Author's Name: Sajal Sinha

Course Title: Face Emotion Recognition

12 September 2021

Face Emotion Recognition

Abstract:

We deployed an Emotion recognition model which can also be used as a microservice as well, using convolutional Neural Network. It has 5 layers which helps in prediction. We achieved an accuracy of 62.21% on with 100 Epochs, batch size of 64 and learning rate as 0.001.

Problem Statement:

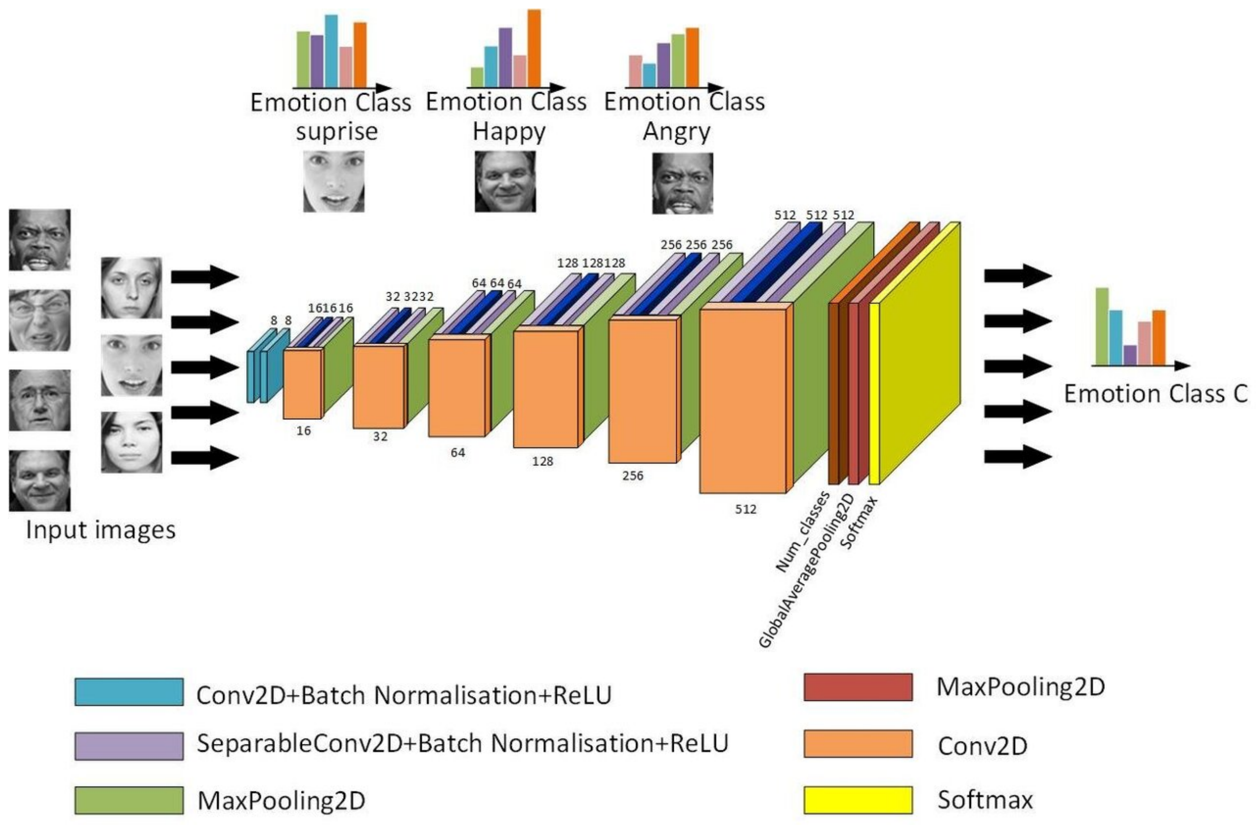
During online classes students often tends to loose attention, which leads to overall non-productivity. For a teacher, its often important for its students to easily grasp concept taught by them. Teachers have skills to observe their students and improve their way throughout their teaching. But due to online teaching, observing has become tough which has eventually disturbed student teacher balance and teaching methods. So, our aim was to develop a Face-Emotion-Recognition Model which can be used a micro service as well so that teachers can understand students much better and enlighten the way to teach.

Introduction:

The Indian education landscape has been undergoing rapid changes for the past 10 years owing to the advancement of web-based learning services, specifically, eLearning platforms. Global E-learning is estimated to witness an 8X over the next 5 years to reach USD 2B in 2021. India is expected to grow with a CAGR of 44% crossing the 10M users mark in 2021. Although the market is growing on a rapid scale, there are major challenges associated with digital learning when compared with brick and mortar classrooms. One of many challenges is how to ensure quality learning for students. Digital platforms might overpower physical classrooms in terms of content quality but when it comes to understanding whether students are able to grasp the content in a live class scenario is yet an open-end challenge. In a physical classroom during a lecturing teacher can see the faces and assess the emotion of the class and tune their lecture accordingly, whether he is going fast or slow. He can identify students who need special attention. Digital classrooms are conducted via video telephony software program (exZoom) where it's not possible for medium scale class (25-50) to see all students and access the mood. Because of this drawback, students are not focusing on content due to lack of surveillance. While digital platforms have limitations in terms of physical surveillance but it comes with the power of data and machines which can work for you. It provides data in the form of video, audio, and texts which can be analysed using deep learning algorithms. Deep learning backed system not only solves the surveillance issue, but it also removes the human bias from the system, and all information is no longer in the teacher's brain rather translated in numbers that can be analysed and tracked.

Discussion:

1. Data loading Mechanism: the *load_data() function in our Colab notebook* access the FER2013.csv dataset from Google Drive and then we start reading it line by line. Our dataset has 35,887 images, which are classified into seven categories - Angry, Disgust, Fear, Happy, Sad, Surprsie, and Neutral. Let us have a look at the file. We are only going to need the Class and the Image Data. Image data is one big string of numbers exactly 2304 numbers. These are pixel intensity values and each number repre-



sents the darkness of that pixel in the image. It can take any value from 0 (White) to 255 (Black). We split the string to get hold of individual numbers and then reshape it into a 48 x 48 array and dividing by 255 normalizes the data.

Once we go through all the images, we expand the dimension of our data array by one to accommodate for the channel value. We one-hot encode the labels then return the numpy arrays.

2. Deployment Mechanism: We have built ourselves a Convolutional Neural Network which has convolutional layers armed with filters that extract features out of images.

We have used Keras to deploy our neural network and if you look at the architecture then you will notice that we have used Dropout layers frequently. Dropout layers inhibit overfitting by randomly dropping out units from the neural network. We will use 20 percent of the training data as validation data.

Hyper-parameter used:

1. epochs = 100
2. batch_size = 64
3. learning_rate = 0.001

Callback functions are those functions which are called at the end of every epoch and in this model, we use two callback functions - ReduceLROnPlateau and EarlyStopping.

1. ReduceLROnPlateau: monitors a certain variable, in this case, validation loss and alters the learning rate when the value stops significantly changing after some certain number of epochs (patience).

2. EarlyStopping: At times our optimiser can land in *local optima* and get stuck there. There is no point in continuing the training as there won't be any further improvements.

3. ModelCheckpoint: Saves the best version of our model along with the weights, so that in case any crash occurs, our model can be recovered.

Steps of CNN:

Pooling :- Convolutional layers in a convolutional neural network systematically apply learned filters to input images in order to create feature maps that summarize the presence of those features in the input.

Convolutional layers prove very effective, and stacking convolutional layers in deep models allows layers close to the input to learn low-level features (e.g. lines) and layers deeper in the model to learn high-order or more abstract features, like shapes or specific objects.

A limitation of the feature map output of convolutional layers is that they record the precise position of features in the input. This means that small movements in the position of the feature in the input image will result in a different feature map. This can happen with re-cropping, rotation, shifting, and other minor changes to the input image.

A common approach to addressing this problem from signal processing is called down sampling. This is where a lower resolution version of an input signal is created that still contains the large or important structural elements, without the fine detail that may not be as useful to the task.

Down sampling can be achieved with convolutional layers by changing the stride of the convolution across the image. A more robust and common approach is to use a pooling layer.

A pooling layer is a new layer added after the convolutional layer. Specifically, after a nonlinearity (e.g. ReLU) has been applied to the feature maps output by a convolutional layer; for example the layers in a model may look as follows:

1. Input Image
2. Convolutional Layer
3. Nonlinearity
4. Pooling Layer

The addition of a pooling layer after the convolutional layer is a common pattern used for ordering layers within a convolutional neural network that may be repeated one or more times in a given model.

The pooling layer operates upon each feature map separately to create a new set of the same number of pooled feature maps.

Pooling involves selecting a pooling operation, much like a filter to be applied to feature maps. The size of the pooling operation or filter is smaller than the size of the feature map; specifically, it is almost always 2×2 pixels applied with a stride of 2 pixels.

This means that the pooling layer will always reduce the size of each feature map by a factor of 2, e.g. each dimension is halved, reducing the number of pixels or values in each feature map to one quarter the size. For example, a pooling layer applied to a feature map of 6×6 (36 pixels) will result in an output pooled feature map of 3×3 (9 pixels).

The pooling operation is specified, rather than learned. Two common functions used in the pooling operation are:

- Average Pooling: Calculate the average value for each patch on the feature map.
- Maximum Pooling (or Max Pooling): Calculate the maximum value for each patch of the feature map.

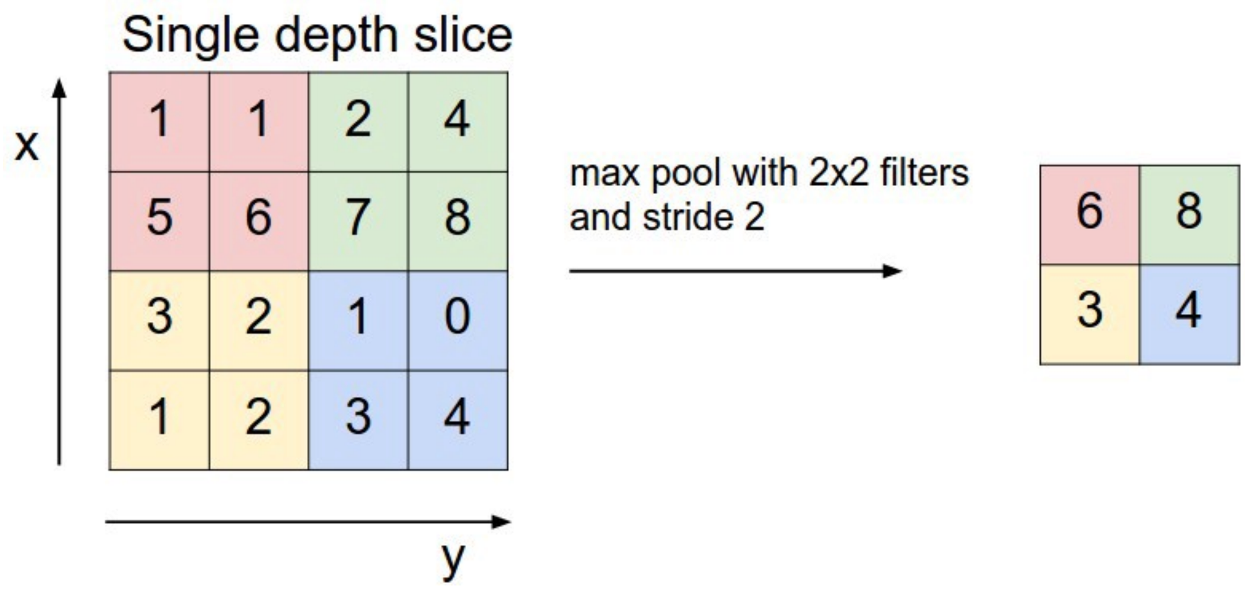
The result of using a pooling layer and creating down sampled or pooled feature maps is a summarized version of the features detected in the input. They are useful as small changes in the location of the feature in the input detected by the convolutional layer will result in a pooled feature map with the feature in the same location. This capability added by pooling is called the model's invariance to local translation.

Batch Normalisation: Batch normalization, or batchnorm for short, is proposed as a technique to help coordinate the update of multiple layers in the model. It does this scaling the output of the layer, specifically by standardizing the activations of each input variable per mini-batch, such as the activations of a node from the previous layer. Recall that standardization refers to rescaling data to have a mean of zero and a standard deviation of one, e.g. a standard Gaussian. Batch normalization reparametrizes the model to make some units always be standardized by definition. This process is also called “whitening” when applied to images in computer vision.

By whitening the inputs to each layer, we would take a step towards achieving the fixed distributions of inputs that would remove the ill effects of the internal covariate shift. Standardizing the activations of the prior layer means that assumptions the subsequent layer makes about the spread and distribution of inputs during the weight update will not change, at least not dramatically. This has the effect of stabilizing and speeding-up the training process of deep neural networks. Normalizing the inputs to the layer has an effect on the training of the model, dramatically reducing the number of epochs required. It can also have a regularizing effect, reducing generalization error much like the use of activation regularization.

Max-pooling, like the name states; will take out only the maximum from a pool. This is actually done with the use of filters sliding through the input; and at every stride, the maximum parameter is taken out and the rest is dropped. This actually down-samples the network.

Unlike the convolution layer, the pooling layer does not alter the depth of the network, the depth dimension remains unchanged.

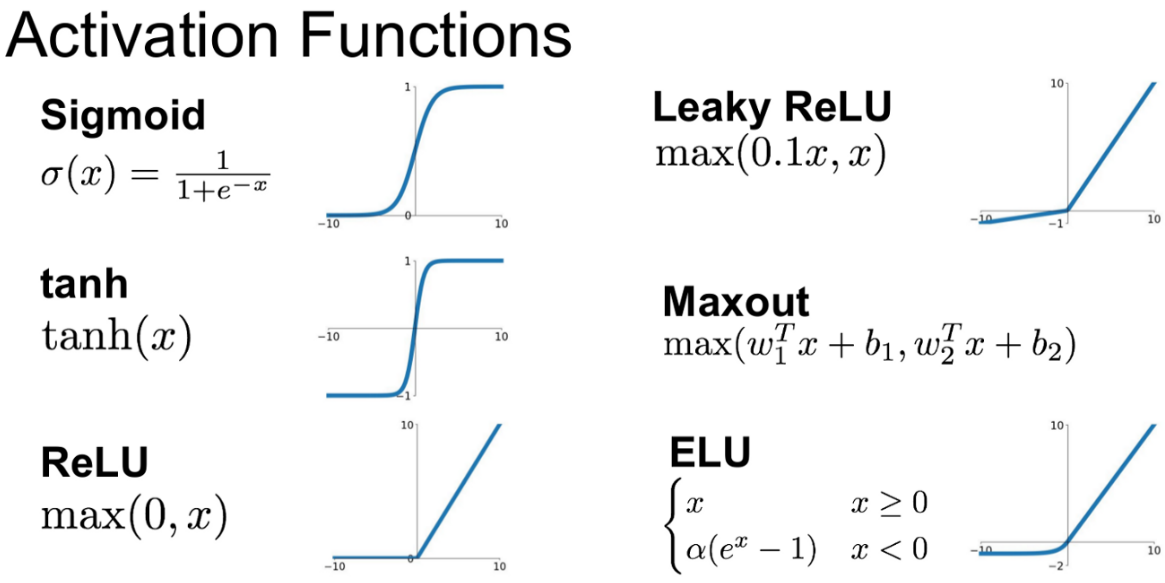


Max-pooling

Formular for the output after Max-pooling:

- (N — F)/ S + 1; where N = Dimension of input to pooling layer
- F = Dimension of filter
- S = Stride

Activation Function: The activation function is a node that is put at the end of or in between Neural Networks. They help to decide if the neuron would fire or not.



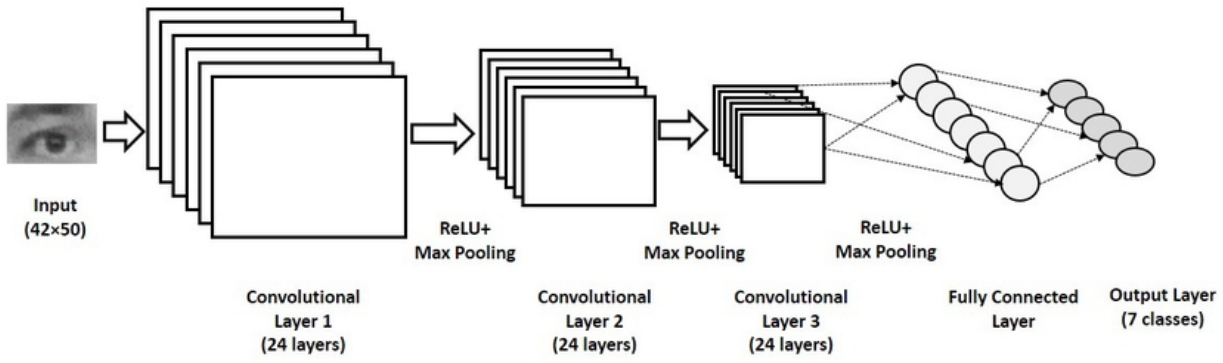
We have different types of activation functions just as the figure above, but for our project my focus will be on Rectified Linear Unit (ReLU).

ReLU function is the most widely used activation function in neural networks to-day. One of the greatest advantage ReLU has over other activation functions is that it does not activate all neurons at the same time. From the image for ReLU function above, we'll notice that it converts all negative inputs to zero and the neuron does not get activated. This makes it very computational efficient as few neurons are activated per time. It does not saturate at the positive region. In practice, ReLU converges six times faster than tanh and sigmoid activation functions.

Some disadvantage ReLU presents is that it is saturated at the negative region, meaning that the gradient at that region is zero. With the gradient equal to zero, during backpropagation all the weights will not be updated, to fix this, we use Leaky ReLU. Also, ReLU functions are not zero-centered. This means that for it to get to its optimal point, it will have to use a zig-zag path which may be longer

Flattening is converting the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector. And it is connected to the final classification model, which is called a **fully-connected** layer. In other words, we put all the pixel data in one line and make connections with the final layer.

Fully-connected Layer :In this layer, the neurons have a complete connection to all the activations from the previous layers. Their activations can hence be computed with a matrix multiplication followed by a bias offset. This is the last phase for a CNN network.



Testing Model:

We trained the neural network and we achieved the highest validation accuracy of 62.43%. After using test data to check how well our model generalize, we score an astounding 63.17% on the test set.

- ## **References:**

- https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480
- https://medium.com/dataseries/basic-overview-of-convolutional-neural-network-cnn-4fcc7dbb4f17#:~:text=The%20activation%20function%20is%20a,neuron%20would%20fire%20or%20not.&text=We%20have%20different%20types%20of,Rectified%20Linear%20Unit%20(ReLU).
- https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/#:~:text=Batch%20normalization%20is%20a%20technique,required%20to%20train%20deep%20networks.