Image De-noising, Face Detection, Facial Expression Classification

Cheng Zeng - Chen Chang - Zwe Naing

Abstract

Facial expressions are a crucial component of humans non-verbal communications and social interactions. Having the ability to detect faces and classify these expressions correctly and efficiently can result in a wide variety of useful applications such as psychological analysis, forensics (lie detection), medical diagnosis, advertisements and so on. In this paper, we will implement a system that could detect faces in the image and classify facial expressions associated with them. The project will be developed three parts: image de-noising, face detection and emotion classification. In image de-noising and facial expression classification, we use convolution neural networks both to improve the quality of the image and classify faces into seven different expressions. For face detection, we extract Haar features and use AdaBoosting to determine an image is a face.

Introduction

Facial expressions play a very important role in humans lives. In fact, very young children quickly learn to recognize basic facial expressions. In older people, understanding emotions and corresponding facial expressions is very crucial in social interactions and people try deliberately to improve their skills in understanding these expressions. Human, despite different races and birth places, develop similar facial expressions that directly correlate to the underlying feelings and emotions in the brain. The muscular movements of the face is universal among humans to express emotions. Thus, a system should be able to recognize emotions of different people independent of the background of the person.

A system that is capable of recognizing humans facial expressions can realize a lot of useful applications. In the medical fields, they can be used to perform psychoanalysis and mental health diagnosis. Also, they can be useful in forensics applications such as lie detection. Robotics can also be benefited from these recognition tasks so that robots can respond to humans appropriately based on their perceived emotions. Last but not least, these systems can help marketing and advertisements more targeted to relevant customers based on their reactions to marketing materials. Therefore, facial expression recognition can bring about abundant useful applications in real world.

With the enormous success in recent decades on the tasks such as speech recognition, object detection, machine translation etc, artificial neural networks proved to be the best performers in certain tasks. Especially, convolution neural networks perform better than traditional methods such as Support Vector Machines, Decision Trees, Boosting etc. with manual feature extraction. Due to the promise of convolution neural networks, we adopt these techniques to solve the problem of facial expression recognition.

Background

Convolution neural networks has been the leading algorithms in performing image related tasks such as object detection, image segmentation and classification, and image captioning. CNNs are different from Multi-layer perceptrons (MLP) in such a way that it makes use of locality of images and weight sharing to reduce the total number of parameters. In a convolution layer, image convolutions of certain filter size is applied across the input image using a stride number. Often, padding is used to prevent the image from shrinking continuously. After convolutions, we apply some form of activations such as sigmoid, tanh or relu although relu activation is mostly used in CNNs. After each convolution layer, there is a down sampling layer to decease the size of the input and in turn, reduce the necessary number of parameters. These kinds of convolution layers and max pooling layers are stacked together multiple layers. At the end, there are a few full-connected layers which connect the flatted output of the last convolution layer in a way MLP network does. At the end, there is an output layer which outputs the class probabilities in case of classification task. We might use softmax activation for multi-label classification and sigmoid activation for binary classification.

Related Work

Convolution Neural Networks have considerably improved many computer vision tasks, especially image classification tasks. AlexNet is the first Convolution Neural Network that outperforms tradition vision algorithms and won the ImageNet ILSVRC challenge in 2012. The network is similar

to LeNet architecture but bigger and deeper and have more layers stacked on top of each other. VGGNet came out a few years later in 2014 and is runner up in ImageNet ILSVRC 2014. The unique characteristic of VGGNet is that it has smaller kernel size, more filters and more layers. As a result, the architecture is simple but powerful. The next one is ResNet which contains skip connections unlike traditional CNNs. It wins ILSVRC 2015. All these networks perform superbly in image classification tasks. In our project, our CNN architectures by VGGNet and include small kernel size, many filters and many layers.

Project Description

Image De-noising

In our project, we are trying to increase the success rate of human face emotion recognition. To increase the percentage, we are trying to increase the resolution image, then piping processed image into the emotion detection algorithm. There are many ways in increasing resolution or de-noising image. In this project, we are using the SRCNN (Super Resolution Convolution Neuron Network)(Dong et al. 2014) complete the objective. The structure of SRCNN is shown below:

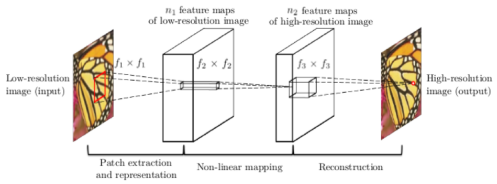


Figure 1: Hello World

Face Detection

Viola-Jones object detection framework will be used in this part. This framework is invented by Paul Viola and Michael Jones in 2003. Its a widely used and fast enough method. The input would be dataset from CBCL Face Database #1 - MIT Center for Biological and Computation Learning and the output would be face images. The general idea would be: First, creating integral images for train sets and use Haar Feature selection to extract the weak features; Second, using AdaBoost algorithm to select the best features and train cascade classifiers; Third, using weak features on test set to get correctness; Fourth, extract faces using cascaded features and compare with CV2 result.

Facial Expression Detection

In this section, we have descried the details of the work that we have done using Convolution Neural Networks to classify facial expressions and present the results and performance of various architectures.

CNN Architecture Details First of all, we present the results of different Convolution Neural Network Architectures to perform facial expression recognition, namely BKVGG8, BKVGG10, BKVGG12 and BKVGG14. (Sang, Van Dat, and Thuan 2017) The comparisons of these architectures

ConvNet Configurations			
BKVGG8 8 layers 3.4 M	BKVGG10 10 layers 4.14 M	BKVGG12 12 layers 4.19 M	BKVGG14 14 layers 4.92 M
Input,layer (48 x 48 x 1)			
Conv3 32	Conv3 32	Conv3 32 Conv3 32	Conv3 32 Conv3 32
Max-pooling layer			
Conv3 64	Conv3 64	Conv3 64 Conv3 64	Conv3 64 Conv3 64
Max-pooling layer			
Conv3 128	Conv3 128 Conv3 128	Conv3 128 Conv3 128	Conv3 128 Conv3 128 Conv3 128
Max-pooling layer			
Conv3 256 Conv3 256	Conv3 256 Conv3 256 Conv3 256	Conv3 256 Conv3 256 Conv3 256	Conv3 256 Conv3 256 Conv3 256 Conv3 256
Fully connected layer - 256 units			
Fully connected layer - 256 units			
Fully connected layer - 7 units			

Table 1: CNN Architectures

are described in the table below. The difference between these architectures is only the number of layers. Each architecture has 8 layers, 10 layers, 12 layers and 14 layers respectively by including convolution layers and fully-connected layers and by excluding max-pooling layers and input layer. BKVGG8 architecture contains 3.4 million parameters, BKVGG10 with 4.14 millions, BKVGG12 with 4.19 millions and BKVGG14 with 4.92 millions respectively. In the table, Conv3 32 stands for a convolution layer with kernel size 3 x 3 and 32 filters while Conv3 128 stands for 128 filters with kernel size 3 x 3. As an example architecture, BKVGG8 network is presented in detail below.

BKVGG8 Architecture In BKVGG8 architecture, the input layer takes batches of grayscale images of size 48 x 48 x 1. All convolution layers use 3x3 kernel size with stride 1 and ReLu activation. We use padding in all convolution layers to keep the dimensions of the input and output. The size of the image is reduced only in maxpooling downsampling layers. However, the number of filters double in each block. The first block has 32 filters, second block 64 filters, third block 128 filters and 256 filters in the last block. Between two blocks, there is a max-pooling layer with 2 x 2 kernel size and stride 2. These blocks are followed by two fully connected layers with 256 units each and finally a SoftMax layer with 7 output, each of which corresponding to the facial expressions that we classify. To avoid overfitting, dropout probability of 0.5 is applied to the first two fully connected layers during training phase. This means there is only 0.5 probability that a particular neuron is activated. This helps reduce the complexity of the model overall and serve as a regularization technique. Finally, SoftMax cross entropy loss is used as the training objective as given by the equa-

tion in which y_i stands for the true label of the data point x_i and p_i stands for the output of SoftMax layer for each class. The neural network is optimized by minimizing this loss. In terms of optimizing, Adam Optimizer is used to train the network. There are 3.4 million parameters in total. (Sang, Van Dat, and Thuan 2017)

$$L = -\sum_{i=1}^{k} y_i \log(p_i)$$

Data Preprocessing Before we train the network, we perform pre-processing to increase performance of the convolution neural network. The preprocessing includes two stages, per-image normalization and pre-pixel normalization.

Per-image normalization - In this stage, first, we compute the mean value of all pixels in a particular image. Then, we abstract the mean value from the pixels value of the image. This essentially sets the mean value of the image to be zero. Afterwards, we also set the standard deviation of the image to 3.125. This is done for all images in the dataset including training, cross validation and test data sets.

Per-pixel normalization - To perform per-pixel normalization, we compute the mean value of a pixel across all images in the dataset. We add up all values of a particular coordinate of all images and then divide by the number of images in the dataset. After that, we set the standard deviation of all pixels to zero. This stage is done separately for training data, cross validation data and test data.

The resulting images after preprocessing stage are shown below. The left images are the original images before preprocessing. The right images are the images after performing preprocessing. We can discover that in original images, there is higher contrast and bigger gaps between pixel values at different locations. However, even after preprocessing, similar image intensities are maintained across the image. In addition, the pixels now have zero mean and zero variance. This helps a lot in the training phase as we elaborated on this later.

Figure 2: Images before and after preprocessing

Data Augmentation Since we have small dataset compared to the size of the network that we are planning to train, data augmentation is used as a way to increase the size of training data and avoid overfitting. As we show in the graphs below, overfitting is a big problem to overcome. Data augmentation helps mitigate this problem by providing different image settings for the network to learn. Data augmentation is performed through successive transformation of an image. First, we perform random horizontal flip with 0.5 probability. There is 0.5 probability that an image will get flipped and 0.5 probability that an image will stay the same as original. Second, we rotate the image randomly from - 45 degrees to 45 degrees. This is done by randomly picking a number between -45 and 45 and rotate by that degree. Finally, we perform random cropping. Before cropping, we resize the image to 54 x 54 using bilinear method and then, take 48 x 48 crops from enlarged image. Performing these three steps provide a wide array of new images. The results of these transformation can be observed below.

Figure 3: Images before and after data augmentation

Traditional Method - SVM Performance The performance of large margin classifiers (Support Vector Machines) is measured in order to compare with deep learning methods (convolution neural networks in this case). SVMs are fairly successful in working with images. For example, the performance of SVMs on canonical MNIST handwritten digits dataset is around 84% which is relatively good. So, we run a SVM on our FERC-2013 training dataset and observe 32% accuracy on test set. In fact, compared to human level performance that we obtained ourselves, this accuracy is comparable. However, feature extraction, careful parameter tuning, and regularization techniques might help improve the performance of the SVMs. For the purpose of gauging against other techniques, this serve as a baseline accuracy so that we can compare with other CNN architectures.

Network Ensembles Among four CNN architectures that we trained, BKVGG12 and BKVGG14 works best at 58% accuracy on the test dataset. These are the results of running on raw original data without any image preprocessing nor data augmentation. We observe doing those two tricks improve the performance of the networks. In addition to preprocessing and data augmentation, performance can be further increased by using ensemble of networks. This idea is inspired from Random Forests algorithm in which multiple decision trees are trained with different subset of features and data. In the testing phase, the prediction is chosen using majority voting. The purpose is to reduce overfitting and to improve generalization. We execute a similar strategy. First, we train BKVGG8, BKVGG10, BKVGG12 and BKVGG14 networks individually until they converge. Then, to predict the label of a particular image, the image is fed through each network separately and use majority voting to determine the predicted class. We observe 8% increase in accuracy from 54% to 62% in BKVGG8 architecture.

Training During training phase, we minimize softmax cross entropy loss using mini-batch Adam optimizer and typical back propagation algorithm. The learning rate is set

at 0.001 which is proved to be best in the Experiments section. Adam parameters are set at 0.9 and 0.999 for Beta 1 value and Beta 2 respectively. All biases are initialized to be zeros while kernel weights are initialized using Xavier initialization, which suggests weights be drawn zero mean and certain variance Gaussian distribution. The variance is given by

$$\sigma^2 = \frac{1}{n_{input}}$$

where n_{input} is the number of units in the input layer. For convolution layers, n_{input} is kernel size x kernel size x number of filters of the previous layer. For fully connected layer, n_{input} is the number of neurons of the previous layer. Dropout probability rate is 0.5 to prevent overfitting. Instead of regular steepest gradient descent, we use minibatch gradient descent to approximate the gradient and speed up the training process. The batch size is 256 and, in each iteration, the training data is shuffled and augmented before feeding into the network. Normally, the training procedure converges around 15 epochs as we show in the plots below.

Testing Our original dataset is divided into three different datasets for training, cross validation and testing datasets. During the training process, training data is used to train the model. Cross validation data for monitoring the performance of each model and to pick the model that performs the best. Finally, test data is used to judge the models overall performance. The accuracy metric is used to rank models. Accuracy is measured by counting the number of images that are correctly classified divided by the number of all images. This can be formulated as below.

$$Score\,(m) = \frac{1}{n}\sum_{i=1}^{n} I\left\{y_i \neq p_i\right\}$$

where m stands for a particular model whose score that we are interested in, n is the number of data points in our test dataset, y_i stands for the true label of image x_i and p_i stands for prediction of image x_i. Finally, the resulting score is multiplied by 100 to present as a percentage value.

Human Level Performance Compared to normal CNN performance on other datasets, the highest accuracy of 64% is relatively low. However, to determine how difficult this task is for humans, we labeled 500 data points from the test set manually and observe that human performance is around 32% which is much worse than the current CNN performance. The reason is because the faces are not obvious but ambiguous facial expressions instead. For example, it is hard to classify between sad and angry, disgust and neutral. This gives a glimpse into the subtlety and difficulty of the problem. The details of misclassifications occurred with CNN is described in the confusion matrix below.

Activations To understand what the network is trying to learn in a particular image, we can visualize the activations in convolution layers. These are the results obtained after performing convolutions on the input image and applying ReLu activations. In BKVGG8 network, the first convolution layer uses same padding which retains the size of the input layer at 48 x 48 x 1. So, the sharpness of the image is still intact. Since the output size quickly shrinks in the second layer and layers afterward and it is difficult to study images of small sizes, we visualize only the output of the first layer below. Since there are 32 filters in the first layer, we obtain 32 images. Below, we can clearly observe that each filter is trying to learn different features of the face, for example, jaw lines, eyes and nose of a person etc. Depending on the facial expression of the input image, different parts of the image get activated. For example, in the happy facial expression of a lady, the contours of the face, the smiling lips and facial landmarks such as eyes and noses get mainly activated. On the other hand, when the input is the angry face of a young man, the network captures the yelling mouth, closing eyes and raised nose as the highest activations. From these, we can assure that the network is learning the correct features of the image.

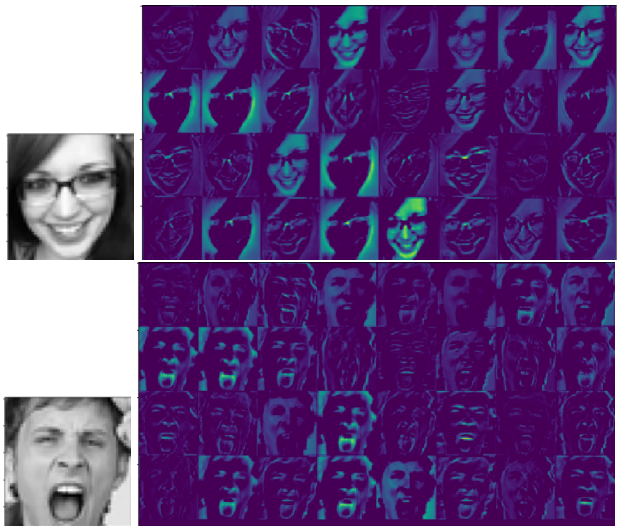


Figure 4: Activations of the corresponding original images in the first layer of BKVGG8 convolution neural network

Experiments

Image De-noising

Before training the network, we need to preprocess the data. The training set for this part is Faces in the Wild, which is from University of Massachusetts, and randomly picking 1000 images from this dataset. We use the same size of training sub-images as in the SRCNN (Dong et al. 2014), $f_sub = 33$. To do this, we used feature_extraction founction from the sklearn library. Patches are assumed to overlap and the image is constructed by filling in the patches from left to right, top to bottom, averaging the overlapping regions, which is very similar to the code provided with the SRCNN (Dong et al. 2014). Since the image from the dataset is already in square shape, we dont have to reshape the image. For each image, we generated 250 sub-images in the size of 3333, which result in total 250000 images in the training set, and this is for the expected data. For the input data, we

used the same set of 250000 images, but we first use gaussian filter to blur images, then resize images to size of 1616, then resize images back to 3333. Both resizing process uses bicubic interpolation algorithm. These steps are to emulate the loss of quality in image resizing using traditional algorithm. After then, is the training part. A batch size of 256 images was used for all experiments. The training parameters for each layer is shown in the table below.

	$n_1 = 128$ $n_2 = 64$			$n_1 = 64$ $n_2 = 32$		
f_1	9	9	9	9	9	9
f_2	1	3	5	1	3	5
f_3	5	5	5	5	5	5

Table 2: Neural Network Configuration

The optimizer using in the network is Adam optimizer, with configuration: $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 0$, $decay = 0$. Loss function is mean squared error, and metrics is $PSNR = 10\log_{10}\left(\frac{R^2}{MSE}\right)$. Each configuration described above was trained on the training set of images for 10 epochs. And we use 10% of images in the tanning set as validation set, and results is shown below.

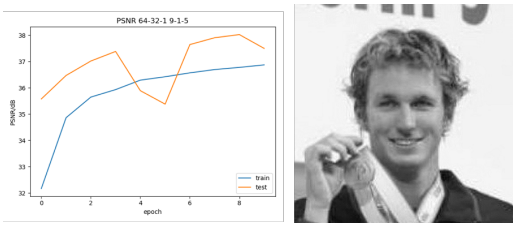


Figure 5: Configuration: 64-32-1 9-1-5

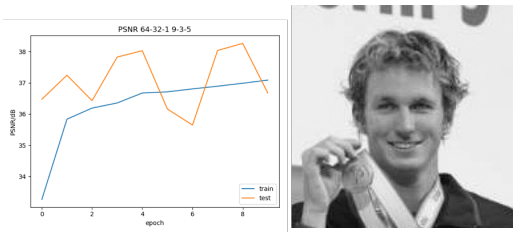


Figure 6: Configuration: 64-32-1 9-3-5

The PSNR from each configuration is very similar, from the PSNR plots, larger number of filters and filter sizes do perfom a little better than other configuration.and also it is very hard to tell which one is better trough the output image by using different configuration. However, it is taking much more times to train the network.

Face Detection

Step1: Load training faces and non-faces and generate integral images; To get Haar features we have to calculate the

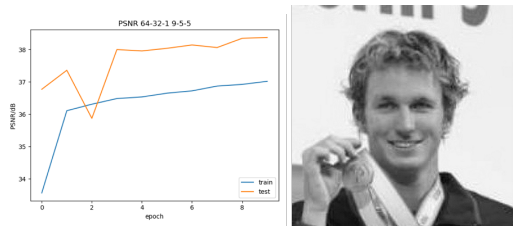


Figure 7: Configuration: 64-32-1 9-5-5

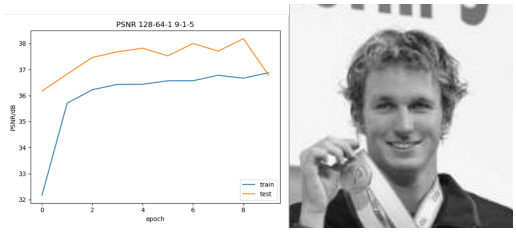


Figure 8: Configuration: 128-64-1 9-1-5

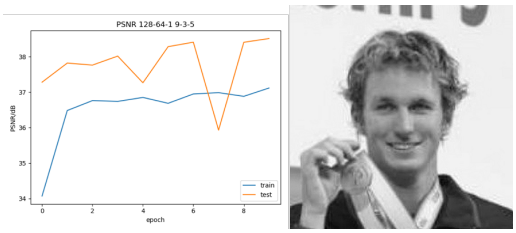


Figure 9: Configuration: 128-64-1 9-3-5

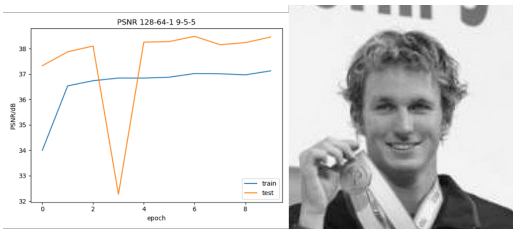


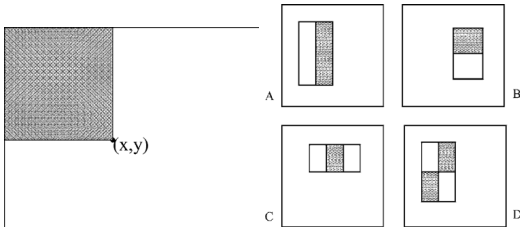
Figure 10: Configuration: 128-64-1 9-5-5

integral image at location x, y which is the sum value for the pixels above and left of x, y. Figure 1.(1)[1] shows the form of integral image. To calculate this, we could use below equation[1]:

$$ii\,(x,y) = \sum_{x^{'}\leq x, y^{'}\leq y} i\left(x^{'}, y^{'}\right)$$

To calculate features we have to use the Haar features. Figure 11b (Viola and Jones 2004) shows the basic four types of Haar features: A & B: two-rectangle; C: three-rectangle; D: four-rectangle feature. The value of each feature is the sum within black rectangles subtracted from the sum in white rectangle.

For 19 x 19 training images, according to the max/min height/width, the total number of features would be signifi-



(a) Integral image (b) Four basic types of Haar features

Figure 11: (a) Integral image; (b) Four basic types of Haar features: A: two vertical rectangle; B: two horizontal rectangle; C: three vertical rectangle; D: A: four rectangle

cantly different. Figure 2 shows the number of features for each heightw̃idth pair.

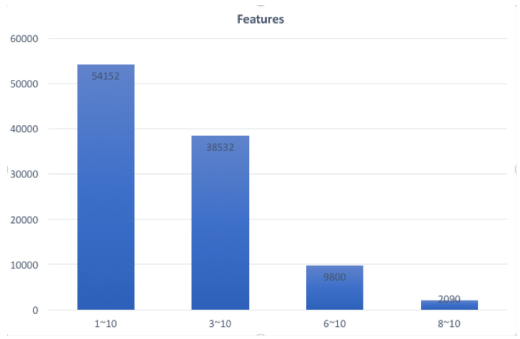


Figure 12: Set max height/width as 10, the total number of features increase as min height/width decrease.

Step2: AdaBoost to get best performance feature Since there are too many features, we are not going to use all of them. To select the best performance features we will use Adaboost. The basic idea is to apply these features to each training image. We set the threshold as 0 and equally assign weights to the training images and features in the beginning. For each feature, it will calculate a score for each image. If the score is below threshold, which is 0, we will assign -1 (non-face) to the image, 1(face) otherwise. But clearly, there will be errors for misclassifications. We select the features with minimum error rate, which means they are the features that best classifies the face and nonface images. Then we reassign weights for the images and features so that the weight of misclassified images will increased. Repeat the steps until we get the required number of features (Mordvintsev and K. 2013).

Step3: Using weak features on test set to detect the faces Since we already get the best performance features, we will apply them on the test sets. Figure 13 shows the relationship between correctness and feature numbers.

Step4: Extract faces using cascaded features and compare with CV2 result To extract faces from image, first we will use the trained best performance features on the input image. If the classification is 1 then we will generate windows for the image and apply trained best performance features on the

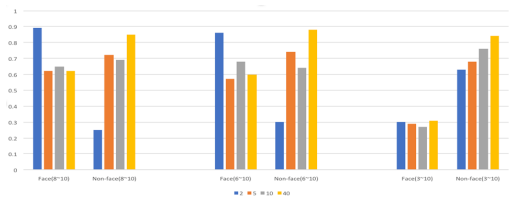


Figure 13: Relationship between correctness and feature numbers for each minm̃ax pair.

window. Check if it is face or not. In an image, most of part is non-face. If we apply all trained best performance features on each window, that will waste a lot of time. To increase the efficiency, instead of using all features we will apply features one by one. If a window fails the first features, discard it. Otherwise, we continue applying the following features until it is classified as face. Then we will merge the overlap face windows and out put the image. To make extraction more precise, we set the iteration time as 10 so that the extract image could have smaller size and more concentrate on face part. Figure 14 shows the result.

(a) Original image (b) Extract part

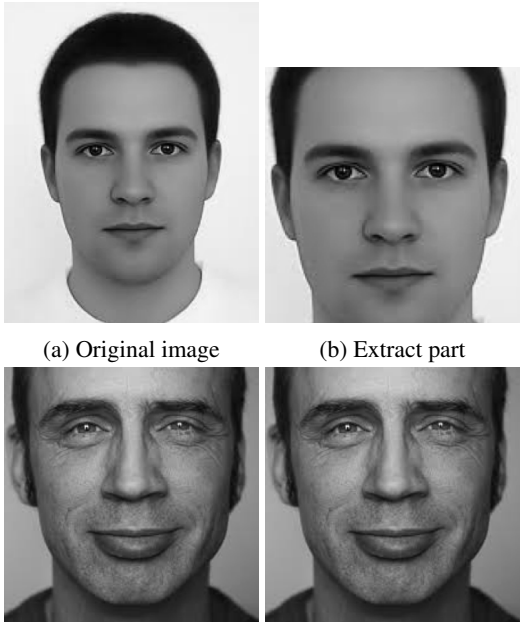
(c) Original image (d) Extract part

Figure 14: Face extraction. (a), (c) are the original images; (b)(d) are the extract face part

Finally we will use OpenCV to process the same input images and get face part and compare them with out result. Figure 15 shows the OpenCV result.

Discussion

1) Relationship between correctness and feature numbers.



(a) Original image (b) Extract part

(c) Original image (d) Extract part

Figure 15: Face extraction. (a), (c) are the original images; (b)(d) are the extract face part

From figure 13 we can find that, as feature numbers increase, the face-detection accuracy decrease. Thats because we select features according to their errors, as the number of features increase, more not accurate features were selected as classifiers which introduce more noise/error. Also, set max height/width as 10, as min height/width decrease the face-detection accuracy decrease. Thats because as rectangle size increase, the Haar feature tends to represent larger areas value, no longer represents the value for a small local area, which introduce the noise. For non-face accuracy, since there is more noise been introduced as min height/width decrease or feature numbers increase, more and more images are counted as non-face.

2) Face extraction From figure 14 and figure 15 we can find that our implementation of Viola-Jones frame work can extract the face part of image. Comparing with OpenCV, our implementation works better in some images. For example, the (c)(d) in figure 15, OpenCV cannot recognize and extract face if the face part is not shown fully in image but our implementation could. But the result of our implementation is not as accurate as OpenCVs. Our implementation only recognizes 2 out of 11 input images as face images but OpenCV could recognize 7 out of 11. Also, our output face image is not fully focus on face part, the left-up area contains a lot of non-face information. Thats because when we run the extraction process, we always start from left-up to right bottom will introduce some errors. In future work, we will try to optimize the merge window process so that the output image could more concentrate on face part. Also, to increase the accuracy, the training set should be change. Since the training images are 19x19 and out input image are 400x400 (or bigger), the classifiers from training images may not fully cover the features, thats also be the reason why in test sets our accuracy is high but in real implementation there are 2 out 11 been recognized.

Facial Expression Classification

All our experiments used the FERC 2013 dataset which is provided on Kaggle Facial Expression Recognition Challenge. The dataset contains images of the wild faces of several people which means they are not posed. Each image are labeled with seven classes: angry, disgust, sad, happy, neutral, surprise and fear.

Data Exploration The whole FERC-2013 dataset contains 35887 images of size 48 x 48. Images are grayscale images of different orientations, brightness and zoom. Kaggle already splits the original dataset into three categories, training, private test set and public test sets. In our experiments, private test set is used for cross validation purposes and public test set is used as a test data set to determine an unbiased performance of the model. Three datasets include 84%, 8% and 8% of the original dataset respectively. The data is split in such a way so that the original distribution is maintained. However, the number of data points in each class are not uniformly distributed. As we can see below, the number of disgust faces are the fewest while the number of happy faces are the most. Other classes are fairly well distributed with average 4500 data points in each class. This is a reasonable split since we observe later that the performance on cross validation dataset is very predictive of the test set performance.

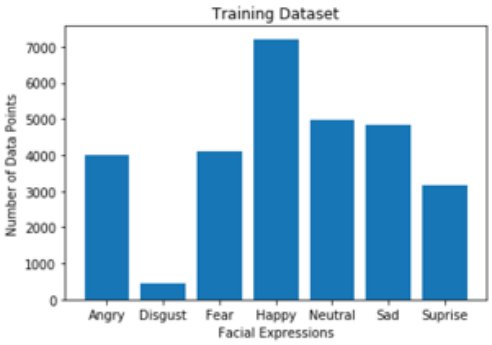


Figure 16: Distribution of Train Dataset

Results

In this section, we present the results obtained by training each CNN architecture for 40 epochs. As we can observe in the graphs below, the loss on the training data decreases overtime close to zero. However, the loss on the cross-validation data increases overtime. This occurs due to overfitting. All architectures converge around 25 epochs. The learning rate is set at 0.001 for Adam optimizer during the whole training period. Beta 1 value and Beta 2 value are

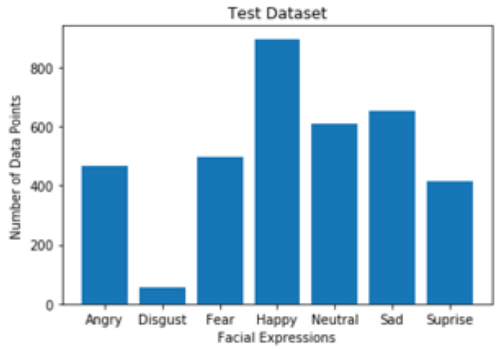


Figure 17: Distribution of Test Dataset

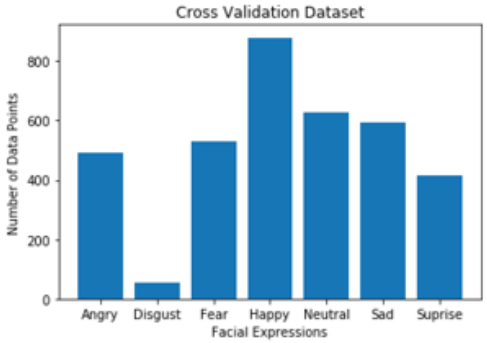


Figure 18: Distribution of Cross Validation Dataset

set at 0.9 and 0.999 respectively. There is no learning rate decay. From these graphs, we can discover that the behaviors of all networks are very similar. All of them are prone to overfitting, even the smallest BKVGG8 architecture.

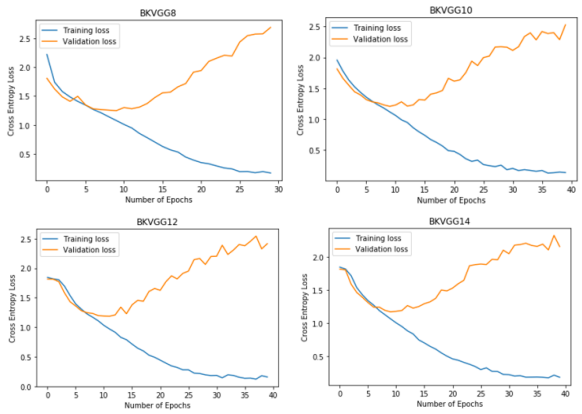


Figure 19: Cross entropy loss of CNN architectures during training

In terms of accuracy, we observe that bigger networks perform better in general. BKVGG8 architecture has 54% accuracy, BKVGG10 with 56% and both BKVGG12 and BKVGG14 with 58% accuracy. As we can observe in the plots below, the algorithm completely converges around 15 epochs.

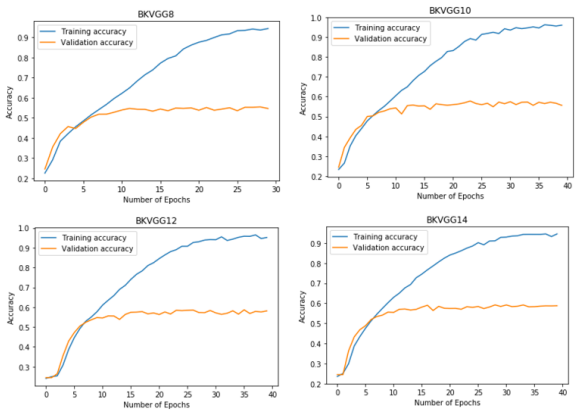


Figure 20: Accuracy of CNN architectures during training

As we can observe clearly in the graphs above, the algorithm has obvious overfitting problems. While the training accuracy keeps increasing close to 100% while the validation accuracy become stagnant around 58%. However, we notice that the accuracy does not decrease as it overfits the training data. So, this implies that regularization technique such as early stopping will not be fruitful.

Architecture	Validation Dataset	Test Dataset
BKVGG8	55%	54%
BKVGG10	56%	56%
BKVGG12	58%	58%
BKVGG14	58%	58%

Table 3: Summary of performances of CNN architectures without preprocessing and data augmentation

Performance Boosting with Preprocessing

Preprocessing by performing per-image normalization and per-pixel normalization is one of many ways to improve performance we discussed here. We describe the performance increase using BKVGG8 architecture. We picked this architecture since this network contains the fewest number of parameters and consequently, less computation power. Moreover, the performance difference between BKVGG8 and other networks are small. So, the observations will be generalizable to other networks too.

We can observe the performance boosting above. The left graph shows the results of training BKVGG8 network without using preprocessing while the right graph describes the results using preprocessed data. With preprocessing, we observe 4% increase in accuracy from 54% to 58%. Moreover, the algorithm converges faster around 6 epochs compared to 12 epochs without preprocessing. Faster convergence happens since the data becomes more centered at zero mean and zero variance after normalization process. This encourages the network learns faster than usual.

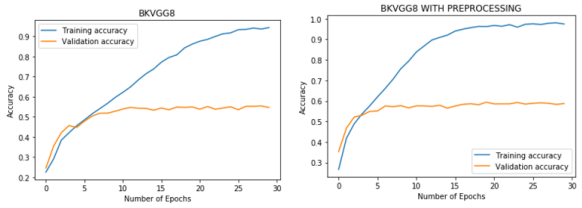


Figure 21: Accuracy of BKVGG8 architecture with and without preprocessing

Architecture	Validation Dataset	Test Dataset
BKVGG8	59%	58%
BKVGG10	60%	59%
BKVGG12	60%	59%
BKVGG14	59%	59%

Table 4: Summary of performances of CNN architectures with preprocessing but not with data augmentation

Performance Boosting with Image Augmentation

Performance can further be improved by performing image augmentation. This will mitigate overfitting problem again. The accuracy improved another 5% from 58% to 63% on the test set. We can also see that there is less overfitting in the comparisons below. In the left graph, there is a huge gap between train accuracy and cross validation accuracy which implies overfitting while the right graph has smaller gap, thus less variance. However, the algorithm takes longer to converge due to data augmentation since augmentation makes the network harder to memorize the training data. Furthermore, we can see that the trend is more noisy with data augmentation. It takes around 100 epochs to reach the plateau.

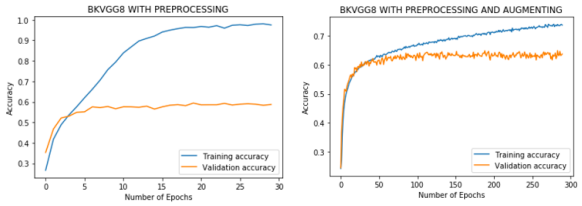


Figure 22: Accuracy of BKVGG8 architecture with and without data augmentation

Architecture	Validation Dataset	Test Dataset
BKVGG8	64%	63%
BKVGG12	63%	63%

Table 5: Summary of performances of CNN architectures with preprocessing and data augmentation

Confusion Matrix

The following table describes what percent of images in a certain class is mislabeled as another class. Each row corresponds to the true labels of the images while each column corresponds to algorithm predictions. From this table, we can observe how much percent of algorithm predictions belong to other class except the correct one. To make the statistics obvious, the true positives are highlighted green while high false positives are highlighted red. For example, 82% of surprised labels are classified correctly. 18% of disgust images are classified as angry. 10% of fear labels are classified as angry. A high percentage of fear images are classified as sad. In fact, these inaccuracies arise from ambiguous facial expressions that we observe in the dataset. The highest accuracies is for happy and surprised classes since we have enough data of happy images. Moreover, they are very disguisable facial expressions compared to other expressions. It comes as a surprise that the accuracy on disgust class is high because the number of training points in that class is very low as we mentioned above.

	Angry	Disgust	Fear	Happy	Sad	Surprised	Neutral
Angry	63%	0%	10%	2%	15%	1%	8%
Disgust	18%	69%	0%	4%	4%	4%	1%
Fear	11%	0%	52%	2%	20%	7%	8%
Happy	1%	0%	2%	90%	3%	1%	3%
Sad	8%	0%	9%	4%	62%	1%	17%
Surprised	1%	0%	5%	5%	2%	82%	3%
Neutral	3%	0%	3%	3%	15%	1%	73%

Figure 23: Confusion Matrix

Training Problems and Solutions

Vanishing Gradients - One of the problems that we encountered during the training phase of the CNNs is that the activations of the convolution layers become extremely close to zeros. Here we describe the results from training BKVGG8 architecture. In the graphs below, we show the distributions of activations which resemble gamma distributions centered at 0. The activation values are plotted on the x-axis, the number of neurons that have a particular value is on y-axis and the number of steps is on the z-axis. We can see that as the training continues and steps increase, the values become closer to zero. This manifest itself as a problem since tiny activations essentially corresponds to tiny gradient values during backpropagation. As a result, the updates to the weights in convolution kernels would be negligible. This will make the training process longer than necessary or even worse, prevent the network to converge at all. This problem is referred to as vanishing gradients.

Solution : Batch Normalization - The solution to overcome vanishing gradient problem is to normalize all activation outputs after each convolution layers to zero mean and standard deviation one. This will make the activation values to have a Gaussian distribution as shown below. We can also see that the activation values are no longer close to zero. As a result, this will speed up the training process and finally reach the convergence.

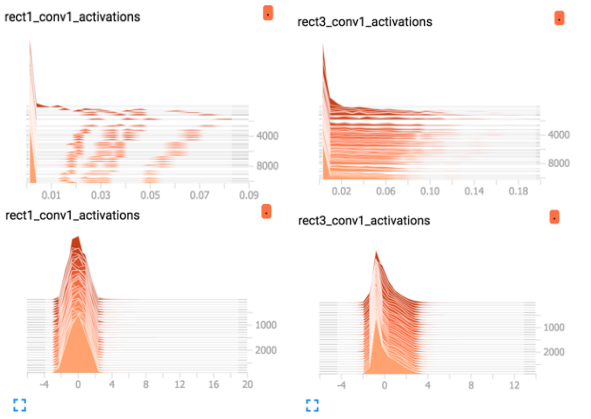


Figure 24: Example activation values with and without batch normalization

Proper Weight Initialization - The other problem that we faced is the proper weight and bias initialization. Typically, kernel weights and biases are initialized randomly so that different filters learn different aspects of the image. However, the disadvantage of initializing this way is it is very likely that the starting weights might quickly lead the network to be stuck in the local optima. One of this problem can be observed below. In the left graph, we show the changes to the biases of the second convolution layer overtime across 3000 steps. We can see that the biases change for 2000 steps and then stop changing at all. Similarly, on the right graph, the kernel weights do not change at all after a few steps. These two observations hinted that the optimization is stuck in local minima. Otherwise, if the network is training overtime, we should the weights should move.

Solution : Xavier Normalization - Instead of initializing all weights randomly, we can use Xavier initialization by initialization weights from a Gaussian distribution as described above. Moreover, the biases are initialized to zeros instead of random. The effect of this initialization procedure can be observed below. On the left graph, the biases are changing quite dramatically overtime during 1000 steps. On the other hand, on the right graph, the weights distribution change overtime. This can be directly compared with random initialization where the changes are less obvious. As a result, this initialization leads our network to converge.

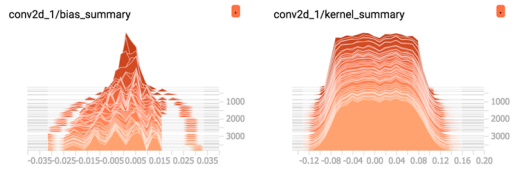


Figure 25: Weight updates with random initialization

Conclusion

From the experiments above, and even after using SRCNN to preprocess images will not have significant result in improving the accuracy of the facial expression classification, which may due to that SRCNN is not suitable for this kind of process. Even it does increasing the image quality comparing to traditional algorithms. But this doesnt mean that increasing image resolution itself is not helpful, In the future, we would like to try other denoising/scaling neural networks such as EDSR(Lim et al. 2017). In face detection, Our implementation of Viola-Jones framework can detect faces better than OpenCV although the faces extracted using our framework contain more non-face information. In future work, we will try to optimize the merge window process so that the output image could more concentrate on face part. Also, to increase the accuracy, the training set should be change. In terms of facial expression classification, we explore different CNN architectures inspired by VGG architecture to tackle the problem of facial expression classification. We discover that the bigger networks perform better compared to smaller networks, but the different is insignificant. Performing preprocessing and data augmentation can help prevent over fitting and increase generalization. Moreover, we realize that the weights initialization is as important as batch normalization for the network to learn reasonably well. Over fitting is the largest problem we need to solve. In future work, more recent regularization techniques need to be employed to increase the accuracy.

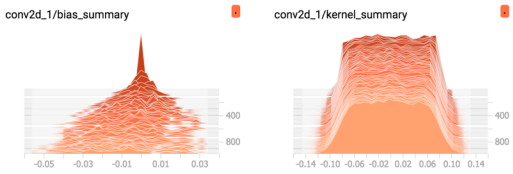


Figure 26: Weight updates with Xavier initialization

References

Dong, C.; Change Loy, C.; He, K.; and Tang, X. 2014. Image super-resolution using deep convolutional networks. *ArXiv e-prints*.

Lim, B.; Son, S.; Kim, H.; Nah, S.; and Lee, K. M. 2017. Enhanced deep residual networks for single image super-resolution. *ArXiv e-prints*.

Mordvintsev, A., and K., A. 2013. Face detection using haar cascades.

Sang, D. V.; Van Dat, N.; and Thuan, D. P. 2017. Facial expression recognition using deep convolutional neural networks. 130–135.

Viola, P., and Jones, M. 2004. Robust real-time face detection. *International Journal of Computer Vision* 57(2):137–154.