Title of Report:
50.039 Theory and Practice of Deep Learning Big Project

Team Number: 17

Name:
Jeremy Ng Kah Jun (1003565),
Zhou Yutong (1003704),
Joey Richie L Tan (1003599),
Prashanth Nair (1003639)

Topic

The human face is an important canvas for us to analyse and understand the human emotion. According to a study conducted by the University of Berkeley, they found out that the human has 27 distinct emotions, instead of 6. These 27 emotions still could be further blended together, combining and creating a plethora of permutations of human emotions(Russell, 1980).One of the many ways that we can analyse the human emotions is through what is reflected and seen, more specifically what can be seen from the human face.

With this fundamental inspiration in mind, the team wanted to study how we can analyse these features and use them to predict the baseline arousal and valence value (Russell, 1980) of these facial expressions. In this paper, it states that emotion is not independent and that affective dimensions are interrelated in a highly systematic function, and can be represented as a spatial model, with affective concepts falling in a circle. This model allowed psychologists to represent and visualise the structure of affective experience and the representation of cognitive structure to conceptualise affect.

As such, we will then be able to deduce the current emotion displayed from their facial expressions according to the following metric: pleasure (0°), excitement (45°), arousal (90°), distress (135°), misery/displeasure (180°), depression (225°), sleepiness (270°), and contentment/relaxation (315°). The angles are calculated using the trigonometry concept, tangent, where the value is calculated the side opposite the angle over the side adjacent to the angle, since we will look towards attaining the values of Arousal and Valence.

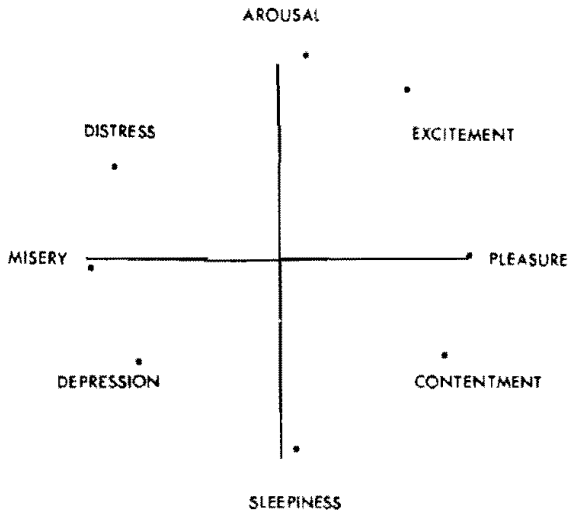

Figure 1: Eight affect concepts in circular order

Data

For our project we used the First Affect-in-the-wild dataset as part of the Aff-Wild Challenge posed by the Intelligent Behaviour Understanding Group (ibug). This dataset includes a total of 252 RGB videos of variable lengths, with the object of focus in the videos are of the various subjects and their displayed emotions. These subjects are of different backgrounds and ethnicity, doing a variety of head poses, with varying illumination conditions and occlusions. These subjects in view had their faces recorded while doing an activity and their facial expression is the focus of our data and our study. Each video in the dataset also has corresponding text files which displays the valence and arousal scores of the subject recorded at every frame. (The length of the text files which display the arousal and valence scores is similar to the number of frames of the video)

Link to dataset original source: https://ibug.doc.ic.ac.uk/resources/first-affect-wild-challenge/

Data Pre-processing

As per mentioned above, the facial expression of the subjects in frame are of paramount importance in our project and they serve as the sole basis to determine the valence and arousal scores. Given that the initial dataset included videos of varying lengths, we look towards standardizing the total number of frames per video to be studied. We believe in the uniformity of our data that is inputted into our model to reduce the number of varying conditions.

The videos were analyzed using only several frames. We cropped every video to their first 400 frames and from there, we extracted a total of 10 frames for each video, at intervals of 40 frames, as shown in Figure 2 below. For its corresponding arousal and valence scores of each video, we extracted 10 arousal and valence scores each from the text files at every 40 intervals which correspond to the index of the video frame extracted.

To help our model analyse the facial emotions in the videos better, we implemented an additional step in the data preprocessing to detect the subjects faces in each frame and crop them. We used a pre-trained model[1] to help us with the identification of faces in the frames. By identifying the faces in the frames and cropping them, this greatly reduces noise fed into the model due to background or other objects in the frame that can have a high possibility to affect the model's prediction. As a form of standardisation to fit into our model pipeline, faces were cropped and resized to a square image of size 224 x 224.

[1] Joint Face Detection and Alignment pre-trained model using Multi-task cascaded Convolutional Networks. https://github.com/TropComplique/mtcnn-pytorch

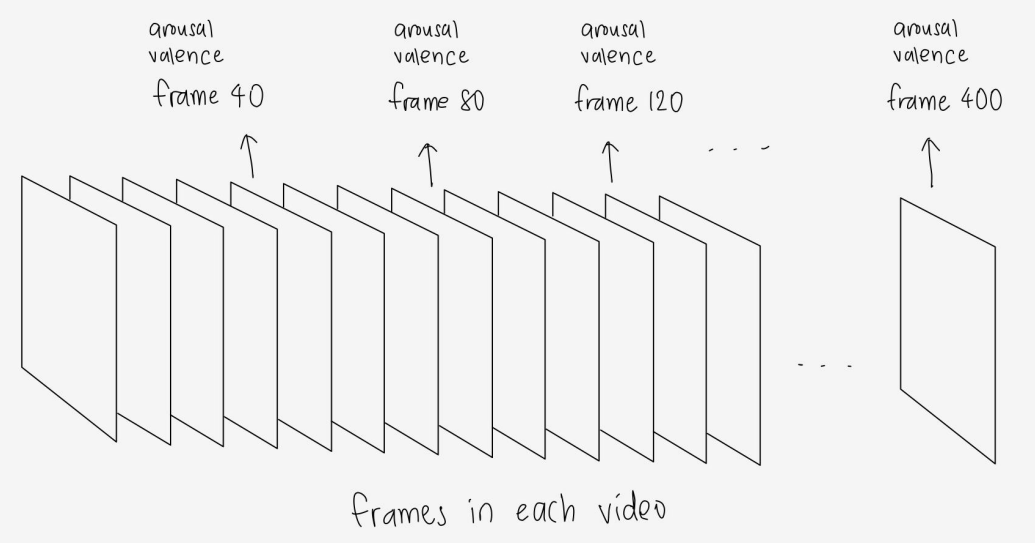

Figure 2: Extraction of frames at intervals of 40 from each video to be inputted to model

Data Visualisation

Below shown in Figure 3 and Figure 4 are snippets showing how a particular frame of a video originally shown in an .mp4 video was cropped into a 224 x 224 sized image showing the subject's face in that frame, to be inputted to our model.

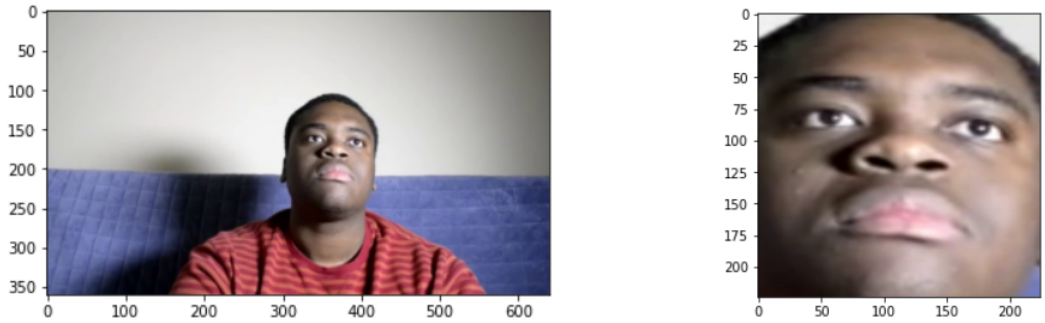

Figure 3: Frame 200 of the 105.mp4 file before cropping(left) and after cropping(right)

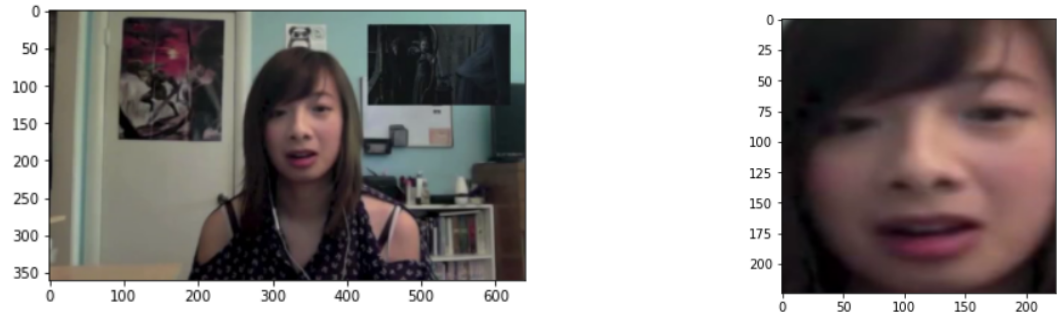

Figure 4: Frame 40 of the 111.mp4 file before cropping(left) and after cropping(right)

Custom Dataloader

We created our own custom dataset for all training, validation and testing data. The dataset is split into a 80-10-10 ratio using scripts. The steps mentioned above in the Data Pre-processing section is applied to every video in the dataset to produce 10 frames with its corresponding arousal and valence score at each frame. There are a total of 198 training data, 26 validation data and 25 testing data. Each of them is then passed into a Dataloader to be grouped together in batches of 8 to speed up the training process.

```
batch_size = 8

train_dataset = Train_Dataset()
valid_dataset = Valid_Dataset()
test_dataset = Test_Dataset()

train_loader = DataLoader(train_dataset, batch_size = batch_size, shuffle = True)
test_loader = DataLoader(test_dataset, batch_size = batch_size, shuffle = True)
val_loader = DataLoader(valid_dataset, batch_size = batch_size, shuffle = True)
```

Model Architecture

Given that we are dealing with Videos here, we would want to look towards using both Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). The underlying idea stems from the fact that videos are continuous frames of image in sequential order. With that in mind, the CNN would be used to analyse and learn the features of each single image, which in our case would be the facial expressions.

The first part of our model architecture will involve CNNs. It will extract significant facial features from the cropped faces in the frames. CNN is largely effective for image classification as the concept of dimensionality reduction is ideal with the large numbers of parameters involved with images, its classification and the processing involved. For our CNN architecture, we took inspiration from the LeNet architecture. We chose the LeNet architecture to model as it is small and easy to understand, yet large enough, with a substantial number of parameters, to allow for

learning and results to be shown. However instead of fully following the LeNet architecture, we have slightly modified it to use the ReLU activation function over the tanh activation function. The Rectified Linear Unit (ReLU) outputs x for all x >=0 and 0 for all x < 0, hence equalling max(x,0). It is also not sensitive to vanishing gradients. In comparison, tanh activation function is a more difficult formula, hence it becomes more computationally expensive and being sensitive to the vanishing gradient, the learning speed of the network is much slower. The extracted features will then be fed into the second part of the architecture.

The code for our CNN architecture can be seen in Figure 5 below.

```
class CnnGru(nn.Module):
    def __init__(self,features,num_layers,hidden_size,batch_size):
        '''
        Initialising parameters used in the cnn-gru model
        '''
        super(CnnGru,self).__init__()
        self.features = features
        self.num_layers = num_layers # the sequence
        self.hidden_size = hidden_size
        self.batch_size = batch_size
        self.device = 'cuda'

        # CNN
        self.conv2d_1 = nn.Conv2d(3, 16, 3, 1, 1) # 224 x224
        self.conv2d_2 = nn.Conv2d(16, 32, 3, 1, 1) # 112 x112
        self.conv2d_3 = nn.Conv2d(32, 64, 3, 1, 1) #

        self.maxPool2d_1 = nn.MaxPool2d((2, 2))
        self.maxPool2d_2 = nn.MaxPool2d((2, 2))
        self.maxPool2d_3 = nn.MaxPool2d((2, 2))

        self.flatten_1 = nn.Flatten()
        self.fc1 = nn.Linear(50176,features)

        # RNN
        self.grucell = nn.GRUCell(input_size=features, hidden_size=hidden_size)
        self.fc2 = nn.Linear(hidden_size, 1)

    def forward_cnn(self, x: torch.Tensor):
        '''
        building custom cnn (inspired from lenet)
        '''
        x = F.relu(self.conv2d_1(x))
        x = self.maxPool2d_1(x)
        x = F.relu(self.conv2d_2(x))
        x = self.maxPool2d_2(x)
        x = F.relu(self.conv2d_3(x))
        x = self.maxPool2d_3(x)
        x = self.flatten_1(x)
        x = self.fc1(x)
        x = F.relu(x)
        return x
```

Figure 5 : CNN part of our model architecture

The second part of the architecture will involve RNNs. It will take in the extracted features from the 10 frames of each video, and based on the features from the frames, evaluate and predict the appropriate arousal and valence scores. RNN is used for the second part due to the requirement of sequential analysis on facial features in order to determine the arousal and valence scores accordingly. Given that each frame will result in an arousal and valence score, it will be most appropriate to implement a many-to-many architecture as our case involves a sequence of multiple frames as input mapped to a sequence with multiple arousal valence scores as output. We then compared the architectures of LSTM against GRU and we concluded on using GRU.

Despite the GRU having fewer parameters, the loss achieved was lower. With more gates for the LSTM model, there are more gates for the gradient to flow through, thus being more computational expensive and causing greater volatility throughout the gradient descent. In addition, the training speed of the GRU is faster than the LSTM. As such, GRU is the chosen architecture type for the RNN. In our solution, with the use of Pytorch, we used the method of creating each GRU cell iteratively. This means that the cell would compute and return only for one frame and timestamp. In our model, each GRU cell would be dependent on the one before. For example the first GRU cell would be dependent solely on the output. However, the second GRU cell would be dependent on both the CNN output as well as the hidden state of the aforementioned first GRU cell. The third RNN cell would then be dependent on the CNN output as well as the hidden states of the first two GRU cells. This dependency trend would continue till the last GRU cell.

The code for our RNN architecture can be seen in Figure 6 below.

```
def forward_rnn(self, x: torch.Tensor):
    '''
    building custom rnn (gru)
    '''
    output = []
    h0 = torch.zeros(x.size(0),self.hidden_size).to(self.device)

    for frame in x.split(1,dim=1):
        if frame.shape[0]==1:
            frame = torch.squeeze(frame,1)
        elif frame.shape[0]>1:
            frame = torch.squeeze(frame)

        h0 = self.grucell(frame,h0)
        out =  self.fc2(h0)
        output += [out]
    output = torch.cat(output, dim=1)
    return output

def forward(self, x: torch.Tensor):
    batch_size, timesteps, C, H, W = x.size()
    gru_in = torch.zeros(batch_size, timesteps, self.features).to(self.device)

    for i in range(self.num_layers):
        temp = x[:,i,:,:,:]
        temp = self.forward_cnn(temp)
        gru_in[:,i,:] = temp

    output = self.forward_rnn(gru_in)
    return output
```

Figure 6: RNN part of our model architecture

As we had both arousal and valence to account for, in order to get each of their corresponding scores for each frame, we created two models. Therefore, one model would serve to output the frame-by-frame arousal scores, and the other would serve to output the frame-by-frame valence scores.

A visualisation of our whole model architecture pipeline can be seen in Figure 7 below.

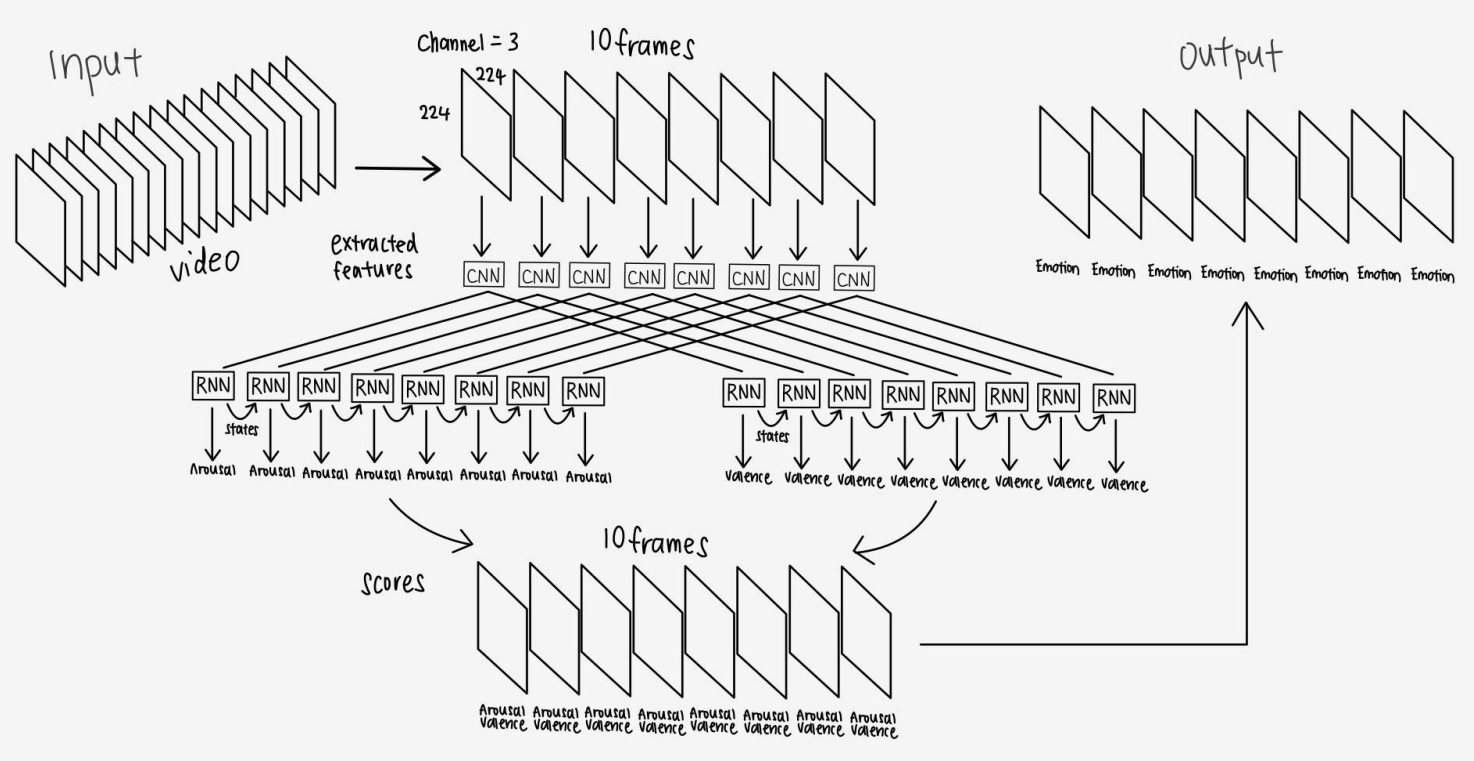

Figure 7: Visual representation of whole model architecture

Parameters chosen

Optimiser

We used the Adam optimizer as we felt it was the best one to work with. From our research, we came across an article by Davide Giordano, titled “7 tips to choose the best optimizer”, where we found it to be rather insightful. We gathered that Adam was better than other adaptive optimizers such as RMSprop , Adadelta and Adagrad. Firstly, Adam trumps Adadelta and RMSProp as unlike the two, an exponentially decaying average of past gradients is stored. This helps to prevent overfitting, which is desirable, especially in the area of healthcare research where models will need to be able to generalize well on the unseen data. To add to the benefit that Adam has over Adadelta and RMSProp , Adam also trumps Adagrad as Adagrad happens to adjust learning rate for each parameter according to all the past gradients, thereby creating a problem where a very small learning would be apparent after many steps, which causes very small update in the weights which prevent the model from learning further. (Giordano, 7 tips to choose the best optimizer, 2020)

Learning Rate

The Learning rate hyperparameter controls the model learning speed, which in turn controls weights of the error updated at the end of each batch of training. At the optimal learning rate, the model will learn to best estimate the function given the limited available resources and epochs. A larger learning rate will allow the model to train faster, but arriving at a sub-optimal final set of weights. A smaller learning rate on the other hand will train the model slower, but arrive at a more optimal set of weights. Hence with a comparatively small set of training data, a learning rate of 0.001 was chosen, to balance between training speed and the optimal set of weights.

Loss - MSE loss

Given that the problem we are looking to solve a regression problem, this would then mean that the loss functions used should not be that of a binary classification one. As such, we would turn to loss functions which would look to minimize the error. For example, should we optimize an erroneous loss function, we would not be able to obtain the optimal minima, hence solving a wrong optimisation problem. As such, we would need to find the appropriate loss function which we would then look to minimize.

Comparing the various regression loss functions, we would be comparing: Sum of Errors (SE), Sum of Absolute Errors (SAE), Sum of Squared Errors (SSE) and Mean Squared Errors (MSE).

Issues with SE: the algorithm will converge and exit prematurely, and this results in a return in a lower than expected error value.

Issues with SAE: looks to overcome issues with SE, but the loss function is not differentiable at the 0 point, hence we would be unable to find the optimum point

Issues with SSE: looks to overcome issues with SAE, but would face the vanishing Gradient problem

MSE: looks to overcome issues from SSE, by taking the average of SSE. The more the data, the lesser the aggregated error. This means that the error decreases as our algorithm gains more experience

From the above evaluations, we can see that MSE would then be the most ideal loss function to make use of for our project.

Mini-Batch size

The minibatch size chosen in our model was 8. According to the paper, “Revisiting Small Batch Training for Deep Neural Network” by Dominic Masters and Carlo Luschi, the best mini-batch

size is from the range of 2 to 32. (Masters & Luschi, 2018). We decided to use a mini-batch size of 8. As we understand that increasing the mini-batch size provides a more stable convergence and acceptable test performance, we wanted to increase our mini-batch to 16. However, we faced ‘out of memory’ issues as we attempted to do so, thus keeping it at 8.

Save Function Implementation

We implemented a save function at the end of the training of our model such that the model weights can be saved so as to ensure reproducibility of the model. Reproducibility of the model is very important as it ensures correctness; as mentioned in the Small Project instructions, should a client want to replicate our results, he/she should be able to recreate the performance results that our group has shown in our report.

Result

Training Loss and Validation loss

In the interest of time, we ran our model for 5 epochs. The visualization of curves reflecting the training and validation losses are depicted in Figure 8 and Figure 9 below. In both the curves it can be that the training and validation losses tend to have a general decreasing trend. Training for more epochs could perhaps make these trends more obvious (Highlighted in Limitations section below).

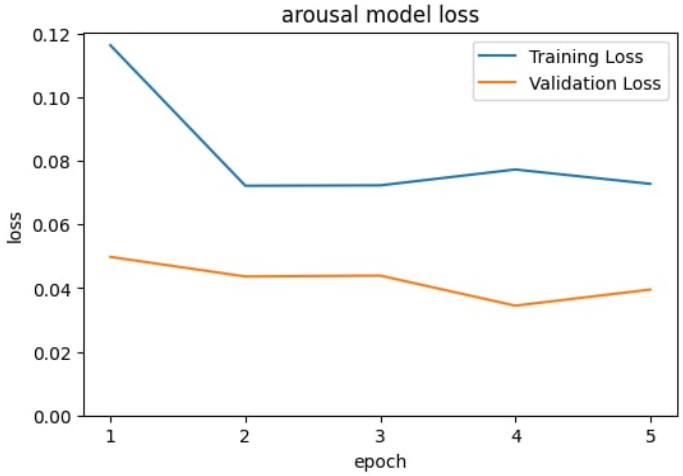

Figure 8: Arousal Model Training and Validation Losses

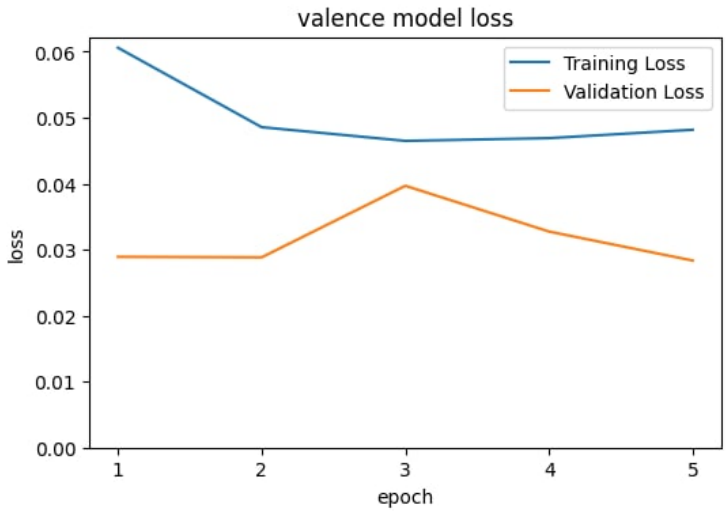

Figure 9: Valence Model Training and Validation Losses

UI

For our User Interface, we came up with a Graphical User Interface(GUI) using PyQt5, as shown in Figure 10 below, where upon inputting the directory of the video and clicking the 'predict' button, the video would load. From there, by clicking the 'k' button, the frames of the video would increase by a factor of 1. The person's emotion only starts being reflected from frame 39 onwards, thereby showing why the person in Figure 11 below is described to have no emotion. Figure 12,13 below show the different emotions displayed by the person at frames 50 and 208 respectively.

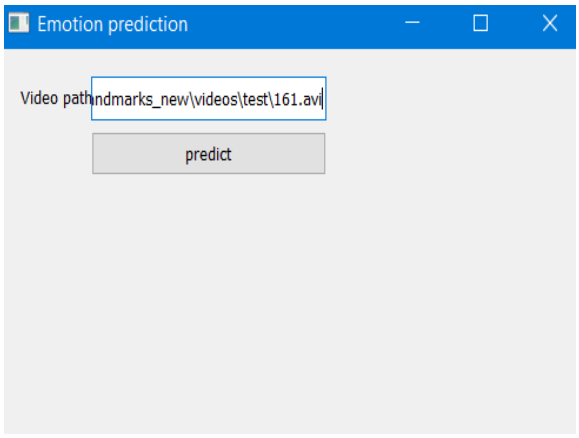

Figure 10: Landing Page

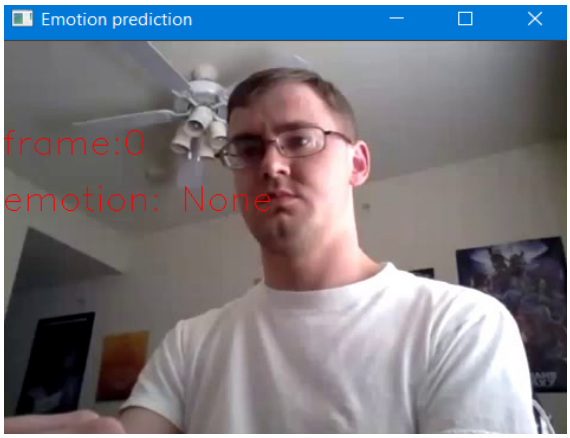

Figure 11: Emotion at Frame 0: None

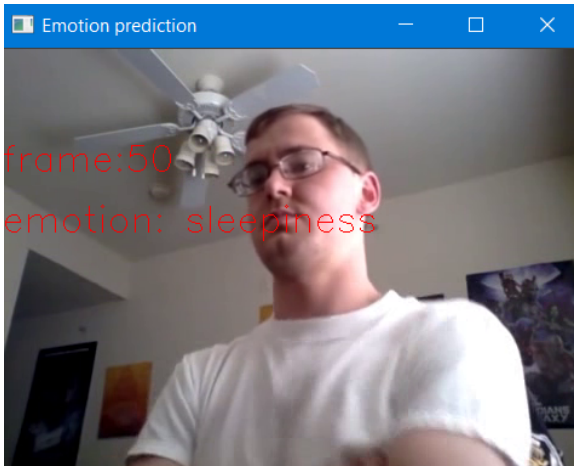

Figure 12: Emotion at Frame 50: Sleepiness

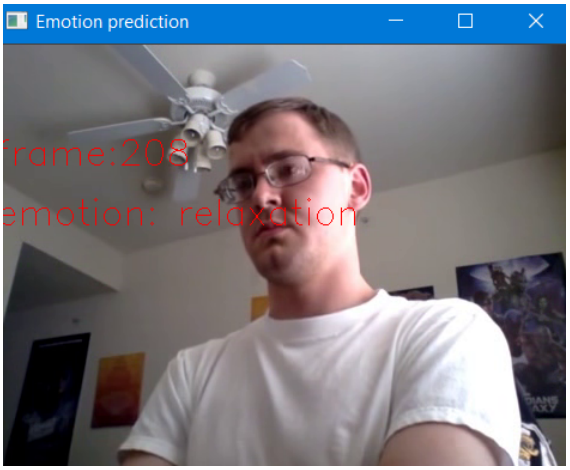

Figure 13: Emotion at Frame 208: Relaxation

Figure 14 below shows the scores of arousal and valence at every 40 frames, whereby the forest array corresponds to the arousal scores, and the second array corresponds to the valence scores.

```
[-0.03532654792070389, -0.030519425868988037, -0.02688261866569519, -0.024053774774074554, -0.021
455207839608192, -0.020694062113761902, -0.02051566354930401, -0.02035398595035076, -0.0201081726
70006752, -0.0197432991117239]
[0.10804653912782669, 0.06844375282526016, 0.04844285547733307, 0.03839634358882904, 0.0334971696
13838196, 0.0316762775182724, 0.030717864632606506, 0.03042556345462799, 0.030621066689491272, 0.
03130599856376648]
```

Figure 14: Arousal and valence scores at every 40 frames of the video used for UI demo

Limitations

As there was high traffic in the cohort accessing the JupyterHub, we were unable to make use of the GPU to carry out the training of our model. This led to us facing time constraints in being able to train our model properly, where although we managed to get the pipeline up, we were unable to optimize the model.

Due to the lack of time and resources, we could not train our model for many epochs as well in order to get a better understanding of our model's performance; where the training/validation losses would begin converging.

Future Works

We could perhaps extend this regression problem to a classification problem, where we could classify facial features into different labels corresponding to emotion; that way we would be able

to compute the accuracy of our training and validation sets and use it as a metric for us to judge if our model is able to generalize well to the validation and test sets.

How to run code

Instructions to run demo UI:

1. Clone directory: git clone https://github.com/jeroee/50.039-Big_Project.git

2. Download dataset from google drive. https://drive.google.com/drive/folders/1GhNLW-MWJbOHdfFbRV_HwPCm3P4kL6zi?usp=sharing. Unzip the folder and replace it with empty 'data' folder in repository

3. Copy full path of any video from data>video>test_trim
    - This is to be input into the GUI (step 5)

4. Install PyQt5:

    Pip install PyQt5

5. Run the GUI : cd to the “Big_Project” directory, run in the terminal:

    python gui.py

    A pyqt gui will pop up and prompt the user to insert a full path of testing video. Click predict to view video frames. Hold 'k' to play the video and

Instructions to train model:

1. Clone directory: git clone https://github.com/jeroee/50.039-Big_Project.git

2. Download dataset from google drive. https://drive.google.com/drive/folders/1GhNLW-MWJbOHdfFbRV_HwPCm3P4kL6zi?usp=sharing unzip the folder and replace it with the empty 'data' folder in repository

3. Change the paths in the Big_Project.ipynb and run all cells to train.

4. Model weights will be saved in 'saved weights' and validations loss will be saved in 'validation_results'.

References

Russell, James. (1980). A Circumplex Model of Affect. Journal of Personality and Social Psychology. 39. 1161-1178. 10.1037/h0077714.

Anwar is a Media Relations Representative at UC Berkeley. (2017). How many different human emotions are there? Retrieved May 3, 2021, from Greater Good website: https://greatergood.berkeley.edu/article/item/how_many_different_human_emotions_are_there

Giordano, D. (2020, July 26). 7 tips to choose the best optimizer. Retrieved from Towards Data Science: https://towardsdatascience.com/7-tips-to-choose-the-best-optimizer-47bb9c1219e

Kathuria, C. (2019, December 5). *Regression — Why Mean Square Error?* Retrieved from Towards Data Science: https://towardsdatascience.com/https-medium-com-chayankathuria-regression-why-mean-square-error-a8cad2a1c96f

Masters, D., & Luschi, C. (2018, April 20). Revisiting Small Batch Training for Deep Neural Network. Retrieved from Cornell University: https://arxiv.org/pdf/1804.07612.pdf