: 1,485,831
Non-trainable params: 0
```

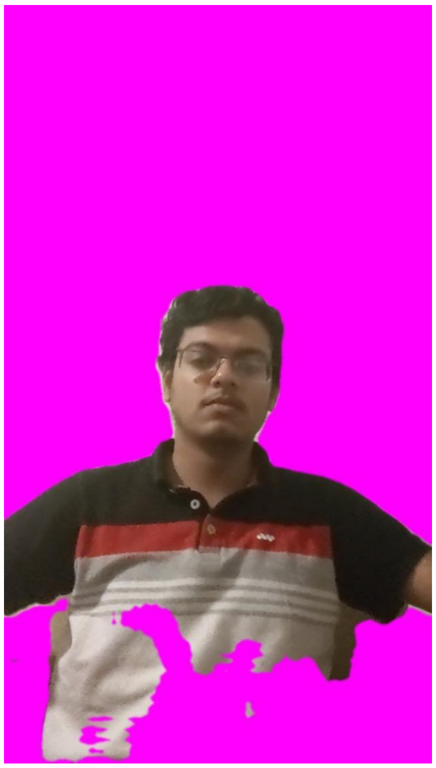
Figure 8: Network Summary showing output shape of tensors after each layer

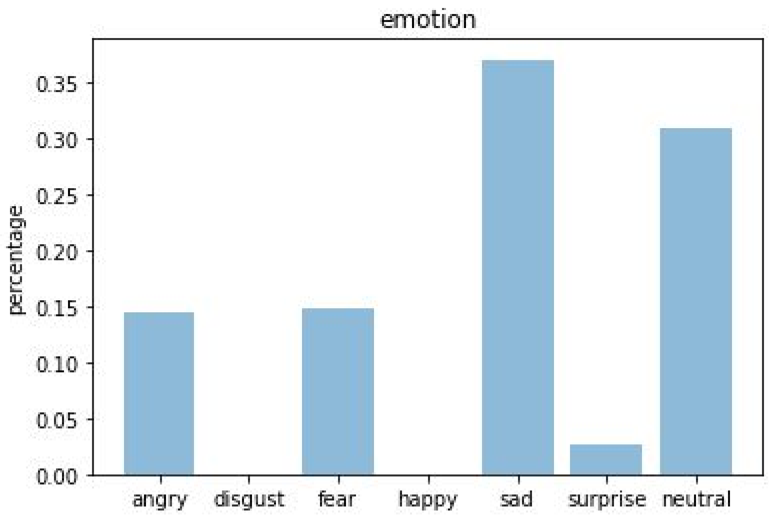
For our custom images, emotions are stored as numeric labels from 0 to 6. Keras would produce an output array including these 7 different emotion scores. We can visualize each prediction as a bar chart.





But sometimes, the model cannot clearly identify which emotion the person is experiencing as it can’t be exactly identified from the expression.


.



As there are seven classes, the accuracy does not express the right impression for multi class classification problems as some emotions have very little change in the facial expressions as we see above.
So, I even tried to calculate the confusion matrix to get a better understanding.

```
from sklearn.metrics import confusion_matrix

pred_list = []; actual_list = []
for i in predictions:
  pred_list.append(np.argmax(i))
for i in y_test:
  actual_list.append(np.argmax(i))

confusion_matrix(actual_list, pred_list)
```

	Angry	Disgust	Fear	Happy	Sad	Surprise	Neutral
Angry	214	9	53	30	67	8	86
Disgust	10	24	9	2	6	0	5
Fear	45	2	208	29	89	45	78
Happy	24	0	40	696	37	18	80
Sad	65	3	83	56	285	10	151
Surprise	7	1	42	27	9	303	26
Neutral	45	2	68	65	88	8	331

The rows represent actual labels whereas columns state predictions. That means that there are 467 angry instances in testset. We can classify 214 angry items correctly. On the other hand, we classified 9 items as disgust but these items are actual angry ones. So, as we can see, the model is not very accurate but facial expression detection is still a difficult and open problem.

For our project of heart attack detection, I decided to take into account all the emotions except “happy” and “surprise”. So, when the person holds the chest initially, if his emotion is happy or surprised, it won’t label it as heart-attack. After a few frames with hand-on-chest detection, even when there is no hand on the chest, the system will classify as heart-attack if the emotion is sad, fear, angry, disgust and even neutral.

This is a very important addition to the project as the person might not have his hands on the chest always but might be having a heart attack. That's where facial expression detection plays an important role. The new system has been tested on old and new test videos and gives satisfactory results.

Instructions on how to run the project:

The whole workflow i.e. codes, videos used, generated plots and json files can be found at the Github repo. https://github.com/Kartikvenkat98/Heart-Attack-Detection

For training the initial model, 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*.

For training the facial expression recognition model, you should run *train_fer.py*. The models will be saved as *model_fer.json* and *model_fer.h5* in the */model* directory.

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

Install OpenPose in your Google Colab working directory by following the instructions found here.

For generating body landmarks on the video, you have to run
./build/examples/openpose/openpose.bin --hand --video path-to-video-file --write_video path-to-video-pose --write_json path-to-json-output --display 0
This command will generate body landmarks for the whole video and save the rendered video in the write-video directory.
For example,
./build/examples/openpose/openpose.bin --hand --video ../videos/video_1.mp4 --write_video ../video_pose/video_1.mp4 --write_json ../json_pose/video_1 --display 0

For detecting heart attack from a video, there are four steps in the whole process:

1. Getting frames from the video.
   For this, you have to run *frame_generator.py path-to-video-file.*
   This command will generate frames for a given video and save it in the */frames* directory.
   For example, *python frame_generator.py ./videos/video_1.mp4*

2. Third step is the instance segmentation and background removal from each of the extracted frames. For this, we will run *seg_backrem.py.*
   The background is changed to magenta color for having the contrast for the segmented image. This command will generate the segmented image for the frames and save them in */fg-extract* folder.

3. The final step is testing each of the background removed frames by running *test_modified.py path-to-video-file.*
   This will also generate the required plot of the predicted probability of heart attack at a specific time instant and also creates the corresponding json file.
   For example, *python test_modified.py ./videos/video_1.mp4*

Future Work:

Detecting heart attack using Deep Learning is quite a novel idea and many improvements can be made to the model by including more features. I would definitely like to keep working on this project.