Facial Expression Recognition

Henry Kam, Thomas Kong

1. Introduction

Facial expression recognition (FER) is an extremely important part of human communication, and thus has become a significant field of research in machine learning. The ability to recognize and interpret facial expressions can allow machines to better interpret human emotions, intentions, and social cues, ultimately allowing machines to have more personalized interactions. This research can be applied to many facets of technology, from virtual reality to marketing.

We wanted to take part in improving human-computer interactions, so we decided to apply our knowledge from our class to ultimately attempt to provide rudimentary facial expression and sentiment analysis.

2. Data Preparations

Our dataset, FER-2013, was sourced from Kaggle, consisting of 48x48 greyscale images of faces with different facial expressions. The images were already preprocessed so that each face is roughly in the center of the screen, and occupies similar amounts of space in each image.

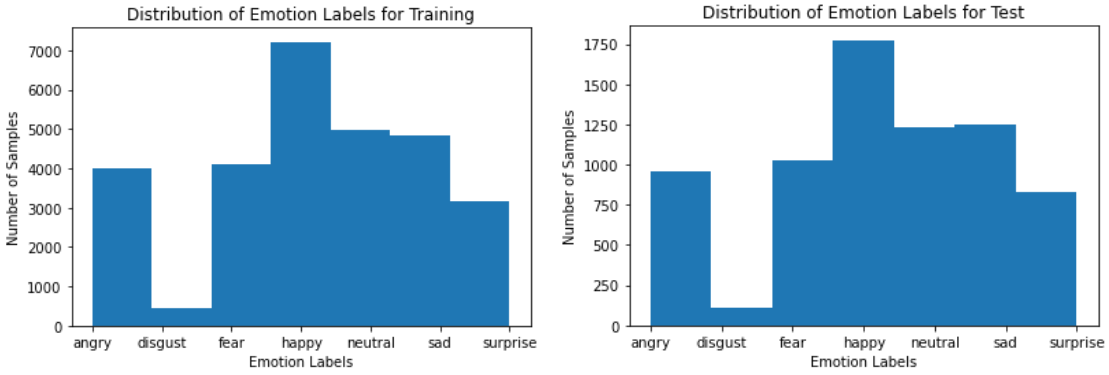
Our objective was to categorize each image into different types of emotions. The images were given to us in seven different categories: angry, disgust, fear, happy, sad, surprise, and neutral. The training set had over 28,000 examples and the test set had over 3,500 examples. We decided to frame this as a multiclass classification problem given the nature of the input.

In regards to wrangling and cleaning the dataset, not much was necessary, as the pictures were uniform in size, and data was given to us in a very clear and clean manner. However, we did normalize the pixel values and convert them to arrays. With certain models, some information such as pixel position was lost as it was all converted into a 1-dimensional array.

We decided to use dummy variable encoding to encode the different emotions. We simply decided to use alphabetical order out of convenience.

We noticed that disgust had a lot less data than the other labels, but we decided that it was fine if our classification was one vs rest.

In regards to splitting the data for test and training, the data set already did it for us. The dataset was split to around 90% training data, and 10% test data. We decided not to change it because it was already given to us. Fortunately, the distribution for the number of images for each label was roughly similar, so we did not need to do cross validation.



3. Models

Given that it was a multiclass classification problem, we decided that we wanted to use more complex models to experiment with. Thus, we decided to use three methods learned in class: Logistic Regression, Support Vector Machines, and Neural Networks.

I. Logistic Regression

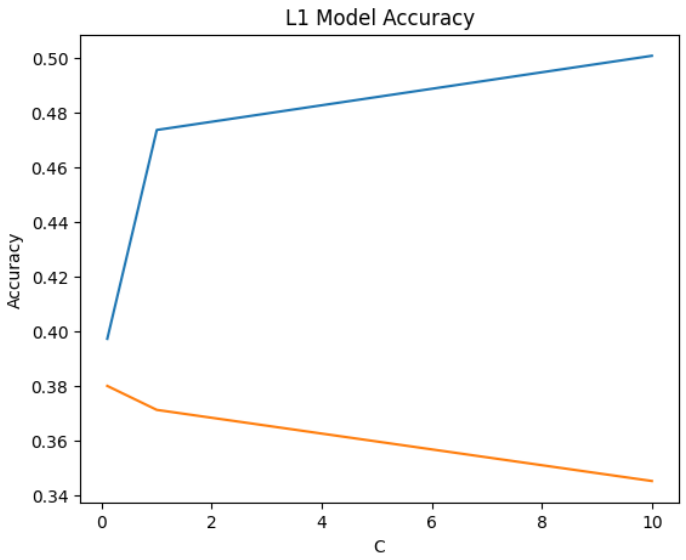
Logistic Regression is a popular algorithm in machine learning for classification problems and was the first model we created. Although it is more commonly used for binary classification, its applications can be extended to multi-classification through the use of the “one vs rest” approach. This approach involves training multiple binary logistic regression models, where each model is trained to distinguish one class from all the other classes. Each of the trained models would predict the probability of an image belonging to the corresponding class, and one with the highest probability would be chosen as the

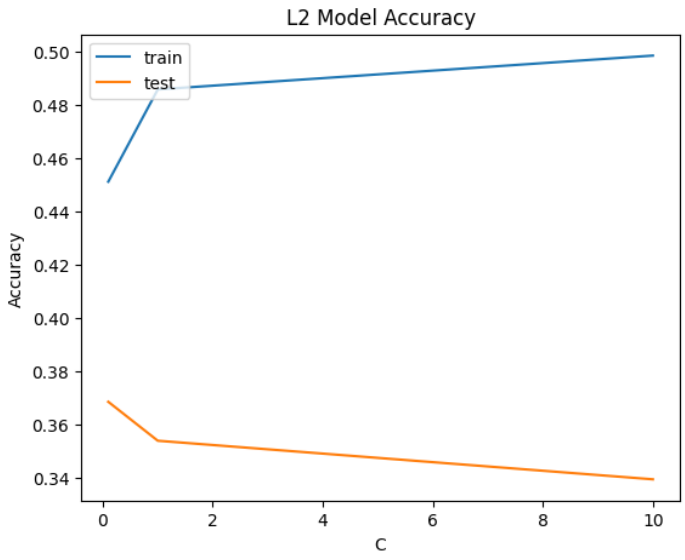
final prediction. In this case, we have 7 different classes in the form of 7 different emotions. Since logistic regression is considered a fairly basic model, it will be serving as the baseline for our project.

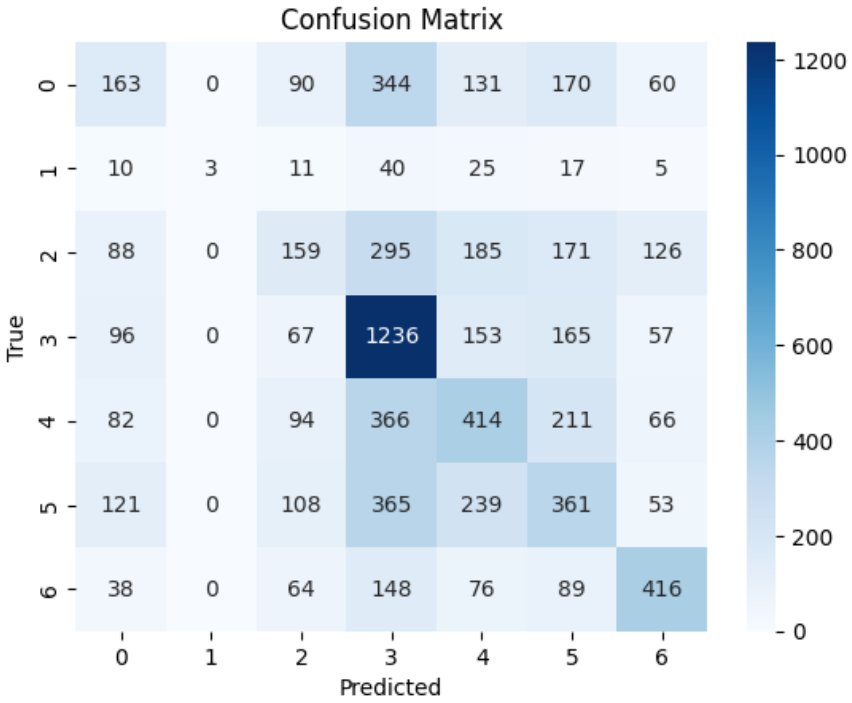
During implementation of the code for this algorithm, we used the LogisticRegression from the sklearn.linear_model library. We first tested keeping the features the way they are with L1 and L2 regularizations, and when the hyperparameter C-values were [0.1, 1, and 10]. Afterwards, we tested the dataset with every feature squared with the same regularizations and C-values. Finally, we performed one last feature transformation on the dataset with every feature cubed and tested it under the same conditions as before.

The data table below was collected when there were no feature transformations with the best testing accuracy of 38.34% produced when C = .1 and L2 regularization was used. Below the table is a graph showcasing the training and testing accuracy for both regularizations over the different C-values.

Cost	Regularization	Train Accuracy	Test Accuracy
.1	L1	0.3972	0.3799
1	L1	0.4736	0.3711
10	L1	0.5008	0.3451
.1	L2	0.4537	0.3834
1	L2	0.4892	0.3571
10	L2	0.5014	0.3422



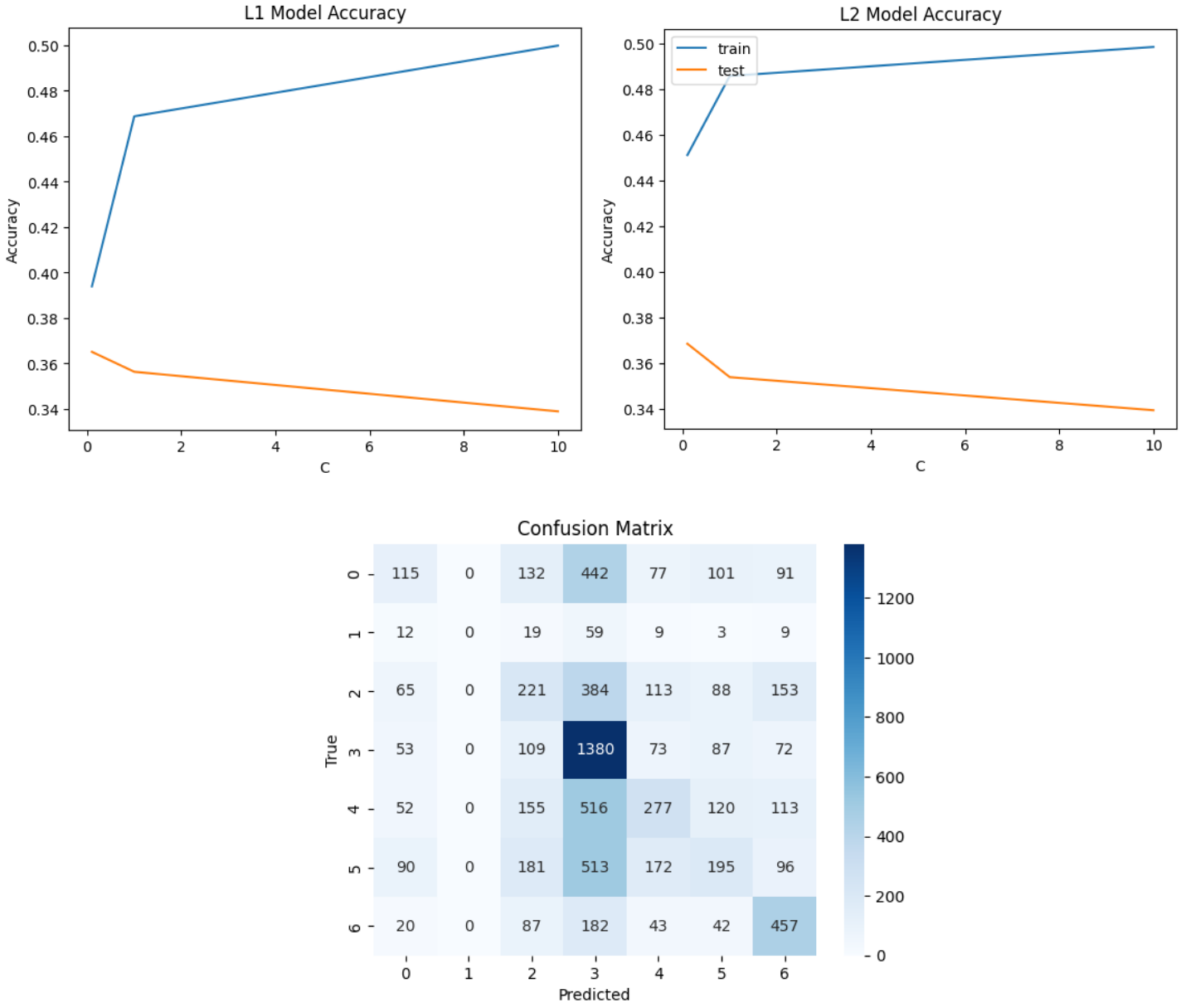




Confusion matrix for no feature transformation (regularization = L2 and C=0.1)

The data table below was collected after squaring every feature and the best test accuracy was produced when C = .1 and L2 regularization was used. Below the table is a graph showcasing the training and testing accuracy for both regularizations over the different C hyperparameters after squaring every feature.

Cost	Regularization	Train Accuracy	Test Accuracy
.1	L1	0.3940	0.3651
1	L1	0.4688	0.3564
10	L1	0.4999	0.3390
.1	L2	0.4511	0.3685
1	L2	0.4858	0.3539
10	L2	0.4985	0.3394

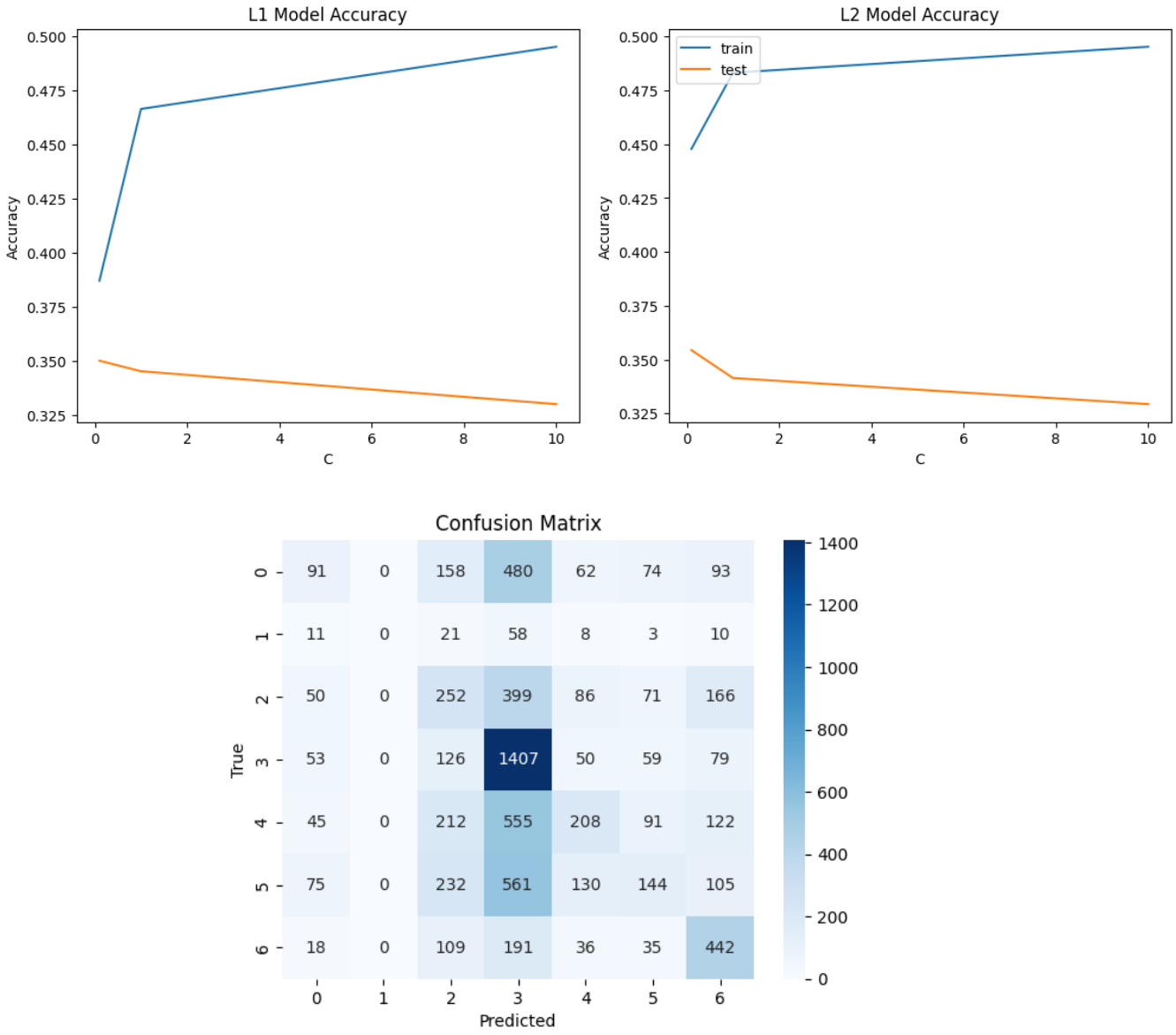


Confusion matrix after squaring every feature (regularization = L2 and C=0.1)

The data table below was collected after cubing every feature and the best test accuracy was produced when C = .1 and L2 regularization was used. Below the table is a graph showcasing the training and testing accuracy for both regularizations over the different C-values after squaring every feature.

Cost	Regularization	Train Accuracy	Test Accuracy
.1	L1	0.3872	0.3502

1	L1	0.4665	0.3454
10	L1	0.4952	0.3302
.1	L2	0.4478	0.3544
1	L2	0.4832	0.3415
10	L2	0.4952	0.3293



Confusion matrix after cubing every feature (regularization = L2 and C=0.1)

After running all the tests for logistic regression, the best testing accuracy we were able to get was approximately 38.34%. This result was produced when we performed no feature transformation and L2 regularization was used when C was initialized to 0.1 as the hyperparameter.

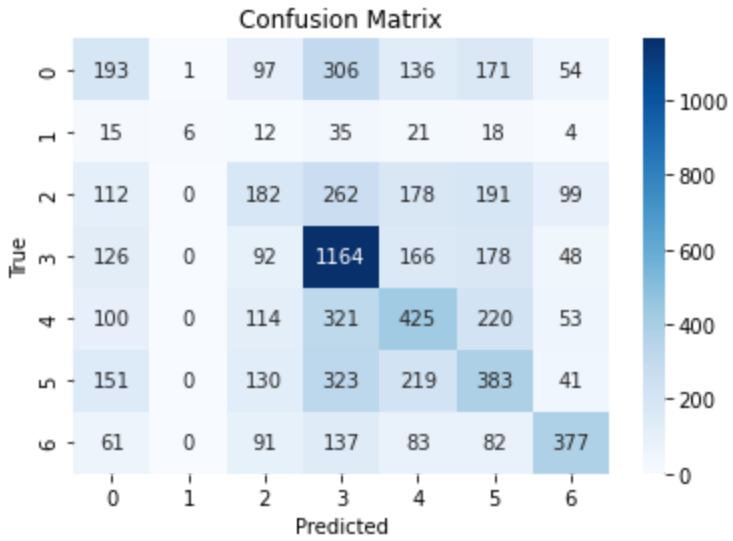
II. SVM

We also decided to use Support Vector Machines for several reasons. For one, images and facial expressions seem to be very complex, so we assumed the relationship may not be linear. Using different kernel functions such as polynomial and radial basis functions may provide a more advantageous classification and provide better accurate modeling of the data. Furthermore, images have a high possibility of noise, given lighting, poses, or other environmental factors. SVMs with soft margins seem to have intrinsic solutions to counteract noise, and could possibly do a better job at classifying images. Most importantly, SVMs and kernel functions have the ability to deal with high dimensional feature spaces, something that images need. SVM allows for more efficient feature selection that can be used for expression recognition.

Our SVM model testing consisted of using multiple kernel functions while changing the cost parameters. We used sklearn's SVC library, and ultimately used accuracy as a metric to evaluate the model because we cared more about the true positives and true negatives because we simply wanted to correctly classify the data.

First we tried the linear kernel function, with varying cost values 0.1, 1, and 10.

Cost	**Train Accuracy**	**Validation Accuracy**
0.1	0.4886	0.3803
1.0	0.5582	0.3526
10.0	0.6106	0.3320

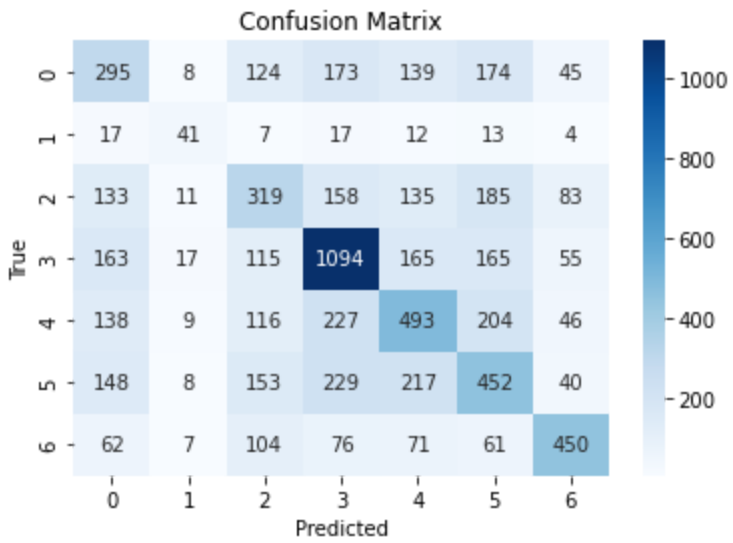


The confusion matrix of the best linear model (C=0.1)

When we lowered C, we raised the margin of error allowing for a better compromise between overfitting and underfitting. However, we knew that having C = 0 meant that these expressions were a hard margin case, and we knew that it was not possible.

Next we decided to try polynomial kernels, which we predicted would yield the best results. To us, polynomials seem to be the most open ended, as there could be endless shapes of function transformations. We did the same test as linear, testing the same C values.

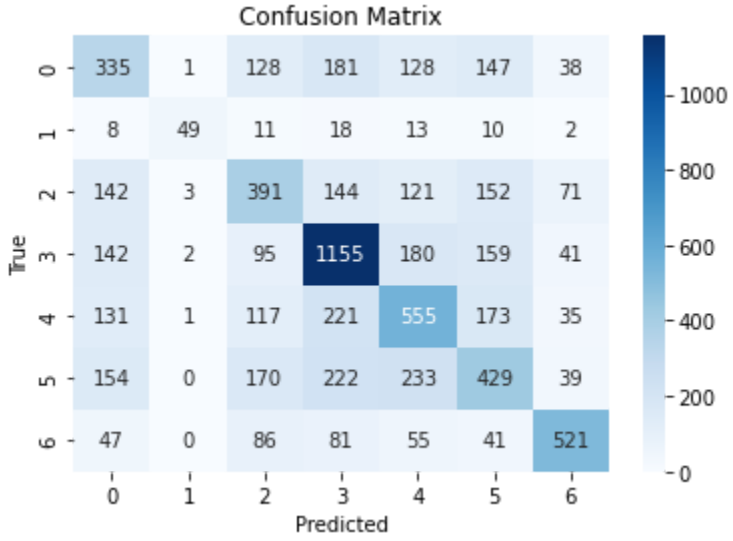
Cost	Training Accuracy	Validation Accuracy
0.1	0.6382	0.4263
1.0	0.9235	0.4380
10.0	0.9929	0.4221



The confusion matrix of the best polynomial model (C=1.0)

Next, we tried RBF with the same data collection method:

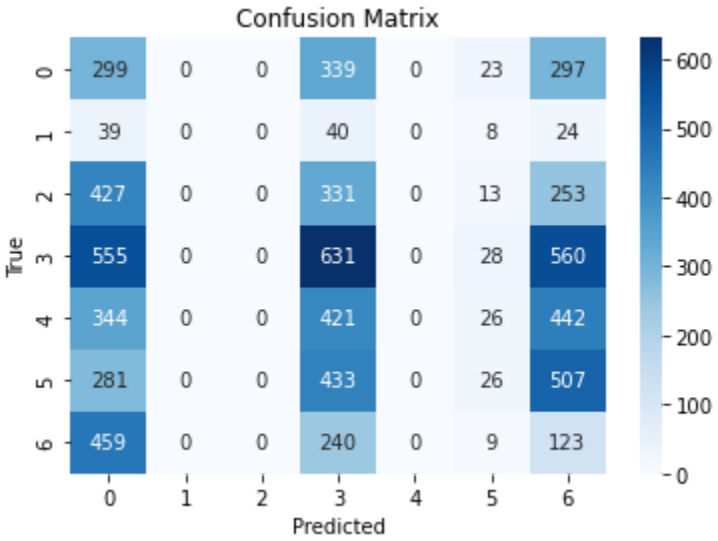
Cost	Training Accuracy	Validation Accuracy
0.1	0.3811	0.3661
1.0	0.6181	0.4464
10.0	0.9770	0.4785



The confusion matrix of the best RBF model (C=10.0)

Finally, we tried the sigmoid kernel function with the same data collection method:

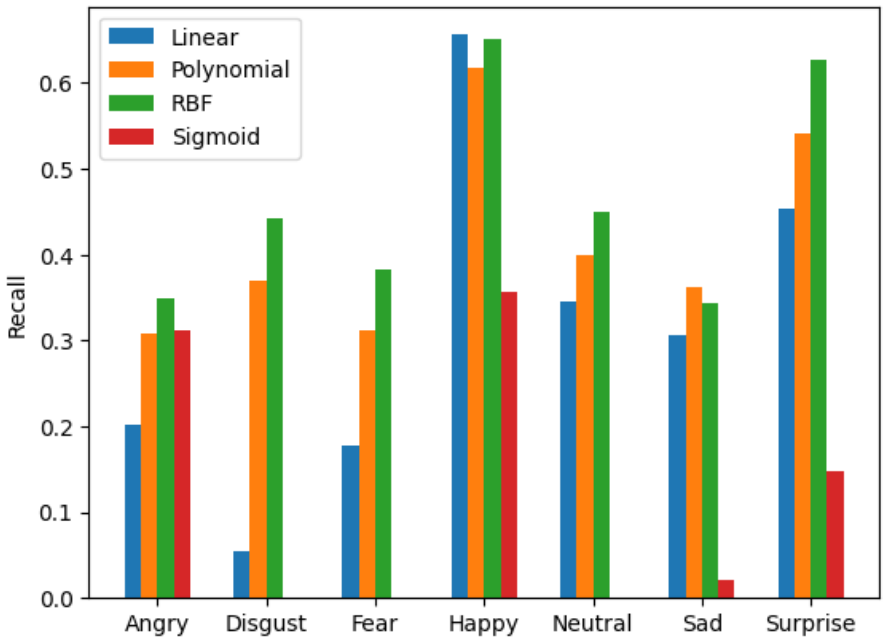
Cost	Training Accuracy	Validation Accuracy
0.1	0.1659	0.1503
1.0	0.1560	0.1256
10.0	0.1554	0.1344



The confusion matrix of the best Sigmoid model (C=0.1)

Ultimately, we found that the RBF kernel function with cost hyperparameter C=10.0, yielding an accuracy of 0.4785. We found that the polynomial function was pretty close as well, achieving a maximum of 0.4380.

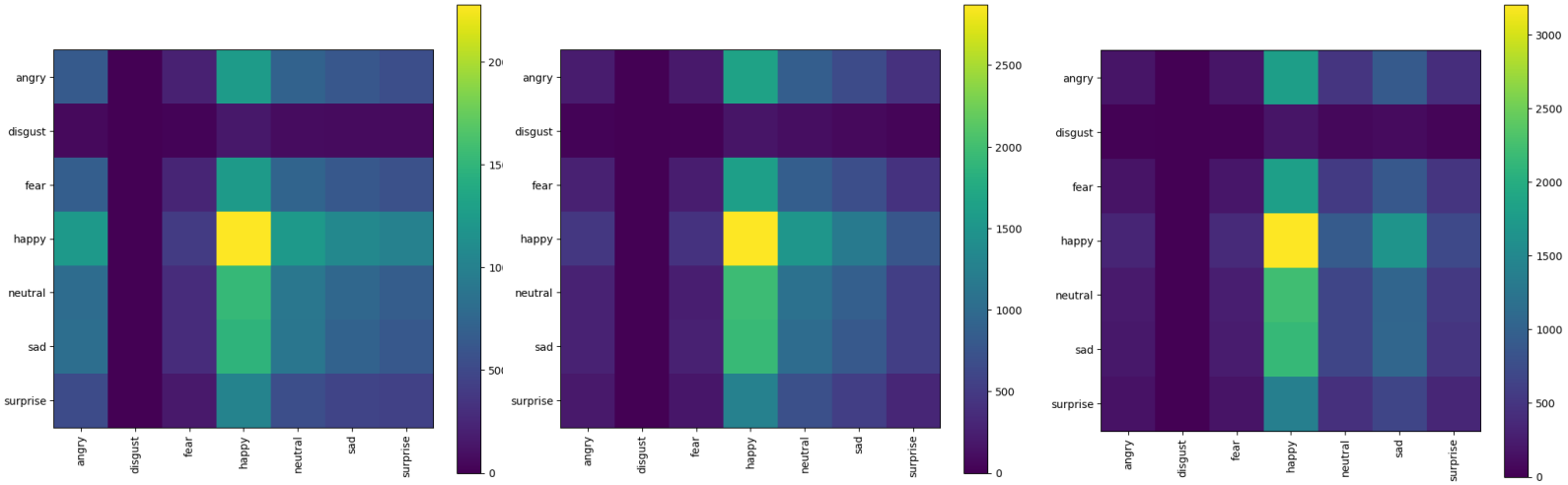
In regards to predicting certain emotions, happiness seemed to have the best true positive ratio, whereas fear, sadness, and anger had the least. Sigmoid was unable to predict disgust, fear, and neutral, giving a recall of 0.



III. Neural Networks

The last model we implemented was Neural Networks, more specifically Convolutional Neural Networks (CNN). A CNN is a type of Neural Network that uses convolutional layers, hence the name, for feature extraction. The basic architecture of the CNN we implemented consists of convolutional layers each using ReLU as its activation function. The last layer of our CNN is a dense layer with the number of neurons equal to the number of classes and uses softmax activation function. The softmax activation function is a variant of the sigmoid activation function that can handle multiple classifications. It produces a probability distribution over the classes, therefore it is used in the last layer of a neural network. The model is compiled with the Adam optimizer with a learning rate of 0.0001, categorical cross-entropy loss function, and accuracy metric. For our feature transformation, we experimented with using 1 to 3 convolutional layers and got the following results.

Layers	Training Accuracy	Testing Accuracy
1	0.3947	0.4267
2	0.4203	0.4521
3	0.4808	0.5068



Confusion Matrix for 1-3 Layer Basic CNN

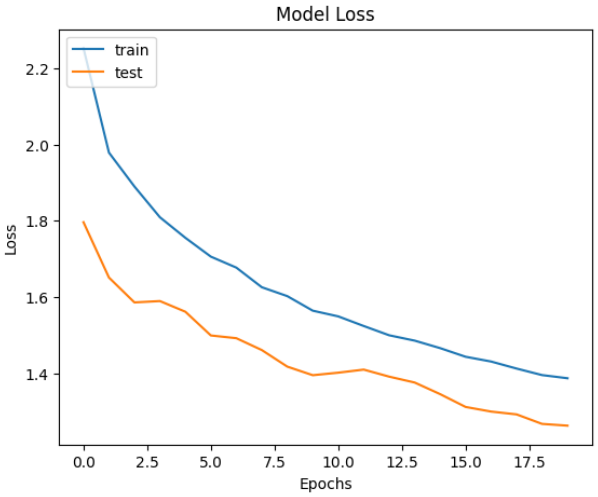
Since 3 layers produced the best results, this structure serves as the basic architecture for our CNN. We further experimented with different variations of it including using different activation functions for the convolutional layers, adding regularization and normalization into the configuration process. Specifically, we tested using tanh and sigmoid for the activation function for the convolutional layers. For regularization we used dropout, which is a popular regularization technique in CNN, that randomly drops a proportion of the neurons in the network during each training epoch by setting them to zero. Although L2 regularization is frequently used in machine learning models, it is not often used in CNN due to them already having a built-in structure that acts as a form of implicit regularization. In addition, we tested adding batch normalization to our CNN model which normalizes the input to each layer of the network, so that the mean activation is close to zero and the standard deviation is close to one.

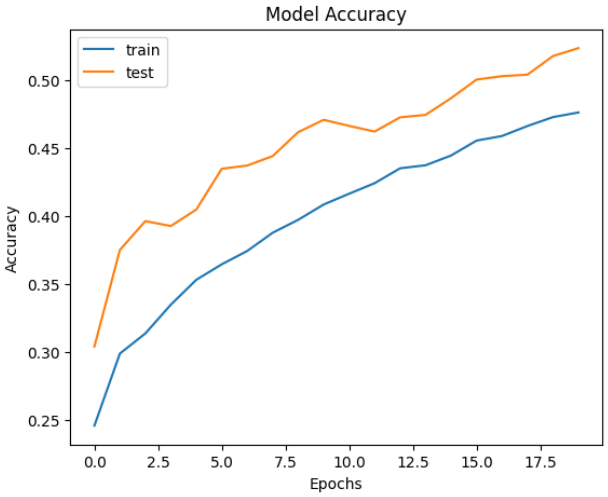
The table below shows the results of using ReLU as the activation function for the convolutional layers for the basic CNN, the CNN with regularization and the CNN with both regularization and batch normalization. From the data, CNN with both regularization and batch normalization produced the best testing accuracy of 52.35% among the models.

Activation Function	Model	Training Accuracy	Testing Accuracy
ReLU	Basic CNN	0.4808	0.5068
ReLU	CNN with Regularization	0.4391	0.4521
ReLU	CNN with	0.5123	0.5235

	Regularization & Batch Normalization		

The following graphs are based on the ReLU CNN with both regularization and batch normalization. The first graph represents the model loss over every epoch for both the training and testing sets. The second graph shows the model accuracy over every epoch step for both the training and testing sets.

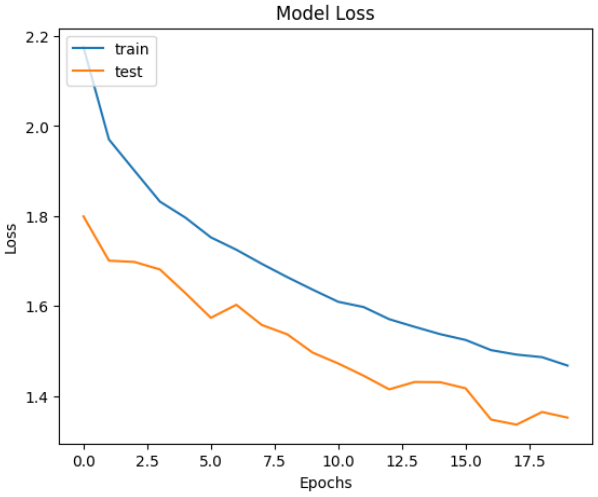


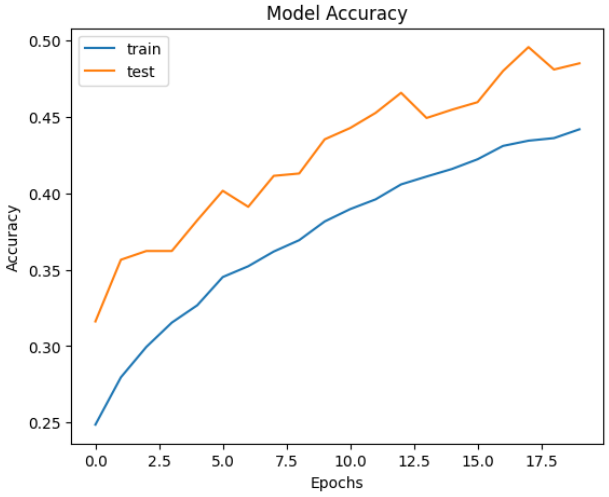


The following table below shows the results of using tanh as the activation function for the convolutional layers for the basic CNN, the CNN with regularization and the CNN with both regularization and batch normalization. From the data showcased in the data, CNN with both regularization and batch normalization produced the best testing accuracy of 48.47% among the models.

Activation Function	Model	Training Accuracy	Testing Accuracy
Tanh	Basic CNN	0.4518	0.4675
Tanh	CNN with Regularization	0.41.71	0.4391
Tanh	CNN with Regularization & Batch Normalization	0.4610	0.4847

Since the tanh CNN model with both regularization and batch normalization produced the best results, below are two graphs corresponding to the model. The first graph represents the model loss over every epoch for both the training and testing sets. The second graph shows the model accuracy over every epoch step for both the training and testing sets.

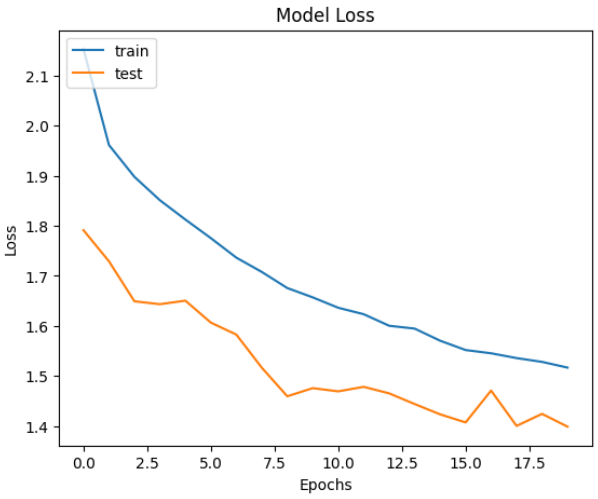


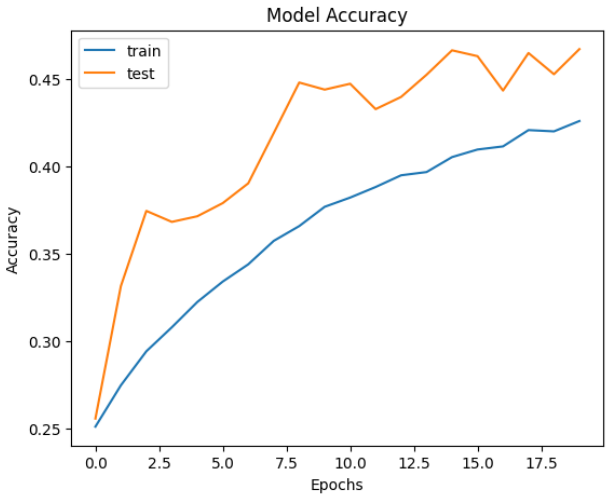


The final table below shows the results of using sigmoid as the activation function for the convolutional layers for the basic CNN model, the CNN model with regularization and the CNN model with both regularization and batch normalization. From the data, CNN with both regularization and batch normalization produced the best testing accuracy of 46.73% among the models.

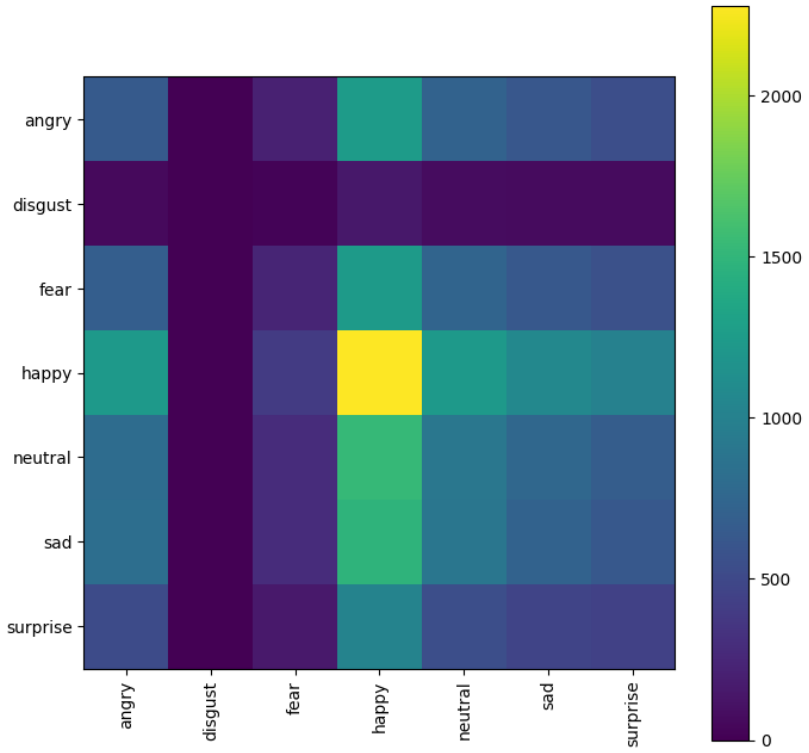
Activation Function	Model	Training Accuracy	Testing Accuracy
Sigmoid	Basic CNN	0.3160	0.3383
Sigmoid	CNN with Regularization	0.2513	0.2471
Sigmoid	CNN with Regularization & Batch Normalization	0.4588	0.4673

The two graphs below represent the sigmoid CNN model (with both regularization and batch normalization) loss and accuracy over every epoch for both the training and testing sets.





After comparing the best result from all three activation functions, it seems that the CNN model produces the greatest testing accuracy when the activation function is ReLU and the model uses regularization and batch normalization. It is important to note that it is possible to improve the testing accuracy through two methods. First, is by making the CNN architecture more complex by adding more layers, however, doing so would not guarantee better results and would increase the time complexity when running the CNN model. The second method is to increase the epochs while running the models, however that would also increase the time complexity when running each CNN model.



Confusion matrix for the best CNN model
(Activation function is ReLU, with dropout & batch normalization)

4. Final Conclusion

Ultimately, the model that produced the best results was CNN when using ReLU as the activation function while incorporating dropout regularization and batch normalization. This model was able to achieve a testing accuracy of 52.35% while the model that achieved the second highest testing accuracy was a basic CNN model using ReLU activation at 50.68%. The third highest testing accuracy was 48.47% which was produced using a CNN model using tanh activation and possessed regularization and batch normalization. Below is a datatable showcasing the top 4 models with the highest testing accuracy.

Model	Testing Accuracy
CNN using ReLU, dropout regularization and basic normalization	53.35%
Basic CNN using ReLU	50.68%
CNN using tanh, dropout regularization and basic normalization	48.47%

SVM using RBF kernel function with cost hyperparameter C = 10.0	47.85%

After experimenting with logistic regression, we found that the most effective hyperparameter value was when C = .1 and that the most effective regularization method was L2 for all logistic models. The range of accuracy for all models of logistic regression was approximately 33% to approximately 38%. In the end, the best testing accuracy logistic regression was 38.34% when there was no feature transformation and L2 regularization was used when C = .1 for the hyperparameter. Given that as the hyperparameter decreases, the training accuracy begins to increase, overfitting is likely involved as regularization becomes less of a factor in the objective function. Therefore, for future improvements on logistic regression, it would be best to test other values for the value C specifically when the C-value is less than 0.1.

SVM provided a method to add complexity while maintaining reliability. Ultimately, each kernel function provided varying output ranging from 0.13 to 0.48 accuracy. However, the regularization with each kernel function changed the output marginally. The kernel function that provided the highest accuracy was RBF, while the polynomial kernel scored consistently the highest. With RBF, increasing the cost parameter ultimately increased the accuracy, which meant that there was some overfitting. However, other kernel functions struggled with underfitting, with adding larger cost parameters making it worse. In the end, if we had more time, we would try more cost parameters and try to formulate a stronger relationship between the accuracy and regularization, as it may not be a linear relationship.

Convolutional Neural Networks is a popular model frequently used in image classification as its built-in convolutional layer reduces the high dimensionality of images without losing its information. Therefore, it is unsurprising that the top 3 testing accuracy throughout this project was achieved by CNN models. In general, each CNN model improved their testing accuracy after incorporating both dropout regularization and batch normalization. It was also interesting to see that the testing accuracy decreased after incorporating only dropout regularization to the basic structure of the CNN. As mentioned, it is possible to improve the testing accuracy through two methods. First, is by making the CNN architecture more complex by adding more layers, however, doing so would not guarantee better results and would increase the time complexity when running the CNN model. The second method is to increase the epochs while running the models, however that would also increase the time complexity when running each CNN model. Therefore, for future improvements on CNN, it would be best to try to add more layers to the architecture of the CNN. Although currently, it takes approximately an hour

to run each CNN model so constructing the architecture to be more complex will likely increase this time substantially. Another future experiment would be to test the CNN while mixing the activation functions so every convolutional layer does not all use the same activation function.

Overall, facial expression recognition is a very complex task that possibly requires more complex knowledge, or a combination of all of these different techniques, something that we did not get the chance to fully explore. Given more time to experiment and time to learn different approaches, we may find a better method to getting greater testing accuracy. However, we are happy with our results and being able to apply the knowledge of the class to something important.

5. Works Cited

https://github.com/gulpinhenry/sentiment-analysis-ml