Facial expression recognition using Convolutional Neural Networks

Kothapalli, Anirudh(20093112) Marulasidda Swamy K S (20116267)

anirudh.matreya@gmail.com marulasiddas@gmail.com

Naveenkumar A R (20098468)

naveenkumar.adi.raman@umontreal.ca

Abstract

Convolutional Neural Networks (CNNs) are a category of Neural Networks that have proven very effective in areas such as image recognition and classification. CNN have been successful in identifying faces, objects and traffic signs apart from powering vision in robots and self-driving cars. We explore how the same can be used to explore human emotions from image data.

1 Introduction

The goal of the project was to implement a classification algorithm that can automatically predict the categories of human expressions given human facial images. The ck++ dataset consists of total 648 images and each image is labelled with one of the eight basic expressions("neutral", "anger", "contempt", "disgust", "fear", "happy", "sadness", "surprise"). As the first step we have prepared the data from ck++ data set which can be directly fed to any machine learning algorithm.

We have done the data preparation in three steps:
1) Copy images to respective folder as per their labelled expression.
2) Detect human face in each image using cascade classifier and cut the image to the size of human face detected.
3) Divide the data in to train and test data of size 521 and 127 respectively. Each image is of size 350px x 350px and these images have been modified by converting in to grayscale images.

We tried with the same CNN architecture with which we got good results in Kaggle competition this method is also called as Transfer Learning, and we were able to get approximately 60% test accuracy. The CNN model with current architecture performed better than any other architectures with the test accuracy of 88.5% as test accuracy.

1.1 Feature Design

- Each element of the input data was reshaped to 48 x 48 arrays Reason : Spatial relationship between the pixel values is not disturbed after resizing. Convolution is expensive with high dimension data.
- Conversion to Python Imaging Library (PIL / Pillow) format Reason : To transform the data to be interpreted as images for the CNN training
- Data augmentation was done with both the operations of Random horizontal flipping as well as Random vertical flipping
- Reason : To enrich the dataset with possible variants of the input images in order to allow the CNN filters to be robust enough to learn and predict the labels even if the images are slightly rotated or flipped.

CNN models are translation invariant but are prone to variation in rotation and Scaling. So the networks need to

- Normalization of image pixel intensity values : The pixel intensity values of the images are maintained between 0 to 1.

  Reason : To transform the input pixel values to a reasonable range to help the neural network to learn faster since gradients act uniformly.

- As a CNN model is being used, the features are not hand-designed and the features with their hierarchies are learned by the filters of the convolutional layers of the CNN during the training process.

Problem Statement: The face is an important information source for communication and interaction. How can CNN be used to provide a robust facial feature tracking method based on active shape models and develop models for facial recognition tasks.

Using the Cohn-Kanade Dataset as the base dataset how can we apply CNN to get an accurate classification measure over the expressions of the image.

2 Data Preparation

To be able to recognize emotions on images we will use OpenCV. OpenCV has a few 'facerecognizer' classes that we can also use for emotion recognition. They use different techniques.

We make two datasets source_emotion and source_images. We input the folders containing the images in a folder called source_images. Along with that folder another folder "sorted_set", to house our sorted emotion images.

Within this folder, create folders for the emotion labels ("neutral", "anger", etc.). for the various images to be sorted. Each image sequence consists of the forming of an emotional expression, starting with a neutral face and ending with the emotion.

So, from each image sequence we want to extract two images; one neutral (the first image) and one with an emotional expression (the last image). The classifier will work best if the training and classification images are all of the same size and have only a face on them so as to prevent any form of clutter and noise. Even if we wanted to avoid that we could use a HAAR filter from OpenCV to automate face finding.

Because of the probable reason that most participants may have had multiple expressions along with the neutral image we may have to clean up the "neutral" folder. This could bias the classifier accuracy unfairly, it may recognize the same person on another picture or be triggered by other characteristics rather than the emotion displayed.

Finally we actually need to teach the classifier to work on what certain emotions look like. The usual approach is to split the complete dataset into a training set and a classification set.

We use the training set to teach the classifier to recognize the to-be-predicted labels, and use the classification set to estimate the classifier performance. The other main reason for splitting the dataset is to get an idea how well the classifier generalizes its recognition capability to never seen before and unknown data.

Overall as the classifier runs over the dataset the possibility that the images can be wrongly classified so we make a separate folder to house those images and name it "difficult".

3 Architecture Of CNN

The Convolutional Neural Network is similar in architecture to the original LeNet and classifies an input image into four categories: dog, cat, boat or bird (the original LeNet was used mainly for character recognition tasks).

There are four main operations in processing of CNN: 1. Convolution 2. Non Linearity (ReLU) 3. Pooling or Sub Sampling 4. Classification (Fully Connected Layer).

Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data.

In CNN terminology, the $3 * 3$ matrix is called a 'filter' or 'kernel' or 'feature detector' and the matrix formed by sliding the filter over the image and computing the dot product is called the 'Convolved Feature' or 'Activation Map' or the 'Feature Map'. It is important to note that filters acts as feature detectors from the original input image.

A CNN learns the values of these filters on its own during the training process (although we still need to specify parameters such as number of filters, filter size, architecture of the network etc. before the training process). The more number of filters we have, the more image features get extracted and the better our network becomes at recognizing patterns in unseen images.

3.1 Depth:

Depth corresponds to the number of filters we use for the convolution operation. Stride: Stride is the number of pixels by which we slide our filter matrix over the input matrix.

3.1.1 Zero-padding:

A nice feature of zero padding is that it allows us to control the size of the feature maps. Adding zero-padding is also called wide convolution, and not using zero-padding would be a narrow convolution.

3.1.2 Non Linearity (ReLU):

The purpose of ReLU is to introduce non-linearity in our ConvNet, since most of the real-world data we would want our ConvNet to learn would be non-linear (Convolution is a linear operation - element wise matrix multiplication and addition, so we account for non-linearity by introducing a non-linear function like ReLU).

3.1.3 Spatial Pooling :

: Spatial Pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum The function of Pooling is to progressively reduce the spatial size of the input representation.

3.1.4 Flow Of CNN

The term "Fully Connected" implies that every neuron in the previous layer is connected to every neuron on the next layer.
The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset.

4 Facial Expression Recognition Using Convolutional Neural Networks:

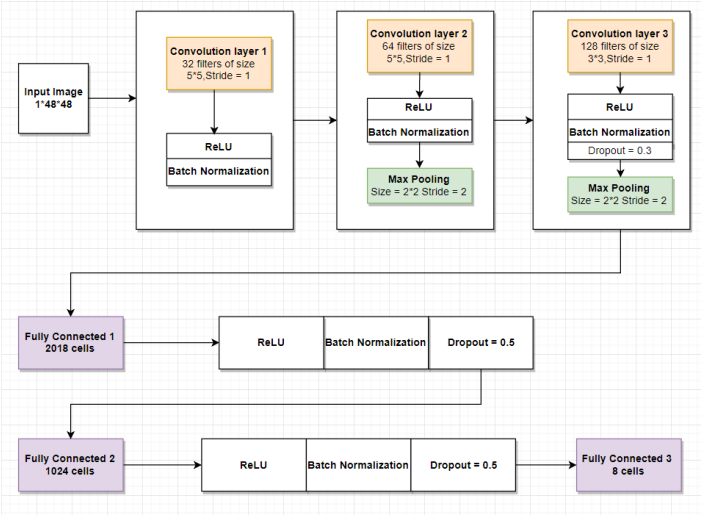
Facial expressions convey rich emotional information in human communication and interpersonal relations.

Automatic facial expression recognition is a challenging problem and has many applications in multiple disciplines.

It has been proven that convolutional neural networks have advantages in object classification, especially for high-dimensional objects like faces. In this project.

CNN models are trained on the Extended Cohn-Kanade (CK+) Dataset to recognize 6 basic emotions and neutral faces. We experimented different CNN architectures and methods such as dropout and batch normalization.

CK+ Dataset $\rightarrow CNNModel \rightarrow Image \rightarrow Result$



The above mathematical expression shows the method in which the CNN Image Classifier works in the classification of human emotions.
The CK+ dataset is first preprocessed into the various folders for each respective emotions and then the same is sent in the flow of data for the CNN Model which then iterates over the image and then gives the result facial expression.

4.1 Algorithm:

Given below are overviews of the learning algorithm used without going into too much detail.

4.1.1 Convolutional Neural Networks (CNN)

We have taken the methods of horizontal and vertical flip to get the augmented data for application in the training and validation set. Other methods such as random cut could not be applicable since the entire of the CK+ dataset is already normalized and resized to 48∗48 matrix.

Then we take the various epochs over 25,50,75,100. And run the same training data over the various epochs to get an idea over how much the accuracy of the image being classified is varying with the epochs.

- Given a set of images, design a sequence of convolution and pooling layers with respective activations.
- For each set of input(s), perform forward propagation performing the convolution and pooling operations to find the output.
- Compute the cost using the computed output and actual target of the input data and back-propagate the gradients to update the weights.

4.1.2 Network Architecture

Rectified Linear Unit: ReLU activations are the simplest non-linear activation function you can use being computationally less expensive compared to other activation functions.

Softmax: Softmax is an activation function. It is frequently used in classifications. Softmax output is large if the score (input called logit) is large. Its output is small if the score is small. Softmax has the property of summation of all output will be equal to 1.0 always.

Cross Entropy Loss: Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label.

Stochastic Gradient Descent(SGD) [3]: The weights are optimized using what we call an optimization algorithm. The optimization process depends on the chosen optimization algorithm.In

our model we have chosen to use the SGD algorithm.
so we multiply the gradient with the learning rate, and we subtract this product from the weight, which will give us the new updated value for this weight.

new weight = old weight $-$ (learning rate $*$ gradient)

this same process is going to happen with each of the weights in the model each time data passes through it. The only difference is that when the gradient of the loss function is computed, the value for the gradient is going to be different for each weight because the gradient is being calculated with respect to each weight. It's learning what values to assign to each weight based on how those incremental changes are affecting the loss function. As the weights change, the network is getting smarter in terms of accurately mapping inputs to the correct output. As this process continues over multiple epochs the loss continuously reduces.

Adagrad (Gradient Descent) [1]: It adapts the learning rate to the parameters, performing smaller updates (i.e. low learning rates) for parameters associated with frequently occurring features, and larger updates (i.e. high learning rates) for parameters associated with infrequent features. For our case, adagrad and SGD with different gradients behaves the same resulting in evidence that adagrad changes learning rate as required. And the results when we used adagrad also have been extrapolated.

4.1.3 Other Techniques Implemented To Assist CNN:

The following operations are performed on the actual data along with the main CNN namely.

Batch Normalization [2]**:** We intend to apply Batch Normalization due to the case that the larger data points in these non-normalized data sets can cause instability in neural networks because the relatively large inputs can cascade down through the layers in the network, which may cause imbalanced gradients, which may therefore cause the famous exploding gradient problem.

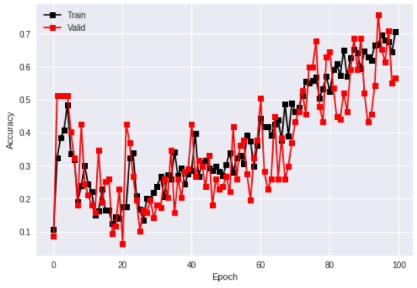
When applying batch norm to a layer, the first thing batch norm does is normalize the output from the activation function. After normalizing the output from the activation function, batch norm multiplies this normalized output by some arbitrary parameter and then adds another arbitrary parameter to this resulting product.

With batch norm, we can normalize the output data from the activation functions for individual layers within our model as well. batches in epoch = training set size / batch_size

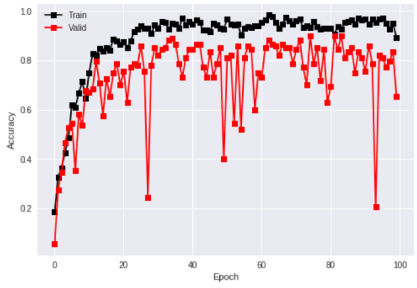
the larger the batch size, the quicker our model will complete each epoch during training. The trade-off, however, is that even if our machine can handle very large batches, the quality of the model may degrade as we set our batch larger and may ultimately cause the model to be unable to generalize well on data it hasn't seen before. In general, the batch size is another one of the hyperparameters that we must test and tune based on how our specific model is performing during training.

In our training model we have decided to keep the size of the batches at 30.

Max Pooling: Max pooling is a type of operation that is typically added to CNNs following individual convolutional layers as it helps in reducing the dimensionality of images by reducing the number of pixels in the output from the previous convolutional layer. There are a couple of reasons why adding max pooling to our network may be helpful.
Reducing computational load: Since max pooling is reducing the resolution of the given output of a convolutional layer, the network will be looking at larger areas of the image at a time going forward, which reduces the amount of parameters in the network and consequently reduces computational load.
Reducing Overfitting Additionally, max pooling may also help to reduce overfitting. The intuition for why max pooling works is that, for a particular image, our network will be looking to extract some particular features.
Dropout [4]**:** Dropout is a form of regularization (it constrains network adaptation to the data at training time, to avoid it becoming too smart in learning the input data; it thus helps to avoid overfitting Dropout, then, can be seen as an extreme version of bagging. At each training step in a mini-batch, the dropout procedure creates a different network by randomly removing some units.
Zero Padding: When a filter convolves a given input channel which in turn results in an output channel, which is a matrix of pixels with the values that were computed during the convolutions that occurred on the input channel.The dimensions of our image are reduced.

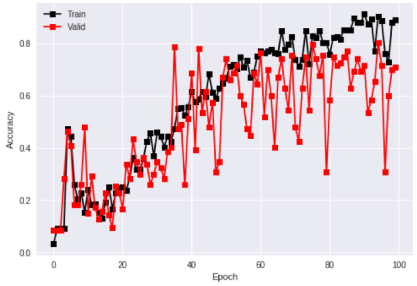


(a) 2a

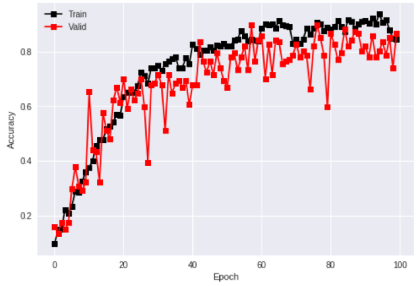


(b) 2b

Figure 1: plots of....



(a) 3a



(b) 3b

Figure 2: plots of....

5 Figures and Results

Figure 2a : Without batch normalization but with dropout
Figure 2b : With batch normalization but without dropout
Figure 3a : Without batch normalization and without dropout
Figure 3b : With weighted loss
Figure 4 : With final architecture

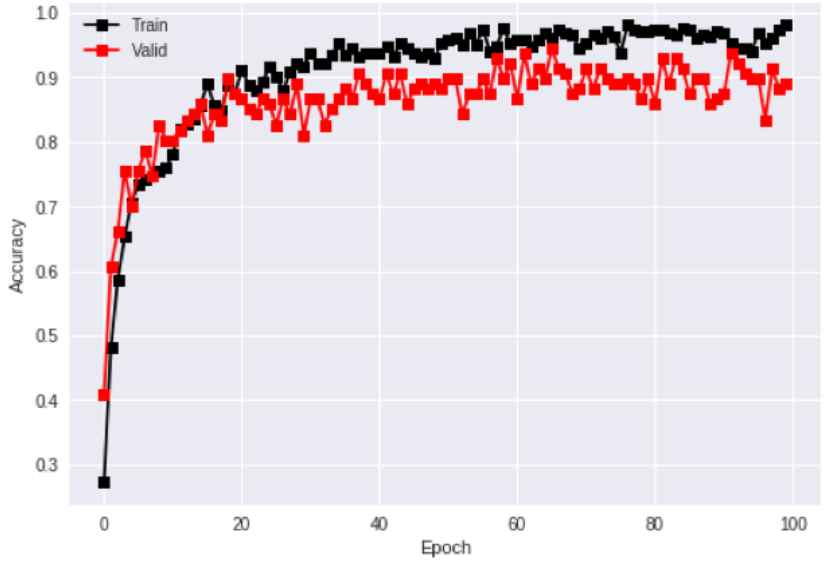


Figure 3

Table 1: Variation By Epochs

Methods	Final Test Accuracy
With SGD and fixed learning rate	83.464
SGD with changing learning rate after few epochs	88.188
With Adagrad	88.976

6 Hyperparameters

The following hyperparameters were tuned using either cross-validation, random search or inspiration from the existing benchmarks of the Google QuickDraw dataset.

- Number or layers = 3 (Conv + Pool) layers + 3 fully-connected layers
- of weights / connections / filter sizes as indicated in Figure 3. Conv layers : 32 (5 x 5) filters -> 64 (5 x 5) filters -> 128 (3 x 3) filters
- Pool layers : Max Pooling with 2 x 2 filters with stride 2 (Common choice for natural images)
- of fully-connected neurons = dimension of the final output from Conv + Pool layer
- Batch size = 30 (Set by trial-and-error performances)
- Learning rate = 0.01 (Tuned randomly with multiple trials)
- Momentum = 0.9 (Tuned randomly for fastest convergence)

7 Results

We present our results with method that gave us the best performance.

- Model CNN
- Train accuracy using CNN = 96.73704414587331
- Test accuracy using CNN = 92.91338582677166

7.1 Why the method pursued performs better than the basic CNN

Higher Capacity: The CNN used for our model has a higher capacity than normal CNN.It is evident from the learning curves of different CNN architecture that it overfits the data without regularization.The final CNN has multiple neurons and layers and had much higher capacity than the basic CNN.
Representational Power: The CNN model learns the representations of the image over the layers in the training procedure. The initial layers learn lower level features like edges, corner and the higher layers compose these to identify contours, boundaries, segments and objects.As the input data samples are images, CNN is a good choice for a model.
Regularization: In the CNN model implemented, regularization was achieved by using dropouts and batch normalization. This ensured that the model did not end fitting the noise more than the data.

8 Acknowledgments

The implementation of CNN model was done by Marulasidda Swamy.The data analysis work and report preparation was done by Anirudh Kothapalli.The literature survey and some part of report preparation was handled by Naveenkumar Adi Raman In all, it was a team effort.

We are grateful to Google Inc. for their generous provision of access to GPUs on the Google Colab platform. It immensely helped us to improve our models with lesser computation time.

References

[1] Duchi, J., Hazan, E., Singer, Y. (2011). *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research, 12, 2121–2159.*

[2] Bower, J.M. & Beeman, D. (1995) *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*

[3] J. Kiefer and J. Wolfowitz *Stochastic Estimation of the Maximum of a Regression Function Ann. Math. Statist. Volume 23, Number 3 (1952), 462-466.*

[4] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov; 15(Jun):19291958, 2014.*Dropout: A Simple Way to Prevent Neural Networks from Overfitting*