Models for FER

- Gaurav Sharma

All the below models are trained and tested by myself and the results are trustable.

Datasets Under Consideration -

1. FER-13 Cleaned

   Angry - 966 samples
   Sadness - 1326 samples
   Fear - 859 samples
   Neutral - 1446 samples
   Happy - 2477 samples

   This dataset has a total of **7074 images**. All the images are grayscale and have dimension (48x48x1). This dataset contains images taken in **un-controlled settings** and has a lot of variations.

2. FER-13 Aligned

   Angry - 803 samples
   Sadness - 1069 samples
   Fear - 653 samples
   Neutral - 1230 samples
   Happy - 1996 samples

   I applied face alignment to every image of the above dataset and removed all images which fail during this process. The dataset is reduced to a total of **5751 images**.

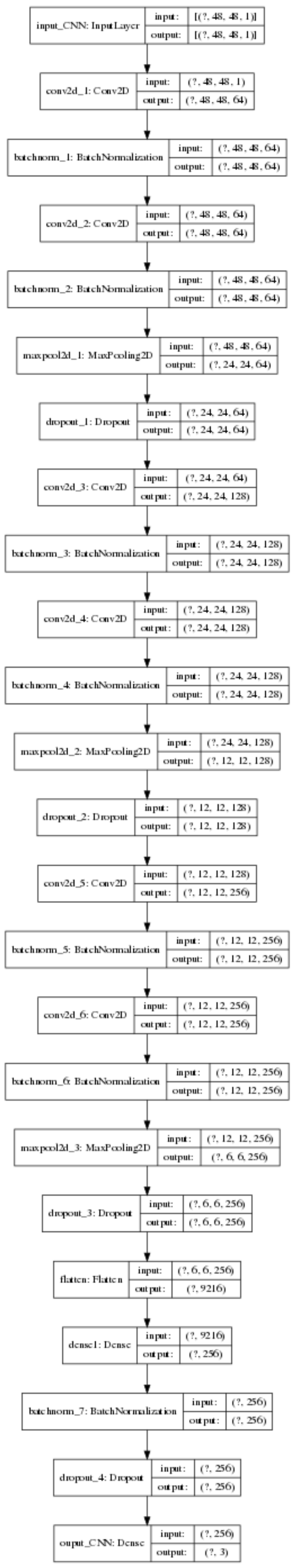
3. CK

   Fear - 75 samples
   Surprise - 249 samples
   Happy - 207 samples
   Sadness - 84 samples
   Angry - 135 samples

   This dataset has a total of **750 images**. All the images are grayscale and have dimension (48x48x1). This dataset is constructed in **controlled settings** and has very clear frontal images.

4. FER-13 Aligned + CK
   I simply combined the **2nd** and **3rd** datasets, it not only results in more data but also 1 more emotion. But the class **surprise** is very low as compared to others (we noticed this data imbalance during testing our model).

1. Convolutional Neural Network (CNN)


input_CNN: InputLayer | input: [(?, 48, 48, 1)] | output: [(?, 48, 48, 1)]
conv2d_1: Conv2D | input: (?, 48, 48, 1) | output: (?, 48, 48, 64)
batchnorm_1: BatchNormalization | input: (?, 48, 48, 64) | output: (?, 48, 48, 64)
conv2d_2: Conv2D | input: (?, 48, 48, 64) | output: (?, 48, 48, 64)
batchnorm_2: BatchNormalization | input: (?, 48, 48, 64) | output: (?, 48, 48, 64)
maxpool2d_1: MaxPooling2D | input: (?, 48, 48, 64) | output: (?, 24, 24, 64)
dropout_1: Dropout | input: (?, 24, 24, 64) | output: (?, 24, 24, 64)
conv2d_3: Conv2D | input: (?, 24, 24, 64) | output: (?, 24, 24, 128)
batchnorm_3: BatchNormalization | input: (?, 24, 24, 128) | output: (?, 24, 24, 128)
conv2d_4: Conv2D | input: (?, 24, 24, 128) | output: (?, 24, 24, 128)
batchnorm_4: BatchNormalization | input: (?, 24, 24, 128) | output: (?, 24, 24, 128)
maxpool2d_2: MaxPooling2D | input: (?, 24, 24, 128) | output: (?, 12, 12, 128)
dropout_2: Dropout | input: (?, 12, 12, 128) | output: (?, 12, 12, 128)
conv2d_5: Conv2D | input: (?, 12, 12, 128) | output: (?, 12, 12, 256)
batchnorm_5: BatchNormalization | input: (?, 12, 12, 256) | output: (?, 12, 12, 256)
conv2d_6: Conv2D | input: (?, 12, 12, 256) | output: (?, 12, 12, 256)
batchnorm_6: BatchNormalization | input: (?, 12, 12, 256) | output: (?, 12, 12, 256)
maxpool2d_3: MaxPooling2D | input: (?, 12, 12, 256) | output: (?, 6, 6, 256)
dropout_3: Dropout | input: (?, 6, 6, 256) | output: (?, 6, 6, 256)
flatten: Flatten | input: (?, 6, 6, 256) | output: (?, 9216)
dense1: Dense | input: (?, 9216) | output: (?, 256)
batchnorm_7: BatchNormalization | input: (?, 256) | output: (?, 256)
dropout_4: Dropout | input: (?, 256) | output: (?, 256)
ouput_CNN: Dense | input: (?, 256) | output: (?, 3)

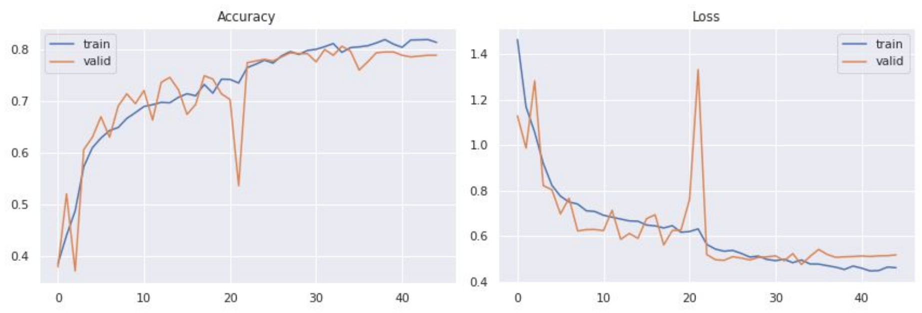
This is a naive CNN, nothing fancy. It has three blocks stacked sequentially, where each block consists of **conv -> batchnorm -> conv -> batchnorm -> maxpool ->dropout** and at the end, there is a single dense layer followed by output layer.

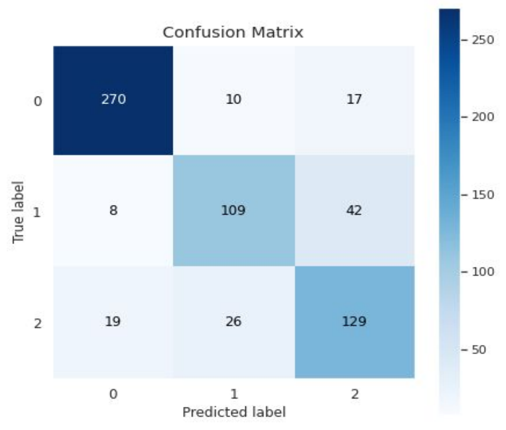
Model evaluation on the following datasets:-

Dataset: FER-13 Cleaned

Emotions: Happy, Sadness, Neutral
Input: Single image
Output: Emotion in that image

Performance: train-test split of 0.88
Best Train Accuracy ~ 82%
Best Test Accuracy ~ 80.09%



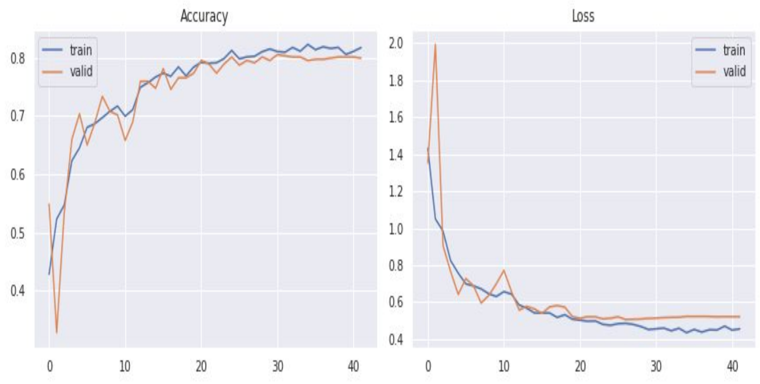


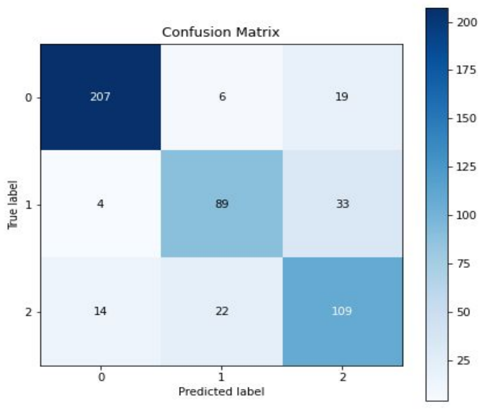
	0	1	2
0	270	10	17
1	8	109	42
2	19	26	129

Dataset: FER-13 Aligned

Emotions: Happy, Sadness, Neutral
Input: Single image
Output: Emotion in that image

Performance: train-test split of 0.88
Best Train Accuracy ~ 82%
Best Test Accuracy ~ 80%

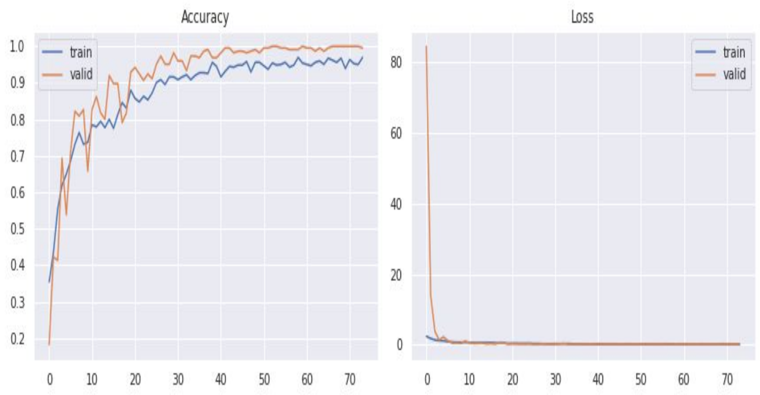


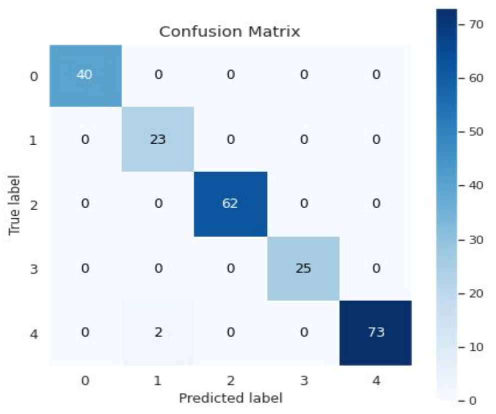


Dataset: CK

Emotions: Happy, Sadness, Surprise, Fear, Angry
Input: Single image
Output: Emotion in that image

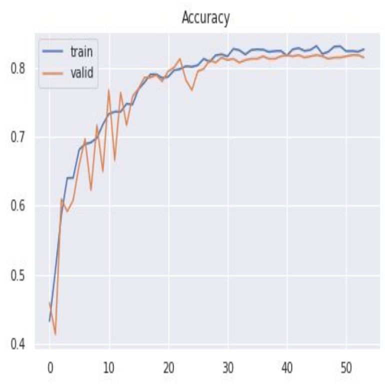
Performance: train-test split of 0.75
Best Train Accuracy ~ 100%
Best Test Accuracy ~ 96%

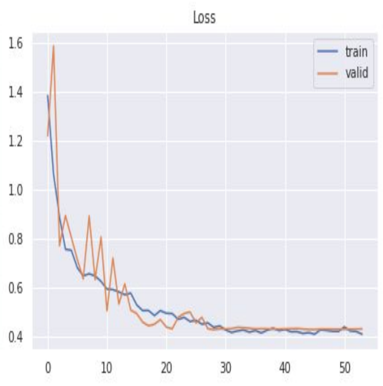


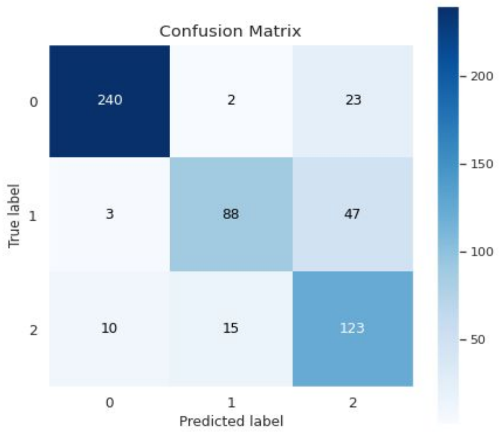


Dataset: FER-13 Aligned + CK
Input: Single image
Output: Emotion in that image
Emotions: Happy, Sadness, Neutral

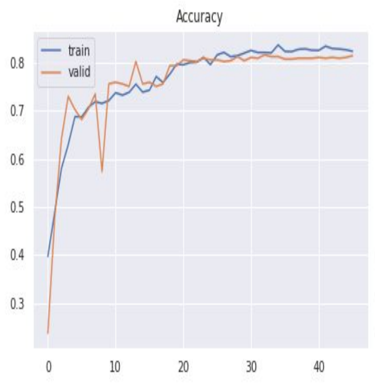
Performance: train-test split of 0.86
Best Train Accuracy ~82%
Best Test Accuracy ~ 81%

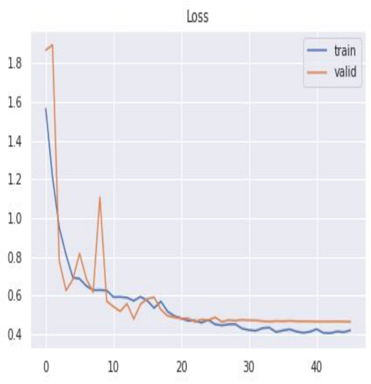


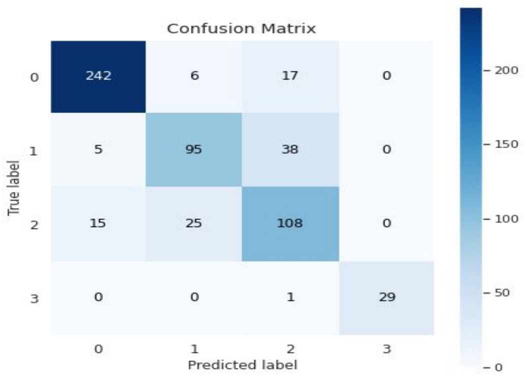




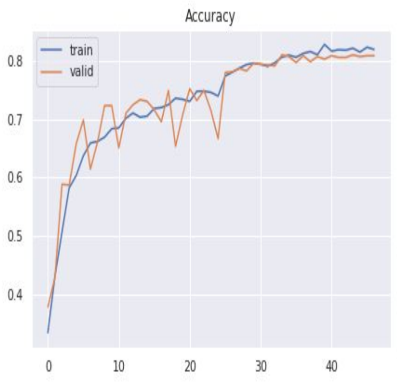
Emotions: Happy, Sadness, Neutral, Surprise
Performance: train-test split of 0.86
Best Train Accuracy ~ 82%
Best Test Accuracy ~ 81%

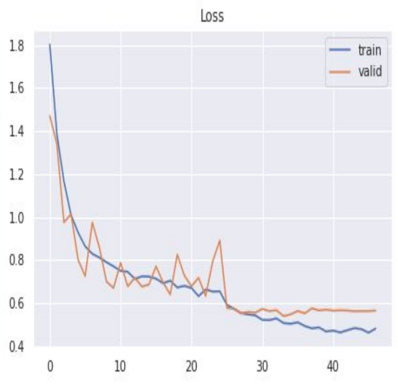


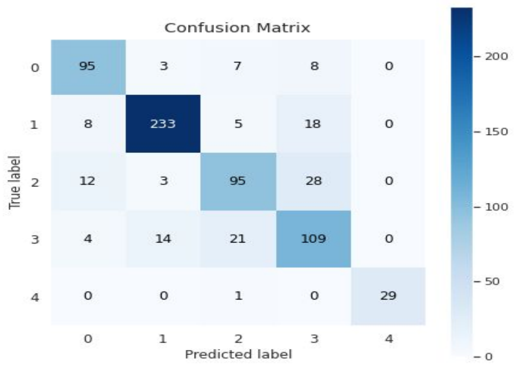




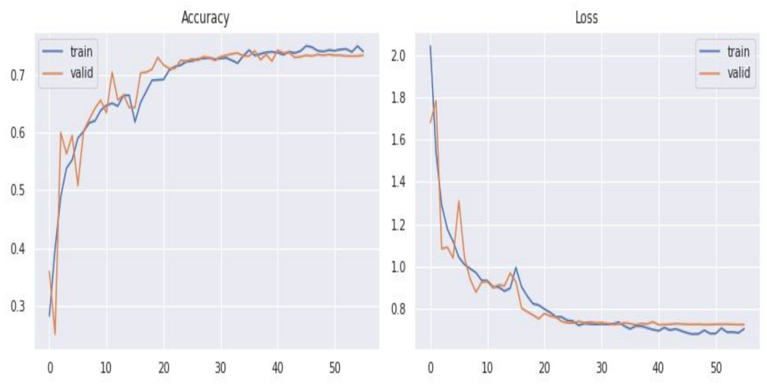
Emotions: Happy, Sadness, Neutral, Surprise, Angry
Performance: train-test split of 0.86
Best Train Accuracy ~ 82%
Best Test Accuracy ~ 80%

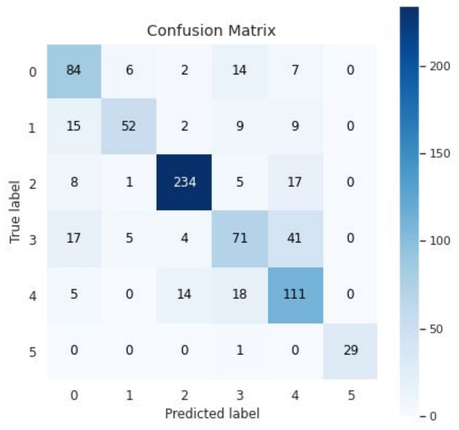






Emotions: Happy, Sadness, Neutral, Surprise, Angry, Fear
Performance: train-test split of 0.86
Best Train Accuracy ~ 76%
Best Test Accuracy ~ 75%





	0	1	2	3	4	5
0	84	6	2	14	7	0
1	15	52	2	9	9	0
2	8	1	234	5	17	0
3	17	5	4	71	41	0
4	5	0	14	18	111	0
5	0	0	0	1	0	29

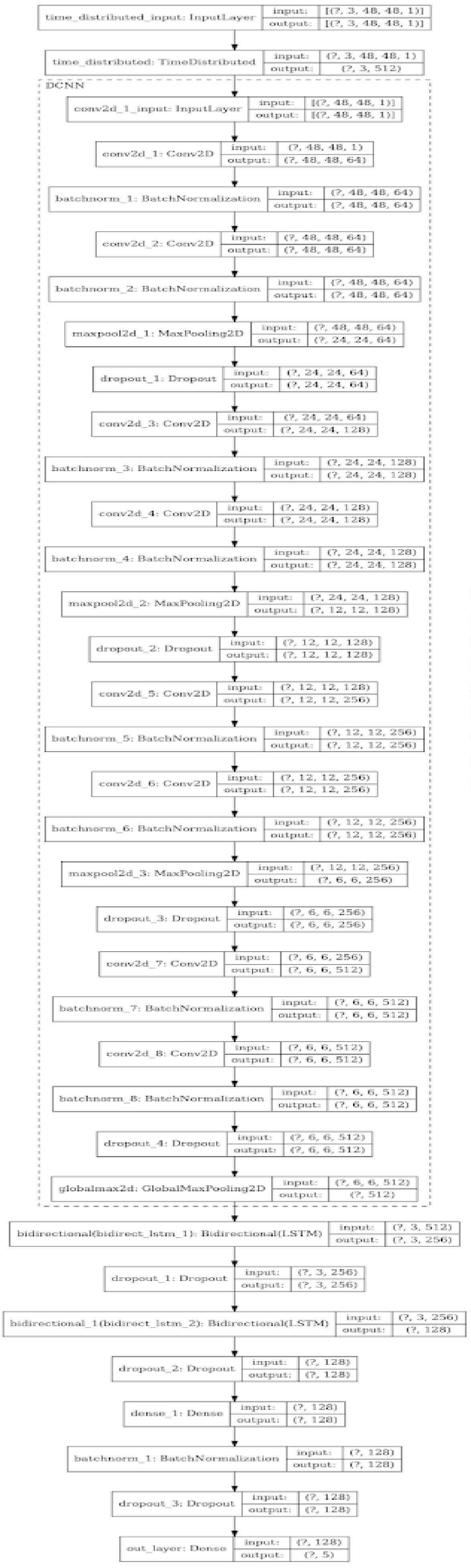
Pros:

- A very simple model (nothing fancy)
- Fast to train and test
- Easy to debug

Cons:

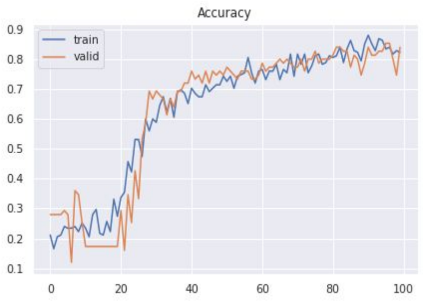
- Model is simple as compared to complex task in hand
- Sometimes focuses on unwanted things as well like nose

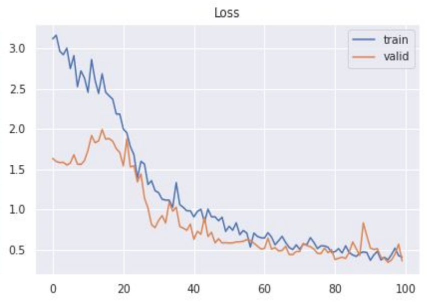
2. TimeDistributed Convolutional Bidirectional LSTM

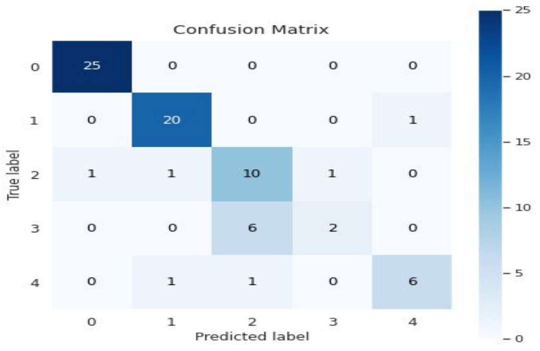

time_distributed_input: InputLayer | input: [(?, 3, 48, 48, 1)] | output: [(?, 3, 48, 48, 1)]
time_distributed: TimeDistributed | input: (?, 3, 48, 48, 1) | output: (?, 3, 512)
DCNN
conv2d_1_input: InputLayer | input: [(?, 48, 48, 1)] | output: [(?, 48, 48, 1)]
conv2d_1: Conv2D | input: (?, 48, 48, 1) | output: (?, 48, 48, 64)
batchnorm_1: BatchNormalization | input: (?, 48, 48, 64) | output: (?, 48, 48, 64)
conv2d_2: Conv2D | input: (?, 48, 48, 64) | output: (?, 48, 48, 64)
batchnorm_2: BatchNormalization | input: (?, 48, 48, 64) | output: (?, 48, 48, 64)
maxpool2d_1: MaxPooling2D | input: (?, 48, 48, 64) | output: (?, 24, 24, 64)
dropout_1: Dropout | input: (?, 24, 24, 64) | output: (?, 24, 24, 64)
conv2d_3: Conv2D | input: (?, 24, 24, 64) | output: (?, 24, 24, 128)
batchnorm_3: BatchNormalization | input: (?, 24, 24, 128) | output: (?, 24, 24, 128)
conv2d_4: Conv2D | input: (?, 24, 24, 128) | output: (?, 24, 24, 128)
batchnorm_4: BatchNormalization | input: (?, 24, 24, 128) | output: (?, 24, 24, 128)
maxpool2d_2: MaxPooling2D | input: (?, 24, 24, 128) | output: (?, 12, 12, 128)
dropout_2: Dropout | input: (?, 12, 12, 128) | output: (?, 12, 12, 128)
conv2d_5: Conv2D | input: (?, 12, 12, 128) | output: (?, 12, 12, 256)
batchnorm_5: BatchNormalization | input: (?, 12, 12, 256) | output: (?, 12, 12, 256)
conv2d_6: Conv2D | input: (?, 12, 12, 256) | output: (?, 12, 12, 256)
batchnorm_6: BatchNormalization | input: (?, 12, 12, 256) | output: (?, 12, 12, 256)
maxpool2d_3: MaxPooling2D | input: (?, 12, 12, 256) | output: (?, 6, 6, 256)
dropout_3: Dropout | input: (?, 6, 6, 256) | output: (?, 6, 6, 256)
conv2d_7: Conv2D | input: (?, 6, 6, 256) | output: (?, 6, 6, 512)
batchnorm_7: BatchNormalization | input: (?, 6, 6, 512) | output: (?, 6, 6, 512)
conv2d_8: Conv2D | input: (?, 6, 6, 512) | output: (?, 6, 6, 512)
batchnorm_8: BatchNormalization | input: (?, 6, 6, 512) | output: (?, 6, 6, 512)
dropout_4: Dropout | input: (?, 6, 6, 512) | output: (?, 6, 6, 512)
globalmax2d: GlobalMaxPooling2D | input: (?, 6, 6, 512) | output: (?, 512)
bidirectional(bidirect_lstm_1): Bidirectional(LSTM) | input: (?, 3, 512) | output: (?, 3, 256)
dropout_1: Dropout | input: (?, 3, 256) | output: (?, 3, 256)
bidirectional_1(bidirect_lstm_2): Bidirectional(LSTM) | input: (?, 3, 256) | output: (?, 128)
dropout_2: Dropout | input: (?, 128) | output: (?, 128)
dense_1: Dense | input: (?, 128) | output: (?, 128)
batchnorm_1: BatchNormalization | input: (?, 128) | output: (?, 128)
dropout_3: Dropout | input: (?, 128) | output: (?, 128)
out_layer: Dense | input: (?, 128) | output: (?, 5)

This is a **memory-based model**. In this model, all frames of a video are first passed to CNN, after that the output is passed to bidirectional LSTM and then to dense layers. Because this model requires sequential (video-based data) so I only applied this on CK. The CK dataset we have in our hand is the cropped one, in which for each video we have the images of the last three frames. So I stacked those 3 frames in time dimension resulting in a **3 frame video** (reverse engineering).

Dataset: Ck
Emotions: Happy, Surprise, Angry, Sadness, Fear
Input: Sequence of images(frames of video)
Output: Single emotion in that entire image sequence
Performance: train-test split of 0.7
Best Train Accuracy ~ 87%
Best Test Accuracy ~ 85%







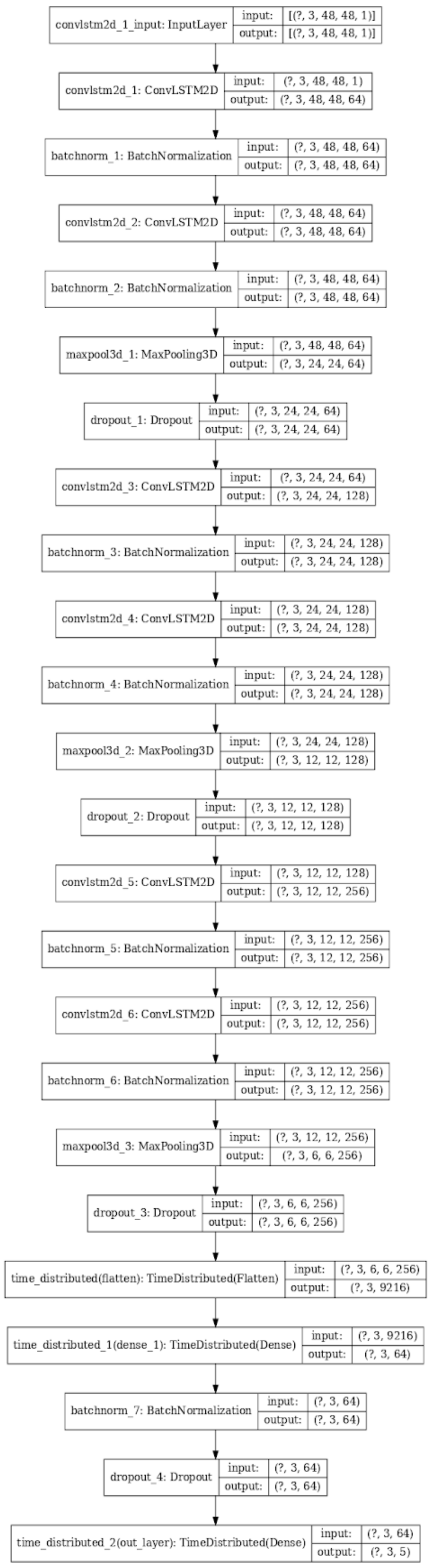
Pros:

- Learn the context from past frames.

Cons:

- Takes more time to train on large data

3. ConvLSTM2D


convlstm2d_1_input: InputLayer | input: [(?, 3, 48, 48, 1)] | output: [(?, 3, 48, 48, 1)]
convlstm2d_1: ConvLSTM2D | input: (?, 3, 48, 48, 1) | output: (?, 3, 48, 48, 64)
batchnorm_1: BatchNormalization | input: (?, 3, 48, 48, 64) | output: (?, 3, 48, 48, 64)
convlstm2d_2: ConvLSTM2D | input: (?, 3, 48, 48, 64) | output: (?, 3, 48, 48, 64)
batchnorm_2: BatchNormalization | input: (?, 3, 48, 48, 64) | output: (?, 3, 48, 48, 64)
maxpool3d_1: MaxPooling3D | input: (?, 3, 48, 48, 64) | output: (?, 3, 24, 24, 64)
dropout_1: Dropout | input: (?, 3, 24, 24, 64) | output: (?, 3, 24, 24, 64)
convlstm2d_3: ConvLSTM2D | input: (?, 3, 24, 24, 64) | output: (?, 3, 24, 24, 128)
batchnorm_3: BatchNormalization | input: (?, 3, 24, 24, 128) | output: (?, 3, 24, 24, 128)
convlstm2d_4: ConvLSTM2D | input: (?, 3, 24, 24, 128) | output: (?, 3, 24, 24, 128)
batchnorm_4: BatchNormalization | input: (?, 3, 24, 24, 128) | output: (?, 3, 24, 24, 128)
maxpool3d_2: MaxPooling3D | input: (?, 3, 24, 24, 128) | output: (?, 3, 12, 12, 128)
dropout_2: Dropout | input: (?, 3, 12, 12, 128) | output: (?, 3, 12, 12, 128)
convlstm2d_5: ConvLSTM2D | input: (?, 3, 12, 12, 128) | output: (?, 3, 12, 12, 256)
batchnorm_5: BatchNormalization | input: (?, 3, 12, 12, 256) | output: (?, 3, 12, 12, 256)
convlstm2d_6: ConvLSTM2D | input: (?, 3, 12, 12, 256) | output: (?, 3, 12, 12, 256)
batchnorm_6: BatchNormalization | input: (?, 3, 12, 12, 256) | output: (?, 3, 12, 12, 256)
maxpool3d_3: MaxPooling3D | input: (?, 3, 12, 12, 256) | output: (?, 3, 6, 6, 256)
dropout_3: Dropout | input: (?, 3, 6, 6, 256) | output: (?, 3, 6, 6, 256)
time_distributed(flatten): TimeDistributed(Flatten) | input: (?, 3, 6, 6, 256) | output: (?, 3, 9216)
time_distributed_1(dense_1): TimeDistributed(Dense) | input: (?, 3, 9216) | output: (?, 3, 64)
batchnorm_7: BatchNormalization | input: (?, 3, 64) | output: (?, 3, 64)
dropout_4: Dropout | input: (?, 3, 64) | output: (?, 3, 64)
time_distributed_2(out_layer): TimeDistributed(Dense) | input: (?, 3, 64) | output: (?, 3, 5)

This is a **memory-based model**. In this model, all frames of a video are first passed to convLSTM2D and then to time distributed dense layers.

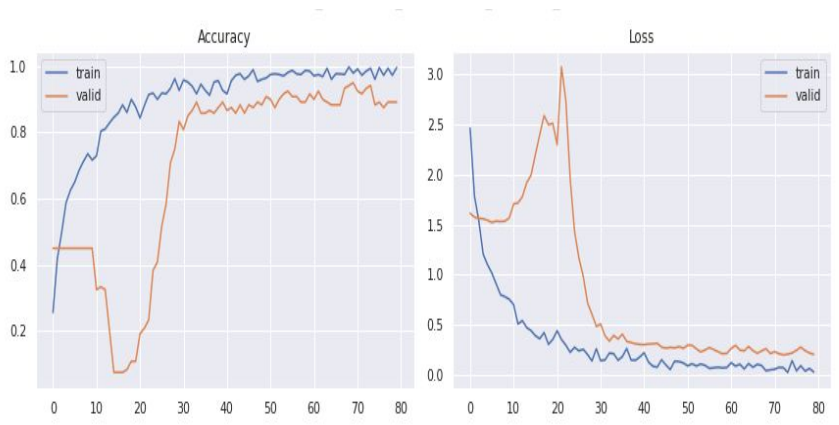
Dataset Used: CK

Emotions: Happy, Surprise, Angry, Sadness, Fear

Input: Sequence of images(frames of video)

Output: One emotion for each frame in that sequence

Performance: As the dataset is very small and the results are not very great for now so I just showed the end results because until it is tested on good quantity data these results may miss-lead.



Pros:

- Learn the context from past frames.

Cons:

- Takes more time to train on large data

4. CNN Wrapped with NSL

This model is the same as the previous CNN but there is an extra regularization added to the model.

Dataset: FER-13 Cleaned

Emotions: Happy, Sadness, Neutral

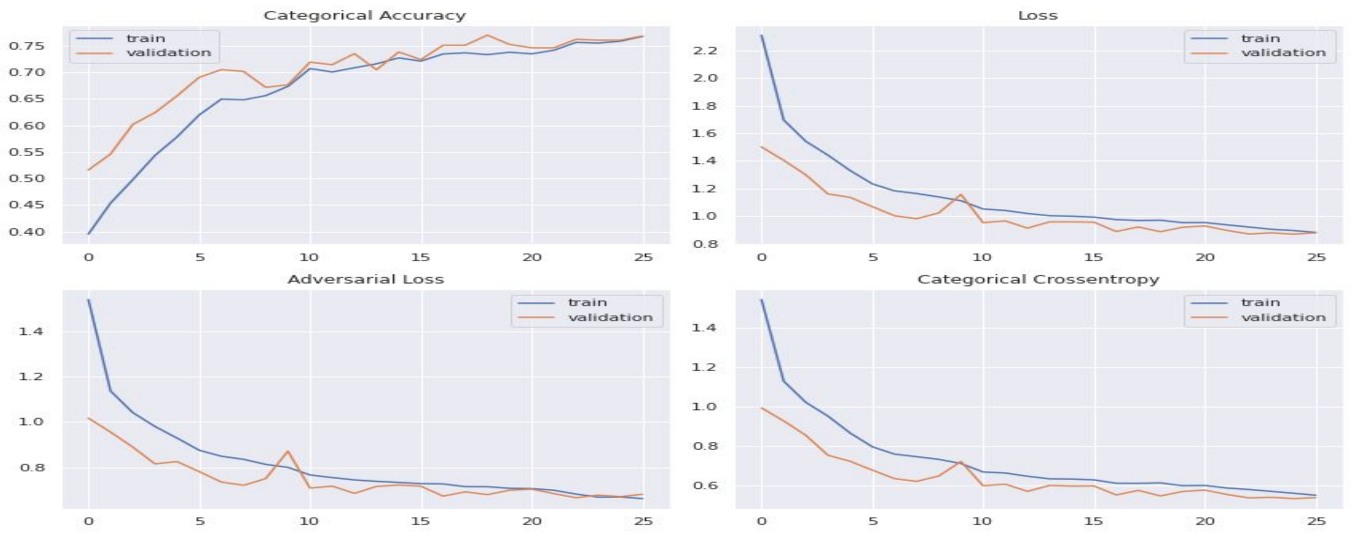
Input: Single image

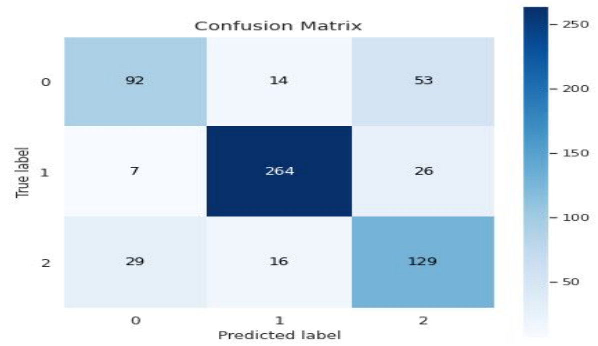
Output: Emotion in that image

Performance: train-test split of 0.7

Best Train Accuracy ~ 76%

Best Test Accuracy ~ 76%





I was expecting the performance to be better than the base CNN model but the accuracy achieved is slightly low. Maybe we can tweak the parameter more and see the results. I tried 9 different configurations for the two NSL parameters and got almost the same result for all configurations. You can find my results here. As this NSL thing is recently introduced so there are no resources out there except official TensorFlow blogs.

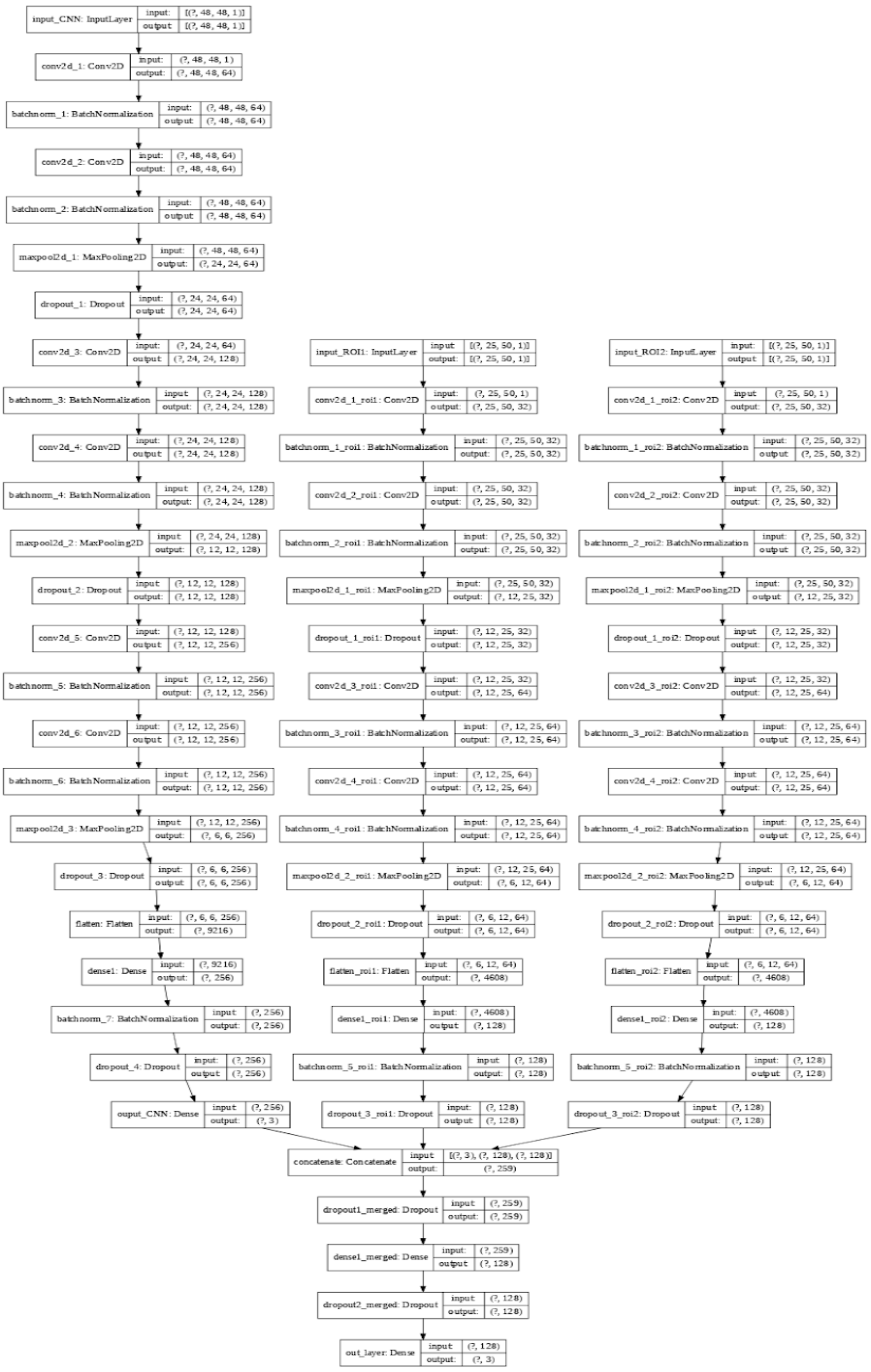
Pros:

- Add an extra regularization term
- Good for adversarially perturbed images

Cons:

- As it is introduced very recently so fewer resources are out there.
- Generally overfits
- Takes more time to train

5. Raw Image + ROI_1 + ROI_2

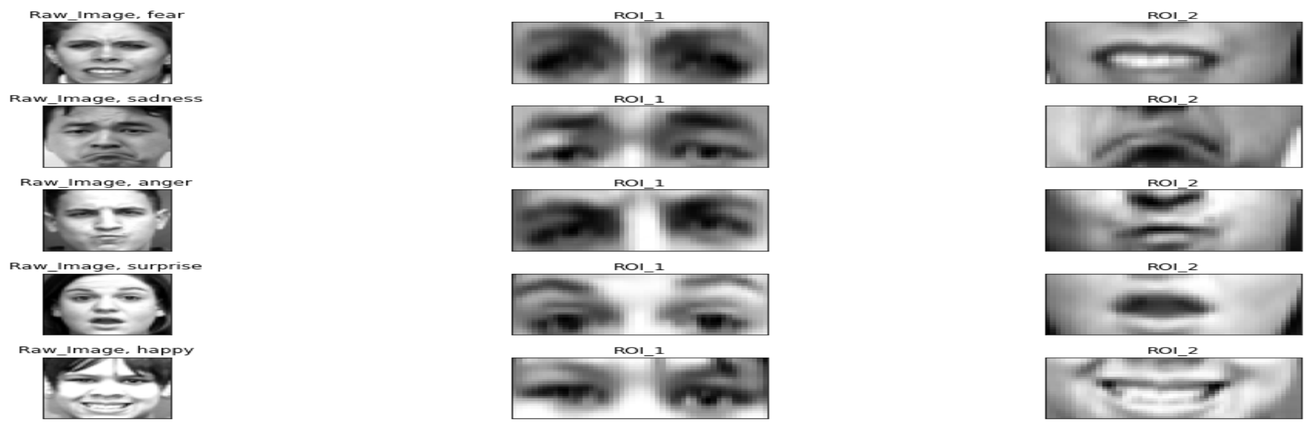


input_CNN: InputLayer | input: [(?, 48, 48, 1)] | output: [(?, 48, 48, 1)]
conv2d_1: Conv2D | input: (?, 48, 48, 1) | output: (?, 48, 48, 64)
batchnorm_1: BatchNormalization | input: (?, 48, 48, 64) | output: (?, 48, 48, 64)
conv2d_2: Conv2D | input: (?, 48, 48, 64) | output: (?, 48, 48, 64)
batchnorm_2: BatchNormalization | input: (?, 48, 48, 64) | output: (?, 48, 48, 64)
maxpool2d_1: MaxPooling2D | input: (?, 48, 48, 64) | output: (?, 24, 24, 64)
dropout_1: Dropout | input: (?, 24, 24, 64) | output: (?, 24, 24, 64)
conv2d_3: Conv2D | input: (?, 24, 24, 64) | output: (?, 24, 24, 128)
batchnorm_3: BatchNormalization | input: (?, 24, 24, 128) | output: (?, 24, 24, 128)
conv2d_4: Conv2D | input: (?, 24, 24, 128) | output: (?, 24, 24, 128)
batchnorm_4: BatchNormalization | input: (?, 24, 24, 128) | output: (?, 24, 24, 128)
maxpool2d_2: MaxPooling2D | input: (?, 24, 24, 128) | output: (?, 12, 12, 128)
dropout_2: Dropout | input: (?, 12, 12, 128) | output: (?, 12, 12, 128)
conv2d_5: Conv2D | input: (?, 12, 12, 128) | output: (?, 12, 12, 256)
batchnorm_5: BatchNormalization | input: (?, 12, 12, 256) | output: (?, 12, 12, 256)
conv2d_6: Conv2D | input: (?, 12, 12, 256) | output: (?, 12, 12, 256)
batchnorm_6: BatchNormalization | input: (?, 12, 12, 256) | output: (?, 12, 12, 256)
maxpool2d_3: MaxPooling2D | input: (?, 12, 12, 256) | output: (?, 6, 6, 256)
dropout_3: Dropout | input: (?, 6, 6, 256) | output: (?, 6, 6, 256)
flatten: Flatten | input: (?, 6, 6, 256) | output: (?, 9216)
dense1: Dense | input: (?, 9216) | output: (?, 256)
batchnorm_7: BatchNormalization | input: (?, 256) | output: (?, 256)
dropout_4: Dropout | input: (?, 256) | output: (?, 256)
ouput_CNN: Dense | input: (?, 256) | output: (?, 3)
input_ROI1: InputLayer | input: [(?, 25, 50, 1)] | output: [(?, 25, 50, 1)]
conv2d_1_roi1: Conv2D | input: (?, 25, 50, 1) | output: (?, 25, 50, 32)
batchnorm_1_roi1: BatchNormalization | input: (?, 25, 50, 32) | output: (?, 25, 50, 32)
conv2d_2_roi1: Conv2D | input: (?, 25, 50, 32) | output: (?, 25, 50, 32)
batchnorm_2_roi1: BatchNormalization | input: (?, 25, 50, 32) | output: (?, 25, 50, 32)
maxpool2d_1_roi1: MaxPooling2D | input: (?, 25, 50, 32) | output: (?, 12, 25, 32)
dropout_1_roi1: Dropout | input: (?, 12, 25, 32) | output: (?, 12, 25, 32)
conv2d_3_roi1: Conv2D | input: (?, 12, 25, 32) | output: (?, 12, 25, 64)
batchnorm_3_roi1: BatchNormalization | input: (?, 12, 25, 64) | output: (?, 12, 25, 64)
conv2d_4_roi1: Conv2D | input: (?, 12, 25, 64) | output: (?, 12, 25, 64)
batchnorm_4_roi1: BatchNormalization | input: (?, 12, 25, 64) | output: (?, 12, 25, 64)
maxpool2d_2_roi1: MaxPooling2D | input: (?, 12, 25, 64) | output: (?, 6, 12, 64)
dropout_2_roi1: Dropout | input: (?, 6, 12, 64) | output: (?, 6, 12, 64)
flatten_roi1: Flatten | input: (?, 6, 12, 64) | output: (?, 4608)
dense1_roi1: Dense | input: (?, 4608) | output: (?, 128)
batchnorm_5_roi1: BatchNormalization | input: (?, 128) | output: (?, 128)
dropout_3_roi1: Dropout | input: (?, 128) | output: (?, 128)
input_ROI2: InputLayer | input: [(?, 25, 50, 1)] | output: [(?, 25, 50, 1)]
conv2d_1_roi2: Conv2D | input: (?, 25, 50, 1) | output: (?, 25, 50, 32)
batchnorm_1_roi2: BatchNormalization | input: (?, 25, 50, 32) | output: (?, 25, 50, 32)
conv2d_2_roi2: Conv2D | input: (?, 25, 50, 32) | output: (?, 25, 50, 32)
batchnorm_2_roi2: BatchNormalization | input: (?, 25, 50, 32) | output: (?, 25, 50, 32)
maxpool2d_1_roi2: MaxPooling2D | input: (?, 25, 50, 32) | output: (?, 12, 25, 32)
dropout_1_roi2: Dropout | input: (?, 12, 25, 32) | output: (?, 12, 25, 32)
conv2d_3_roi2: Conv2D | input: (?, 12, 25, 32) | output: (?, 12, 25, 64)
batchnorm_3_roi2: BatchNormalization | input: (?, 12, 25, 64) | output: (?, 12, 25, 64)
conv2d_4_roi2: Conv2D | input: (?, 12, 25, 64) | output: (?, 12, 25, 64)
batchnorm_4_roi2: BatchNormalization | input: (?, 12, 25, 64) | output: (?, 12, 25, 64)
maxpool2d_2_roi2: MaxPooling2D | input: (?, 12, 25, 64) | output: (?, 6, 12, 64)
dropout_2_roi2: Dropout | input: (?, 6, 12, 64) | output: (?, 6, 12, 64)
flatten_roi2: Flatten | input: (?, 6, 12, 64) | output: (?, 4608)
dense1_roi2: Dense | input: (?, 4608) | output: (?, 128)
batchnorm_5_roi2: BatchNormalization | input: (?, 128) | output: (?, 128)
dropout_3_roi2: Dropout | input: (?, 128) | output: (?, 128)
concatenate: Concatenate | input: [(?, 3), (?, 128), (?, 128)] | output: (?, 259)
dropout1_merged: Dropout | input: (?, 259) | output: (?, 259)
dense1_merged: Dense | input: (?, 259) | output: (?, 128)
dropout2_merged: Dropout | input: (?, 128) | output: (?, 128)
out_layer: Dense | input: (?, 128) | output: (?, 3)

This is a **multi-input model** with 3 image inputs,

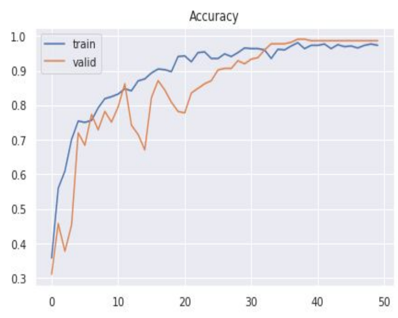
1. Raw image, input_shape = 48x48x1
2. Region of interest 1 (ROI_1), this contains the region of eyebrows and eyes, input_shape = 25x50x1
3. Region of interest 2 (ROI_2), this contains region of lips, input_shape = 25x50x1

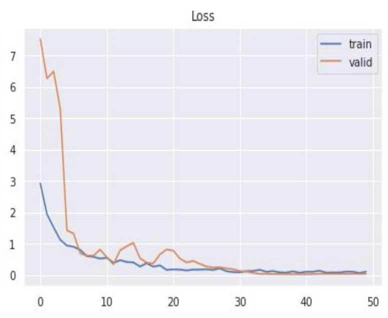
In the below image the first column is the raw image whereas the second and third columns are the ROI_1 and ROI_2 of that image respectively.

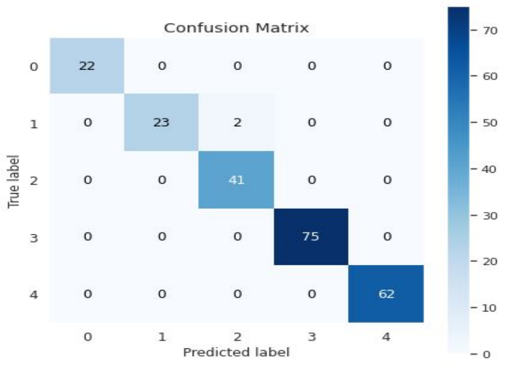


Dataset Used: CK

Emotions: Happy, Surprise, Anger, Sadness, Fear
Input: Single image
Output: Emotion in that image
Performance: train-test split of 0.7
Best Train Accuracy ~ 97%
Best Test Accuracy ~ 99%





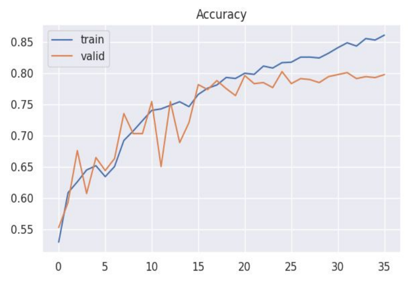


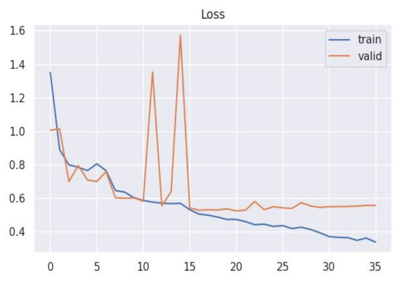
Dataset Used: FER-13 Aligned + CK

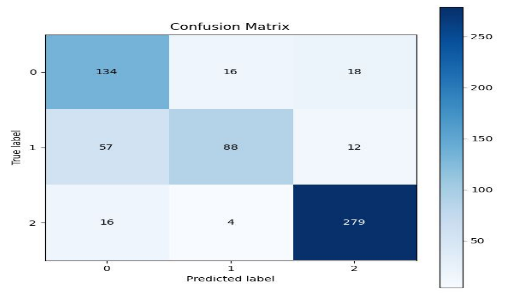
Input: Single image
Output: Emotion in that image

Emotions: Happy, Sadness, Neutral
Performance: train-test split of 0.86
Best Train Accuracy ~86%
Best Test Accuracy ~ 80%

Emotions: Happy, Sadness, Neutral, Surprise



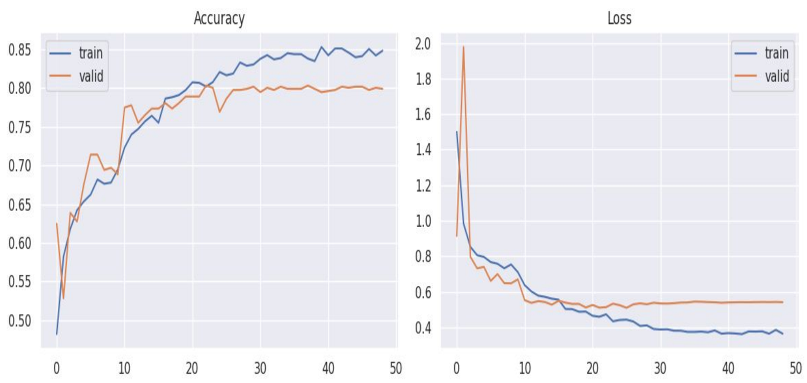


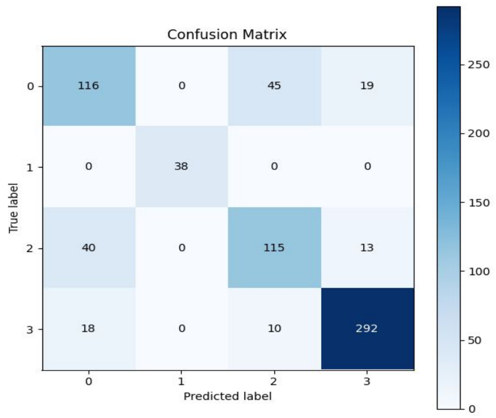


Performance: train-test split of 0.85

Best Train Accuracy ~ 85%

Best Test Accuracy ~ 80%



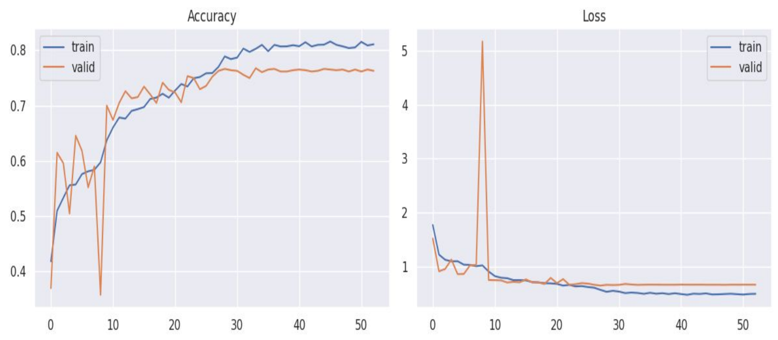


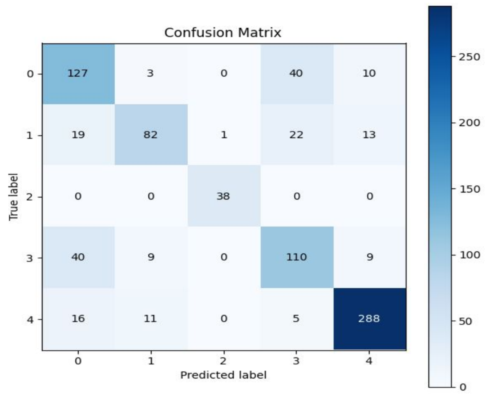
Emotions: Happy, Sadness, Neutral, Surprise, Angry

Performance: train-test split of 0.85

Best Train Accuracy ~ 81%

Best Test Accuracy ~ 76%





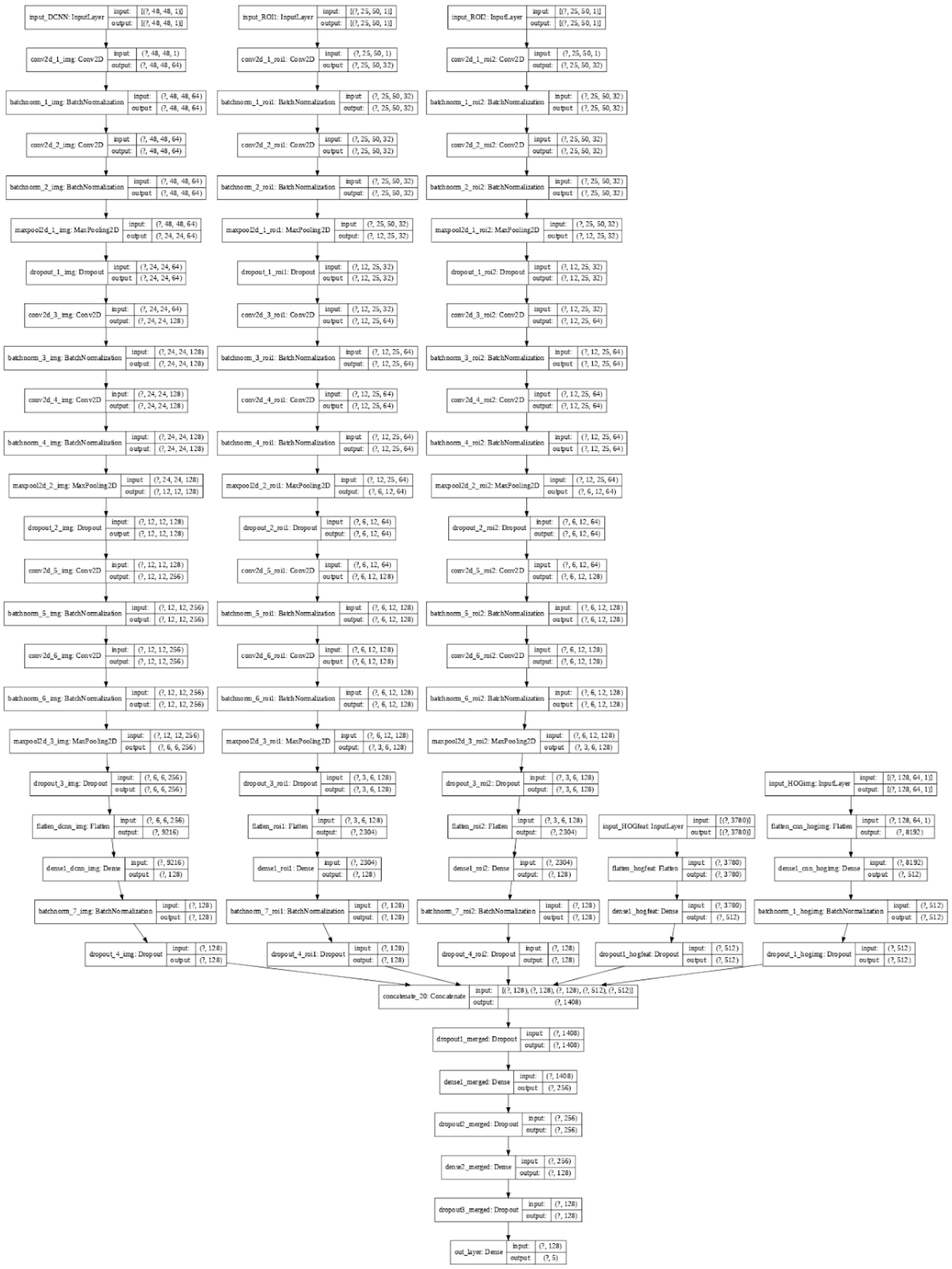
Pros:

- Process important regions separately
- Performs well especially for CK dataset

Cons:

- Doesn't work well with unaligned images

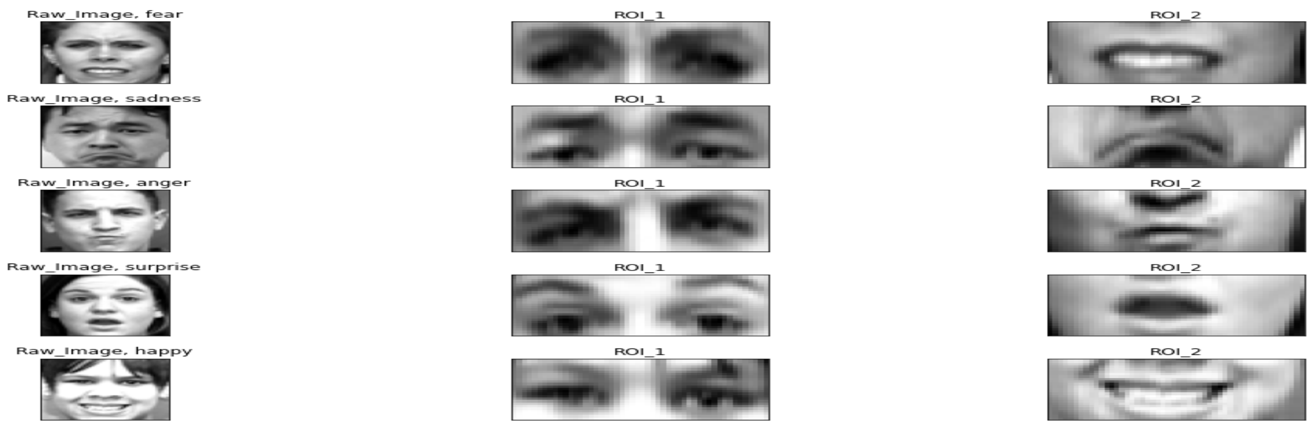
6. Raw Image + ROI_1 + ROI_2 + HOG_FEAT + HOG_IMG


input_DCNN: InputLayer | input: [(?, 48, 48, 1)] | output: [(?, 48, 48, 1)]
conv2d_1_img: Conv2D | input: (?, 48, 48, 1) | output: (?, 48, 48, 64)
batchnorm_1_img: BatchNormalization | input: (?, 48, 48, 64) | output: (?, 48, 48, 64)
conv2d_2_img: Conv2D | input: (?, 48, 48, 64) | output: (?, 48, 48, 64)
batchnorm_2_img: BatchNormalization | input: (?, 48, 48, 64) | output: (?, 48, 48, 64)
maxpool2d_1_img: MaxPooling2D | input: (?, 48, 48, 64) | output: (?, 24, 24, 64)
dropout_1_img: Dropout | input: (?, 24, 24, 64) | output: (?, 24, 24, 64)
conv2d_3_img: Conv2D | input: (?, 24, 24, 64) | output: (?, 24, 24, 128)
batchnorm_3_img: BatchNormalization | input: (?, 24, 24, 128) | output: (?, 24, 24, 128)
conv2d_4_img: Conv2D | input: (?, 24, 24, 128) | output: (?, 24, 24, 128)
batchnorm_4_img: BatchNormalization | input: (?, 24, 24, 128) | output: (?, 24, 24, 128)
maxpool2d_2_img: MaxPooling2D | input: (?, 24, 24, 128) | output: (?, 12, 12, 128)
dropout_2_img: Dropout | input: (?, 12, 12, 128) | output: (?, 12, 12, 128)
conv2d_5_img: Conv2D | input: (?, 12, 12, 128) | output: (?, 12, 12, 256)
batchnorm_5_img: BatchNormalization | input: (?, 12, 12, 256) | output: (?, 12, 12, 256)
conv2d_6_img: Conv2D | input: (?, 12, 12, 256) | output: (?, 12, 12, 256)
batchnorm_6_img: BatchNormalization | input: (?, 12, 12, 256) | output: (?, 12, 12, 256)
maxpool2d_3_img: MaxPooling2D | input: (?, 12, 12, 256) | output: (?, 6, 6, 256)
dropout_3_img: Dropout | input: (?, 6, 6, 256) | output: (?, 6, 6, 256)
flatten_dcnn_img: Flatten | input: (?, 6, 6, 256) | output: (?, 9216)
dense1_dcnn_img: Dense | input: (?, 9216) | output: (?, 128)
batchnorm_7_img: BatchNormalization | input: (?, 128) | output: (?, 128)
dropout_4_img: Dropout | input: (?, 128) | output: (?, 128)
input_ROI1: InputLayer | input: [(?, 25, 50, 1)] | output: [(?, 25, 50, 1)]
conv2d_1_roi1: Conv2D | input: (?, 25, 50, 1) | output: (?, 25, 50, 32)
batchnorm_1_roi1: BatchNormalization | input: (?, 25, 50, 32) | output: (?, 25, 50, 32)
conv2d_2_roi1: Conv2D | input: (?, 25, 50, 32) | output: (?, 25, 50, 32)
batchnorm_2_roi1: BatchNormalization | input: (?, 25, 50, 32) | output: (?, 25, 50, 32)
maxpool2d_1_roi1: MaxPooling2D | input: (?, 25, 50, 32) | output: (?, 12, 25, 32)
dropout_1_roi1: Dropout | input: (?, 12, 25, 32) | output: (?, 12, 25, 32)
conv2d_3_roi1: Conv2D | input: (?, 12, 25, 32) | output: (?, 12, 25, 64)
batchnorm_3_roi1: BatchNormalization | input: (?, 12, 25, 64) | output: (?, 12, 25, 64)
conv2d_4_roi1: Conv2D | input: (?, 12, 25, 64) | output: (?, 12, 25, 64)
batchnorm_4_roi1: BatchNormalization | input: (?, 12, 25, 64) | output: (?, 12, 25, 64)
maxpool2d_2_roi1: MaxPooling2D | input: (?, 12, 25, 64) | output: (?, 6, 12, 64)
dropout_2_roi1: Dropout | input: (?, 6, 12, 64) | output: (?, 6, 12, 64)
conv2d_5_roi1: Conv2D | input: (?, 6, 12, 64) | output: (?, 6, 12, 128)
batchnorm_5_roi1: BatchNormalization | input: (?, 6, 12, 128) | output: (?, 6, 12, 128)
conv2d_6_roi1: Conv2D | input: (?, 6, 12, 128) | output: (?, 6, 12, 128)
batchnorm_6_roi1: BatchNormalization | input: (?, 6, 12, 128) | output: (?, 6, 12, 128)
maxpool2d_3_roi1: MaxPooling2D | input: (?, 6, 12, 128) | output: (?, 3, 6, 128)
dropout_3_roi1: Dropout | input: (?, 3, 6, 128) | output: (?, 3, 6, 128)
flatten_roi1: Flatten | input: (?, 3, 6, 128) | output: (?, 2304)
dense1_roi1: Dense | input: (?, 2304) | output: (?, 128)
batchnorm_7_roi1: BatchNormalization | input: (?, 128) | output: (?, 128)
dropout_4_roi1: Dropout | input: (?, 128) | output: (?, 128)
input_ROI2: InputLayer | input: [(?, 25, 50, 1)] | output: [(?, 25, 50, 1)]
conv2d_1_roi2: Conv2D | input: (?, 25, 50, 1) | output: (?, 25, 50, 32)
batchnorm_1_roi2: BatchNormalization | input: (?, 25, 50, 32) | output: (?, 25, 50, 32)
conv2d_2_roi2: Conv2D | input: (?, 25, 50, 32) | output: (?, 25, 50, 32)
batchnorm_2_roi2: BatchNormalization | input: (?, 25, 50, 32) | output: (?, 25, 50, 32)
maxpool2d_1_roi2: MaxPooling2D | input: (?, 25, 50, 32) | output: (?, 12, 25, 32)
dropout_1_roi2: Dropout | input: (?, 12, 25, 32) | output: (?, 12, 25, 32)
conv2d_3_roi2: Conv2D | input: (?, 12, 25, 32) | output: (?, 12, 25, 64)
batchnorm_3_roi2: BatchNormalization | input: (?, 12, 25, 64) | output: (?, 12, 25, 64)
conv2d_4_roi2: Conv2D | input: (?, 12, 25, 64) | output: (?, 12, 25, 64)
batchnorm_4_roi2: BatchNormalization | input: (?, 12, 25, 64) | output: (?, 12, 25, 64)
maxpool2d_2_roi2: MaxPooling2D | input: (?, 12, 25, 64) | output: (?, 6, 12, 64)
dropout_2_roi2: Dropout | input: (?, 6, 12, 64) | output: (?, 6, 12, 64)
conv2d_5_roi2: Conv2D | input: (?, 6, 12, 64) | output: (?, 6, 12, 128)
batchnorm_5_roi2: BatchNormalization | input: (?, 6, 12, 128) | output: (?, 6, 12, 128)
conv2d_6_roi2: Conv2D | input: (?, 6, 12, 128) | output: (?, 6, 12, 128)
batchnorm_6_roi2: BatchNormalization | input: (?, 6, 12, 128) | output: (?, 6, 12, 128)
maxpool2d_3_roi2: MaxPooling2D | input: (?, 6, 12, 128) | output: (?, 3, 6, 128)
dropout_3_roi2: Dropout | input: (?, 3, 6, 128) | output: (?, 3, 6, 128)
flatten_roi2: Flatten | input: (?, 3, 6, 128) | output: (?, 2304)
dense1_roi2: Dense | input: (?, 2304) | output: (?, 128)
batchnorm_7_roi2: BatchNormalization | input: (?, 128) | output: (?, 128)
dropout_4_roi2: Dropout | input: (?, 128) | output: (?, 128)
input_HOGfeat: InputLayer | input: [(?, 3780)] | output: [(?, 3780)]
flatten_hogfeat: Flatten | input: (?, 3780) | output: (?, 3780)
dense1_hogfeat: Dense | input: (?, 3780) | output: (?, 512)
dropout1_hogfeat: Dropout | input: (?, 512) | output: (?, 512)
input_HOGimg: InputLayer | input: [(?, 128, 64, 1)] | output: [(?, 128, 64, 1)]
flatten_cnn_hogimg: Flatten | input: (?, 128, 64, 1) | output: (?, 8192)
dense1_cnn_hogimg: Dense | input: (?, 8192) | output: (?, 512)
batchnorm_1_hogimg: BatchNormalization | input: (?, 512) | output: (?, 512)
dropout_1_hogimg: Dropout | input: (?, 512) | output: (?, 512)
concatenate_20: Concatenate | input: [(?, 128), (?, 128), (?, 128), (?, 512), (?, 512)] | output: (?, 1408)
dropout1_merged: Dropout | input: (?, 1408) | output: (?, 1408)
dense1_merged: Dense | input: (?, 1408) | output: (?, 256)
dropout2_merged: Dropout | input: (?, 256) | output: (?, 256)
dense2_merged: Dense | input: (?, 256) | output: (?, 128)
dropout3_merged: Dropout | input: (?, 128) | output: (?, 128)
out_layer: Dense | input: (?, 128) | output: (?, 5)

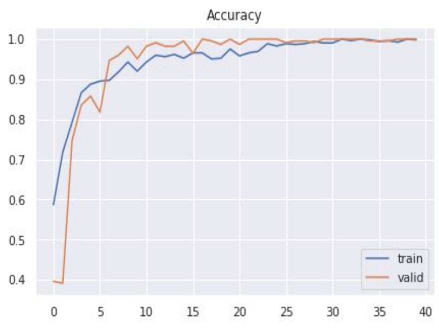
This is a **multi-input model** with 5 inputs,

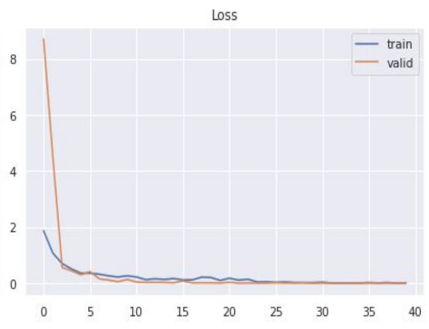
1. Raw image, input_shape = 48x48x1
2. Region of interest 1 (ROI_1), this contains the region of eyebrows and eyes, input_shape = 25x50x1
3. Region of interest 2 (ROI_2), this contains the region of lips, input_shape = 25x50x1
4. HOG features (HOG_FEAT), this contains hog feature descriptors, input_shape = 3780x1
5. HOG img (HOG_IMG), this contains hog feature image, input_shape = 128x64x1

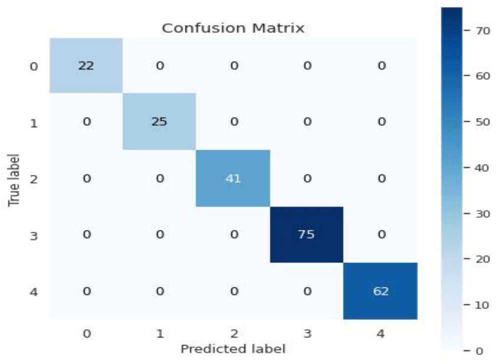
In the below image the first column is the raw image whereas the second and third columns are the ROI_1 and ROI_2 of that image respectively.



Dataset Used: Ck
Emotions: Happy, Surprise, Angry, Sadness, Fear
Input: Single image
Output: Emotion in that image
Performance: train-test split of 0.7
Best Train Accuracy = 100%
Best Test Accuracy = 100%







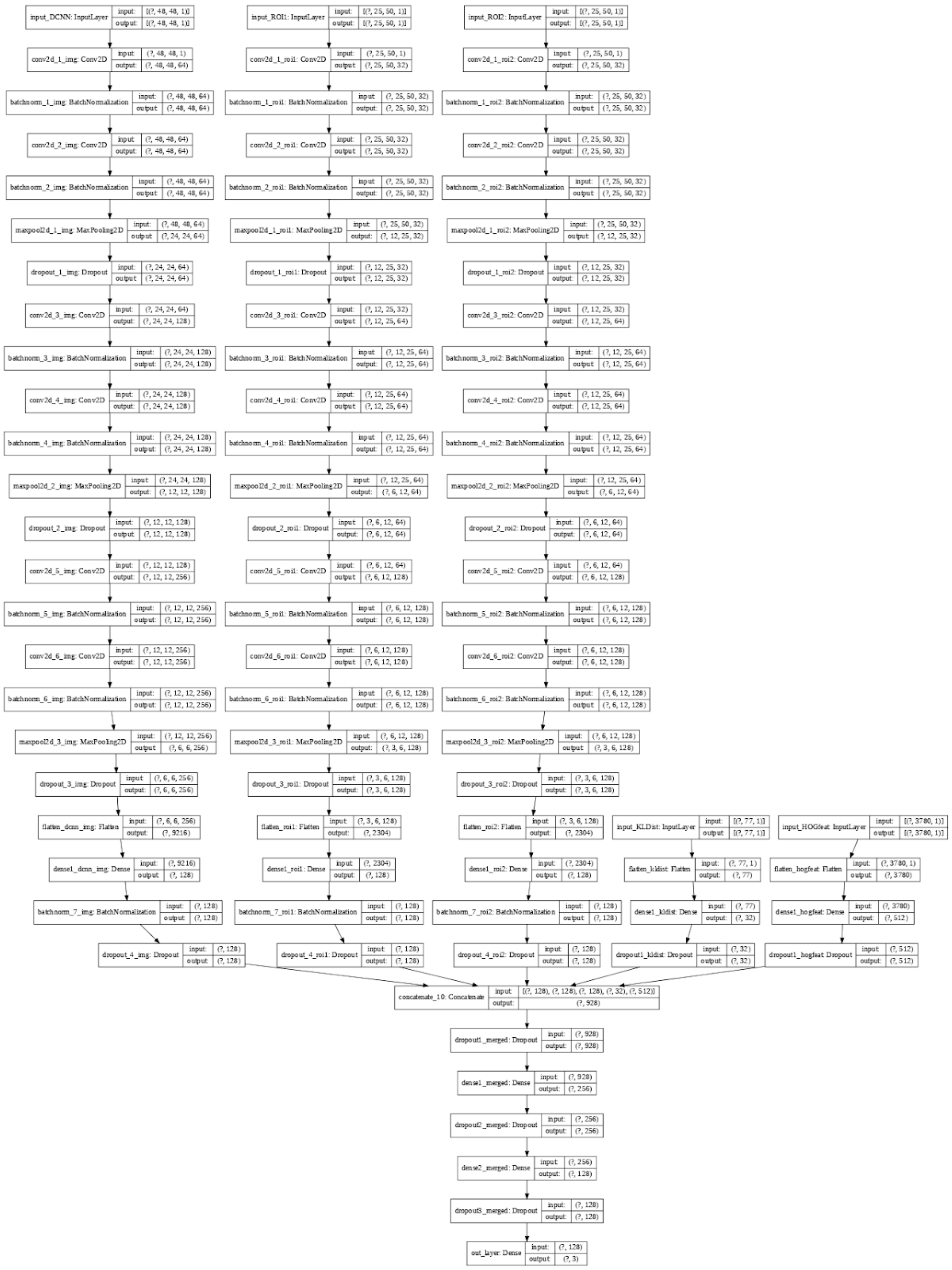
Pros:

- Takes other important features of the image.
- Works really good with CK dataset

Cons:

- Performance is not that good on FER dataset
- Takes more time to train and test

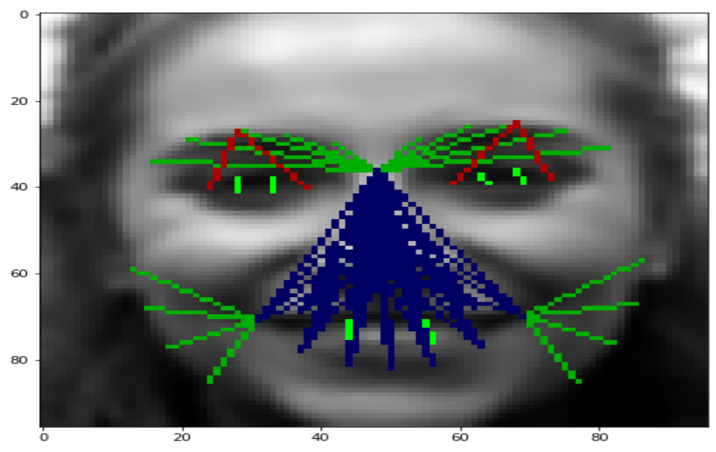
7. Raw Image + ROI_1 + ROI_2 + KL_DIST + HOG_FEAT


input_DCNN: InputLayer input: [(?, 48, 48, 1)] output: [(?, 48, 48, 1)]
conv2d_1_img: Conv2D input: (?, 48, 48, 1) output: (?, 48, 48, 64)
batchnorm_1_img: BatchNormalization input: (?, 48, 48, 64) output: (?, 48, 48, 64)
conv2d_2_img: Conv2D input: (?, 48, 48, 64) output: (?, 48, 48, 64)
batchnorm_2_img: BatchNormalization input: (?, 48, 48, 64) output: (?, 48, 48, 64)
maxpool2d_1_img: MaxPooling2D input: (?, 48, 48, 64) output: (?, 24, 24, 64)
dropout_1_img: Dropout input: (?, 24, 24, 64) output: (?, 24, 24, 64)
conv2d_3_img: Conv2D input: (?, 24, 24, 64) output: (?, 24, 24, 128)
batchnorm_3_img: BatchNormalization input: (?, 24, 24, 128) output: (?, 24, 24, 128)
conv2d_4_img: Conv2D input: (?, 24, 24, 128) output: (?, 24, 24, 128)
batchnorm_4_img: BatchNormalization input: (?, 24, 24, 128) output: (?, 24, 24, 128)
maxpool2d_2_img: MaxPooling2D input: (?, 24, 24, 128) output: (?, 12, 12, 128)
dropout_2_img: Dropout input: (?, 12, 12, 128) output: (?, 12, 12, 128)
conv2d_5_img: Conv2D input: (?, 12, 12, 128) output: (?, 12, 12, 256)
batchnorm_5_img: BatchNormalization input: (?, 12, 12, 256) output: (?, 12, 12, 256)
conv2d_6_img: Conv2D input: (?, 12, 12, 256) output: (?, 12, 12, 256)
batchnorm_6_img: BatchNormalization input: (?, 12, 12, 256) output: (?, 12, 12, 256)
maxpool2d_3_img: MaxPooling2D input: (?, 12, 12, 256) output: (?, 6, 6, 256)
dropout_3_img: Dropout input: (?, 6, 6, 256) output: (?, 6, 6, 256)
flatten_dcnn_img: Flatten input: (?, 6, 6, 256) output: (?, 9216)
dense1_dcnn_img: Dense input: (?, 9216) output: (?, 128)
batchnorm_7_img: BatchNormalization input: (?, 128) output: (?, 128)
dropout_4_img: Dropout input: (?, 128) output: (?, 128)
input_ROI1: InputLayer input: [(?, 25, 50, 1)] output: [(?, 25, 50, 1)]
conv2d_1_roi1: Conv2D input: (?, 25, 50, 1) output: (?, 25, 50, 32)
batchnorm_1_roi1: BatchNormalization input: (?, 25, 50, 32) output: (?, 25, 50, 32)
conv2d_2_roi1: Conv2D input: (?, 25, 50, 32) output: (?, 25, 50, 32)
batchnorm_2_roi1: BatchNormalization input: (?, 25, 50, 32) output: (?, 25, 50, 32)
maxpool2d_1_roi1: MaxPooling2D input: (?, 25, 50, 32) output: (?, 12, 25, 32)
dropout_1_roi1: Dropout input: (?, 12, 25, 32) output: (?, 12, 25, 32)
conv2d_3_roi1: Conv2D input: (?, 12, 25, 32) output: (?, 12, 25, 64)
batchnorm_3_roi1: BatchNormalization input: (?, 12, 25, 64) output: (?, 12, 25, 64)
conv2d_4_roi1: Conv2D input: (?, 12, 25, 64) output: (?, 12, 25, 64)
batchnorm_4_roi1: BatchNormalization input: (?, 12, 25, 64) output: (?, 12, 25, 64)
maxpool2d_2_roi1: MaxPooling2D input: (?, 12, 25, 64) output: (?, 6, 12, 64)
dropout_2_roi1: Dropout input: (?, 6, 12, 64) output: (?, 6, 12, 64)
conv2d_5_roi1: Conv2D input: (?, 6, 12, 64) output: (?, 6, 12, 128)
batchnorm_5_roi1: BatchNormalization input: (?, 6, 12, 128) output: (?, 6, 12, 128)
conv2d_6_roi1: Conv2D input: (?, 6, 12, 128) output: (?, 6, 12, 128)
batchnorm_6_roi1: BatchNormalization input: (?, 6, 12, 128) output: (?, 6, 12, 128)
maxpool2d_3_roi1: MaxPooling2D input: (?, 6, 12, 128) output: (?, 3, 6, 128)
dropout_3_roi1: Dropout input: (?, 3, 6, 128) output: (?, 3, 6, 128)
flatten_roi1: Flatten input: (?, 3, 6, 128) output: (?, 2304)
dense1_roi1: Dense input: (?, 2304) output: (?, 128)
batchnorm_7_roi1: BatchNormalization input: (?, 128) output: (?, 128)
dropout_4_roi1: Dropout input: (?, 128) output: (?, 128)
input_ROI2: InputLayer input: [(?, 25, 50, 1)] output: [(?, 25, 50, 1)]
conv2d_1_roi2: Conv2D input: (?, 25, 50, 1) output: (?, 25, 50, 32)
batchnorm_1_roi2: BatchNormalization input: (?, 25, 50, 32) output: (?, 25, 50, 32)
conv2d_2_roi2: Conv2D input: (?, 25, 50, 32) output: (?, 25, 50, 32)
batchnorm_2_roi2: BatchNormalization input: (?, 25, 50, 32) output: (?, 25, 50, 32)
maxpool2d_1_roi2: MaxPooling2D input: (?, 25, 50, 32) output: (?, 12, 25, 32)
dropout_1_roi2: Dropout input: (?, 12, 25, 32) output: (?, 12, 25, 32)
conv2d_3_roi2: Conv2D input: (?, 12, 25, 32) output: (?, 12, 25, 64)
batchnorm_3_roi2: BatchNormalization input: (?, 12, 25, 64) output: (?, 12, 25, 64)
conv2d_4_roi2: Conv2D input: (?, 12, 25, 64) output: (?, 12, 25, 64)
batchnorm_4_roi2: BatchNormalization input: (?, 12, 25, 64) output: (?, 12, 25, 64)
maxpool2d_2_roi2: MaxPooling2D input: (?, 12, 25, 64) output: (?, 6, 12, 64)
dropout_2_roi2: Dropout input: (?, 6, 12, 64) output: (?, 6, 12, 64)
conv2d_5_roi2: Conv2D input: (?, 6, 12, 64) output: (?, 6, 12, 128)
batchnorm_5_roi2: BatchNormalization input: (?, 6, 12, 128) output: (?, 6, 12, 128)
conv2d_6_roi2: Conv2D input: (?, 6, 12, 128) output: (?, 6, 12, 128)
batchnorm_6_roi2: BatchNormalization input: (?, 6, 12, 128) output: (?, 6, 12, 128)
maxpool2d_3_roi2: MaxPooling2D input: (?, 6, 12, 128) output: (?, 3, 6, 128)
dropout_3_roi2: Dropout input: (?, 3, 6, 128) output: (?, 3, 6, 128)
flatten_roi2: Flatten input: (?, 3, 6, 128) output: (?, 2304)
dense1_roi2: Dense input: (?, 2304) output: (?, 128)
batchnorm_7_roi2: BatchNormalization input: (?, 128) output: (?, 128)
dropout_4_roi2: Dropout input: (?, 128) output: (?, 128)
input_KLDist: InputLayer input: [(?, 77, 1)] output: [(?, 77, 1)]
flatten_kldist: Flatten input: (?, 77, 1) output: (?, 77)
dense1_kldist: Dense input: (?, 77) output: (?, 32)
dropout1_kldist: Dropout input: (?, 32) output: (?, 32)
input_HOGfeat: InputLayer input: [(?, 3780, 1)] output: [(?, 3780, 1)]
flatten_hogfeat: Flatten input: (?, 3780, 1) output: (?, 3780)
dense1_hogfeat: Dense input: (?, 3780) output: (?, 512)
dropout1_hogfeat: Dropout input: (?, 512) output: (?, 512)
concatenate_10: Concatenate input: [(?, 128), (?, 128), (?, 128), (?, 32), (?, 512)] output: (?, 928)
dropout1_merged: Dropout input: (?, 928) output: (?, 928)
dense1_merged: Dense input: (?, 928) output: (?, 256)
dropout2_merged: Dropout input: (?, 256) output: (?, 256)
dense2_merged: Dense input: (?, 256) output: (?, 128)
dropout3_merged: Dropout input: (?, 128) output: (?, 128)
out_layer: Dense input: (?, 128) output: (?, 3)

This is a **multi-input model** with 5 inputs,

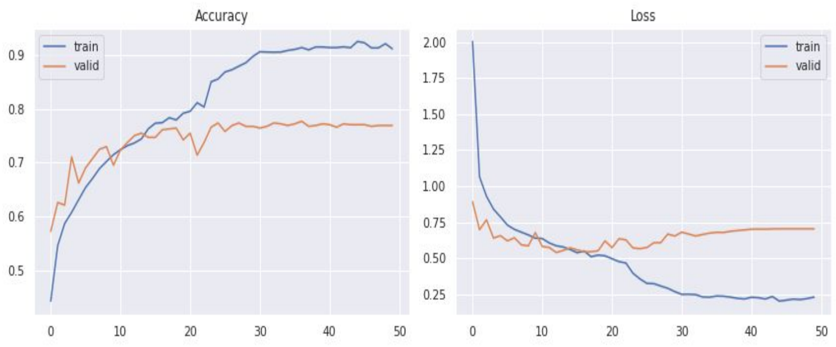
1. Raw image, input_shape = 48x48x1
2. Region of interest 1 (ROI_1), this contains the region of eyebrows and eyes, input_shape = 25x50x1
3. Region of interest 2 (ROI_2), this contains region of lips, input_shape = 25x50x1
4. Key Landmarks distance (KL_DIST), this is the distance of important facial landmarks, input_shape = 77x1
5. HOG features (HOG_FEAT), this contains hog feature descriptors, input_shape = 3780x1

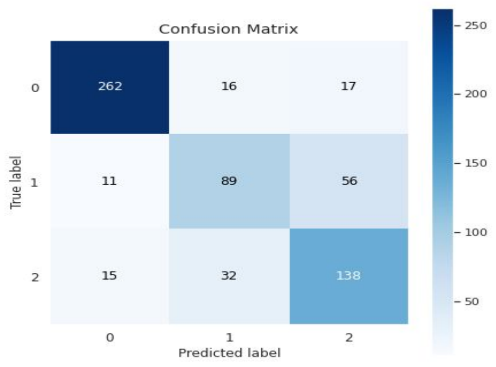
ROI_1 and ROI_2 are the same as before.



These are 77 important euclidean distances from a face. These contain distance between upper and lower lip, the distance between eyebrow and eyes, etc.

Dataset: FER-13 Cleaned
Emotions: Happy, Sadness, Neutral
Input: Single image
Output: Emotion in that image
Performance: train-test split of 0.86
Best Train Accuracy ~ 92%
Best Test Accuracy ~ 77%





	0	1	2
0	262	16	17
1	11	89	56
2	15	32	138

Pros:

- Calculate distances between import landmarks
- Works well on CK dataset

Cons:

- These models work well on video-based datasets not image-based because of the change in landmarks shares context with previous and future frames.
- Overfits in case of FER-13

Apart from these I also tried many state-of-the-art architectures like **ResNet**, **DenseNet**, **GoogLeNet**, **XceptionNet**. You can find them here.