NP Easy: Facial Expression Recognition

Shahmir Ali, Anmol Joshi, Kartik Saravana Arvind
CSC420

1. Abstract

The program will determine the person's facial expressions in real time by assigning labels on frames captured periodically. We use a pre-trained Haar-cascade model to detect a face and use our classifier model to classify an emotion label on the detected face. We intend to use videos where the people are expressive, like YouTube vloggers or Twitch streamers.

We used two datasets: Kaggle FER2013 dataset consisting of 37k images and CK+ dataset consisting of 981 images with the emotions "angry", "disgust", "fear", "happy", "neutral", "sad", "surprise". Some emotions tended to provide a greater testing accuracy than others. This is largely due to the imbalance of data in the Kaggle dataset and perceived ambiguities in the facial expressions.

For each convolutional block, we used a convolution layer, batch normalization, Relu activation, and max pooling. We will also be using ReduceLROnPlateau to slow down our learning rate when approaching an extrema. The impact of not using ReduceLROnPlateau is that the model overfits and loss is unstable.

In order to artificially expand the size of the dataset, we experimented with some image augmentations in order to find a set of augmentations that optimizes accuracy. We found that adding too many augmentations, and adding complicated augmentations severely lowered the testing accuracy due to overfitting. We found that an augmentation of a horizontal flip with vertical and horizontal shifts produced the best accuracy (60.9%).

For the SVM model, we wanted to use the CK dataset since SVM works well with small datasets. However, our SVM model performed very poorly (~11%) accuracy. We researched that the problem was that we needed to tune our hyperparameters which were difficult to tune. So we experimented by using our original much bigger kaggle dataset which seemed to produce better results (up to ~45% classification accuracy).

We used a pretrained model called 'shape_predictor.dat' from the library dlib to detect landmarks in important places. We tested various combinations using features together such as

(HOG + landmarks) to see if our accuracy would increase, and these two worked well. Using FL and HOG together yielded the best accuracy of 45.7%.

2. Introduction & Literature Review

2.1. Introduction to the problem

The problem we will be tackling is that of identifying emotional states by analyzing facial expressions in an image. In particular, we will be using a video of a person as input. The program will determine the person's facial expressions in real time by assigning the labels "Angry", "Disgust", "Fear", "Happy", "Neutral", "Sad" or "Surprise" on frames captured periodically.

In order to predict emotions in real-time, we select a frame periodically from a video and process the frame in two steps. First, we use a pre-trained Haar-cascade model to detect a face in the frame and extract an image of the detected face. Next, we preprocess this image and use our classifier model to classify an emotion label on the detected face. Combined with the Haar-cascade model, our model is fast enough to provide predictions in real-time to video frames.

We intend to use videos where the people are expressive, like YouTube vloggers or Twitch streamers. We believe that these videos would be good examples of test cases that can be presented to an audience. Currently, the application can process video input from a webcam. It is also capable of predicting emotions in live Twitch/YouTube streamers if a method of capturing the video from an internet stream is implemented.
We were severely limited by the amount of data available to use to train our model. The CNN model was trained using the Kaggle FER2013 dataset. Although it consists of ~29000 images, some emotions have far more images compared to other emotions. This disparity in distribution of emotions in the dataset, and the inherent similarity of facial expressions for different emotions, leads to a lower accuracy for some emotions like "disgust".

We were able to find only one other dataset that is compatible with our labels: the CK+ dataset which consists of 981 images. We attempted to use this dataset for the SVM model, since the SVM model performs better with a smaller dataset. For other datasets, we were unable to convert their labels into labels compatible with our classifier model.

After experimenting with different models, we noticed that the CNN model had a high testing accuracy, especially with emotions "Happy", "Neutral", "Anger" and "Surprise". In a general context, the inability for the model to correctly predict the other emotions like "Disgust" and

“Fear” would be bigger drawbacks, but for streamers, we can make a claim that they display emotions “Happy”, “Anger”, “Surprise” and “Neutral” disproportionately more than other emotions. For this reason, our model would perform better with streamers’ videos than what the test accuracy would suggest.

2.2. Datasets

In all the research we have done for CNN and SVMs, the two datasets that were the most popular were:

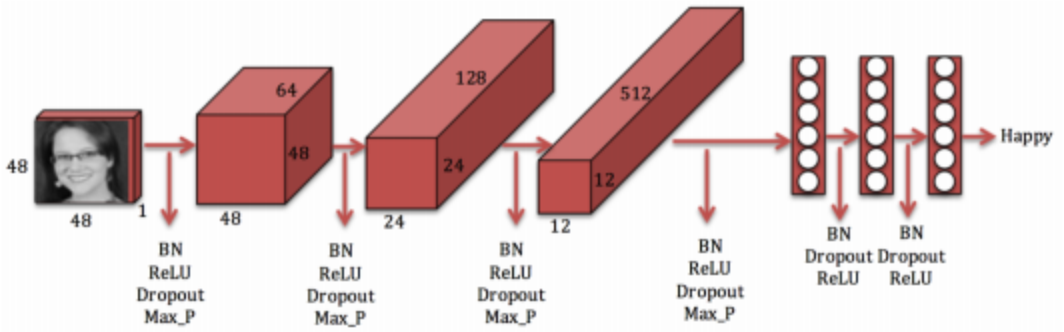
Kaggle FER2013 dataset: consists of 37k images (29k for training, 4k for validation and 4k for testing) with the emotions angry, disgust, fear, happy, neutral, sad, surprise [7].

CK+ dataset: consists of 981 images with the same emotion set as FER2013 [8].

We will primarily be using these two datasets.

2.3. CNN

Convolutional neural networks are one of the best ways to train for a model that recognizes facial expressions. CNNs consist of convolution layers, pooling layers and fully connected layers. There are two ways to go about building the network: shallow learning, or deep learning. Shallow learning consists of creating two convolution layers (one with 32 filters 3x3 window, and other 64 filters 3x3 windows), one fully connected layer with 512 neurons [1]. A convolution layer should consist of batch normalization, an activation layer (ReLu is a good choice), max pooling and a dropout. The Kaggle FER2013 dataset was used in the stanford project [1], achieving ~55% validation accuracy using the shallow learning technique [1]. To go deeper, two more convolution layers were added (512 filters each) along with an extra fully connected layer with softmax activation. This architecture resulted in a higher validation accuracy ~62.5% [1].

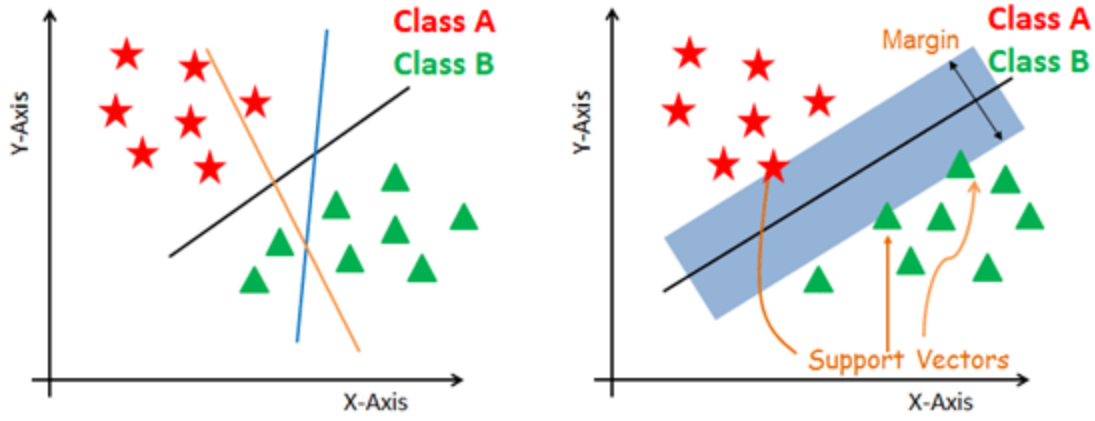


CNN architecture used in stanford report for FER [1].

We used the same dataset and closely followed this paper in the process of constructing our neural network, and tried to match the accuracies obtained in this experiment.

2.4. SVM

SVM is a supervised learning technique for Facial expression analysis that has high generalization performance [2]. The SVM classifier uses various kernels like linear, polynomial, and rbf (Radial Basis Function), which take low dimensional input and transform them to a higher dimension [5]. This is known as the 'kernel trick'. Different kernels are used for different problems, however for facial expression analysis it is popular to use rbf, due to its ability to map input to infinite dimensions and outperform the other kernels [5][6]. We will be using this in our SVM implementation. It is known to work well for datasets that are of moderate size [2]. We use this fact and attempt to try a new dataset called the CK+ dataset which is a smaller subset of the bigger CK dataset which when used with LBP (Local Binary Pattern) produced a test accuracy of 87.6% [3].



SVM generalising data points for each class [5].

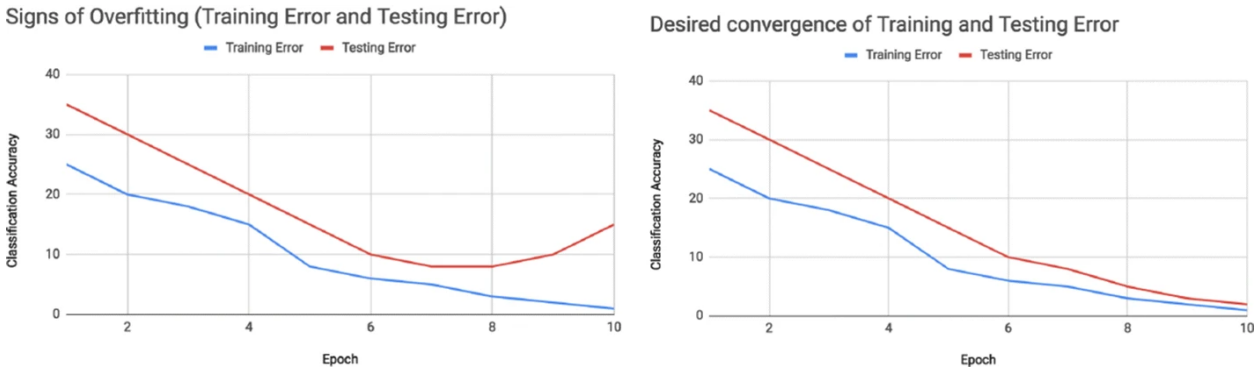
LBP is an efficient descriptor that extracts rich features. Each pixel analyses its neighbours to produce a binary number that represents texture [4]. It is good at detecting gray-scaled changes and illumination variations in images which have been proven to work well with facial features [4]. Another advantage of LBP is that the computational cost is low, which makes it ideal for our goal of analysing video streams like from twitch.

HOG is also a good descriptor to use for facial expression recognition. HOG uses bins that store occurrences of gradients in specific directions, this is useful for facial expressions, because it can be used to differentiate between the various emotions. The gradients around the facial muscles, eyes, mouth can be used with SVM to generalise facial expressions.

We explore various descriptors, including the ones mentioned above to use with an SVM model to see how it compares with our deep learning CNN.

2.5. Augmentation

Preventing the overfitting of training data remains a great challenge in deep learning [16]. In order to improve generalizability, the training error must decrease along with the validation error. When the training error and validation error begin to diverge, we know that there is overfitting [16].



[reference]

The desired outcome is that of the training error and validation error *converging*, as shown in Fig 1b. Augmenting the dataset in use can provide ways of achieving this. Augmentation works under the assumption that more data can be extracted by simply warping or oversampling the existing data. This way, we can 'increase' the size of the dataset as we add more and more augmentations. There are many ways to augment data, however the augmentations that one decides to include depends on the nature of the dataset; transforming an image in the dataset must not change its label. For example, if we horizontally flip an image of the lowercase letter 'd' (typed or handwritten), the resulting image will display a lowercase 'b', which means that a horizontal flip will not generally preserve the label for this particular dataset [16].

In our project, we experimented with some geometric transformations and some colour transformations. The geometric transformations we tried involved rotations, flips, and shifts, and colour transformations involved Gaussian blurring, pixel intensity thresholding, brightness

shifts, and noise injection. Most of the colour transformations were custom made preprocessing functions used as arguments in the `ImageDataGenerator` function.

3. Methodology, Results, & Experiments

3.1. Theoretical part

3.1.1 CNN

Layers

For each convolutional block, we used a convolution layer, batch normalization, Relu activation, and max pooling. The convolution layer convolves the input image with filters from a filter bank which allows our CNN to extract useful features in order to classify facial expressions. Batch normalization standardizes the batch which in turn accelerates the deep learning process [9]. To introduce non-linearity to our network, we used the activation function 'ReLu'. This function outputs the input if it's positive, and is known to produce better results. The reason it's better than sigmoid in the context of a CNN with many layers is because of the 'vanishing gradient' problem [10]. Another layer that introduces non-linearity (prevents overfitting) is the max pooling layer. This layer increases performance of the CNN by reducing image size. We use these layers in our CNN.

Learning rate

We want our model to be efficient so choosing a good learning rate to accomplish that is important. We don't want training to take too long, since we don't have powerful machines. We will also be using a keras function (ReduceLROnPlateau) to slow down our learning rate when approaching an extrema so our model performs its best and doesn't overstep/understep.

Back Propagation and Loss function

Our model adjusts weights every epoch using back propagation internally (built in Keras model). The loss function we use is categorical cross entropy, which is good for classification tasks such as facial expression analysis [11]. It is used to help classify if a face (data point) belongs to one of the 7 emotions .

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output}\\ \text{size}}} y_i \cdot \log \hat{y}_i$$

Categorical cross-entropy loss function [11].

3.1.2 SVM

SVM descriptors

In order for SVM to generalize facial expressions, we must feed it rich features. Features that we researched that would be computationally efficient were HOG, LBP and facial landmarks. We used a pretrained model to calculate 68 facial landmarks called “shape_predictor.dat” which belongs to the ‘dlib’ library.

3.2. Empirical work (Quality of Analyses)

3.2.1 CNN Empirical work

First of all, we split the Kaggle dataset 40% training, 20% validation, 40% testing. We couldn’t find any other dataset that had as big of a size. We couldn’t use the CK+ dataset because it was too small, and our research suggested we need large datasets to train a CNN well.

Using the Stanford paper as reference [1], we constructed our CNN using the keras library with all the layers we mentioned in part 3.1 (model.py). In addition to adding the 4 convolutional layers (consisting of BN, Relu, Max Pooling, dropout) we had two fully connected dense layers at the end to use as our features. Finally we have a softmax layer to transform them into probabilities to use for predicting what emotion the image contains.

```
# First CNN layer
# 64 filters 3x3, images are 48x48 and 1 channel because grayscale
model.add(Conv2D(64, (3, 3), padding='same', input_shape=(IMG_DIM, IMG_DIM, 1)))
model.add(BatchNormalization())
model.add(Activation('relu'))
# Divide height and width of conv block by 2
model.add(MaxPooling2D(pool_size=(2, 2)))
# can mess around with this hyperparameter (drops random data)
model.add(Dropout(dropout))
```

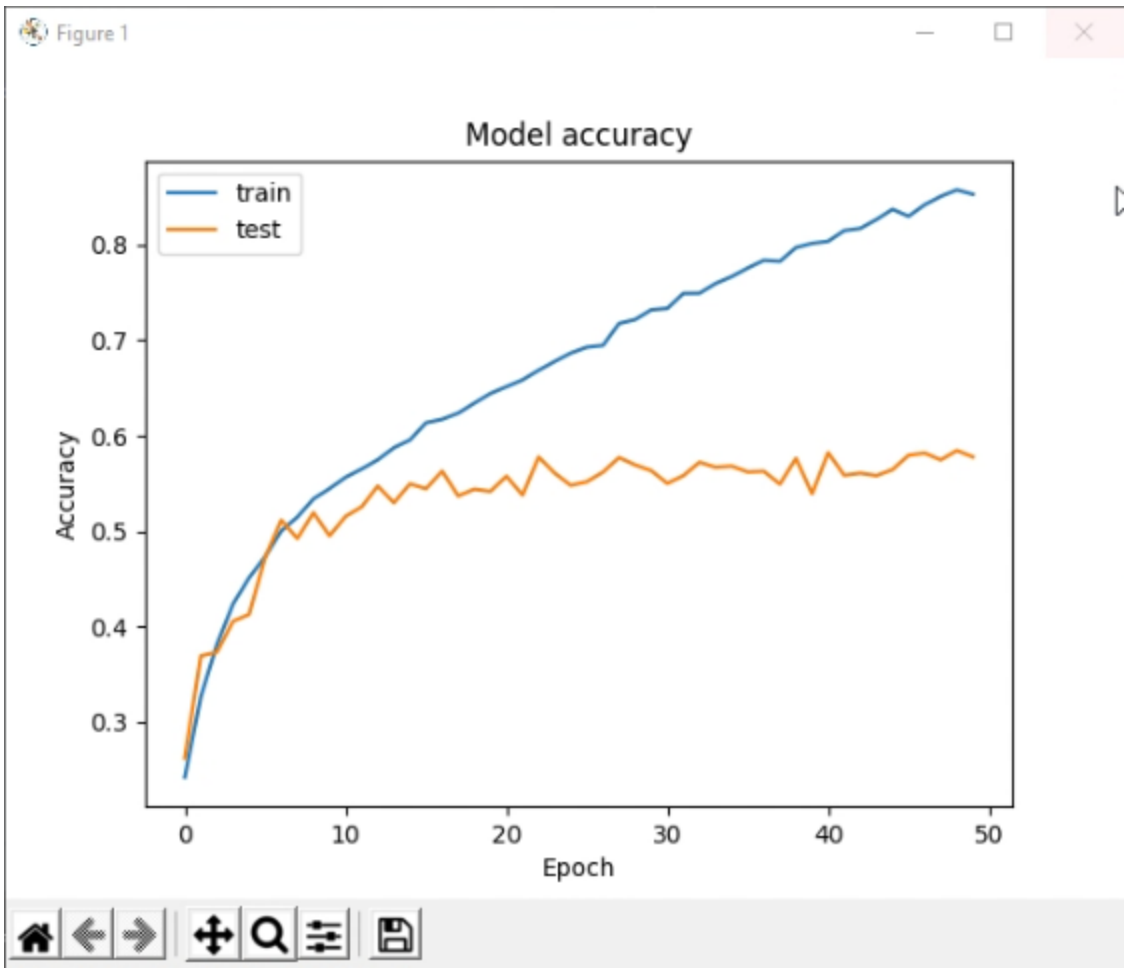
One convolutional layer (code from model.py)

In order to optimize our training to produce the best test accuracy, we tuned the hyper parameters by passing in non-augmented data. The first one we tuned was the learning rate. We mentioned that we wanted our model to train for an efficient amount of time such that it wouldn’t take extremely long and still produce decent results. We experimented with various rates and agreed upon one that produced the results we were looking for, lr = 0.0008.

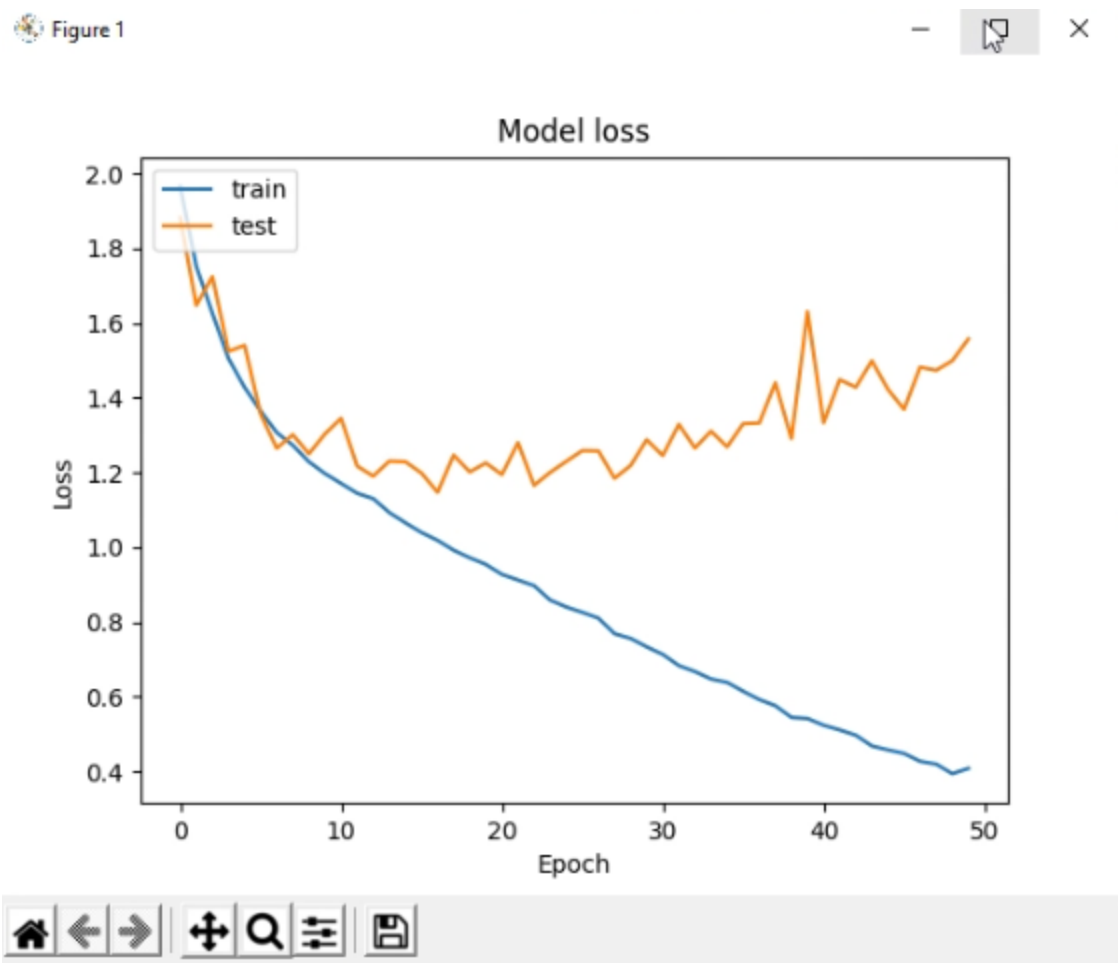
Additionally we used a keras function called ‘ReduceLROnPlateau’ which lowers the learning rate as we approach an extrema. It monitors the validation loss, and reduces the learning rate by a factor of 10 if there is no improvement after 2 epochs [12]. Below we observe the impact of not using ‘ReduceLROnPlateau’.

```
rlr_function = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=2, min_lr=0.00005, mode='auto')
```

ReduceLROnPlateau located in train.py



Model overfits when we don’t use ReduceLROnPlateau because we skip over (test = validation here)



Since val_loss ('test' in graph) is not monitored, our loss is unstable

Dropout is also an important factor in our model which randomly drops nodes to prevent overfitting. The intuition of dropout is to not make our CNN rely too much on one weight or feature. So next, we picked a good probability to use for our dropout. Our values ranged from 0.15 to 0.3, and we noticed around 0.27 worked the best (highest accuracy) when we tested our model with unseen data (no overfitting).

In order to artificially expand the size of the dataset, we experimented with some image augmentations in order to find a set of augmentations that optimizes accuracy. We found that adding too many augmentations, and adding complicated augmentations severely lowered the testing accuracy due to overfitting. In the end we decided to use a horizontal flip and vertical and horizontal shifts of 5 pixels each way. The shifts are small because the training images have dimensions of only 48 x 48 pixels, so large shifts may not preserve the labels.

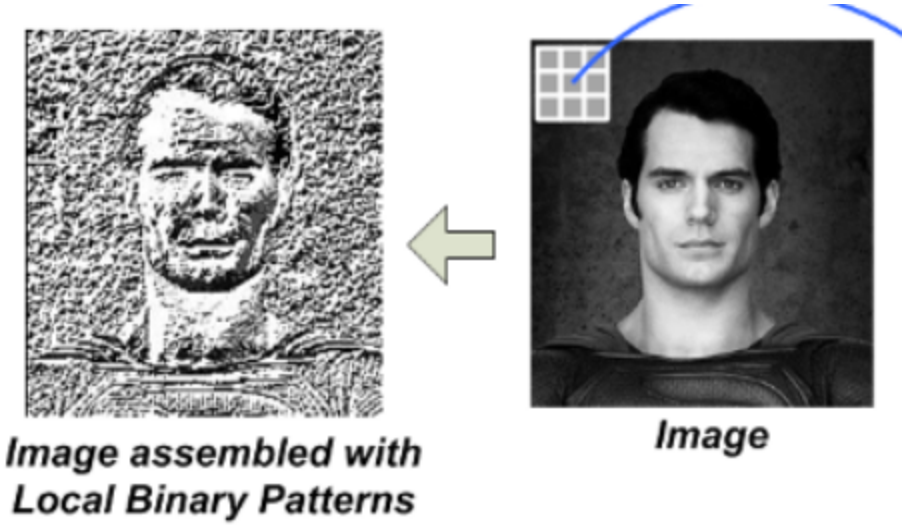
3.2.2 SVM Empirical work (all code for this is in model_svm.py and descriptors.py)

We wanted to use the CK dataset for SVM training since it was a small one and our research indicated that SVM works well with it. However, our SVM model (SVC from sklearn library) performed very poorly (~11%) accuracy, and we noticed it was labelling everything as the last label (surprise). We researched what the problem could be, and learned that we needed to tune our hyperparameters C (regularization parameter) and gamma [13]. This turned out to be a hard parameter to tune, as it resulted in very little changes in performance. So we experimented by using our original much bigger kaggle dataset which seemed to produce better results (up to ~45% classification accuracy).

We wanted to compare our CNN's performance against an SVM model, so we experimented on various kernels and features to use against it.

We experimented with two SVM kernels: Linear and rbf (the recommended choice from our research) [5][6]. By feeding our SVM model raw pixels and analysing the test accuracy, our rbf kernel outperformed the linear one as expected.

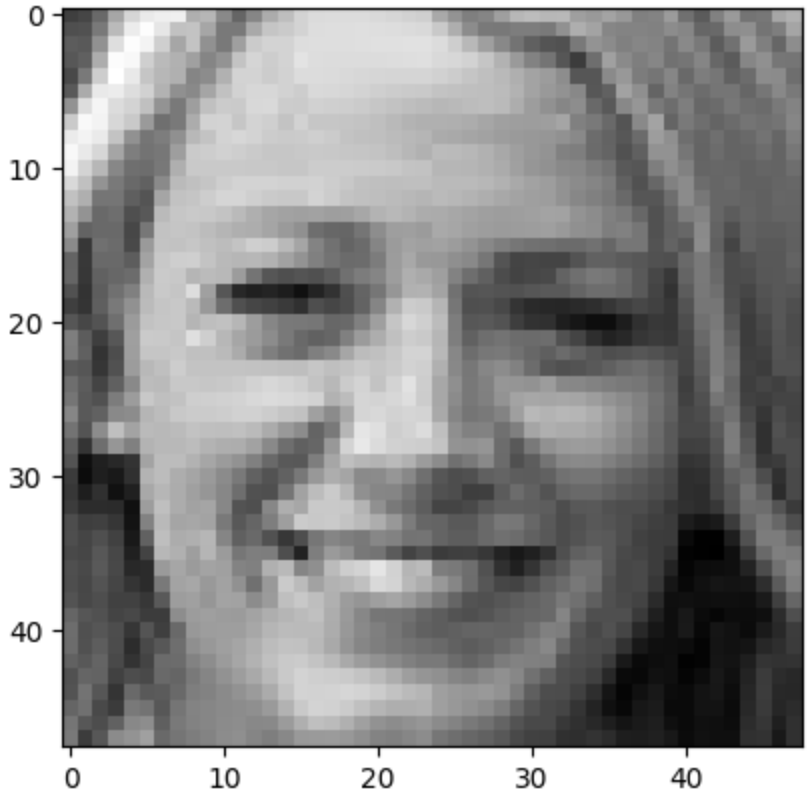
After picking 'rbf' as our primary kernel, we experimented with various descriptors to see what features allowed our model to perform best. Initially we expected LBP would outperform all the other features because of our research about it, however it turned out to perform the worst with our model in terms of test accuracy (all results in section 3.3.2).

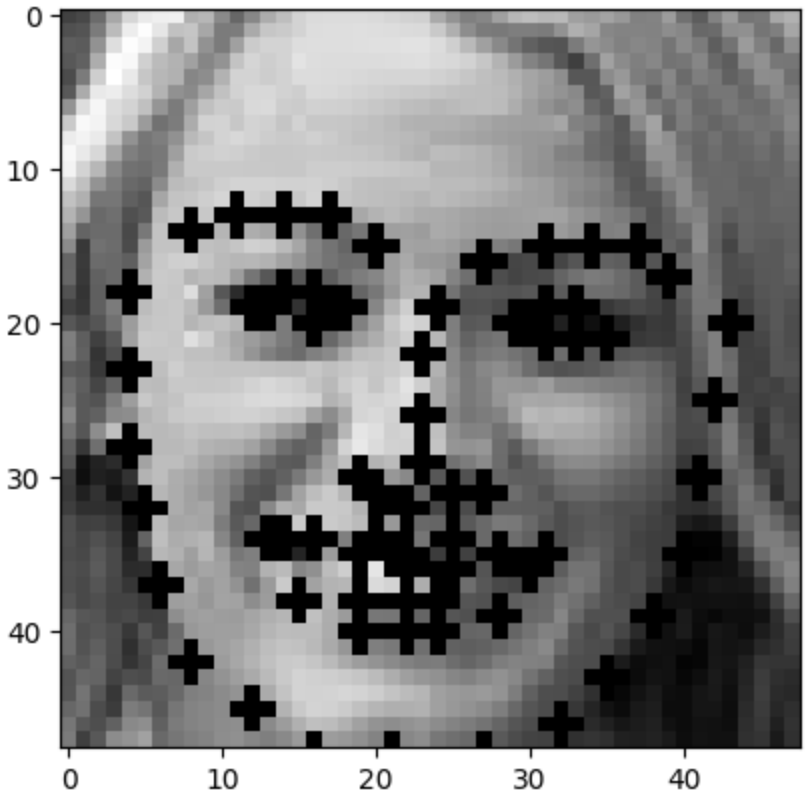


LBP feature visualization [14].

We experimented with HOG and facial landmarks next. HOG performed better than LBP and facial landmarks was the best (implementations can be found in descriptors.py). We believe hog

performed well because emotions can be detected well by analysing the HOG around facial features. For example, hog around a smile will be different than hog around a neutral face. Landmarks performed the best because we get the coordinates to where the eyebrows, eyes, nose and mouth are. Our SVM model is able to detect landmarks in important places such as the corner of the mouth (to see the position of it relative to the mouth center). We used a pretrained model to figure these out for us called ‘shape_predictor.dat’ from the library dlib.



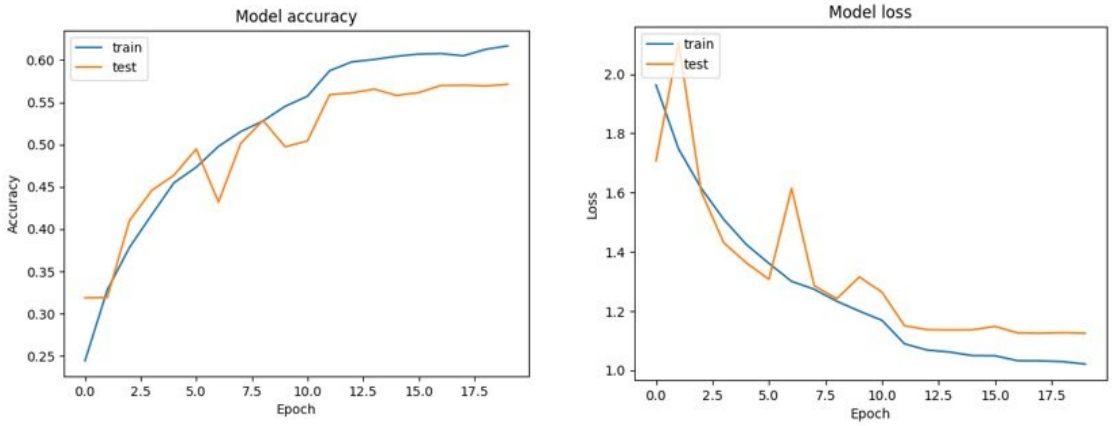


Facial landmarks visualized by running descriptor.py on a random img in our dataset.

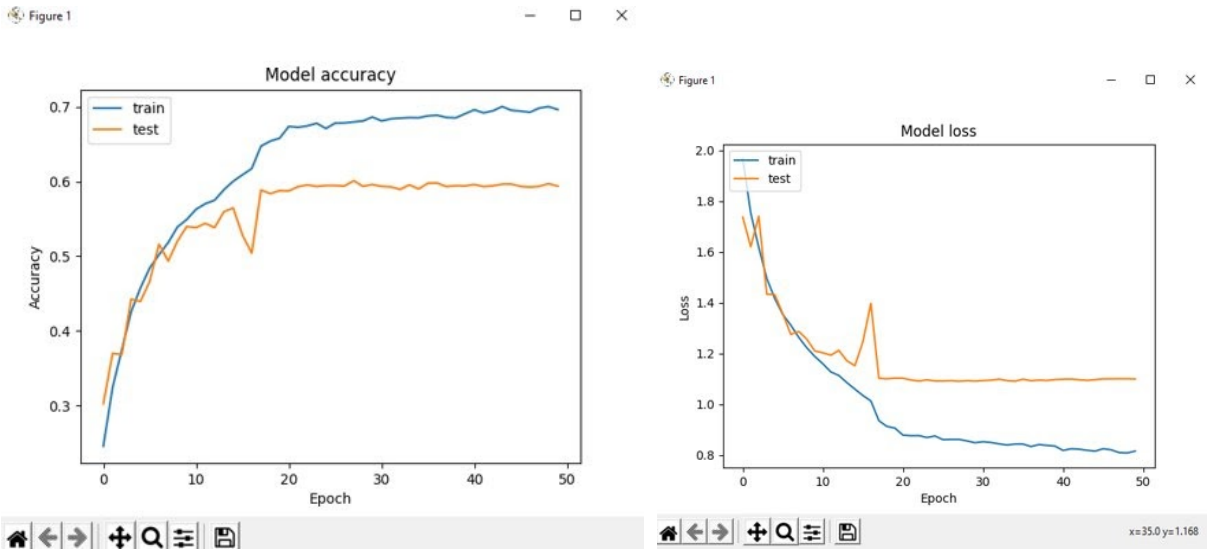
We tested various combinations using features together such as (HOG + landmarks) to see if our accuracy would increase, and these two worked well (results displayed in section 3.3.2).

3.3. Presentation of Results/Failure

3.3.1 CNN results (all code for this in model.py, train.py and predict.py)
We initially trained our model for 20 epochs and 50 epochs (no augmentations) and compared the results per emotion. Testing CNN models takes place in 'predict.py' after setting the boolean 'test' to true.



Model accuracy/loss for training/validation sets (20 epochs)(generated by 'train.py')
Final loss: 1.02 val_accuracy: 0.5713



Model accuracy/loss for training/validation sets (50 epochs)
Final loss: 0.81 val_accuracy: 0.5939

Emotion	Test accuracy % (20 epochs)	Test accuracy % (50 epochs)
Angry	*42.5*	*50.3*
Disgust	*26.4*	*39.3*
Fear	*26.2*	*34.2*
Happy	*81.7*	*82.4*
Neutral	*55.2*	*56.7*
Sad	*56.5*	*52.6*
Surprise	*72.5*	*74.3*
Overall	*57.9*	*60.2*

Testing CNN models takes place in 'predict.py' after setting the boolean 'test' to true.

Test accuracy for 'angry', 'disgust' and 'fear' has improved by a noticeable amount and overall accuracy has risen 2.3%.

By observing the graph for 20 epochs and 50 epochs, it seems that training for more than 50 epochs would result in the same val_accuracy (converges at 0.6), which means we must tune other parameters in order to improve test accuracy.

Next, we trained our model using augmentations. The below table conveys each set of augmentations used, and the accuracies achieved using them. Note that the numbers in the headings have the following conversions:

{0 → Angry, 1 → Disgust, 2 → Fear, 3 → Happy, 4 → Neutral, 5 → Sad, 6 → Surprise}

Augmentations	**Validation (%)**	**0 (%)**	**1 (%)**	**2 (%)**	**3 (%)**	**4 (%)**	**5 (%)**	**6 (%)**	**Overall (%)**
horizontal flip	58.5	45.5	38.3	31.2	83.3	56.4	54.7	74.8	59.7
horizontal flip, rotation range [-30°, 30°]	59.5	33.9	31.5	28.0	70.0	34.4	77.1	68.3	60.1
horizontal flip, width shift	60.5	51.3	39.5	39.6	83.7	52.3	53.9	74.5	60.9

range [-4, 4] pixels, height shift range [-4, 4] pixels									
horizontal flip, zca whitening	57.9	44.7	32.5	29.2	81.9	56.1	56.1	74.3	59.0
Horizontal flip, preprocessing function: threshold *(custom)*	58.0	40.8	33.3	36.8	79.0	54.9	50.1	74.3	57.3
Horizontal flip, preprocessing function: noise injection *(custom)*	56.5	39.3	34.1	37.0	80.4	52.8	49.7	70.2	57.0

The sets of augmentations experimented with, and their corresponding accuracies

We found that, out of the augmentations tested, a horizontal flip with vertical and horizontal shifts produced the best accuracy (60.9%, 0.7% increase). This may be because the shifts do not result in too much lost information, but provides a more diverse dataset.

Notice how some emotions tended to provide a greater testing accuracy than others. This is largely due to the imbalance of data in the Kaggle dataset. 'Disgust' had the lowest number of images to train with, so the model was less likely to predict disgust in the testing images. Meanwhile, since 'Happy' had the highest number of images to train with, we can see that 'Happy' yielded the highest accuracy in most of the trials.

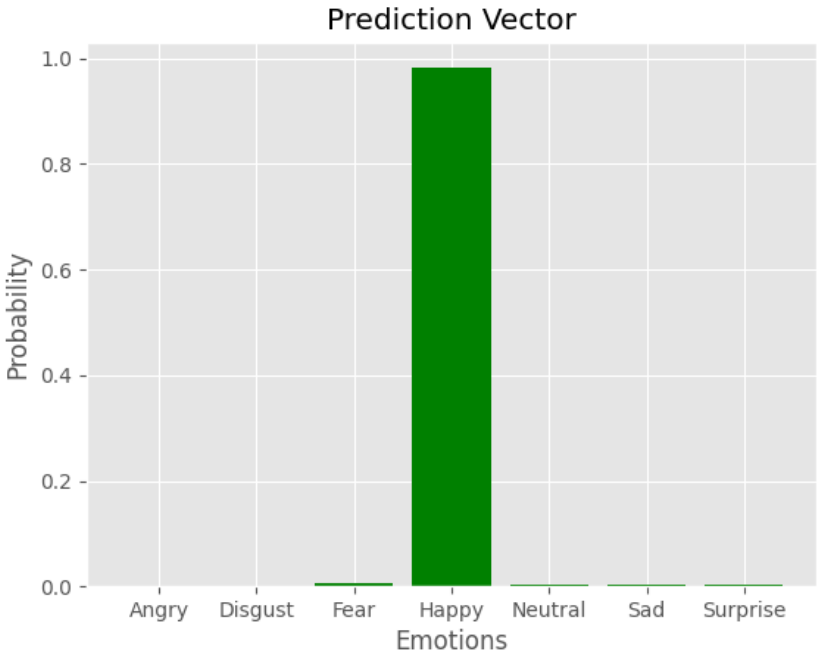
Another reason for the imbalance is perceived ambiguities in the facial expressions. A frown can be perceived as a sad expression, but also an angry, disgusted, or fearful expression. In contrast, a smile is most often perceived as a happy expression, so the model is less likely to perceive the overall image as any other emotion. While there are, of course, other distinguishing features for each emotion (e.g. furrowed brows often indicate anger), the mouth is one of the most important factors due to its size and variability in shape.

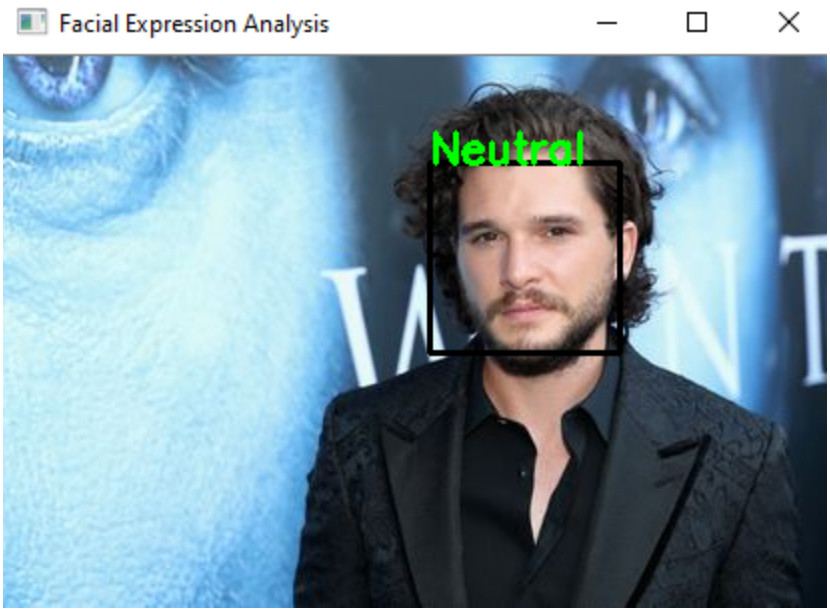
Our test results were close to the stanford project (60.9% vs ~62.5%) [1]

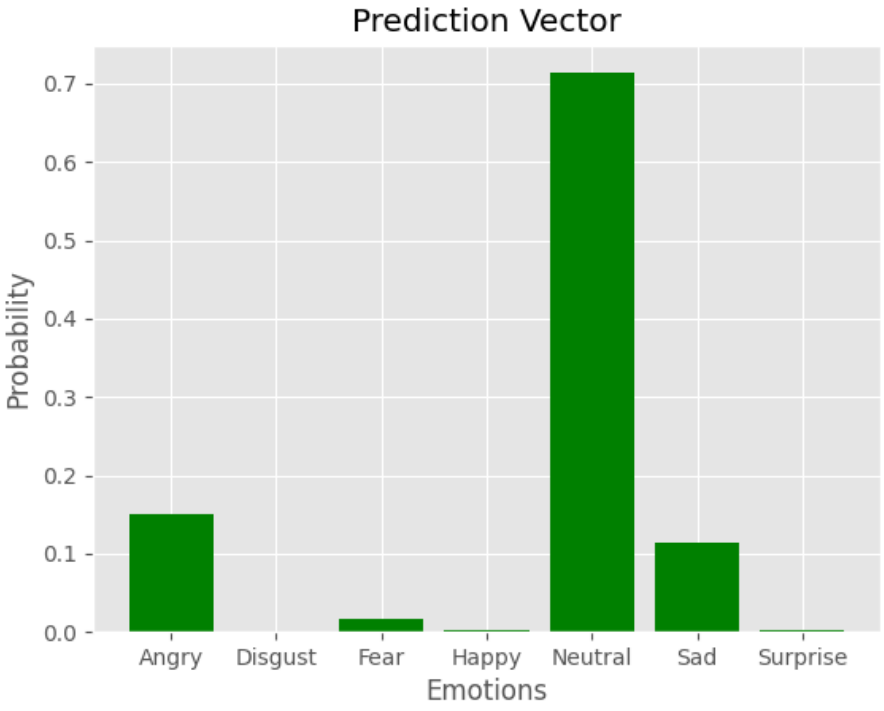
Our CNN model is designed to take in a 48x48 image, so we test images of various sizes as follows (steps on how a prediction is made in 'prediction.py'):

Single images:

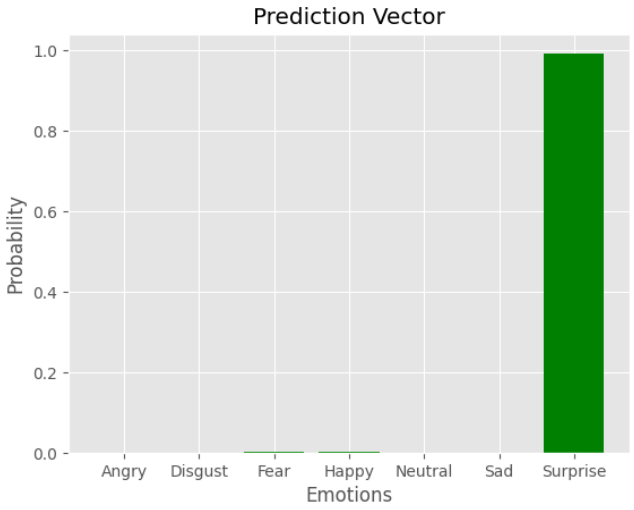
Pass an image (ndarray) through `display_expression(img, model, mode=0)`, along with a CNN model. Mode 0 means it's for a single image and will show a prediction vector along with the image's prediction. This function uses a pretrained model 'haarcascade' to detect a face and create a bounding box around it. The resulting image is then extracted and rescaled to 48x48 and sent through our model. The resulting prediction is displayed on the image.


Prediction Vector
1.0
0.8
0.6
0.4
0.2
0.0
Probability
Angry
Disgust
Fear
Happy
Neutral
Sad
Surprise
Emotions


Facial Expression Analysis
Neutral


Prediction Vector
Probability
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Angry
Disgust
Fear
Happy
Neutral
Sad
Surprise
Emotions


Facial Expressio...
Surprise
2:58 / 15:12



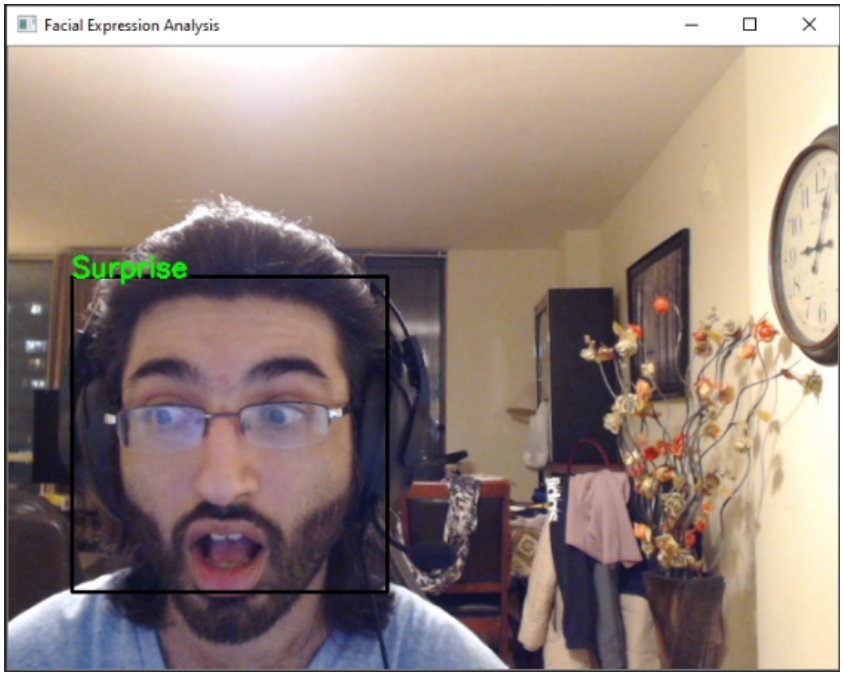
Single image prediction for 3 images in ‘./Test pics’

Webcam feed:

We call the function `detect_emotions_webcam(model),` which runs your webcam feed and displays it to you if a face is detected. It calls `display_expression(img, model, mode=1)` but this time in video mode, so it doesn’t constantly show a prediction vector. It predicts expression on each frame in real-time due to haar cascades fast frontal face detection, which would be ideal for a twitch streamer.


Facial Expression Analysis
Happy


Facial Expression Analysis
Angry



Frames from Shahmir's webcam feed. Detection for these four emotions seems the most accurate.

3.3.2 SVM results (all code for this in model_svm.py and descriptors.py)
First we load our data using `get_x_and_y(DIR_NAME, features)` since our data is going to strictly be features. Features could be either 'HOG', 'LBP', 'FL' or 'HOGFL' which is a concatenation of HOG and Facial Landmarks. Features are extracted using descriptors in descriptors.py. The model takes a numpy array of 1D arrays that represent feature vectors. Additionally it takes a list of correct labels for the respective input. The following test accuracies were obtained by evaluating our model with various features:

Features	Test accuracy (%)
None (raw pixels)	28.6
Local binary patterns (LBP)	25.4
Histogram of Gradients (HOG)	30.0
Facial Landmarks (FL)	41.7
FL + LBP	43.0

FL + HOG	45.7
FL + HOG + LBP	45.6

Testing SVM models takes place in 'model_svm.py' after setting the boolean 'testing' to true.

By observation, in our experiment FL had the biggest impact (+13.1%) since it uses a pretrained model to figure out facial feature locations and marks them. LBP performed worse than simply using raw pixels (-3.2%) HOG was better than LBP (+4.6%). Using FL and HOG together yielded the best accuracy of 45.7%

We were expecting a much higher accuracy with LBP and SVM because of our research, so we might have not tuned the hyper parameters correctly/well enough to achieve this result.

4. Conclusion

In conclusion, our CNN model outperforms our SVM one, with a difference of +15.2% (60.9% vs 45.7%). By observing our results, it seems like we did not tune our SVM model well enough as it was lower than what we expected when used with LBP. However, deep learning models are generally better than other classifiers in terms of performance, as deep learning models make their own intelligent decisions after forming an artificial neural network (higher capability than ML), while machine learning is a super set of deep learning, with less predictive power [15].

Another thing we noticed was that emotions like fear and disgust were harder to detect than happy, angry, sad, surprise, due to the fact that we had limited data present for them. It is also the case that in the dataset even as a human, it was sometimes hard to detect differences between disgust/angry or fear/surprised. So we'd imagine it's even harder for an artificial neural network to differentiate between the two. Generally, twitch streamers show the 4 main emotions (happy, angry, sad, surprise), so our test accuracy would be higher if we only analysed them for our problem.

We also noticed that it was faster to get a prediction using our CNN since we were passing in the image without using some sort of descriptor on it. When we pass an image to our SVM model, it is required to pass some sort of features. Our best performing feature (HOG + FL with 45.7% accuracy) first calculates HOG (1x900 feature vector), then calculates a 1x68 FL feature

vector and inputs the concatenation into the model. This requires a preprocessing step which may not be ideal for video streams, as it may cause delays when predicting emotions. Additionally, if we decide to make our CNN take in bigger images as input, our feature vectors will also get bigger resulting in more preprocessing time.

In the future we plan to create an API of our model that works with twitch. For example, there is an event that happens when a streamer has more viewers than usual. In this event, viewers who subscribe or donate to the streamer, gain additional rewards like extra perks. We had an idea to use our CNN model to trigger these events periodically (for eg something exciting happens and the streamer shows a surprise reaction) whenever a streamer was displaying a certain emotion (with the help of twitch API).

5. Author's Contributions

Anmol

- Searched for and provided Cohn-Kanade datasets
- Researched on dataset augmentation for deep learning
- Rigorously experimented with dataset augmentations to find most accurate model
- Wrote custom preprocessing functions to augment dataset with

Kartik

- Setup scripts for training model and using model on Google Colab
- Created server for saving different models to the cloud and selecting one of them to use to predict facial emotion in any uploaded images

Shahmir

- Trained CNN model with no major augmentations (train.py and model.py)
- Worked on CNN model prediction, along with prediction on webcam stream (predict.py)
- Worked on SVM training and prediction along with descriptor experimentation(model_svm.py and descriptors.py)

6. References

[1] Alizadeh, Shima, and Azar Fazel. *Convolutional Neural Networks for Facial Expression Recognition*, cs231n.stanford.edu/reports/2016/pdfs/005_Report.pdf.

[2] Michel, Philipp, and Rana El Kaliouby. 2005, *Facial Expression Recognition Using Support Vector Machines* , www.cs.cmu.edu/~pmichel/publications/Michel-FacExpRecSVMAbstract.pdf.

[3] Căleanu, Cătălin. 2013, *Face Expression Recognition: A Brief Overview of the Last Decade*, www.researchgate.net/publication/261498182_Face_expression_recognition_A_brief_overview_of_the_last_decade.

[4] Vasanth, P C, and K R Nataraj. 2015, *Facial Expression Recognition Using SVM Classifier*, www.researchgate.net/publication/282522466_Facial_Expression_Recognition_Using_SVM_Classifier.

[5] Navlani, Avinash. "(Tutorial) Support Vector Machines (SVM) in Scikit-Learn." *DataCamp Community*, 27 Dec. 2019, www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python.

[6] Michel, Philipp, and Rana El Kaliouby. 2003, *Real Time Facial Expression Recognition in Video Using Support Vector Machines*, www.cs.cmu.edu/~cga/behavior/FER-SVM-ICMIpaper.pdf.

[7] "Challenges in Representation Learning: Facial Expression Recognition Challenge." *Kaggle*, 2013, www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data?select=train.csv.

[8] Shawon, Ashadullah. "CKPLUS." *Kaggle*, 16 Oct. 2018, www.kaggle.com/shawon10/ckplus.

[9] Brownlee, Jason. "A Gentle Introduction to Batch Normalization for Deep Neural Networks." *Machine Learning Mastery*, 3 Dec. 2019, machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/.

[10 Brownlee, Jason. "A Gentle Introduction to the Rectified Linear Unit (ReLU)." *Machine Learning Mastery*, 20 Aug. 2020, machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

[11] “Categorical Crossentropy Loss Function: Peltarion Platform.” *Peltarion*, peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy.

[12 “Tf.keras.callbacks.ReduceLROnPlateau ： TensorFlow Core v2.4.0.” *TensorFlow*, www.tensorflow.org/api_docs/python/tf/keras/callbacks/ReduceLROnPlateau.

[13] “Output of Scikit SVM in Multiclass Classification Always Gives Same Label.” *Cross Validated*, 24 Nov. 2014, stats.stackexchange.com/questions/125353/output-of-scikit-svm-in-multiclass-classification-always-gives-same-label.

[14] Kyrkou, Christos. “Object Detection Using Local Binary Patterns.” *Medium*, Medium, 2 Nov. 2017, ckyrkou.medium.com/object-detection-using-local-binary-patterns-50b165658368.

[15] Grossfeld, Brett, and Associate content marketing manager. “Deep Learning vs Machine Learning.” *Zendesk*, 23 Jan. 2020, www.zendesk.com/blog/machine-learning-and-deep-learning/.

[16] Shorten, Connor, and Taghi M. Khoshgoftaar. Springer Open, 2019, *A Survey on Image Data Augmentation for Deep Learning*, journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0.

[17] Brownlee, Jason. “Image Augmentation for Deep Learning With Keras.” *Machine Learning Mastery*, 12 Sept. 2019, machinelearningmastery.com/image-augmentation-deep-learning-keras/.

[18] Brownlee, Jason. “How to Configure Image Data Augmentation in Keras.” *Machine Learning Mastery*, 5 July 2019, machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/.