Computer Vision and Deep Learning
Project Report

Facial Attribute Recognition using Custom CNN architecture

Edera Venkata Naga Sai Bharadwaja
SC20M104
Machine Learning and Computing
Indian Institute of Space Sciene and Technology

Table of Contents

List of Tables

List of Figures

1. McCulloch and Pitts Neuron

In 1943, McCulloch and Pitts introduced a mathematical model of a neuron. It consisted of three components:

1. A set of **weights** w_i corresponding to synapses (inputs) 2. An **adder** for summing input signals; analogous to cell membrane that collects charge 3. An **activation function** for determining when the neuron fires, based on accumulated input

The neuron model is shown schematically below. On the left are input nodes $\{x_i\}$, usually expressed as a vector. The strength with which the inputs are able to deliver the signal along the synapse is determined by their corresponding weights $\{w_i\}$. The adder then sums the inputs from all the synapses:

$$h = \sum_i w_i x_i$$

The parameter θ determines whether or not the neuron fires given a weighted input of h. If it fires, it returns a value $y = 1$, otherwise $y = 0$. For example, a simple **activation function** is using θ as a simple fixed threshold:

$$y = g(h) = \begin{cases} 1, \text{if } h\theta \\ 0, \text{if } h \leq \theta \end{cases}$$

this activation function may take any of several forms, such as a logistic function.

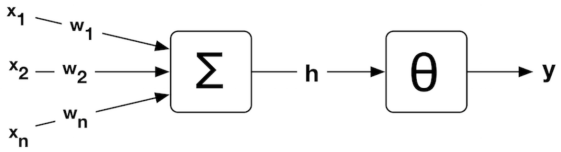


Fig. 1. McCulloch and Pitts Neuron

A single neuron is not interesting, nor useful, from a learning perspective. It cannot learn; it simply receives inputs and either fires or not. Only when neurons are joined as a **network** can they perform useful work.

Learning takes place by changing the weights of the connections in a neural network, and by changing the parameters of the activation functions of neurons.

2. Perceptron

A collection of McCullough and Pitts neurons, along with a set of input nodes connected to the inputs via weighted edges, is a perceptron, the simplest neural network.

Each neuron is independent of the others in the perceptron, in the sense that its behavior and performance depends only on its own weights and threshold values, and not of those for the other neurons. Though they share inputs, they operate independently.

The number of inputs and outputs are determined by the data. Weights are stored as a 'N x K' matrix, with N observations and K neurons, with w_{ij} specifying the weight on the *i*th observation on the *j*th neuron.

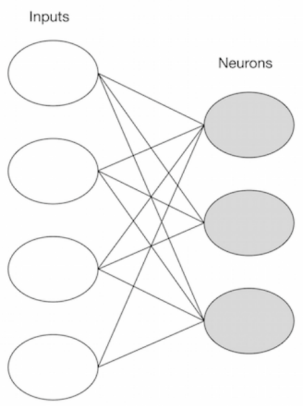


Fig. 2. Perceptron

In order to use the perceptron for statistical learning, we compare the outputs y_j from each neuron to the obervation target t_j, and adjust the input weights when they do not correspond (**e.g.** if a neuron fires when it should not have).

$$t_j - y_j$$

We use this difference to update the weight w_{ij}, based on the input and a desired **learning rate**. This results in an update rule:

$$w_{ij} \leftarrow w_{ij} + \eta(t_j - y_j)x_i$$

After an incremental improvement, the perceptron is shown the training data again, resulting in another update. This is repeated until the performance no longer improves. Having a learning rate less than one results in a more stable learning rate, though this stability is traded off against having to expose the network to the data multiple times. Typical learning rates are in the 0.1-0.4 range.

An additional input node is typically added to the perceptron model, which is a constant value (usually -1, 0, or 1) that acts analogously to an intercept in a regression model. This establishes a baseline input for the case when all inputs are zero.

3. Learning with Perceptrons

- Initialize weights w_{ij} to small, random numbers.
- For each t in T iterations * compute activation for each neuron *j* connected to each input vector *i*

$$y_j = g\left(h = \sum_i w_{ij} x_i\right) = \begin{cases} 1, \text{if } h0 \\ 0, \text{if } h \leq 0 \end{cases}$$

- update weights

$$w_{ij} \leftarrow w_{ij} + \eta(t_j - y_j)x_i$$

4. Multi-layer Perceptron

The solution to fitting more complex (**i.e.** non-linear) models with neural networks is to use a more complex network that consists of more than just a single perceptron. The take-home message from the perceptron is that all of the learning happens by adapting the synapse weights until prediction is satisfactory. Hence, a reasonable guess at how to make a perceptron more complex is to simply **add more weights**.

There are two ways to add complexity:

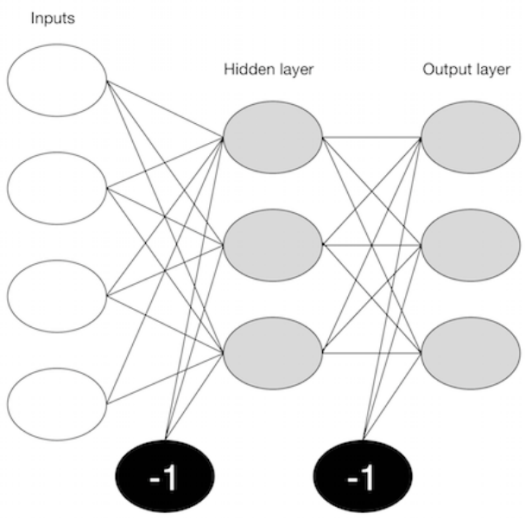


Fig. 3. Multi Layer Perceptron

How to train a multilayer network is not intuitive. Propagating the inputs forward over two layers is straightforward, since the outputs from the hidden layer can be used as inputs for the output layer. However, the process for updating the weights based on the prediction error is less clear, since it is difficult to know whether to change the weights on the input layer or on the hidden layer in order to improve the prediction.

Updating a multi-layer perceptron (MLP) is a matter of:

1. moving forward through the network, calculating outputs given inputs and current weight estimates
2. moving backward updating weights according to the resulting error from forward propagation.

In this sense, it is similar to a single-layer perceptron, except it has to be done twice, once for each layer.

5. Backpropagation

Backpropagation is a method for efficiently computing the gradient of the cost function of a neural network with respect to its parameters. These partial derivatives can then be used to update the network's parameters using, e.g., gradient descent. This may be the most common method for training neural networks. Deriving backpropagation involves numerous clever applications of the chain rule for functions of vectors.

5.1 Review: The chain rule

The chain rule is a way to compute the derivative of a function whose variables are themselves functions of other variables. If C is a scalar-valued function of a scalar z and z is itself a scalar-valued function of another scalar variable w, then the chain rule states that

$$\frac{\partial C}{\partial w} = \frac{\partial C}{\partial z}\frac{\partial z}{\partial w}$$

For scalar-valued functions of more than one variable, the chain rule essentially becomes additive. In other words, if C is a scalar-valued function of N variables $z_1, \ldots, z_N$, each of which is a function of some variable w, the chain rule states that

$$\frac{\partial C}{\partial w} = \sum_{i=1}^{N} \frac{\partial C}{\partial z_i}\frac{\partial z_i}{\partial w}$$

5.2 Notation

In the following derivation, we'll use the following notation:

L - Number of layers in the network.

N^n - Dimensionality of layer $n \in \{0, \ldots, L\}$. N^0 is the dimensionality of the input; N^L is the dimensionality of the output.

$W^m \in \mathbb{R}^{N^m \times N^{m-1}}$ - Weight matrix for layer $m \in \{1, \ldots, L\}$. W^m_{ij} is the weight between the i^{th} unit in layer m and the j^{th} unit in layer $m-1$.

$b^m \in \mathbb{R}^{N^m}$ - Bias vector for layer m.

σ^m - Nonlinear activation function of the units in layer m, applied elementwise.

$z^m \in \mathbb{R}^{N^m}$ - Linear mix of the inputs to layer m, computed by $z^m = W^m a^{m-1} + b^m$.

$a^m \in \mathbb{R}^{N^m}$ - Activation of units in layer m, computed by $a^m = \sigma^m(h^m) = \sigma^m(W^m a^{m-1} + b^m)$. a^L is the output of the network. We define the special case a^0 as the input of the network.

$y \in \mathbb{R}^{N^L}$ - Target output of the network.

C - Cost/error function of the network, which is a function of a^L (the network output) and y (treated as a constant).

5.3 Backpropagation Derivation

In order to train the network using a gradient descent algorithm, we need to know the gradient of each of the parameters with respect to the cost/error function C; that is, we need to know $\frac{\partial C}{\partial W^m}$ and $\frac{\partial C}{\partial b^m}$. It will be sufficient to derive an expression for these gradients in terms of the following terms, which we can compute based on the neural network's architecture:

- $\frac{\partial C}{\partial a^L}$: The derivative of the cost function with respect to its argument, the output of the network - $\frac{\partial a^m}{\partial z^m}$: The derivative of the nonlinearity used in layer m with respect to its argument

To compute the gradient of our cost/error function C to W^m_{ij} (a single entry in the weight matrix of the layer m), we can first note that C is a function of a^L, which is itself a function of the linear mix variables z^m_k, which are themselves functions of the weight matrices W^m and biases b^m. With this in mind, we can use the chain rule as follows:

$$\frac{\partial C}{\partial W^m_{ij}} = \sum_{k=1}^{N^m} \frac{\partial C}{\partial z^m_k} \frac{\partial z^m_k}{\partial W^m_{ij}}$$

Note that by definition

$$z^m_k = \sum_{l=1}^{N^m} W^m_{kl} a^{m-1}_l + b^m_k$$

It follows that $\frac{\partial z^m_k}{\partial W^m_{ij}}$ will evaluate to zero when $i \neq k$ because z^m_k does not interact with any elements in W^m except for those in the k¡sup¿th¡/sup¿ row, and we are only considering the entry W^m_{ij}. When $i = k$, we have

$$\begin{aligned}\frac{\partial z_i^m}{\partial W_{ij}^m} &= \frac{\partial}{\partial W_{ij}^m}\left(\sum_{l=1}^{N^m} W_{il}^m a_l^{m-1} + b_i^m\right) \\ &= a_j^{m-1} \\ \rightarrow \frac{\partial z_k^m}{\partial W_{ij}^m} &= \begin{cases} 0 & k \neq i \\ a_j^{m-1} & k = i \end{cases}\end{aligned}$$

The fact that $\frac{\partial C}{\partial a_k^m}$ is 0 unless $k = i$ causes the summation above to collapse, giving

$$\frac{\partial C}{\partial W_{ij}^m} = \frac{\partial C}{\partial z_i^m} a_j^{m-1}$$

or in vector form

$$\frac{\partial C}{\partial W^m} = \frac{\partial C}{\partial z^m} a^{m-1\top}$$

Similarly for the bias variables b^m, we have

$$\frac{\partial C}{\partial b_i^m} = \sum_{k=1}^{N^m} \frac{\partial C}{\partial z_k^m} \frac{\partial z_k^m}{\partial b_i^m}$$

As above, it follows that $\frac{\partial z_k^m}{\partial b_i^m}$ will evaluate to zero when $i \neq k$ because z_k^m does not interact with any element in b^m except b_k^m. When $i = k$, we have

$$\begin{aligned}\frac{\partial z_i^m}{\partial b_i^m} &= \frac{\partial}{\partial b_i^m}\left(\sum_{l=1}^{N^m} W_{il}^m a_l^{m-1} + b_i^m\right) \\ &= 1 \\ \rightarrow \frac{\partial z_i^m}{\partial b_i^m} &= \begin{cases} 0 & k \neq i \\ 1 & k = i \end{cases}\end{aligned}$$

The summation also collapses to give

$$\frac{\partial C}{\partial b_i^m} = \frac{\partial C}{\partial z_i^m}$$

or in vector form

$$\frac{\partial C}{\partial b^m} = \frac{\partial C}{\partial z^m}$$

Now, we must compute $\frac{\partial C}{\partial z_k^m}$. For the final layer ($m = L$), this term is straightforward to compute using the chain rule:

$$\frac{\partial C}{\partial z_k^L} = \frac{\partial C}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_k^L}$$

or, in vector form

$$\frac{\partial C}{\partial z^L} = \frac{\partial C}{\partial a^L} \frac{\partial a^L}{\partial z^L}$$

The first term $\frac{\partial C}{\partial a^L}$ is just the derivative of the cost function with respect to its argument, whose form depends on the cost function chosen. Similarly, $\frac{\partial a^m}{\partial z^m}$ (for any layer m includling L) is the derivative of the layer's nonlinearity with respect to its argument and will depend on the choice of nonlinearity. For other layers, we again invoke the chain rule:

$$\begin{aligned}
\frac{\partial C}{\partial z_k^m} &= \frac{\partial C}{\partial a_k^m} \frac{\partial a_k^m}{\partial z_k^m} \\
&= \left(\sum_{l=1}^{N^{m+1}} \frac{\partial C}{\partial z_l^{m+1}} \frac{\partial z_l^{m+1}}{\partial a_k^m} \right) \frac{\partial a_k^m}{\partial z_k^m} \\
&= \left(\sum_{l=1}^{N^{m+1}} \frac{\partial C}{\partial z_l^{m+1}} \frac{\partial}{\partial a_k^m} \left(\sum_{h=1}^{N^m} W_{lh}^{m+1} a_h^m + b_l^{m+1} \right) \right) \frac{\partial a_k^m}{\partial z_k^m} \\
&= \left(\sum_{l=1}^{N^{m+1}} \frac{\partial C}{\partial z_l^{m+1}} W_{lk}^{m+1} \right) \frac{\partial a_k^m}{\partial z_k^m} \\
&= \left(\sum_{l=1}^{N^{m+1}} W_{kl}^{m+1\top} \frac{\partial C}{\partial z_l^{m+1}} \right) \frac{\partial a_k^m}{\partial z_k^m}
\end{aligned}$$

where the last simplification was made because by convention $\frac{\partial C}{\partial z_l^{m+1}}$ is a column vector, allowing us to write the following vector form:

$$\frac{\partial C}{\partial z^m} = \left(W^{m+1\top} \frac{\partial C}{\partial z^{m+1}} \right) \circ \frac{\partial a^m}{\partial z^m}$$

Note that we now have the ingredients to efficiently compute the gradient of the cost function with respect to the network's parameters: First, we compute $\frac{\partial C}{\partial z_k^L}$ based on the choice of cost function and nonlinearity. Then, we recursively can compute $\frac{\partial C}{\partial z^m}$ layer-by-layer based on the term $\frac{\partial C}{\partial z^{m+1}}$ computed from the previous layer and the nonlinearity of the layer (this is called the "backward pass").

6. Convolutional Neural Networks

Convolutional Neural Networks (ConvNets or CNNs) are a category of Neural Networks that have proven very effective in areas such as image recognition and classification. ConvNets have been successful in identifying faces, objects and traffic signs apart from powering vision in robots and self driving cars.

A Convolutional Neural Network (CNN) is comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural network. The architecture of a CNN is designed to take advantage of the 2D structure of an input image (or other 2D input such as a speech signal). This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features. Another benefit of CNNs is that they are easier to train and have many fewer parameters than fully connected networks with the same number of hidden units. In this article we will discuss the architecture of a CNN and the back propagation algorithm to compute the gradient with respect to the parameters of the model in order to use gradient based optimization.

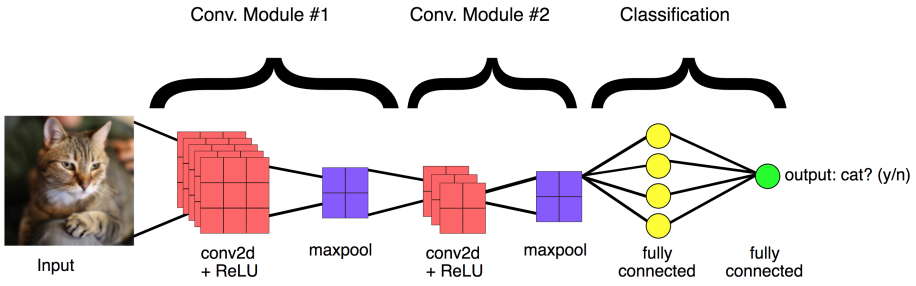


Fig. 4. CNN

6.1 Correlation and Convolution

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

$$F = \begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{bmatrix}$$

$$A * F,\ A * (F)_{\text{rotate180}^o}$$

- $A * (F)_{\text{rotate}180^o}$
  - Flip F vertically and horizontally (rotating 180^o)
  - Perform Correlation of A with rotated F

Full Convolution

$$\begin{bmatrix} \frac{\partial L}{\partial X_{11}} & \frac{\partial L}{\partial X_{12}} & \frac{\partial L}{\partial X_{13}} \\ \frac{\partial L}{\partial X_{21}} & \frac{\partial L}{\partial X_{22}} & \frac{\partial L}{\partial X_{23}} \\ \frac{\partial L}{\partial X_{31}} & \frac{\partial L}{\partial X_{32}} & \frac{\partial L}{\partial X_{33}} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \frac{\partial L}{\partial O_{11}} & \frac{\partial L}{\partial O_{12}} & 0 \\ 0 & \frac{\partial L}{\partial O_{21}} & \frac{\partial L}{\partial O_{22}} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} F_{22} & F_{21} \\ F_{12} & F_{11} \end{bmatrix}$$

- Correlation of the rotated filter with the gradients of the Loss function with respect to the input
- Real Convolution
- Padded with zero for adjusting the size of the left hand matrix

6.2 CNN Backprpogation

$$X = \begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix} \qquad *F = \begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{bmatrix}$$

6.2.1 Convolution

$$X * F = \begin{bmatrix} O_{11} & O_{12} \\ O_{21} & O_{22} \end{bmatrix}$$

$$\begin{aligned} O_{11} &= X_{11}F_{11} + X_{12}F_{12} + X_{21}F_{21} + X_{22}F_{22} \\ O_{12} &= X_{12}F_{11} + X_{13}F_{12} + X_{22}F_{21} + X_{23}F_{22} \\ O_{21} &= X_{21}F_{11} + X_{22}F_{12} + X_{31}F_{21} + X_{32}F_{22} \\ O_{22} &= X_{22}F_{11} + X_{23}F_{12} + X_{32}F_{21} + X_{33}F_{22} \end{aligned}$$

6.2.2 Filter Gradient

$$\frac{\partial L}{\partial F} = \begin{bmatrix} \frac{\partial L}{\partial F_{11}} & \frac{\partial L}{\partial F_{12}} \\ \frac{\partial L}{\partial F_{21}} & \frac{\partial L}{\partial F_{22}} \end{bmatrix}$$

$$\begin{aligned} \frac{\partial L}{\partial F_{11}} &= \frac{\partial L}{\partial O_{11}}\frac{\partial O_{11}}{\partial F_{11}} + \frac{\partial L}{\partial O_{12}}\frac{\partial O_{12}}{\partial F_{11}} + \frac{\partial L}{\partial O_{21}}\frac{\partial O_{21}}{\partial F_{11}} + \frac{\partial L}{\partial O_{22}}\frac{\partial O_{22}}{\partial F_{11}} \\ &= \frac{\partial L}{\partial O_{11}}X_{11} + \frac{\partial L}{\partial O_{12}}X_{12} + \frac{\partial L}{\partial O_{21}}X_{21} + \frac{\partial L}{\partial O_{22}}X_{22} \end{aligned}$$

$$\frac{\partial L}{\partial F_{12}} = \frac{\partial L}{\partial O_{11}}X_{12} + \frac{\partial L}{\partial O_{12}}X_{13} + \frac{\partial L}{\partial O_{21}}X_{22} + \frac{\partial L}{\partial O_{22}}X_{23}$$

$$\frac{\partial L}{\partial F_{21}} = \frac{\partial L}{\partial O_{11}}X_{21} + \frac{\partial L}{\partial O_{12}}X_{22} + \frac{\partial L}{\partial O_{21}}X_{31} + \frac{\partial L}{\partial O_{22}}X_{32}$$

$$\frac{\partial L}{\partial F_{22}} = \frac{\partial L}{\partial O_{11}}X_{22} + \frac{\partial L}{\partial O_{12}}X_{23} + \frac{\partial L}{\partial O_{21}}X_{32} + \frac{\partial L}{\partial O_{22}}X_{33}$$

$$\begin{aligned} \begin{bmatrix} \frac{\partial L}{\partial F_{11}} & \frac{\partial L}{\partial F_{12}} \\ \frac{\partial L}{\partial F_{21}} & \frac{\partial L}{\partial F_{22}} \end{bmatrix} &= \begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix} * \begin{bmatrix} \frac{\partial L}{\partial O_{11}} & \frac{\partial L}{\partial O_{12}} \\ \frac{\partial L}{\partial O_{21}} & \frac{\partial L}{\partial O_{22}} \end{bmatrix} \\ &= \text{conv}(X, \frac{\partial L}{\partial O}) \end{aligned}$$

6.2.3 Input Gradient

$$\frac{\partial L}{\partial X_{11}} = \frac{\partial L}{\partial O_{11}} * F_{11}$$

$$\frac{\partial L}{\partial X_{12}} = \frac{\partial L}{\partial O_{11}} * F_{12} + \frac{\partial L}{\partial O_{12}} * F_{11}$$

$$\frac{\partial L}{\partial X_{13}} = \frac{\partial L}{\partial O_{12}} * F_{12}$$

$$\frac{\partial L}{\partial X_{21}} = \frac{\partial L}{\partial O_{11}} * F_{21} + \frac{\partial L}{\partial O_{21}} * F_{11}$$

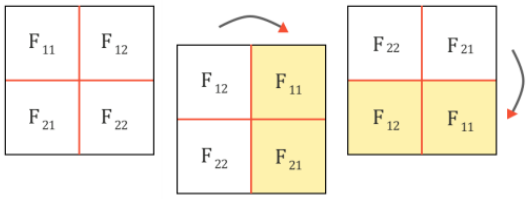
$$\frac{\partial L}{\partial X_{22}} = \frac{\partial L}{\partial O_{11}} * F_{22} + \frac{\partial L}{\partial O_{12}} * F_{21} + \frac{\partial L}{\partial O_{21}} * F_{12} + \frac{\partial L}{\partial O_{22}} * F_{11}$$

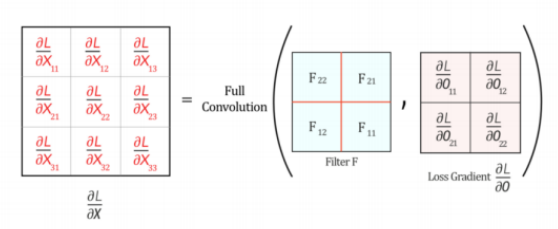
$$\frac{\partial L}{\partial X_{23}} = \frac{\partial L}{\partial O_{12}} * F_{22} + \frac{\partial L}{\partial O_{22}} * F_{12}$$

$$\frac{\partial L}{\partial X_{31}} = \frac{\partial L}{\partial O_{21}} * F_{21}$$

$$\frac{\partial L}{\partial X_{32}} = \frac{\partial L}{\partial O_{21}} * F_{22} + \frac{\partial L}{\partial O_{22}} * F_{21}$$

$$\frac{\partial L}{\partial X_{33}} = \frac{\partial L}{\partial O_{22}} * F_{22}$$





7. Facial Attribute Recognition using CNN

The Main objective of this system is to predict different types of Facial Attributes such as

1. **Eyes: Open/Closed**
   (get_frontal_face_detector() and shape_predictor_68_face_landmarks.dat)

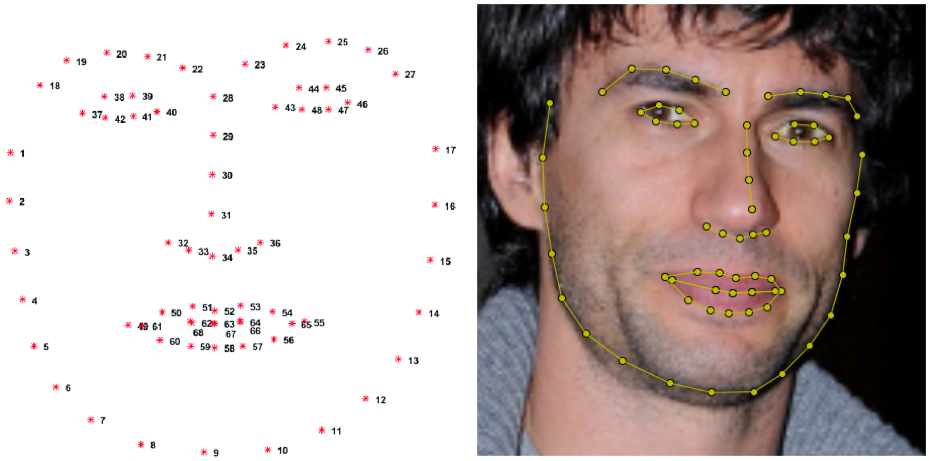
2. **Smiling: Yes/No**
   (Trained the model using fer13 Dataset)

3. **Glasses: Wearing/Not Wearing**
   (Trained the model using “glasses” and “without Glasses” Data set)

4. **Gender**
   (Trained the model using Gender Classification Dataset)

7.1 Eyes: Open/Closed

The dlib library in python has a method called get_frontal_face_detector(), which internally has a pretrained model **shape_predictor_68_face_landmarks.dat**. This pretrained model is used to get the 68 facial landmarks on the identified face in an image.



	Start	End	(Top,Bottom)
Left Eye	37	42	(38,42),(39,41)
Right Eye	43	48	(44,48),(45,47)

Table 1. Eye Points in 68 facial Landmarks

To verify whether Eyes are opened or Closed , then we use the following formula:

- A = dist.euclidean(leftEyePts[1], leftEyePts[5])
- B = dist.euclidean(leftEyePts[2], leftEyePts[4])
- C = dist.euclidean(leftEyePts[0], leftEyePts[3])
- result = (A + B) / (2.0 * C)
- if result ¿ Threshold : Left Eye is opened
- else : Left Eye is opened

Similarly , we calculate for the right eye

7.2 Emotion: Smile/Neutral

Dataset : fer13

For the prediction of the Emotion of a face in an image , we use CNN architecture. In this project we tried so many custom architectures by making use of LR tuner , Keras Tuner and finally found an architecture that suits best for the given data.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 34, 34, 32)        896
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 5, 5, 256)         295168
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 2, 2, 256)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 2, 2, 256)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               262400
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 514
=================================================================
Total params: 651,330
Trainable params: 651,330
Non-trainable params: 0
```

Fig. 5. CNN Architecture for Smile Model

For the above Trained Model after 30 Epochs

- loss: 0.0154
- accuracy: 0.9982
- val_loss: 0.4759
- val_accuracy: 0.8993

```
Epoch 30/30
552/552 [==============================] - 7s 12ms/step - loss: 0.0154 - accuracy: 0.9982 - val_loss: 0.4759 - val_accuracy:
0.8993
```

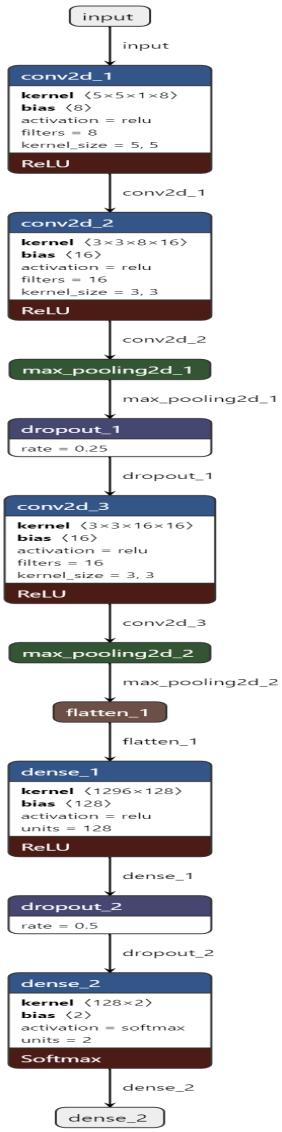


Fig. 6. Block Diagram for Smile Model

7.3 Glasses : yes/No

Dataset : with and without glasses

For the prediction of wearing Glasses of a face in an image , we use CNN architecture. In this project we tried so many custom architectures by making use of LR tuner , Keras Tuner and finally found an architecture that suits best for the given data.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 34, 34, 32)        896
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 5, 5, 256)         295168
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 2, 2, 256)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 2, 2, 256)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               262400
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 514
=================================================================
Total params: 651,330
Trainable params: 651,330
Non-trainable params: 0
```

For the above Trained Model after 25 Epochs

- loss: 0.0403
- accuracy: 0.9873
- val_loss: 0.2950
- val_accuracy: 0.8908

```
Epoch 25/25
473/473 [==============================] - 5s 11ms/step - loss: 0.0403 - accuracy: 0.9873 - val_loss: 0.2950 - val_accuracy:
0.8908
```

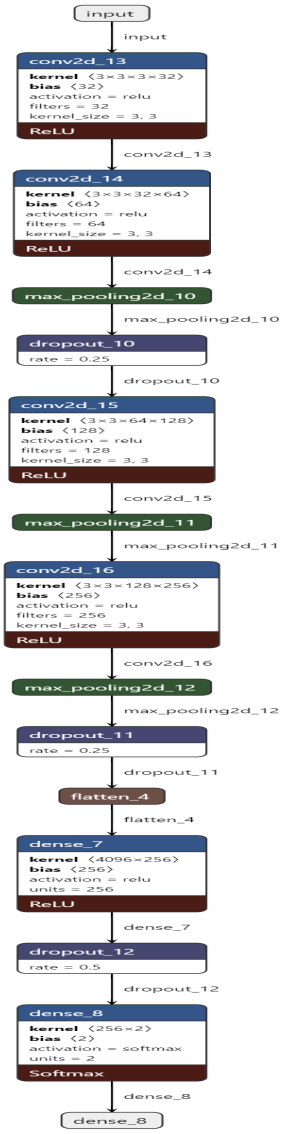


Fig. 7. Block Diagram for Glass Model

7.4 Gender : M/F

Dataset : Gender Dataset

For the prediction of Gender in an image , we use CNN architecture. In this project we tried so many custom architectures by making use of LR tuner , Keras Tuner and finally found an architecture that suits best for the given data.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 34, 34, 32)        896
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 5, 5, 256)         295168
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 2, 2, 256)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 2, 2, 256)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               262400
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 514
=================================================================
Total params: 651,330
Trainable params: 651,330
Non-trainable params: 0
```

Fig. 8. CNN Architecture for Gender Model

For the above Trained Model after 30 Epochs

- loss: 0.0154
- accuracy: 0.9982
- val_loss: 0.4759
- val_accuracy: 0.8993

```
Epoch 30/30
552/552 [==============================] - 7s 12ms/step - loss: 0.0154 - accuracy: 0.9982 - val_loss: 0.4759 - val_accuracy:
0.8993
```

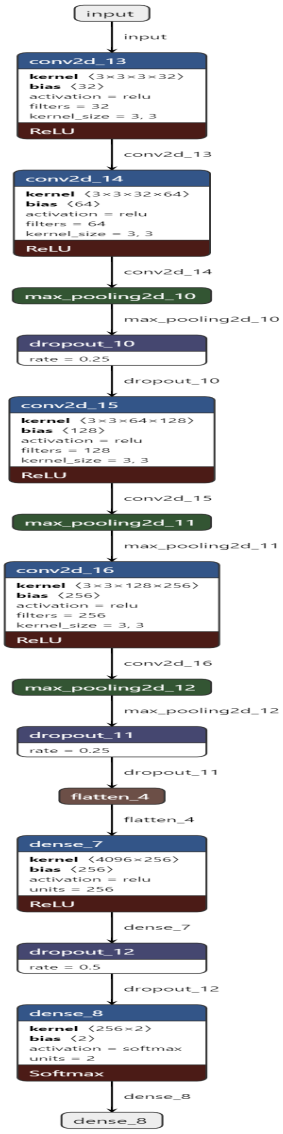
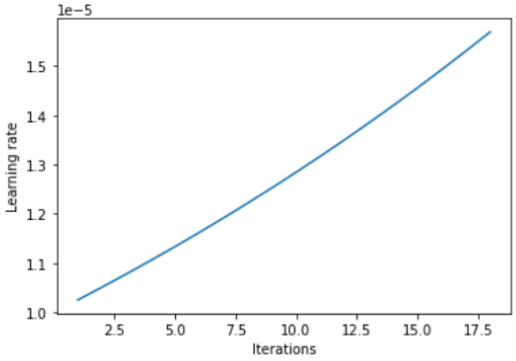
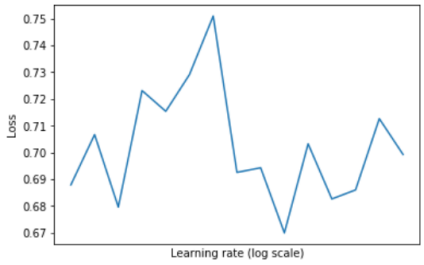
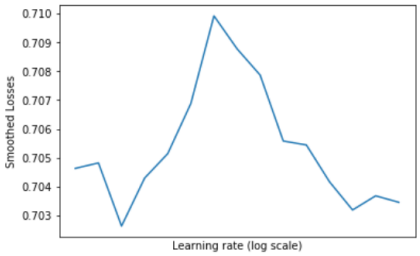
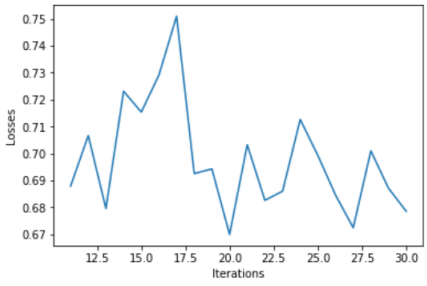


Fig. 9. Block Diagram for Gender Model


1e−5
Learning rate
Iterations


Loss
Learning rate (log scale)


Smoothed Losses
Learning rate (log scale)


Losses
Iterations

8. Result

8.1 image 1

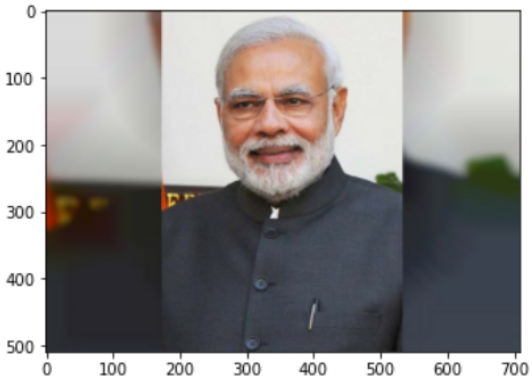


Fig. 10. image 1 : input

```
rectangles[[(242, 77) (428, 263)]]
(186, 186)
face # 0 ,
 Eyes:  Both eyes are open ,
 Smiling:  YES ,
 Glasses:  YES ,
 Gender: M
```

Fig. 11. image 1 : output

8.2 image 2

Fig. 12. image 2 : input

```
rectangles[[(304, 284) (489, 469)], [(139, 304) (324, 490)]]
(185, 185)
face # 0 ,
 Eyes:  Both eyes are open ,
 Smiling:  YES ,
 Glasses:  NO ,
 Gender: F
(186, 185)
face # 1 ,
 Eyes:  Both eyes are open ,
 Smiling:  YES ,
 Glasses:  YES ,
 Gender: M
```

Fig. 13. image 2 : output

8.3 image 3

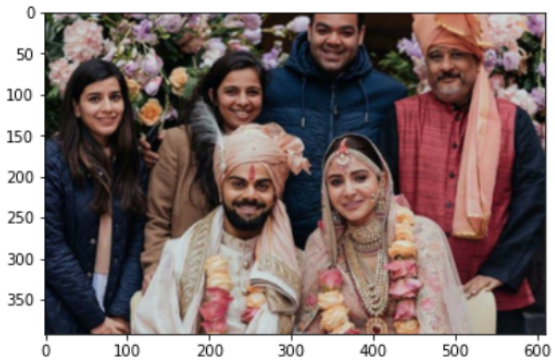


Fig. 14. image 3 : input

```
face # 0 ,
 Eyes:  Both eyes are open ,
 Smiling:  YES ,
 Glasses:  NO ,
 Gender: F
(75, 75)
face # 1 ,
 Eyes:  Both eyes are open ,
 Smiling:  YES ,
 Glasses:  NO ,
 Gender: F
(75, 74)
face # 2 ,
 Eyes:  Left Eye is open and right eye is closed ,
 Smiling:  YES ,
 Glasses:  NO ,
 Gender: M
(74, 75)
face # 3 ,
 Eyes:  Both eyes are open ,
 Smiling:  YES ,
 Glasses:  NO ,
 Gender: F
(75, 74)
face # 4 ,
 Eyes:  Both eyes are open ,
 Smiling:  YES ,
 Glasses:  NO ,
 Gender: M
(75, 75)
face # 5 ,
 Eyes:  Both eyes are open ,
 Smiling:  YES ,
 Glasses:  NO ,
 Gender: M
```

Fig. 15. image 3 : output

8.4 image 4

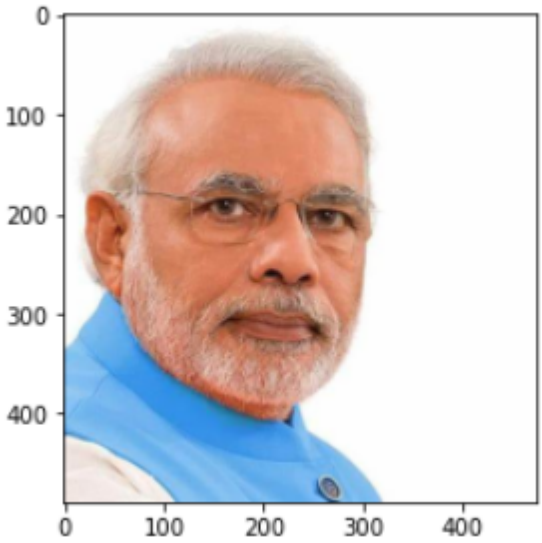


Fig. 16. image 4 : input

```
rectangles[[(118, 162) (504, 547)]]
(385, 386)
face # 0 ,
 Eyes:  Both eyes are open ,
 Smiling:  NO ,
 Glasses:  YES ,
 Gender: M
```

Fig. 17. image 4 : output

9. Applications

Fig. 18. Emotion Recognition

Fig. 19. Facial Recognition Attendance

Fig. 20. Face Unlock

References

- https://arxiv.org/pdf/1602.01827v1.pdf
- https://arxiv.org/pdf/1805.01290v1.pdf
- http://dap.vsb.cz/wsc17conf/Media/Default/Page/online_wsc17_submission_59.pdf
- https://medium.com/themlblog/how-to-do-facial-emotion-recognition-using-a-cnn-b7bbae79cd8f