Emotion Detection: Facial Expression Recognition

Micheal Du, Maxine Zhang, Chenyang Zhang
{q6du, m342zhan, c529zhan}@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Abstract

Facial Expression Recognition technologies have a wide range of applications: from creating a more interactive personal assistant to emotion and mental health monitoring systems that may save lives. However, in order to achieve these goals, it is crucial to choose a suitable model. In this paper, we investigated if Convolutional Neural Networks (CNN) will achieve better performance for identifying emotions from photos than a simpler model, K-Nearest Neighbours (KNN). In this process, we trained and optimized these two models and compared their test accuracy and confusion matrix.

After we conducted relevant research and experiments on the two models, we found that a simple CNN model outperforms KNN as convolution layers are designed to capture high and low-level features from a photo. Our optimized CNN model achieved an accuracy of 65.1% compared to 34.7% for KNN, and we believe this performance can be greatly improved with a larger and balanced data-set.

Introduction

The facial expression, a form of nonverbal communication, is a very powerful signal of human emotions. A facial expression is built by the motions and positions of muscles beneath the skin. In normal communication, the ability to read other people's facial expressions correctly is vital in maintaining relationships and pursuing career goals. Facial expression recognition (FER), a research field in computer vision and machine learning, tackles the problem of determining human emotions based on facial expressions. In the 1970s, Ekman and Keltner suggested that some facial behaviours are universally associated with certain emotions regardless of cultures (Ekman and Friesen 1971); these basic emotions are happiness, sadness, anger, surprise, disgust, and fear. Nowadays, describing other emotions by combining these basic discrete emotions is one of the most popular research topics of FER as it pioneers investigation and sets a comprehensive definition for facial expressions (Li and Deng 2020).

For real-life applications, FER technologies can be applied in a wide range of industries such as robotics and human-computer interaction systems. For example, a specific FER system that recognizes facial behaviors for fatigue can be applied to driver fatigue surveillance systems, an active warnings of fatigue to drivers can reduce fatalities. For the field of sociable robotics, robots that can recognize the emotion of the speaker based on facial expressions in addition to speech can respond more accurately to people and can better mimic a human's response.

To develop a FER system, choosing a suitable machine learning algorithm/model is crucial to optimize the performance. From machine learning to deep learning, many models were introduced that can be applied to FER, such as K-Nearest Neighbours (KNN), Convolutional Neural Networks (CNN), etc. To quantify the target classes for our FER model, we limited the output space to the 6 basic facial expressions (happiness, sadness, anger, surprise, disgust, fear) plus the neutral expression. In this paper, we investigated the answer to the research question "Will a CNN model achieve a higher accuracy for identifying basic emotions from photos compared to a model based on KNN?"

We evaluated each model based on its accuracy as well as its confusion matrix. We define the accuracy of each model by calculating the percentage of images that were identified correctly. On the other hand, we used the confusion matrix to determine the precision and recall for each target class. We leveraged an existing labeled data-set from Kaggle for training and testing (Kaggle 2014). When building the model, we used KNN as the baseline model and investigated the different structure of convolutional networks. We increased the number of convolutional layers and added regularization layers to find the most optimal representation of our data. We tested and concluded the ability of KNN model on classifying data with multiple categories.

For KNN, we used a pre-trained facial landmark detector to achieve a test accuracy of 34.7% and significantly reduced the training time compared to training KNN with full pictures. On the other hand, our baseline CNN model achieved 53.8% test accuracy, and after adding regularization layers and fine-tuning hyper-parameters of the model, we improved the test accuracy to 65.1%. Overall, the CNN model outperformed the KNN model as convolution layers contain filters that can extract both high and low-level features from an image. However, since we were using a small and unbalanced data-set, both models had relatively lower accuracy, and performed best at predicting Happy.

Our results showed that facial expression recognition is a

complex problem that is better represented with a more complex model. While optimizing the CNN model, we found that our results were consistent with previous researches of hyper-parameters, with the exception of adding batch normalization. In our case, we found that it may disturb the training processing of the model. More importantly, since our CNN model only contains about 1.6 million neurons with a small filter size, it can be easily deployed into a mobile application once we improve the performance by training this model with more data.

Related Work

Facial recognition is used in a wide range of applications. Many researchers have published papers regarding image data-sets, facial features extracting techniques, and training models. Since most people share similar facial structures, it is challenging to differentiate an individual from a crowd. Therefore, most researchers focus on topics involving general facial features. Human emotion recognition is an example of such a topic: researchers have taken advantage of the similarity between people's facial structures to accurately extract facial expressions and proposed a compact frame-based facial expression recognition framework that requires minimal parameters during training (Kuo, Lai, and Sarkis 2018).

Since facial expressions can be categorized, researchers have considered using k-nearest neighbours (KNN) for facial expression recognition. Dino et al., in a recent publication, used the Viola-Jones algorithm for face detection and KNN for facial expression classification (Dino and Abdulrazzaq 2019). The accuracy of this model was 79.97% (Dino and Abdulrazzaq 2019). To obtain a model with higher accuracy, one possible approach is to perform FER using Neural Networks. For example, Saket S. Kulkarni created a model that applies committee Neural Networks with several facial features extracted from facial images (Saket S Kulkarni and Hariharan 2009). The system achieved a test accuracy of 90.43% though it was not robust enough to detect the exact expression correctly if the input data does not contain both spontaneous and deliberate expressions (Saket S Kulkarni and Hariharan 2009).

Other researchers have implemented more complex models with more parameters to further increase the accuracy of FER models. For example, Deepface used a nine-layer deep neural network, and the network takes more than 120 million parameters. The model was implemented using several locally connected layers without weight sharing, rather than standard convolutional layers (Taigman et al. 2014). The researchers also trained the model on a more complex data-set, and in the end, achieved an accuracy of 97.35%, a performance level that is close to human behaviour (Taigman et al. 2014).

Methodology

In order to select a model that best identify emotions from photos, we first needed to choose a good data-set.

We leveraged the Kaggle Facial Expression Recognition Challenge data-set for training and testing. This labeled data-set consists of 28709 training examples, 3589 testing examples, and 3589 validation examples. The inputs of this data-set are 48p $\times$ 48p gray-scale close-up face photos and the target values are categorized emotions.

Value	Emotion
0	Angry
1	Disgust
2	Fear
3	Happy
4	Sad
5	Surprise
6	Neutral

Table 1: Categorical Value for Each Basic Emotion

The distribution of target classes in the given data-set is shown in Table 2. Since the distribution of classes is unbalanced, this may cause difficulty in obtaining a fair and robust model. Predicting classes with fewer examples such as disgust would be more challenging since there is less data for the model to learn the characteristics of this class. Hence the model would struggle to differentiate examples from the minor classes from the other classes with more data.

Value	Emotion
Angry	3995
Disgust	436
Fear	4097
Happy	7125
Sad	4830
Surprise	3171
Neutral	4965

Table 2: Target Distribution

We used sampling techniques to decrease the bias that may stem from the large discrepancy between samples for the happy class and samples for disgust. We used the Synthetic Minority Over-sampling Technique (SMOTE) to create synthetic samples for the minor classes and undersampled the major classes.

```
# Oversample with SMOTE
smote = SMOTE()
X, y = smote.fit_sample(X, y)

# Undersample with NearMiss
undersample = NearMiss()
X, y = undersample.fit_resample(X, y)
```

With the pre-processed and properly-sampled training set, we explored different models to find the model that yields the highest accuracy. Particularly investigated K-Nearest Neighbour (KNN) and convolutional Neural Networks (CNN) to find the most optimal representation of our data.

K-Nearest Neighbour (KNN)

We trained KNN and used the result as a baseline score. The rationale was that KNN has one single hyper-parameter to

tune, and is a simple and intuitive model to represent the data for our research question. KNN provides a non-linear separator to the data points, and it determines the label of a new data-point by the most frequent label among its k-nearest neighbors:

$$y_x \leftarrow \text{mode}(\{y_{x'} | x' \in \text{knn}(x)\})$$

where y_x is the label of point x, and x' are the k-nearest neighbors of x.

To optimize the performance of KNN, we extracted the facial landmarks using the facial detector from Dlib. This model detects 68 2-D points on a human face. These are points on the face such as the corners of the mouth, the eyebrows, the eyes, and so forth (King 2009).

For this model, the k-nearest neighbors were calculated by the Euclidean distance between input point and points in the data-set and were defined as:

$$d(x, x') = (\sum_{j=1}^{M} |x_j - x_{j'}|^2)^{\frac{1}{2}}$$

with M being the number of features of a single input.

The hyper-parameter for KNN is K, which is the number of nearest neighbors the model uses to determine the label for a single input. The common practice is to use $\sqrt{N}$ as the K value where N is the number of samples in the training set, thus we used the range of $[\sqrt{N} - 10, \sqrt{N} + 10]$ in cross validation to find the most optimal K.

Convolutional Neural Network (CNN)

CNN is a class of deep Neural Networks. CNN introduces convolutional kernels which are considered as a feature map. CNNs have applications in image and video recognition and neural language processing. The basic idea of CNN is to extract small features and assemble these features into more complex features. CNN generally has higher accuracy for predictions from image inputs than a simple multi-layer perceptron neural network.

For pre-processing, we loaded the image as a $48 \times 48 \times 1$ 3d NumPy array, then divided every term by 255. The reason to extend the matrix by 1 dimension was that it is required for the Keras model input; the reason for normalizing data was to avoid large values of weights during training.

For our experimental model, the input layer has a shape of $48 \times 48 \times 1$. The model has 3 convolutional layers, 3 max-pooling layers, a flatten layer, and 3 fully connected layers. Each CNN layer applied a filter of size 2×2, and set RELU as its activation function. Each max-pooling layer has size 2×2. Before the fully connected layers, there was a Flatten layer that reshaped the data to a 1D array. For the first 2 fully connected layers, RELU was applied as their activation function. For the output layer, which is the last layer, we selected softmax as its activation function.

$$\text{Softmax } \delta(z)_i = \frac{e^{z_i}}{\sum_{j=0}^{k-1} e^{z_j}}$$

with z being the result of the linear combinations of the last hidden layer, $0 \leq i \leq k-1$ representing the k labels, z_i is the i^{th} item of z, and $\delta(z)_i$ is the result. One advantage of softmax is that it converts the outputs to normalized probability distribution, i.e. every term of the converted outputs is a real value between 0 and 1, and the sum of them is 1. Figure 1 shows the shape of each layer. This model has approximately 1.36 million trainable parameters.

Layer (type)	Output Shape
conv2d (Conv2D)	(None, 48, 48, 32)
max_pooling2d (MaxPooling2D)	(None, 24, 24, 32)
conv2d_1 (Conv2D)	(None, 24, 24, 64)
max_pooling2d_1 (MaxPooling2	(None, 12, 12, 64)
conv2d_2 (Conv2D)	(None, 12, 12, 128)
max_pooling2d_2 (MaxPooling2	(None, 6, 6, 128)
flatten (Flatten)	(None, 4608)
dense (Dense)	(None, 256)
dense_1 (Dense)	(None, 512)
dense_2 (Dense)	(None, 7)

Figure 1: Keras CNN Model with Default Hyperparams

We chose categorical cross-entropy (CE) as the loss function and $Adam$ for the optimizer to train weights.

$$\text{CE} = -\sum_{i=0}^{k-1} y_i \log \hat{y}_i$$

where $\hat{y}_i$ is the i^{th} scalar value of the output, and y_i is the corresponding target value.

$Adam$ is an optimizer for stochastic gradient descent algorithms. This algorithm is computationally efficient, requires little memory, and suitable to large data sets or number of parameters (Kingma and Ba 2014).

In general, to train this model, we set the batch size to 128, and we ran 50 epochs on the training data, validate using the validation data set, and save the model with the lowest validation loss during the training process for future prediction.

The model discussed in this section was implemented in Python with the Keras library. Keras is a famous neural network library for Python, and it is simple and intuitive to use. We used NumPy library to manipulate training and testing data, such as reshape matrices.

Results

The output space of our data to the 6 basic facial expressions plus the neutral expression, and we used accuracy as the primary performance metric. We defined the accuracy of each model by the percentage of test images that are identified correctly.

$$\text{Accuracy} = \frac{\text{\# of images with correct emotion identified}}{\text{total \# of images}}$$

Accuracy is the most suitable performance metric for the research problem since this metric can lead to the most appropriate solution to our problem. Our goal is to obtain a model that is able to identify emotions based on photos, which is aligned with the definition of accuracy. In addition, since the classes are discrete and non-ordinal, the concept of getting a less wrong answer is not applicable, which accuracy is not able to capture. However, we did not choose the final model based on accuracy alone. In order to gain insight into the performance of our model, we analyzed each model with a confusion matrix to identify precision, recall, and any common misclass between target classes.

The confusion matrix is takes the form of Table 3, and each entry is the occurrence for the true value and predicted value pair.

		True Value			
		class 1	class 2	...	class n
Predicted Value	class 1			...	
	class 2			...	
	⋮			⋮	
	class n			...	

Table 3: Example Confusion Matrix

We were able to construct this table with scikit-learn library function and calculate precision and recall value for each class.

```
# Print the confusion matrix
metrics.confusion_matrix(y_true, y_pred)

# Print precision, recall and other metrics
metrics.classification_report(
    y_true, y_pred, digits=3)
```

```
[[4 1 1]
 [6 2 2]
 [3 0 6]]
              precision    recall  f1-score   support

     class 1      0.308     0.667     0.421         6
     class 2      0.667     0.200     0.308        10
     class 3      0.667     0.667     0.667         9

    accuracy                          0.480        25
   macro avg      0.547     0.511     0.465        25
weighted avg      0.581     0.480     0.464        25
```

Figure 2: Scikit-Learn Confusin Matrix Summary

Experimental Design

Intuitively, the KNN model is expected to differentiate facial expressions since similar facial expressions tend to have similar facial feature points, which allows KNN to group similar emotions. However, the disadvantage of using KNN is that, due to the low complexity of KNN, the model may not be able to distinguish different expressions that suggest similar emotions effectively. Furthermore, the accuracy of KNN also depends on the quality of inputs, and the model is sensitive to noise in the data. Therefore, the result from KNN was used as a baseline score for other algorithms. To determine the hyper-parameter K that gives the best test accuracy, we researched the KNN model with the K values in the range of $[\sqrt{N} - 10, \sqrt{N} + 10]$. Since FER-2013 has split the data into training and validation parts, we used them for training and validation respectively. Since FER-2013 has a relatively small data size, we used 10-fold cross-validation to train for best K.

To find the CNN model with the highest test accuracy, we built more sophisticated CNN models by increasing the number of convolutional layers. We also added regularization layers such as batch normalization and dropout layers. Then we ran those models on the same testing examples provided by FER-2013. At last, we compared those models using accuracy and the confusion matrices.

The main hyper-parameters that we were adjusting includes dropout rate of dropout layers, the kernel size, the number of kernels in a CNN layer, the number of neurons in a fully connected layer. The performance of a model while adjusting the hyper-parameters is measured by the categorical cross-entropy loss on the validation set provided by FER-2013. If the validation loss was rising in 10 consecutive epochs, we would stop the training process because it suggested that the model was overfitting. We set the default number of epochs to 50. If the model had no sign of overfitting in 50 epochs, we would double the epochs. We kept track of the lowest validation loss during the training of a given set of hyper-parameters and selected the set with the lowest validation loss among all sets.

The model shown in Figure 1 has some of the hyper-parameters that were inspired by the famous VGG model (Simonyan and Zisserman 2015). VGG achieves 92.7% top-5 test accuracy on ImageNet, which has over 14 million images belonging to 1000 categories, in the Large Scale Visual Recognition Challenge 2014 (ILSVRC2014). In VGG16, consisting of 13 CNN layers and 3 fully connected layers, the number of filters of the first 3 CNN layers before each max-pooling is 64, 128, and 256. Since we were limited by the size of our data and wanted to reduce training time, we halved those values as the number of filters for our model. Kernel size was set as 2×2 since we wanted to start with capturing the smallest features. 256 and 512, the number of neurons we chose for the first 2 fully connected layers, were reduced from VGG's 4096 and 4096 to improve training performance.

Implementation

We have explored two approaches to the implementation of KNN. The first approach used the KNN model on the raw data without first extracting any facial features. This was also used as our baseline KNN model. To implement this model, we flattened the $48 * 48$ image into a $1 * 2304$ normalized array and split the data-set into training and test sets. We then trained the KNN model using the training set and tested its accuracy. Since this KNN model was trained on complete images, which means that the input was large, thus the training time was significantly higher. As a result, we were only

able to train on a subset of the data-set. [1]

For the second implementation, we leveraged a pre-trained detector for 68 key facial landmark points to make a more robust model. We first grouped the images by the output class and generated x-y coordinates of the 68 landmark points for each image. We compiled these landmark points into an array of length 68 and feed these data points to a KNN model and tested its accuracy.

We then experimented with different values to find the best hyper-parameter K. We limited the range of possible K values to have a mean of $\sqrt{N}$ and depended the window width on data size. We then trained the model with each hyper-parameter K and compared the test accuracy. The K with the highest test accuracy was chosen to be the best hyper-parameter.

To implement CNN, we first loaded in the data-set, pre-processed the data, and divided the inputs and targets into training, validating, and testing sets. Then we created the model with default parameters (Figure 1) and trained the model with 50 epochs. This model was used as our baseline CNN model. After building the baseline model, we explored the influence of dropout layers and batch normalization layers (batch norm). Generally dropout layers can be used as a regularization method and batch normalization are used to make the model more stable.

After comparing models with different combinations of dropout and batch normalization layers, we performed hyper-parameter optimization on the best structure out of four based on the magnitude of validation loss. We experimented with different dropout rates, the number of normalized convolutional layers, kernel size, and the number of filters. To fine-tune the hyper-parameters, we isolated one hyper-parameter at a time, created new models with different values for the hyper-parameter, tested the models, and compared the results. After finding the best value for a hyper-parameter, we applied the hyper-parameter to the model and test other parameters with the new model. We obtain our best CNN model for facial expression detection after testing different values for all hyper-parameters.

Performance

K-Nearest Neighbour (KNN)

The KNN model without facial detector had a worse performance and yielded an accuracy of 27.2% (Figure 3). The reasons for the poorer performance were that the training set is smaller and this model was prone to noise in the training images. Since this model was trained on full images, it is influenced by irrelevant features in the image such as the location of the face, background colour, and skin tone.

The KNN model that leveraged the pre-trained facial landmark detector performed better than our baseline KNN model. The test accuracy with the best hyper-parameter was approximately 34.7% (Figure 4).

The improvement in performance is expected since facial feature extractions decrease the influence of noise on the model. In addition, being able to use all of the images in our data-set may also be a factor.

[1]The models in this paper can be found in this repository: `github.com/meixinzhang/ExpressionRecognizer`

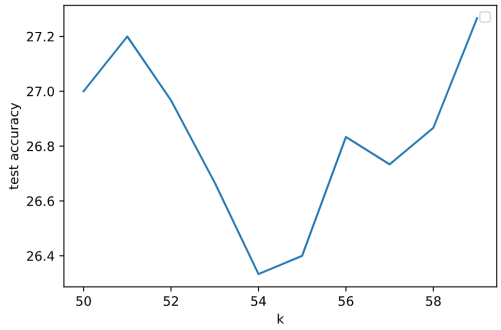

Figure 3: Validation Accuracy of Baseline KNN Model

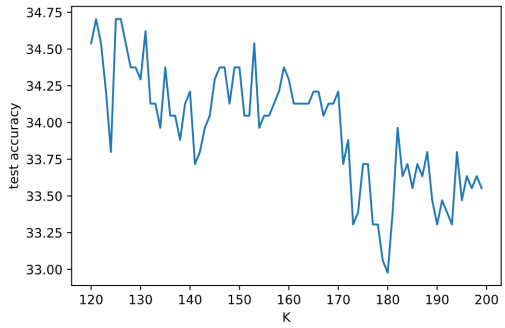

Figure 4: Validation Accuracy of KNN with Detector

To further analyze the performance of our best KNN model, we generated the confusion matrix based on the KNN model with facial detector (Figure 5, 6).

	precision	recall	f1-score	support
Angry	0.17	0.01	0.01	183
Disgust	0.00	0.00	0.00	3
Fear	0.00	0.00	0.00	101
Happy	0.35	0.91	0.50	378
Sad	0.28	0.23	0.25	280
Surprise	0.00	0.00	0.00	182
Neutral	0.00	0.00	0.00	89
accuracy			0.34	1216
macro avg	0.11	0.16	0.11	1216
weighted avg	0.20	0.34	0.22	1216

Figure 5: Precision/Recall Summary of KNN with Facial Detector

Figure 5 and 6 show that the KNN model classified most of the input images as Happy and Sad. Observe that the recall of class Happy is very high while the precision is low, this suggests that the model has trouble distinguishing between different facial expressions and is categorizing most images as Happy. This is supported by high values in the column for Happy in the confusion matrix. This shows that KNN is not a good classifier for facial expressions as it is not complex enough to extract information from the positions of facial landmarks to recognize different facial expressions.

Convolutional Neural Network (CNN)

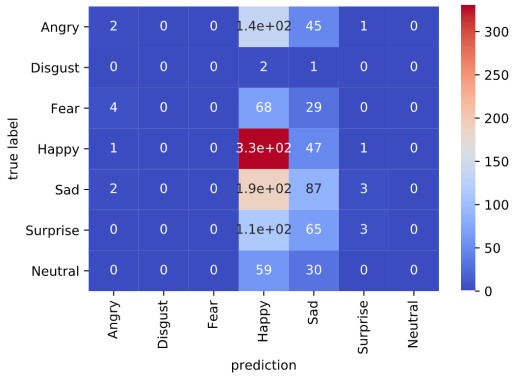

Figure 6: Normalized Confusion Matrix of KNN with Detector

Our baseline CNN network achieved a validation loss of 1.205 and a testing accuracy of 53.80%. We used these values as the benchmarks for the other CNN models. We noticed that this model also had a very low training loss (≈ 0.06), but a high validation loss (> 4), which suggests that the model is overfitting.

After building the baseline model, we explored the influence of dropout layers and batch normalization layers (also called batch norm). We trained three more models, a model with dropout layers, one with batch normalization layers, and one with both dropout and normalization layers. The dropout layers were appended after each max-pooling layer and each fully connected layer. We chose a dropout rate of 0.25. For batch normalization layers, they were appended after each convolution and dropout layers and fully connected layer. We then compare the performance of these four CNN structures.

Model	lowest validation loss	test accuracy
Base model	1.228	53.80%
With dropout	1.156	57.54%
With batch norm	1.406	51.38%
With both	1.154	57.68%

Table 4: Result for adding dropout and batch norm layers

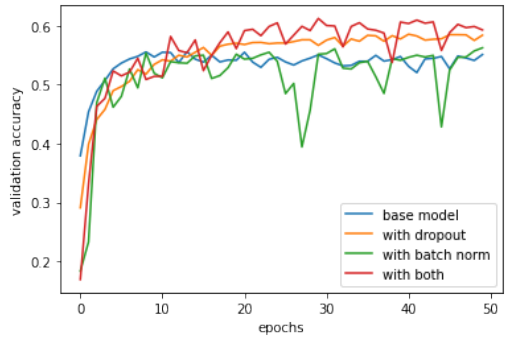

Figure 7: Validation accuracy of the 4 models

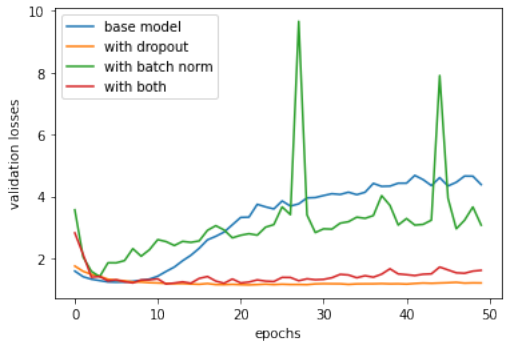

Figure 8: Validation losses of the 4 models

From Figure 7 and 8, and Table 4, we found that the batch normalization layer in the third model did not make training the model more stable and lead to performance improvements. The dropout layers in the second model reduced overfitting and hence led to lower validation loss. The model with both dropout layers and batch norms has the best test accuracy, validation accuracy, and test accuracy. Therefore we decided to use both dropouts and batch normalization layers to test hyper-parameters.

Hyper-parameter Optimization

We first experimented with different values to find the best dropout rate. We tested 4 dropout rates, 0.1, 0.2, 0.3, 0.4, using the model with dropouts and batch norms. We then trained the model with each of these dropout rates and compared the results. In this part, we trained each model with 100 epochs, since adding dropout layers will slow down the learning process.

Dropout rate	best validation loss	test accuracy
0.1	1.260	54.78%
0.2	1.203	58.46%
0.3	1.167	60.10%
0.4	1.124	59.60%

Table 5: Best validation loss and test accuracy of different dropout rate (NOTE: 'best validation loss' means 'lowest validation loss')

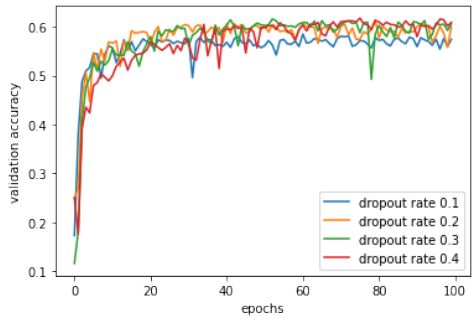

Figure 9: Validation accuracy with different dropout rate

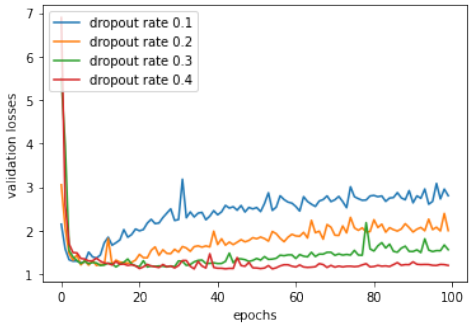

Figure 10: Validation losses with different dropout rate

Figure 9, Figure 10, and Table 5 suggests that the model with dropout rate 0.3 and the model with dropout rate 0.4 have similar validation accuracy and performs better than other models. The former model has higher test accuracy, while the latter has a lower best validation loss. We trained the model with the mean of these two values, 0.35, as we want a balanced result. We noticed that using 0.35 as the dropout rate allows the model to converge faster than the model using 0.4. The model achieved the lowest validation loss of 1.162 and test accuracy of 60.63%. Since the test accuracy is higher than all other values, we used 0.35 for the dropout rate for other hyper-parameters optimizations.

We then explored the batch normalization layers. The two options were FER-CNN6, which has 6 layers of perceptrons: 3 CNN layers and 3 fully connected layers, and FER-CNN9. FER-CNN9 was inspired by VGG networks, where we double each CNN layer. This network has 9 layers of perceptrons. Table 6 shows the changes. We have 3 CNN blocks in both FER-CNN6 and FER-CNN9.

Block In FER-CNN6	**Block In FER-CNN9**
conv2d + batch norm	conv2d + batch norm
dropout	conv2d + batch norm
max pooling	dropout
	max pooling

Table 6: Illustration of CNN Blocks in FER-CNN6 and FER-CNN9

Then we experimented on the kernel size of the CNN layers on both FER-CNN6 and FER-CNN9. The kernel size we chose to test is $(2, 2)$ and $(3, 3)$. We chose smaller kernels as concatenating 2 small kernels has the same effect as applying a larger kernel.

Model	best val loss	test accuracy
FER-CNN6 kernel (2,2)	1.162	60.63%
FER-CNN6 kernel (3,3)	1.097	60.07%
FER-CNN9 kernel (2,2)	1.041	64.14%
FER-CNN9 kernel (3,3)	0.9823	65.09%

Table 7: Best validation loss and test accuracy of FER-CNN6 and FER-CNN9 with different kernel size

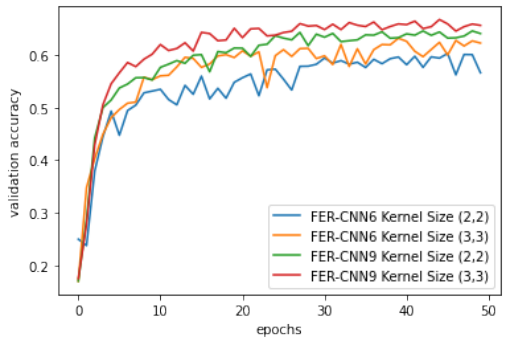

Figure 11: Validation accuracy of the FER-CNN6 and FER-CNN9 with different kernel size

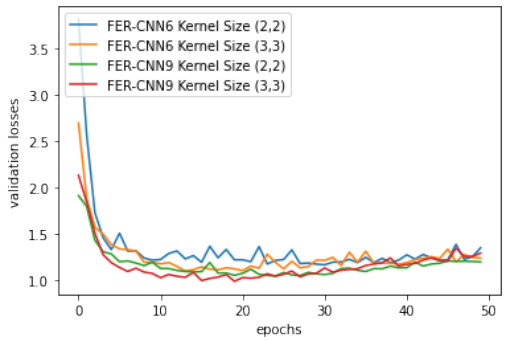

Figure 12: Validation losses of the FER-CNN6 and FER-CNN9 with different kernel size

According to Figure 11 and Table 7, FER-CNN9 has considerably better testing accuracy than FER-CNN6 and switching from kernel size $(2, 2)$ to $(3, 3)$ will reduce the validation losses and increase validation accuracy. Hence, we applied kernel size $(3, 3)$ to FER-CNN6 and FER-CNN9.

The final hyper-parameter we fine-tuned was the number of filters in CNN layers and the number of neurons in the fully connected layers. We doubled these numbers and compared the performance with default hyper-parameters.

Model	best val loss	test accuracy
FER-CNN6 default	1.097	60.07%
FER-CNN6 doubled	1.12	62.83%
FER-CNN9 default	0.9823	65.09%
FER-CNN9 doubled	1.065	62.25%

Table 8: Best validation loss and test accuracy of FER-CNN6 and FER-CNN9 with default/doubled numbers of filters in CNN layers and default/doubled numbers of neurons in fully connected layers

From Table 8, we observed that doubling the numbers of filters and numbers of neurons increased validation losses. For FER-CNN6, it increased test accuracy; for FER-CNN9, it decreased test accuracy.

Finally, after performing optimization on all hyper-parameters, the model with the lowest validation loss and highest test accuracy is FER-CNN9, with dropout layers

(dropout rate 0.35) and batch normalization layers, and a kernel size of (3,3) and the default number of filters. This model had an improved test accuracy of 65.09% from 53.80%, which makes this model our final and best CNN model.

From Figure 13 and 14, we observed that though the accuracy is only 65%, the model has learned to differentiate between different facial expressions, especially Happy, Disgust, and Neutral. The category with the best performance is "Happy", which has the highest precision and recall, and based on the confusion matrix when given a face that is happy, the model rarely categorize it wrong. This category may be performing better since the data-set has the most samples in this category.

	precision	recall	f1-score	support
Angry	0.60	0.52	0.56	491
Disgust	0.70	0.67	0.69	55
Fear	0.58	0.40	0.47	528
Happy	0.81	0.88	0.84	879
Sad	0.58	0.68	0.62	626
Surprise	0.49	0.54	0.52	594
Neutral	0.77	0.75	0.76	416
accuracy			0.65	3589
macro avg	0.65	0.63	0.64	3589
weighted avg	0.65	0.65	0.65	3589

Figure 13: Precision/Recall Summary of our Best CNN Model

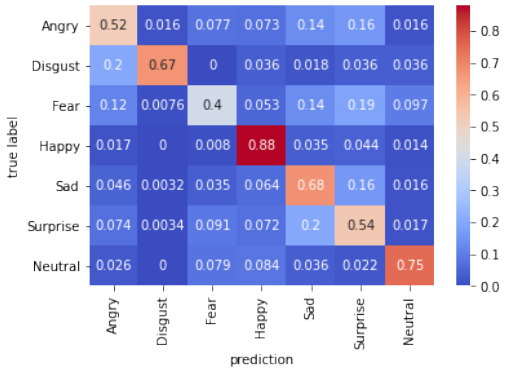

Figure 14: Normalized Confusion Matrix of our Best CNN Model

The category with the lowest precision is Surprise, from the confusion matrix, we found that the model is categorizing a portion of the photos to Fear, Angry, and Sad. This is reasonable since Surprise can be associated with all of these emotions. On the other hand, the category with the lowest recall is Fear, the confusion matrix indicates that the model is categorizing some of the images in these categories to surprise and sad, which is reasonable since fear sometimes has a component of surprise.

Overall, our best CNN model is able to learn facial features and derive a basic emotion from images better than KNN. However, due to the size of the data-set, the model is not robust in some classes, and it failed to distinguish between some expressions that may have similar facial features.

Insights Gained

While we were collecting data for the models, we learned the different situations to apply data normalization and data standardization. For this particular data-set, neither methods are crucial to improve performance since the pixel values are on the same scale (0 to 255).

When we were building KNN models, we gained insights into the disadvantages of KNN: KNN will be slow train on large input data-set and KNN is very prone to noise in the data. In addition, since the data-set has seven different output classes and the data is very complex (images), KNN was not able to find a good separator. Therefore, we learned that though KNN finds a non-linear separator, it is still not a good choice for classifying complex data types with a large output space.

In the process of training CNNs, we learned how to use Keras libraries to implement, train, and test a convolutional neural network. We also gained insights into the different structures of a CNN block and their advantages. While we fine-tuned the model, we learned how dropout layers with different dropout rates and batch normalization layers could affect the training process, train/validation/test accuracy, and train/validation/test losses. When analyzing the performance of each model, we learned how to interpret precision and recall of a class as well as how to extract key information from the confusion matrix.

Discussion

In our experiments, SMOTE and NearMiss that aim to solve the data imbalance did not lead to a satisfactory result as these techniques are best suited for numerical data with a distinct underlying distribution. Though images can be represented using pixels, treating the data points as pure numeric data may lead to the loss of high-level features in the images. In addition, flattening an image leads to loss of localized information, thus the generated flattened array may not be meaningful to the training. We omitted SMOTE in later training, which led to data imbalance negatively affecting the performance of our model.

As for the models, the result of the KNN models matched our previous intuition of only using this model as a baseline. The validation scores indicated that this model is not well-suited for classifying images. Since this model predicts based on distances between key pixels, it is susceptible to noise and change in position of the faces. This method did show a positive correlation between key pixels in a portrait photo and prediction class. However, by simply calculating a non-linear separator using nearest neighbours, the hypothesis may not be strong enough to classify all seven classes correctly.

The confusion matrix of the KNN model showed that the model had a preference to classify images to Happy. This could be as a result of the imbalanced data-set and the way we select the facial features for classification. The landscape detector model gives 68 key pixels as input for training the model, which theoretically extracts key information from an image and this reduces the cost of training; however, since the data-set consists of 7 different categories, 68 points

may not be sufficient for the model to distinguish between different facial expressions. Overall, the performance of KNN agreed with the result by (Dino and Abdulrazzaq 2019). KNN models provide a low-accuracy prediction model as a baseline comparison for more complex models we trained.

When experimenting with different CNN structures, we found that the dropout layers have a significant effect in preventing overfitting. Adding dropout layers will improve the test accuracy in general ($> 4\%$) and the suitable dropout rate is between 0.2 and 0.4 for a CNN network. If the dropout rate is below 0.2, the overfitting problem is still severe. If the dropout rate is above 0.4, it significantly slows the learning rate, and the model converges to an optimal with a slight potential improvement on the test accuracy ($< 0.5\%$), which is not a worthwhile trade-off. On the other hand, doubling CNN layers, a technique that is inspired by VGG, is effective in improving the test accuracy as shown in Table 6. Finally, Table 7 shows that increasing the size of the kernel in the CNN model doesn't a significant impact on test accuracy but can improve the validation losses. This implies more information is lost using 2-by-2 kernels.

A surprising result from our research is that batch normalization layers have little impact on improving the model performance. As researchers state (Ioffe and Szegedy 2015), batch normalization increases the training rate, stabilizes the input distribution of each layer, and improves the accuracy of the model. However, adding batch normalization showed little effect in our model. These layers also made the training process more unstable which is not a result of other related work. This implies that either we added batch normalization layers in the wrong place, or batch normalization layers are not suitable for all CNN models, specifically as classifiers. In addition, this result may also be data-set dependent.

Using an open data-set eliminates the step of data cleaning and format standardization, making pre-processing much simpler. However, it also has the limitation of being restrictive and potentially making data augmentation more difficult. Since the provided images are already compressed to a smaller size, we could not choose a larger size and build a deeper network for more insights; the loss of details also decreases the variance of generated images. This caused our test accuracy to be relatively low compared to other results. In fact, many different CNN models were developed by other researchers that have better performance on different data-sets (Li and Deng 2020), with some models having an accuracy of over 90%.

The confusion matrix of the test results from the CNN model suggests that the CNN model can identify Happy and Neutral with high precision. It implies that for our CNN network, key features for Happy and Neutral emotions are extracted most successfully. This may be due to the imbalanced data-set, as well as these two expressions are very distinguishable from other emotions. For other emotions, such as Angry and Surprise, since they share some key features, such as opened mouth, and there are fewer examples in this category in the training data, the CNN network may misclassify these expressions. For example, our model categorizes about 20% of images with emotion Surprise to category Sad. However, our CNN model achieves much higher test accuracy than a KNN model overall after optimizing the hyper-parameters, adding regularization layers, and adding more CNN layers. The CNN modeal can also distinguish a few expressions that are different and well-represented in the data-set while KNN failed to learn different features for different expressions.

Finally, since our training data is unbalanced, both of our models have a tendency of predicting Happy, though CNN has a much higher precision in comparison to KNN. In addition, our model is likely to perform poorly with people who wear glasses, since that is also a feature that is underrepresented in our data-set. These shortcomings limit the application of our best CNN model, however, we believe that the CNN structure will perform much better with a larger and more diversified data-set.

Conclusion

Facial expression recognition can be very helpful in people's daily life. For example, if personal assistants such as Siri can recognize the speakers' emotions through their tone and expression from the camera, the technology will be able to better assist the user. As more machine learning models have been designed and implemented in the past few years, FER became a task that can be complete almost to the human level by computers. In this report, we aimed to compare two machine learning models, KNN and CNN, to decide the most suitable one for facial expression recognition by analyzing their test accuracy and confusion matrix.

As a baseline model, KNN provided a simple way to distinguish between different facial expressions and achieved an accuracy of 34.7%. However, even with the 68 facial landmarks extracted, this model is unable to learn the features of each expression due to its low complexity and images are relatively complex data. Since our performance from the model does suggest a positive correlation between facial recognition and key pixels extracted from images, therefore, we may achieve a better result with different detectors and standardizing the pixels to relative pixel positions.

A convolutional neural network by design can extract both high and lower level features from different convolutional layers, leading to higher test accuracy. We first trained CNN with default structure and parameters, Then we gradually improved the performance of the model by modifying the structure, adding dropout and regularization layers, and adjusting hyper-parameters. In the final model, we improved the test accuracy from 53.08% to 65%. During this process, we found that the performance improvement with different hyper-parameters is mostly consistent with previous researches, except in our case, adding batch normalization layers will make the training process unstable, which is caused by an unbalanced training set. Nevertheless, the CNN model can predict emotions with higher accuracy than what the KNN model can predict. This shows that facial expression recognition is a complex problem that a more complex model provides a better hypothesis.

As we directly trained the CNN model on images, but the data-set is smaller, we may experiment with transfer learning to improve the performance of the model. This can be achieved by feeding images into a pre-trained model to extract key features from a portrait image and then train a neural network with the outputs from this pre-trained model. This approach may also reduce training time, thus we may experiment with more combinations of hyper-parameters.

Other than the model designs, we may also improve the performance by properly handling data imbalance. Since SMOTE and NearMiss are both for numerical data, incorporating these did not lead to a satisfactory result. To improve the training result, we may use proper data augmentation techniques for image data such as ImageDataGenerator given more time. This tool is also well-integrated into Keras framework.

```
datagen = ImageDataGenerator( ... )
datagen.fit(X)
CNN = model.fit(datagen.flow(X, y),
                ... )
```

Furthermore, we would like to further the research by identifying emotions as a percentage combination of the 6 basic emotions instead, these are known as the compound emotions, which is much more common in real life than simple basic emotions.

References

Dino, H. I., and Abdulrazzaq, M. B. 2019. Facial expression classification based on svm, knn and mlp classifiers. In *2019 International Conference on Advanced Science and Engineering (ICOASE)*, 70–75.

Ekman, P., and Friesen, W. 1971. Constants across cultures in the face and emotion. 17(2):124–129.

Ioffe, S., and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift.

Kaggle. 2014. Challenges in representation learning: Facial expression recognition challenge.

King, D. E. 2009. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research* 10:1755–1758.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

Kuo, C.; Lai, S.; and Sarkis, M. 2018. A compact deep learning model for robust facial expression recognition. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2202–22028.

Li, S., and Deng, W. 2020. Deep facial expression recognition: A survey. *IEEE Transactions on Affective Computing* 1–1.

Saket S Kulkarni, N. P. R., and Hariharan, S. 2009. Facial expression (mood) recognition from facial images using committee neural networks. *BioMedical Engineering OnLine* 8(16).

Simonyan, K., and Zisserman, A. 2015. Very deep convolutional networks for large-scale image recognition.

Taigman, Y.; Yang, M.; Ranzato, M.; and Wolf, L. 2014. Deepface: Closing the gap to human-level performance in face verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 1701–1708.