CS-354N
PROGRESS REPORT

By

Khushi Verma (200001036)
Sanskar Verma (200001069)

Under – Dr. Aruna Tiwari

FACE EXPRESSION / EMOTION RECOGNITION

Human emotion classification from images is an emerging field of research in the Artificial Intelligence domain. Optimal facial expression recognition for basic emotions like anger, happiness, frustration, fear, surprise, etc., plays an imminent role in effective non-verbal communication. This helps in easier market research or surveying when companies need to collect user feedback about their products. It can also help cluster videos based on emotional content on any video player like Netflix or YouTube. Facial Expression recognition also glimpses a prospect for the future, like Google Assistant, based on the user's emotional state as detected by the camera; it would provide appropriate suggestions for songs, books, movies, and so on.

Most current algorithms employ Deep Learning networks like Artificial Neural Networks, KNN, Random Forest Classifier, etc. Tensorflow libraries are primarily used in such models for proper training and classification of the dataset. Several traditional methods that exist exploit facial features like eye depth, contraction in length measurements of mouth, nose, or eyes, whether the mouth is closed or open, and local binary patterns.

DATA COLLECTION / PRE-PROCESSING

- We have used the dataset provided on Kaggle under the link – https://www.kaggle.com/datasets/jonathanoheix/face-expression-recognition-dataset
- The dataset contains 48 x 48 pixel grayscale images of faces. In grayscale images, the higher intensities are denoted by 1 (white) and lower intensity by 0 (black). The dataset is ordered in such a way that the main directory folder contains labelled sub-directories of images for the 7 different types of emotion classes that we will be working with. The classes are: (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral).
- The training set contains a total of about 28,709 images, and an additional test set has been provided that consists of another 3,589 images. The dataset was prepared by some researchers as part of their project.
- We have performed data preprocessing to convert the available data into usable format. First, we labelled/ encode the images in a numeric format that computer understands using Label_Encoding and then we categorized our labels into a vector matrix of 0s and 1s using to_categorical() function in tensorflow Keras, since we are working with multi-class classification problem. We finally normalized the weights associated with each class so that the model does not get biased or unbiased for any particular class.

- Once the labels are sorted, we next preprocessed the images so that they can be worked with easily. After loading each image of the dataset, we convert it into a tensor, using tensorflow in-built function – decode_jpeg() over RGB 3-way channel. We first resized and upscaled the 48 x 48 images to 96 x 96, keeping in mind the noise associated with that. Next, we perform Data Augmentation in the way of Flipping, Rotating, and Zoom-in and Zoom-out of images, so that our model can be trained very effectively on the features. This way we would get a Robust Model that can deal with a variety of input images in the real world.
- The key part involves creating tensor slices of our image dataset and labels using tensorflow. To improve the efficiency of the entire process, we parallelize the task into batches of size 32, and they are autotuned by tensorflow.
- After all these steps, we can verify that the shape of an image is (96,96,3) and there are 7 labels total. If we try and check to print an image along with its label, we get the expected outputs.

ALGORITHM

- We will be building a Convolutional Neural Network using Tensorflow to perform this multi-class image classification problem. The CNN would be provided with our preprocesses input data and it would pass through the Average_Pooling layer to signal out the average of features in specific areas of the input images.
- We would use features like eyebrow position, lip movement, eye length, mouth gesture, etc., and get their measurements for each classification label to figure out patterns for our model so that it can fit well.
- After the pooling and flattening layers, we would add Dropout and Dense layers periodically. We plan to use ‘relu’ activation function for the hidden layers, and for the last layer, we will be using ‘softmax’ activation function with 7-neurons layer since we are dealing with multiple classes here.
- Along with this, we will need Optimizers like Adam Optimizer, and values like learning-rate, error-tolerance etc. that we will figure out experimentally by trying out different parameters and choosing the one that fits best.
- To improve the training performance, we can use training in two phases with different number of epochs, which implies the first training would focus on extracting features out of the training dataset efficiently, and the next phase would emphasize upon the classification task.
- We will further be using OpenCV to perform real-time prediction and classification of facial expressions. We would use EfficientNetB2, if needed,

just as a backbone to improve our model, and build upon it our hyperparameters.

PERFORMANCE EVALUATION

We aim to evaluate the performance of our model using various metrics like Accuracy, Precision and Recall. We will also use 'Categorical-Cross-Entropy' Loss metric to evaluate our training and validation losses, and to assess that they decrease following the elbow method. A brief about each of these performance criteria is:

- ACCURACY:

Accuracy is a metric that generally describes how the model performs across all classes. It is useful when all classes are of equal importance. It is calculated as the ratio between the number of correct predictions to the total number of predictions.

$$Accuracy = \frac{True_{positive} + True_{negative}}{True_{positive} + True_{negative} + False_{positive} + False_{negative}}$$

- PRECISION:

The precision is calculated as the ratio between the number of Positive samples correctly classified to the total number of samples classified as Positive (either correctly or incorrectly). The precision measures the model's accuracy in classifying a sample as positive.

$$Precision = \frac{True_{positive}}{True_{positive} + False_{positive}}$$

- RECALL:

The recall is calculated as the ratio between the number of Positive samples correctly classified as Positive to the total number of Positive samples. The recall measures the model's ability to detect Positive samples. The higher the recall, the more positive samples detected.

$$Recall = \frac{True_{positive}}{True_{positive} + False_{negative}}$$

- LOSS:

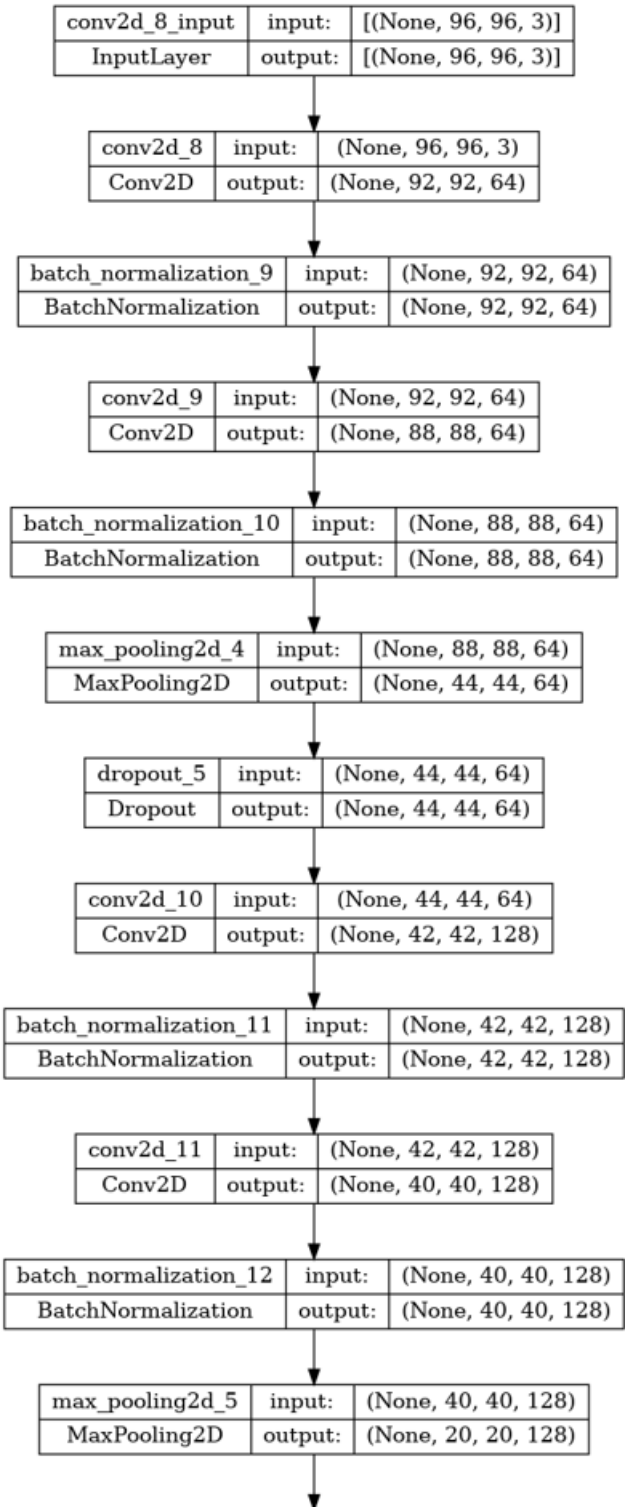
The loss function is used to compute the quantity that the model should seek to minimize during training. We are using categorical cross entropy loss function for multi-class classification model where there are two or more output labels. The output label is assigned one-hot category encoding value in form of 0s and 1. The output label, if present in integer form, is converted into categorical encoding.

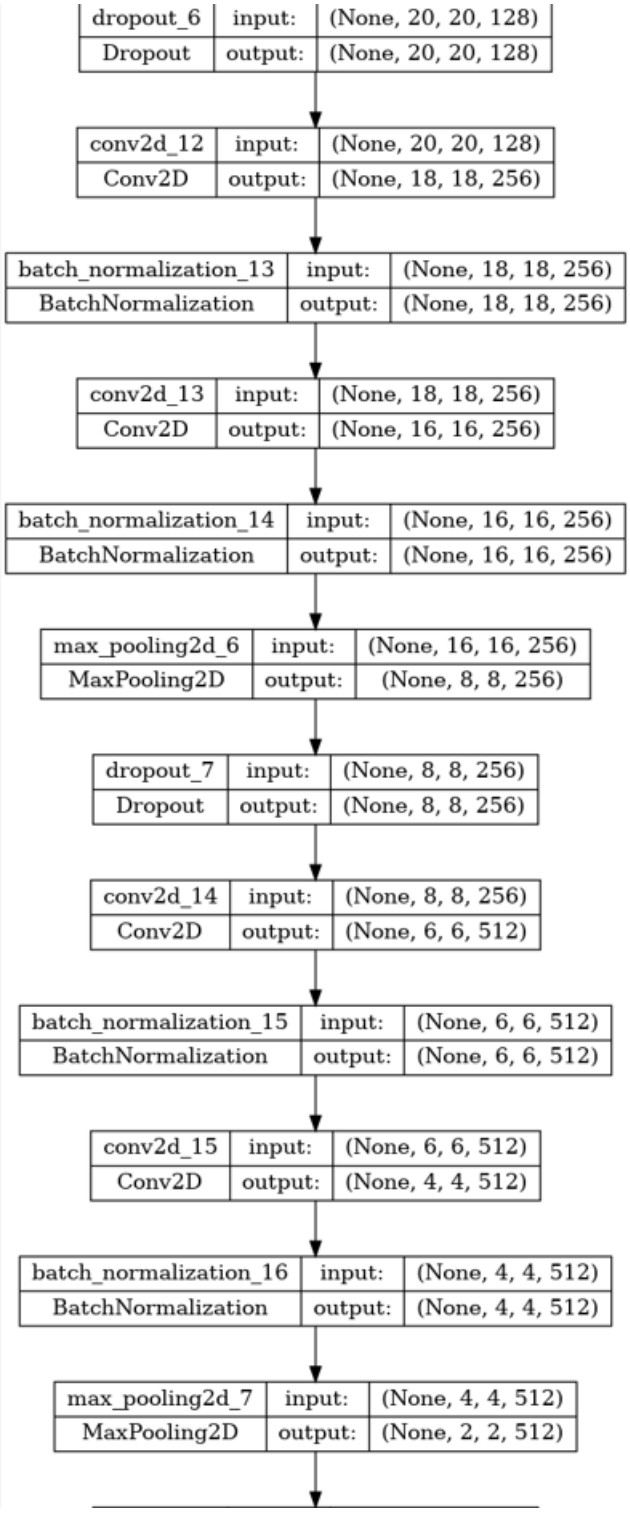
CODING

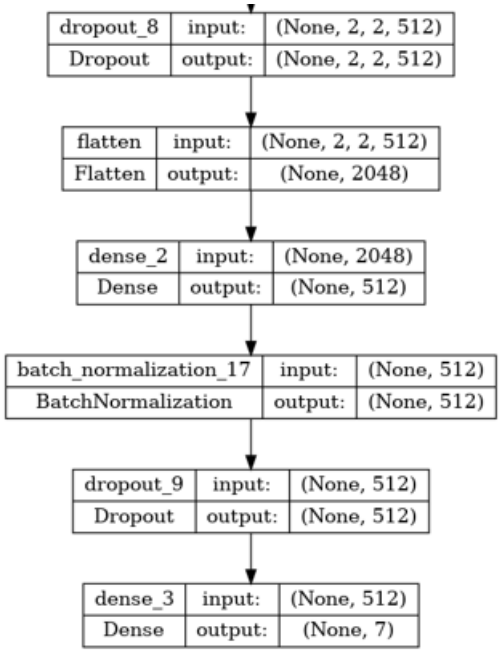
Our code work can be found here on this Kaggle Notebook - https://www.kaggle.com/code/khushikv/face-emotion-recognition

MODEL BUILDING AND ITS ARCHITECTURE

- A CNN model is a type of deep neural network that can process images and other types of data with spatial structure.
- Our sequential CNN model consists of different layers that perform different operations on the input data, such as convolution, pooling, activation, normalization, dropout, flattening and fully connected layers.
- A convolution layer applies a set of filters to the input data, producing a set of feature maps that capture the local patterns in the data.
- A pooling layer reduces the size of the feature maps by applying a function such as max or average over a sliding window, resulting in a more compact and invariant representation of the data.
- An activation layer applies a nonlinear function to the feature maps, such as ReLU or sigmoid, introducing nonlinearity and enhancing the expressive power of the model.
- A normalization layer adjusts the feature maps by scaling and shifting them based on their mean and variance, improving the stability and performance of the model.
- A dropout layer randomly drops out some of the feature maps during training, preventing overfitting and improving generalization.
- A fully connected layer connects all the feature maps to a vector of output values, such as class probabilities or regression scores, depending on the task.
- The architecture of a CNN model stack several convolution-normalization-convolution-normalization-pooling-dropout layers into blocks and followed by the last block which contains flatten-dense-normalization-dropout-dense layers.
- The following image shows the overall structure of our model:


conv2d_8_input | input: | [(None, 96, 96, 3)]
InputLayer | output: | [(None, 96, 96, 3)]
conv2d_8 | input: | (None, 96, 96, 3)
Conv2D | output: | (None, 92, 92, 64)
batch_normalization_9 | input: | (None, 92, 92, 64)
BatchNormalization | output: | (None, 92, 92, 64)
conv2d_9 | input: | (None, 92, 92, 64)
Conv2D | output: | (None, 88, 88, 64)
batch_normalization_10 | input: | (None, 88, 88, 64)
BatchNormalization | output: | (None, 88, 88, 64)
max_pooling2d_4 | input: | (None, 88, 88, 64)
MaxPooling2D | output: | (None, 44, 44, 64)
dropout_5 | input: | (None, 44, 44, 64)
Dropout | output: | (None, 44, 44, 64)
conv2d_10 | input: | (None, 44, 44, 64)
Conv2D | output: | (None, 42, 42, 128)
batch_normalization_11 | input: | (None, 42, 42, 128)
BatchNormalization | output: | (None, 42, 42, 128)
conv2d_11 | input: | (None, 42, 42, 128)
Conv2D | output: | (None, 40, 40, 128)
batch_normalization_12 | input: | (None, 40, 40, 128)
BatchNormalization | output: | (None, 40, 40, 128)
max_pooling2d_5 | input: | (None, 40, 40, 128)
MaxPooling2D | output: | (None, 20, 20, 128)


dropout_6 | input: | (None, 20, 20, 128)
Dropout | output: | (None, 20, 20, 128)
conv2d_12 | input: | (None, 20, 20, 128)
Conv2D | output: | (None, 18, 18, 256)
batch_normalization_13 | input: | (None, 18, 18, 256)
BatchNormalization | output: | (None, 18, 18, 256)
conv2d_13 | input: | (None, 18, 18, 256)
Conv2D | output: | (None, 16, 16, 256)
batch_normalization_14 | input: | (None, 16, 16, 256)
BatchNormalization | output: | (None, 16, 16, 256)
max_pooling2d_6 | input: | (None, 16, 16, 256)
MaxPooling2D | output: | (None, 8, 8, 256)
dropout_7 | input: | (None, 8, 8, 256)
Dropout | output: | (None, 8, 8, 256)
conv2d_14 | input: | (None, 8, 8, 256)
Conv2D | output: | (None, 6, 6, 512)
batch_normalization_15 | input: | (None, 6, 6, 512)
BatchNormalization | output: | (None, 6, 6, 512)
conv2d_15 | input: | (None, 6, 6, 512)
Conv2D | output: | (None, 4, 4, 512)
batch_normalization_16 | input: | (None, 4, 4, 512)
BatchNormalization | output: | (None, 4, 4, 512)
max_pooling2d_7 | input: | (None, 4, 4, 512)
MaxPooling2D | output: | (None, 2, 2, 512)



COMPILING OUR MODEL

- We compiled our model using the following measures.
- Optimizer used is Adam Optimizer.
- Adam stands for Adaptive Moment Estimation. Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients like AdaDelta, Adam also keeps an exponentially decaying average of past gradients M(t), similar to momentum.
- Loss function used is categorical Cross-Entropy.
- Loss function simply measures the absolute difference between our prediction and the actual value.
- Cross-entropy 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. So predicting a probability of .012 when the actual observation label is I would be bad and result in a high loss value.

TRAINING OUR MODEL

- We fitted our model on the training (75%) and validation dataset (25%) which we preprocessed earlier and trained the model in batches which means dividing the input data into smaller subsets and updating the model parameters after processing each batch, instead of processing the whole data at once with.
- Training in batches can reduce the memory requirements and improve the computational efficiency of the model, as well as introduce some randomness and noise that can prevent overfitting and improve generalization.
- Our model runs for 25 number of epochs.
- We got 63.24% accuracy which is considerably very good given our large dataset of facial images incorporating a lot of noise. This value of metric puts our model in one of the best performing models that exist in this domain.

SAVING MODEL FILES

- We have saved our model weights and the model file itself in .h5 file format. Whenever we need to test our model on any dataset, we can simply initialize a model instance and load the model weights into that object.
- After this, we can run the model on any testing dataset of our choice, given it is preprocessed in the same form as our train dataset.

TESTING OUR MODEL

- Firstly, we prepared our test dataset in our preprocessed form to pass into model.
- We labelled/ encode the images in a numeric format, categorized our labels and converted the paths and labels into tensors to make them in a usable form.
- Then we decoded images into jpeg format into 3 channels and resized the images into 96x96 using method bilinear.
- Now as the test dataset is created, we verified it and evaluate our model using precision, recall and accuracy metrics on the test dataset.
- Values for the above metrics are-

  Testing Accuracy : 0.5525

  Testing Precision : 0.6659

  Testing Recall : 0.4416

REALTIME PREDICTION USING OUR MODEL

- Finally, we have used dlib and OpenCV built-in libraries and integrated them with the model that we built to make-realtime predictions. Upon running the code, a windows pop-up will start from the webcam and it starts capturing the video of the person's face before the camera.
- While the code records the video, it will simultaneously use our model and make live predictions on the input it is getting from the video. The program outputs these predictions of the person's facial expression by drawing a bounding box around the face detected and also simultaneously displaying the emotion output that our model predicted, example – sad, happy, surprised or neutral face and so on.
- **The code** for the realtime-prediction model can be found on our Kaggle Notebook here - https://www.kaggle.com/code/khushikv/face-emotion-recognition-realtime-prediction

REFERENCES OF SOME USEFUL PAPERS AND COMPARISON OF ACCURACY METRICS

- Khan, A. R. (2022). Facial Emotion Recognition Using Conventional Machine Learning and Deep Learning Methods: Current Achievements, Analysis and Remaining Challenges. Information, 13(6), 268.
- Teo, W. K., De Silva, L. C., & Vadakkepat, P. (2004). Facial expression detection and recognition system. Journal of The Institution of Engineers, Singapore, 44(3).
- Comparison of metrics – https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/5165369/9964459/9815154/savch.t8-3188390-large.gif
- A. V. Savchenko, L. V. Savchenko and I. Makarov, "Classifying Emotions and Engagement in Online Learning Based on a Single Facial Expression Recognition Neural Network," in IEEE Transactions on Affective Computing, vol. 13, no. 4, pp. 2132-2143, 1 Oct.-Dec. 2022, doi: 10.1109/TAFFC.2022.3188390.

GITHUB REPOSITORY

- Link - https://github.com/KhushiKV/Face-Emotion-Recognition.git