Human Facial Expression Classifier

(Assignment 1 – CNN based)

Praveen Sridhar
2018A7PS0166G
f20180166@goa.bits-pilani.ac.in

Preeyam Sahu
2018A7PS0191G
f20180191@goa.bits-pilani.ac.in

Shreya Mallampalli
2017B4A70313G
f20170313@goa.bits-pilani.ac.in

Abstract*— *As an exploration of the Convolutional Neural Network, we worked to build a model that performs decently well on the dataset of images to classify facial expressions into one of seven different emotions. We experimented with various models and architectures and reported our findings and our analysis cohesively. Our model reached an accuracy of 96% on the training set and 67% on the validation set after a lot of tweaking and modifying and will be presented here shortly.

Keywords—Convolutional Neural Networks, CNNs, Neural Networks, Facial Expression Classifier

1. Introduction

A Convolutional Neural Network (CNN) is a type of deep neural network which can take in an input image, assign learnable weights and biases to various aspects of the image and be able to differentiate one from the other. The images are prepared and processed and given to the algorithm, which then classifies the image after learning the necessary parameters. Such neural networks have multiple layers called convolutional layers which involve sliding a filter of a specified dimension across the entire image which performs an operation called a convolution and outputs a feature map. Multiple such layers are stacked together and flattened at the end into a dense layer which is used to make a classification.

1.1. The Problem Statement:

To develop a CNN for classifying human facial expressions. As a part of this assignment, we would be required to prepare the provided dataset; design a CNN architecture and tune and train it to get the best prediction; and present the final model and weights obtained.

Over the course of the assignment, our aim was to get familiarized with CNNs and their working, experiment with various architectures, and develop a model with the best possible results for the problem given to us.

2. Solution

2.1. Preparation of the Dataset

The data given to us was structured in following way: Two columns in a csv file; a column ‘emotions’ of numbers ranging 0 to 6 and a column ‘pixels’ which was provided as a string containing 48*48 numbers that represent the pixel values of a single-channel greyscale image. (Fig. 1)

The second column into was converted into a numpy array and then reshaped become a matrix of dimensions 48x48. The first column such that each entry represented a label and was a single integer.

Data augmentation was then used. To this end, the ImageGenerator class was used where the images were rescaled by a factor of 1/255. After this, additional values such as width shift, height shift, zoom range, rotation range, shear range, and horizontal flips were also added within reasonable limits.

The normalizer is saved as function in the python notebook attached and needs to be run on the test data before testing the model.

Fig. 1. A small randomly sampled subset of the dataset

2.2. The Architecture

The architecture we have used contains 15 layers (not including dropout, batch norm and flatten), of which 5 were convolution and max pooling layers and 5 were fully connected layers. The model and its layers have been summarized in the table given below (Table 1).

This model was inspired by the architecture of VGG 16 and AlexNet. Similar to the two, our model has convolution layers with increasing number of filters followed by max pooling layers.

To improve performance on the validation set and avoid overfitting, dropout regularization was added. 6 batch normalization layers were added to improve training time and performance.

The activation function used in the model was the ReLU function.

2.3. Training the Model

The model was trained using the Adam optimizer, with a learning rate of 0.01 using sparse categorical cross entropy as the loss function. A batch size of 256 was used for the training and validation generators and was trained for 450 epochs at around 12 second per epoch on Google Colab with GPU as the hardware accelerator.

The model and the weights were stored in a .h5 file after training and evaluation was complete.

TABLE 1. Model Summary

Layer (type)	Output Shape	Param #
conv2d_40 (Conv2D)	(None, 48, 48, 32)	320
max_pooling2d_40 (MaxPooling	(None, 24, 24, 32)	0
batch_normalization_39 (Batc	(None, 24, 24, 32)	128
conv2d_41 (Conv2D)	(None, 24, 24, 64)	18496
max_pooling2d_41 (MaxPooling	(None, 12, 12, 64)	0
batch_normalization_40 (Batc	(None, 12, 12, 64)	256
conv2d_42 (Conv2D)	(None, 10, 10, 128)	73856
max_pooling2d_42 (MaxPooling	(None, 9, 9, 128)	0
batch_normalization_41 (Batc	(None, 9, 9, 128)	512
dropout_16 (Dropout)	(None, 9, 9, 128)	0
conv2d_43 (Conv2D)	(None, 7, 7, 256)	295168
max_pooling2d_43 (MaxPooling	(None, 6, 6, 256)	0
batch_normalization_42 (Batc	(None, 6, 6, 256)	1024
conv2d_44 (Conv2D)	(None, 4, 4, 512)	1180160
max_pooling2d_44 (MaxPooling	(None, 2, 2, 512)	0
batch_normalization_43 (Batc	(None, 2, 2, 512)	2048
flatten_8 (Flatten)	(None, 2048)	0
dense_42 (Dense)	(None, 256)	524544
dropout_17 (Dropout)	(None, 256)	0
dense_43 (Dense)	(None, 128)	32896
dense_44 (Dense)	(None, 64)	8256
batch_normalization_44 (Batc	(None, 64)	256
dense_45 (Dense)	(None, 32)	2080
dense_46 (Dense)	(None, 7)	231

Total params: 2,140,231
Trainable params: 2,138,119
Non-trainable params: 2,112

3. Results and Analysis

Initially a model with a very simple architecture was trained. It had just the input layer, 2 convolutional and max pooling layers, followed by 2 fully connected dense layers.

This model gave good performance on the training set but did not do well on the validation set, and appeared to be overfitting the training set. Reducing the number of layers only made the performance worse. The final architecture was then proposed.

3.1. Performance of our architecture

The final model took significantly longer than the original simple model to train but gave a dramatic improvement in the validation accuracy, which increased from about 30% to 55%.

However, the model stagnating around 55% despite doing well on the training set (around 95% accuracy). We inferred that the model maybe overfitting on the training set (especially since the architecture was quite complex).

To improve performance on the validation set, dropout regularization was added after some of the convolutional and dense layers. Batch normalization was also added at this point to improve training time and performance.

After these modifications, we obtained a training accuracy of about 97% and a validation accuracy of about 63% (Graph, Fig. 2).

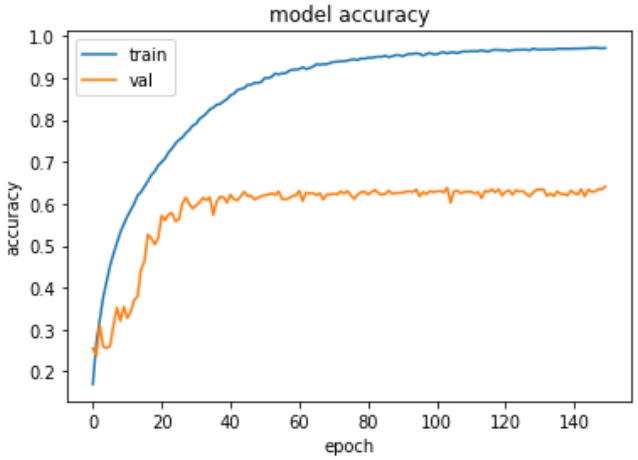

Fig. 2. Accuracy graph of the preliminary model; final training accuracy ~97% and validation accuracy ~63%.

To better our model and make it more robust to newer images, it was trained again from scratch with the augmented data as specified earlier (sec. 2.1), but this time three more batch normalization layers were added, one after every convolution and max pooling layer and one after a fully connected layer, bringing the total number of batch normalization layers to 6. This was done to improve training time. It also improved the performance significantly on the validation set.

The model was trained in 2 parts. a batch size of 1024 was used for the training and validation generators and was trained for 450 epochs at around 15 second per epoch on Google Colab with GPU as the hardware accelerator. At this point, the learning began to plateau. So training was halted and some settings of the Training image generator were tweaked to allow for faster training. The batch size was also changed to 8192. The model was then again trained for 300 epochs but retaining the weights from before. There was a sharp increase in the rate at which the model performance was improving, and the training accuracy plateaued around 99% and the validation accuracy around 68% (Graph, Fig. 3).

But this posed a risk of overfitting as the training was happening on an easier dataset (hence the sharp increase in the graph, Fig. 3).

So, to retain the toughness of the data set, it was trained again, this time for 450 epochs with a batch size of 256. This model gave us similar results to the previous one, with training accuracy around 96% and validation accuracy around 67% (Graph, Fig. 4).

Even though the training accuracy was lower for this model, it presented less of a risk of overfitting the data (having been trained on a tougher dataset) and hence this was the model finally decided upon.

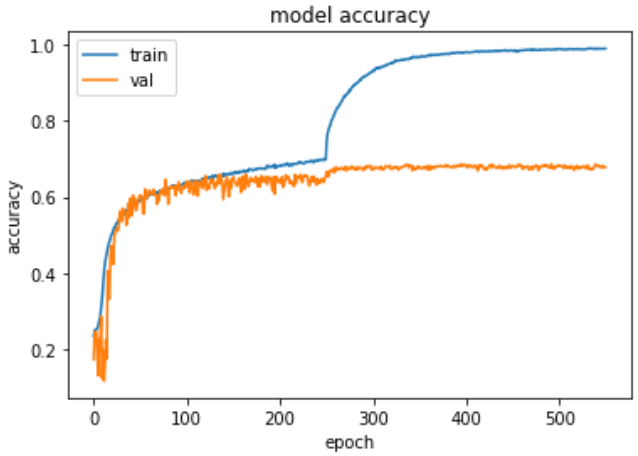

Fig. 3. Accuracy graph of the later model; final training accuracy ~99% and validation accuracy ~68%. Initially the model was trained with a certain set of parameters in the image generator for the first 250 epochs. As the learning plateaued after 300 epochs, these parameters were modified but the weights retained. As a result, there was a sharp increase in learning.

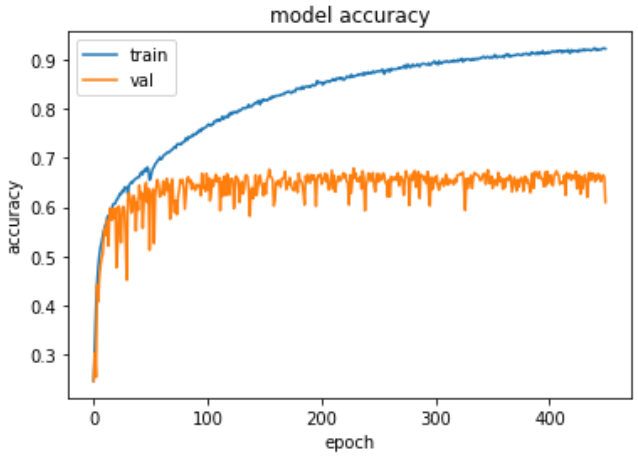

Fig 4. Accuracy graph of the final model; final training accuracy ~96% and validation accuracy ~67%.

Lots of additional models were trained along the way in order to compare or improve performance. Changing the number of convolution layers, pooling layers, number of hidden units in a given layer, dropout rate, etc. were all attempted.

The performance of our model was also compared against modified versions of several classical CNN models, which has been discussed in the following section (3.2).

Tweaking the settings of ImageGenerator class, optimization algorithm and learning rate was also done, however they either diminished or reached the same performance as the one presented here.

One additional metric that was used was the top 2 predictions accuracy. For the final model, the top 2 predictive accuracy was as high as 99.5% on the training set and 82.4% on the validation set. Metrics like precision and recall were not used as this was a multi class problem and it was not of prime importance to prioritize any specific class.

3.2. Performance on some modified CNN architectures

In an attempt to get an idea of how well our model was performing in comparison to other classical existing CNN architectures and try to improve our own model, we trained the data using modified versions of the LeNet[1], the AlexNet[2], and the VGG-16[3] architectures.

3.2.1. AlexNet

A modified version of the AlexNet architecture (Fig. 4) was used for the model. The following changes were made:- kernel size and stride of the first two layers were reduced to better suit the image size given to us (48x48). Also, multiple batch normalization and dropout layers were added to improve the performance. A few more fully connected layers were added with reduced number of units; the number of filters for all other layers was retained as is from the original model.

The accuracy obtained using this model was about 99.5% on the training set and 61.2% on the validation set (Graph, Fig.6). Tweaks on learning rate, batch size and dropout rate yielded similar results.

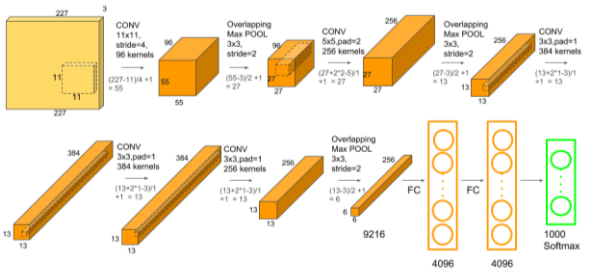

Fig. 5. The original AlexNet architecture*

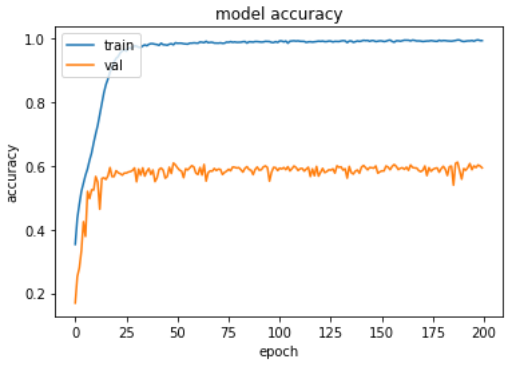

Fig. 6. Accuracy graph for modified AlexNet, final training accuracy ~99% and validation accuracy ~61%

3.2.2. VGG-16

A modified version of the VGG-16 architecture (Fig. 6) was used. In the original architecture, the first 6 convolution and max pooling layers reduced the dimensions of the input to 56x56. This was what was closest to the data set provided and therefore, this was chosen as the starting point to apply the architecture.

After this, the architecture was almost exactly replicated bar one or two layers because the input size dropped below 0. The accuracy obtained using this model was about 96% on the training set and 55% on the validation set.

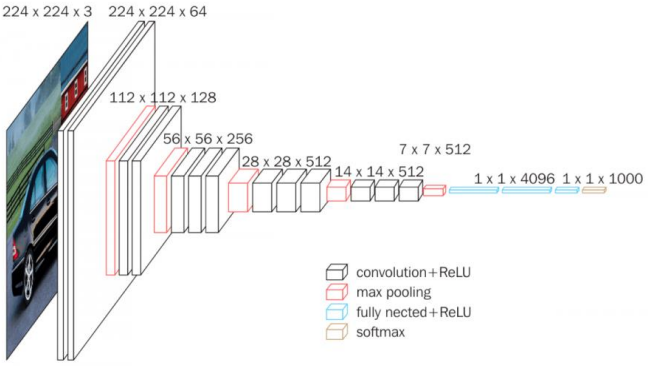

Fig. 7. The original VGG-16 architecture**

4. Conclusion

CNNs are a part of the field of deep learning where extensive research takes place today. While working on this project, we got very well familiarized with the internal working of CNNs and experimented hands-on with many different architectures.

While we managed to get a very good performance on the training data (~96%) our model plateaued after a certain point on the validation data and the maximum accuracy we could obtain was 67%.

On analyzing and comparing with various well-known architectures of today, we came to the conclusion that the possible reasons for not doing very well on the validation set are as follows:

- The size of the dataset (about 35k images) is too small;
- The quality of images is not good enough to be able to capture all the features properly;
- The images were greyscale and hence couldn't capture certain features; and finally,
- Humans being complicated, sometimes it's possible that a person could have multiple emotions at the same time and this could show on their faces, so it may not be possible to so easily strictly classify them.

Considering all the above, we believe that our model performs fairly well. Of course, this field is a very deep and vast one and we may not have been able to do justice in the time given to us, and it is all the more true there is never going to be an end to learning.

Acknowledgment

We give sincere a thank you to Prof. Tanmay Tulsidas Verlekar for taking this course and giving us an opportunity to work on this assignment. We'd also like to thank to all the TA's who helped us with our labs and cleared any doubts regarding this project.

References

[1] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'12). Curran Associates Inc., Red Hook, NY, USA, 1097–1105.

[3] Karen Simonyan, & Andrew Zisserman. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition.

Documentation

tf.keras.Model | TensorFlow Core v2.4.1

Module: tf.keras.layers | TensorFlow Core v2.4.1

tf.keras.preprocessing.image.ImageDataGenerator (tensorflow.org)

Links

*https://neurohive.io/en/popular-networks/alexnet-imagenet-classification with-deep-convolutional-neural-networks/

**https://neurohive.io/en/popular-networks/vgg16/

Attached files

1. modelfinal.h5
2. modelfinalweights.h5
3. NNFL_assignment_final.ipynb