

CS 584 Machine Learning
Project Report on
Image Classification using Convolution Neural Networks

By Group 23:

Akash Chaturvedi Battula (A20549559)

Raja Sharath Chandra Acha (A20549513)

Chandra Santosh Katakam (A20520290)



Department of Computer Science and Engineering

Illinois Institute of Technology

10 West 35th Street

Chicago, IL

Abstract

In the context of image processing, wavelets can be used to separate the fine details in an image from the coarse details. This is done by applying the wavelet transform to the image, which breaks it down into a set of wavelet coefficients. These coefficients represent the image at different scales and orientations. The key idea behind wavelet compression is that many of the wavelet coefficients are small and can be discarded with little loss of information. This is the “redundancy reduction”. The remaining coefficients can then be efficiently encoded, resulting in a compressed version of the image that requires less storage space. Moreover, because wavelets operate at different scales, they are very good at picking up details and patterns that other methods might miss. This makes them particularly useful for tasks like image recognition, where the ability to pick up on small, localized features can be crucial. Convolutional neural network (CNN) is recognized as state of the art of deep learning algorithm, which has a good ability on the image classification and recognition. The problems of CNN are as follows: the precision, accuracy and efficiency of CNN are expected to be improved to satisfy the requirements of high performance. The main work is as follows: Firstly, wavelet convolutional neural network (WCNN) is proposed, where wavelet transform function is added to the convolutional layers of CNN. Secondly, image classification experiments using CNN, WCNN algorithms, and comparison analysis are implemented with custom dataset of images of popular politicians. The effects of the improved methods are Both precision and accuracy are improved. The mean square error and the rate of error are reduced. The complexity of the improved algorithms is increased.

Introduction

Convolutional neural network (CNN) is a typical deep learning method which is based on feature extraction of convolution calculation. It is widely applied to fields of prediction, classification etc. CNN can solve high-dimensional problems which are difficult for traditional machine learning methods. The ability to minimize the system error between the label and the inference of CNN is much more powerful especially in the application of image processing. The neuron weights of CNN are modified by forward propagation and error back propagation. In recent years, the ability of CNN has become more powerful because the distributed computing power has been greatly improved. Apart from image recognition, CNN is also applied in other fields such as text classification, control system and target tracking. Since the appearance of CNNs in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012, CNNs and other deep neural network architectures have received considerably more attention than in previous years. This interest comes from their impressive results in tasks like image classification, speech recognition, and natural language processing.

The development history of CNN is as follows:

The earliest study about CNN can be traced back to Fukushima, who mimicked the visual cortex of an organism and proposed the Neocognitron model. Time-Delay Neural Network (TDNN) was proposed by Alexander Waibel et al. in 1987. It is proved in TDNN that more hidden layers have greater feature extraction capabilities, which becomes the foundation of further optimization of CNN. After a series of improvements, He-Kaiming et al. released ResNet in 2015, the network manages to skip some neuron nodes to achieve higher performance. In 2017, Gao Huang et al. proposed DenseNet.

Two main factors were necessary to enable the success of CNNs:

The availability of very large datasets of manually annotated in-put data, and the high performance of current computing systems. Large datasets are needed to train the deep neural network parameters until a highly accurate result is reached. CNNs differ from other deep neural networks due to the inclusion of convolutional layers. The outputs of these layers are the result of weighted sums of inputs, like in fully connected layers. The difference strives in the inputs that are involved in the computation of each output, and the fact that convolutional layers feature a smaller

number of weights which are reused for several outputs. Outputs of a convolutional layer are not a function of all the inputs as in fully connected layers; they only depend on a contiguous subset of the inputs (known as receptive field). This considerably reduces the computation cost of these layers, when compared to fully connected layers. It is also interesting that, in convolutional layers, computational and storage costs no longer depend on the number of inputs and outputs.

The computational cost depends on the size of the contiguous subset of inputs, while the reuse of weights reduces the memory required to store them. To implement the weighted sums, convolutional layers use an operation called convolution, which gives them the name. Most of the execution time of a convolutional layer is spent performing convolutions. Since most of the layers in CNNs are convolutional, convolutions account for a large part of the overall network execution time. Several works target the optimization of convolution operations for GPUs. Some perform data transformations to be able to exploit already existing high-performance functions like GEMM. Other works rely on algorithmic optimizations to reduce the computational cost of convolutions, thus improving their performance.

Problems of CNN can be summarized as follows:

- (1) The precision, accuracy and efficiency of CNN are expected to be improved.
- (2) High-dimensional information contains more details, which are difficult to learn such as datasets of MINIST and CIFAR. Even the human brain also tends to ignore the high-dimensional information.
- (3) CNN is more complex than classical neural network, but the trained model of CNN cannot be well explained. It is proved that randomly generated network of CNN can solve difficult problems better than the carefully designed network sometimes. A more intelligent module which can identify more detailed information is expected.

Wavelets:

Haar-Wavelet

Wavelets are essentially a type of multiresolution function approximation that allow for the hierarchical decomposition of a signal or image.

They have been applied successfully to various problems including object detection, face recognition and image retrieval. Any given decomposition of a signal into wavelets involves just a pair of waveforms (mother wavelets). The two shapes are translated and scaled to produce wavelets at different locations and on different scales.

The Haar-like features are first proposed by Papageorgiou et al discussed working with an alternate feature set based on Haar wavelets instead of the usual image intensities. Papageorgiou & Poggio made thorough study of the over complete Haar wavelets for the detection of face, car, and pedestrian. The Haar transform provides a multiresolution representation of an image with wavelet features at different scales capturing different levels of detail; the scale wavelets encode large regions while the fine scale wavelets describe smaller, local regions. The wavelet coefficients preserve all the information in the original image. Fig. 1 shows the various forms of wavelet.

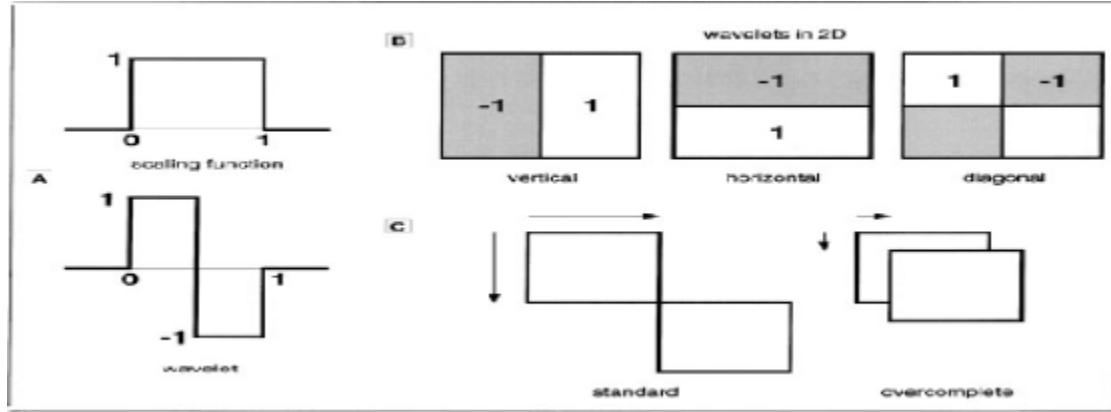


Fig. 1 The haar wavelet framework; (a) the haar scaling function and wavelet, (b) the three types of 2-dimensional non-standard haar wavelets vertical, horizontal, and diagonal, and(c) the shift in the standard transform as compared to our quadruple dense shift.

Viola and Jones adapted the idea of using Haar wavelets and proposed Haar-like rectangle features and the fast evaluation method for face detection. The authors use a set of features which are significant of Haar Basis functions. To compute these features very rapidly at many scales we introduce the integral image representation for images. The simple features used are important of Haar basis functions which have been used by Papageorgiou et al. More specifically, we use three kinds of features. The value of a two-rectangle feature is the difference between the sum of the pixels within two rectangular regions. The regions have the same size and shape and are horizontally or vertically adjacent. A three-rectangle feature computes the sum within two outside rectangles subtracted from the sum in a center rectangle. Finally, a four-rectangle feature computes the difference between diagonal pairs of rectangles.

Wavelet transform (WT) is often used in deep learning. Many features can be obtained by the discrete wavelet transform which has been improved by research. The application fields based on WT and deep learning methods are image classification, computer vision, texture classification etc.

The applications based on wavelet neural network (WNN) in deep learning are as follows:

In 2019, Pengju Liu et al. proposed a Multi-level Wavelet Convolutional Neural Networks (MWCNN), which has been proved to increase the receptive field by reducing the number of maps. The Multi-Path Learnable Wavelet Neural Network for Image Classification was introduced by De Silva et al. This model introduces a multi-path layout with several levels of wavelet decomposition. In the domain of prediction, a convolutional LSTM network using the wavelet decomposition has been proposed in 2018. It takes the wavelet decomposition as the method of feature extraction rather than the manual feature extraction, which has been also proved by Kiskin et al. in 2017.

The advantages of wavelet analysis are as follows:

Wavelet analysis has been widely used in signal processing and analysis. Wavelet analysis method is called mathematical microscope which is considered as a powerful tool for zooming details of sound, image, etc. Although the wavelet transformation has some complexity the powerful detail extraction ability of wavelet transformation is helpful and important to solve the above problems of CNN.

The motivation of this project is to solve CNN's problems based on the advantages of the (Wavelet transform) WT. The importance of the research is that the improvements of CNN neurons are focused. Different from the ability of network with deeper layers, it is believed that the improvements of each neuron of CNN can improve the features identification and learning ability of the whole CNN. Wavelet analysis is adopted to improve the CNN network in this study.

Problem Statement

In this project, we propose to implement a convolutional neural network (CNN), a type of deep learning model, for image classification, a task of assigning a label to an image based on its content, using OpenCV, a library for image processing, computer vision, and machine learning. We also used wavelet transformation, a mathematical tool that decomposes an image into different frequency components, to enhance the image quality and feature representation for CNN. The goal is to overcome the challenges and limitations of image classification, such as high dimensionality, variability, complexity, noise, occlusion, and multi-scale and multi-resolution issues, by using wavelet transformation and CNNs by implementing on the custom dataset of images of popular politicians.

The following sections are organized as follows:

The traditional CNN model is described in the second section. The performance of image classification using CNN and image processing using wavelet transformation and OpenCV is verified, analyzed with custom dataset of images of popular politicians in the third section. Discussion, conclusions, and results are given in the fourth section.

Algorithms

2. Model of convolutional neural network (CNN)

2.1 Structure of CNN

The structure of classical CNN is shown in Fig. 2. There are two parts in CNN: the first part is CPNN, and the second part is FCNN. In CPNN, the first layer is an input layer, and the following layers of CPNN are several pairs of convolutional layers and pooling layers. In FCNN, the first layer is an input layer, and the second layer of FCNN is an output layer, both layers of FCNN are fully connected.

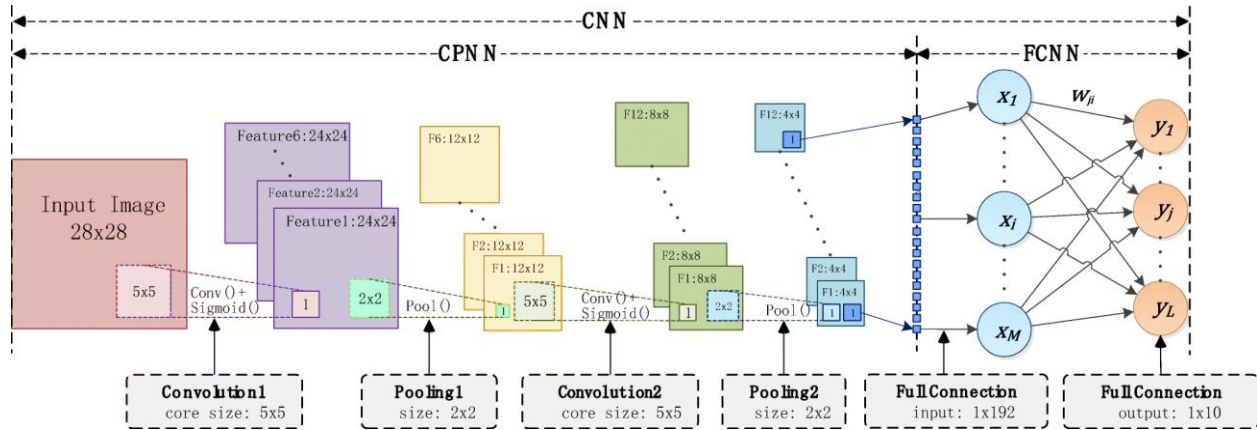


Fig 2: Architecture of CNN

The relation and features of CPNN and FCNN are as follows.

- (1) The input layer of CPNN is the input layer of CNN.
- (2) The last layer of CPNN is the input layer of FCNN.
- (3) The output layer of FCNN is the output layer of CNN.

(4) The activation function of the convolutional layer in CPNN and the output layer in FCNN is sigmoid function. There are not any activation functions in the input layer and pooling layer of CPNN and the input layer of FCNN.

Working of CNN

The input to the network is a 2D image. The network has an input layer which takes the image as the input, output layer from where we get the trained output, and the intermediate layers called the hidden layers. As stated earlier, the network has a series of convolutional and sub-sampling layers. Together the layers produce an approximation of input image data. CNNs exploit spatially local correlation by enforcing a local connectivity pattern between neurons of adjacent layers. Neurons in layer say, 'm' are connected to a local subset of neurons from the previous layer of (m-1), where the neurons of the (m-1) layer have contiguous receptive fields, as shown in below figure.

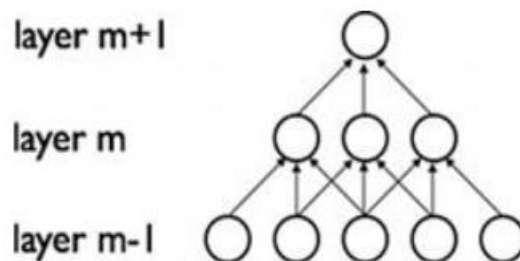


Fig3: Graphical flow of layers showing connection between layers

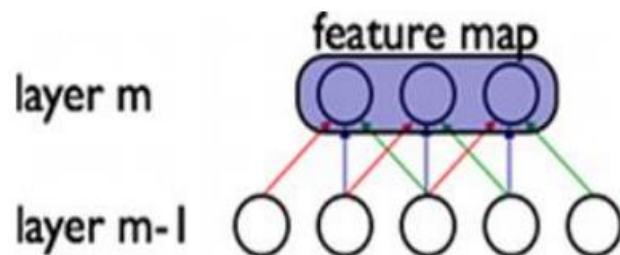


Fig4: Graphical flow of layers showing sharing of weights

2.2 Algorithm of CNN

The algorithm of CNN can be described as follows:

- (1) Initializing weights between layers and bias of neurons.
- (2) Forward propagating.
- (3) Calculating the mean square error (MSE) of all samples according to the loss function.
- (4) Calculating the errors of back propagating for each layer, which are the results of derivation by the chain rule.
- (5) Applying gradient to adjust the weights and bias according to the back-propagated errors.
- (6) Repeating step (2) to step (5) until the MSE is small enough.
- (6) Evaluating accuracy, precision, and efficiency.

2.2.1 Forward propagation of CNN

Forward propagation of CNN is the calculation process from the input layer to the output layer, which can be described as follows:

- (1) The input layer of CNN is filled by a two-dimensional matrix of pixels of an image.
- (2) Forward propagation is calculated in convolutional and pooling layers (CPNN).
- (3) Forward propagation is calculated in fully connected layer (FCNN).

net^l and O^l are the input and the output of neurons in layer l . The output of each neuron can be calculated according to the input and the activation function of each neuron. l is the layer number, e.g. $l = 1$ stands for the first layer, and $l = L$ stands for the last layer. i and j are the row number and column number respectively.

According to the above definition, net^{L-1} , net^{L-2} and net^{L-3} stand for the input of the last FCNN layer, the input of the first FCNN layer and the input of the layer before FCNN (i.e. the last layer of pooling layers) respectively. The data structures of net^l and O^l of each layer of CPNN are two-dimensional matrix, while the net^l and O^l of each layer in FCNN are one-dimensional vectors.

Forward propagation of convolutional layer:

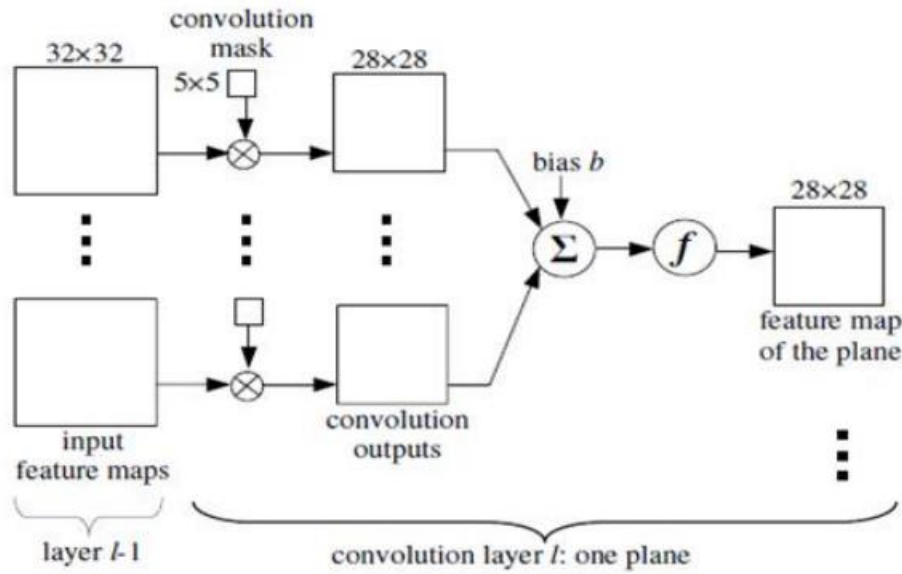


Figure 5: Convolutional layer working

The input of convolutional layer (net^l) can be calculated according to Eq. (1). The net^l_{mn} stands for each input value of neurons in layer l . The convolution (O^{l-1}, w^l, m, n) is the function for convolution calculations. The O^{l-1} is the output of the previous layer. The w^l is the matrix of weights between the input of layer l (net^l) and the output of the previous layer (O^{l-1}). The b^l is the bias of layer l .

$$\begin{aligned}
net_{mn}^l &= \text{convolution}(O^{l-1}, w^l, m, n) + b^l \\
&= \sum_{i=0}^{size^l-1} \sum_{j=0}^{size^l-1} (O_{m+i, n+j}^{l-1} \cdot w_{i,j}^l + b^l)
\end{aligned} \tag{1}$$

An example of convolution operation is provided. If

$$x = \begin{matrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{matrix}, \quad y = \begin{matrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{matrix}, \quad \text{the formula of}$$

convolution(x, y) can be expressed as Eq. (2):

$$\text{convolution}(x, y) = \begin{matrix} x_{11}y_{11} + x_{12}y_{12} + x_{21}y_{21} + x_{22}y_{22} & x_{12}y_{11} + x_{13}y_{12} + x_{22}y_{21} + x_{23}y_{22} \\ x_{21}y_{11} + x_{22}y_{12} + x_{31}y_{21} + x_{32}y_{22} & x_{22}y_{11} + x_{23}y_{12} + x_{32}y_{21} + x_{33}y_{22} \end{matrix} \tag{2}$$

The output of the convolutional layer l (O_{mn}^l) can be calculated as Eq. (3), where $\text{sigmoid}()$ is the activation function.

$$O_{mn}^l = F(net_{mn}^l) = \text{sigmoid}(net_{mn}^l) = \frac{1}{1 + e^{-net_{mn}^l}} \tag{3}$$

Forward propagation of pooling layer:

The function $\text{pool}(x)$ represents the average pooling of matrix x . The formula of $\text{pool}(x)$ can be expressed as Eq. (4). The $size^l$ stands for the size of the pooling window.

$$y_{ij} = \text{pool}(x, i, j) = \frac{\sum_{m=1}^{size^l} \sum_{n=1}^{size^l} x_{size^l \times (i-1) + m, size^l \times (j-1) + n}^{l-1}}{size^l \times size^l} \tag{4}$$

An example of average pooling is provided: If $x = \begin{matrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{matrix}$, The pooling

result is calculated as Eq. (5).

$$\text{pool}(x) = \begin{matrix} \frac{x_{11} + x_{12} + x_{21} + x_{22}}{4} & \frac{x_{13} + x_{14} + x_{23} + x_{24}}{4} \\ \frac{x_{31} + x_{32} + x_{41} + x_{42}}{4} & \frac{x_{33} + x_{34} + x_{43} + x_{44}}{4} \end{matrix} \tag{5}$$

According to Eq. (4), the output of the pooling layer O^l is according to the output of the previous layer (O^{l-1}). In other words, the input of the pooling layer l (net^l) is same as the output of the previous layer (O^{l-1}).

Forward propagation of fully connected layer:

The total number of neurons in the first layer of FCNN ($size^{-2} \times 1$) is equal to the number of neurons of in the last layer of CPNN ($size^{-3} \times size^{-3}$), which can be expressed as $size^{-2} \times 1 = size^{-3} \times size^{-3}$. The output of the first layer of FCNN (O_i^{-2}) is transformed from the output of the last layer of CPNN (O_{mn}^{-3}). The transform relation between O_i^{-2} and O_{mn}^{-3} can be expressed as Eq. (6).

$$O_i^{-2} = O_{mn}^{-3}, \quad m = \text{int}\left(\frac{i}{size^{-2}}\right) + 1, n = i - size^{-2} \times (m - 1) \quad (6)$$

The result of forward propagation is \hat{y}_n , which can be formulated in Eq. (7) to Eq. (9).

$$net_j^{-1} = \sum_{i=1}^{size^{-2}} \left(O_i^{-2} \cdot w_{ij}^{-1} + b^{-1} \right), \quad j = 1, 2, \dots, size^{-1} \quad (7)$$

$$O_j^{-1} = F\left(net_j^{-1}\right) = \text{sigmoid}\left(net_j^{-1}\right) = \frac{1}{1 + e^{-net_j^{-1}}} \quad (8)$$

$$\hat{y}_n = O^{-1} \quad (9)$$

Back propagation of CNN

There are three kinds of back propagation (BP) in CNN algorithm: BP in fully connected layer, BP in pooling layer and BP in convolutional layer.

δ^l is defined as the input error of layer l .

L is the mean square error (MSE) of all samples, which can be formulated as Eq. (10). The closer the values of \hat{y}_n and y_n , the better the training effect is, because \hat{y}_n is the prediction of x_n , and y_n is the label of x_n . If each value of \hat{y}_n is very close to y_n , the value of L will be very small, which means that the training effect is very good, and the trained model has a good fitting.

$$L = \frac{1}{2} \sum_{n=1}^N (\hat{y}_n - y_n)^2 \quad (10)$$

Back propagation of fully connected layer

According to the above definition, δ_j^{-1} is defined as the input error of the last layer of FCNN, which is formulated as Eq. 11. y_n is the labels of training and testing samples. \hat{y}_n is the predictive result of samples. n is the number (ID) of the samples. N is the total number of samples.

$$\delta_j^{-1} = \frac{\partial L}{\partial net_j^{-1}} = \frac{1}{N} \sum_{n=1}^N (\hat{y}_n - y_n) \left(1 - net_j^{-1}\right) net_j^{-1} \quad (11)$$

δ_i^{-2} is defined as the input error of the first layer of FCNN. The size of δ_i^{-1} is $size^{-2} \times 1$. δ_i^{-1} is defined as the back propagation error of previous layer of FCNN (the last layer of CPNN before the first layer of FCNN). The size of δ_{mn}^{-3-3} is $size^{-3} \times size^{-3}$. The transform relation between δ_i^{-2-2} and δ_{mn}^{-3-3} can be expressed as Eq. (12):

$$\delta_{mn}^{-3} = \delta_i^{-2}, \quad i = size^{-2} \times (m - 1) + n \quad (12)$$

The error back propagation from the first layer of FCNN to the last pooling layer in CPNN is expressed as below Eq:

$$\delta_i^{-2} = \frac{\partial L}{\partial net_i^{-2}} = \frac{\partial L}{\partial O_i^{-2}} = \frac{\partial L}{\partial net_j^{-1}} \cdot \frac{\partial net_j^{-1}}{\partial O_i^{-2}} = \sum_{j=1}^{size^{-2}} \delta_j^{-1} \cdot w_{ij}^{-1} \quad i = 1, 2, \dots, size^{-1}$$

Backpropagation of pooling layer

If the layer l is a convolutional layer, the layer $l + 1$ is a pooling layer. Functions of pool calculations can be expressed as Eq. (14) to Eq. (16):

Function of padding(x): matrix x can be expanded with 0 around as Eq. (14):

$$x = \begin{matrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{matrix}, \quad padding(x) = \begin{matrix} 0 & 0 & 0 & 0 \\ 0 & x_{11} & x_{12} & 0 \\ 0 & x_{21} & x_{22} & 0 \\ 0 & 0 & 0 & 0 \end{matrix} \quad (14)$$

Function of rotate(x): matrix x can be rotated 180 degrees as Eq. (15):

$$x = \begin{matrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{matrix}, \quad rotate(x) = \begin{matrix} x_{22} & x_{21} \\ x_{12} & x_{11} \end{matrix} \quad (15)$$

The input error of convolutional layer is calculated by below Eq:

$$\delta_{mn}^l = \frac{\partial L}{\partial net_{mn}^l} = \frac{\partial L}{\partial net_{mn}^{l+1}} \cdot \frac{\partial net_{mn}^{l+1}}{\partial O_{mn}^l} \cdot \frac{\partial O_{mn}^l}{\partial net_{mn}^l} = convolution \left(padding \left(\delta^{l+1} \right), rotate \left(w^l \right) \right)$$

Backpropagation of convolutional layer

The size and data of the output of pooling layer is expanded to the input of pooling layer. For example, matrix x_{uv} (output of pooling layer) is replaced by matrix y_{mn} (input of pooling layer) according to the function poolExpand(x) which is expressed as Eq. (17). $size^l$ is the size of pooling window.

$$y_{mn} = poolExpand(x_{uv}) = \frac{1}{size^l \times size^l} \cdot x_{uv}, \quad u = \text{int} \left(\frac{m-1}{size^l \times size^l} \right) + 1, \quad v = \text{int} \left(\frac{n-1}{size^l \times size^l} \right) + 1$$

For example, if $x = \begin{matrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{matrix}$, the result of $\text{poolExpand}(x)$ is calculated as Eq. (18):

$$\text{poolExpand}(x) = \begin{matrix} \frac{x_{11}}{4} & \frac{x_{11}}{4} & \frac{x_{12}}{4} & \frac{x_{12}}{4} \\ \frac{x_{11}}{4} & \frac{x_{11}}{4} & \frac{x_{12}}{4} & \frac{x_{12}}{4} \\ \frac{x_{21}}{4} & \frac{x_{21}}{4} & \frac{x_{22}}{4} & \frac{x_{22}}{4} \\ \frac{x_{21}}{4} & \frac{x_{21}}{4} & \frac{x_{22}}{4} & \frac{x_{22}}{4} \end{matrix} \quad (18)$$

If the layer l is a pooling layer, then the layer $(l+1)$ is a convolutional layer, and the input error of pooling layer can be calculated as the below Eq:

$$\begin{aligned} \delta_{mn}^l &= \frac{\partial L}{\partial \text{net}_{mn}^l} = \frac{\partial L}{\partial \text{net}_{mn}^{l+1}} \cdot \left(\frac{\partial \text{net}_{mn}^{l+1}}{\partial O_{mn}^l} \right) \cdot \left[\frac{\partial O_{mn}^l}{\partial \text{net}_{mn}^l} \right] = \text{poolExpand} \left(\delta^{l+1} \right) \cdot \left(\frac{\partial \text{net}_{mn}^{l+1}}{\partial O_{mn}^l} \right) \cdot \left[F'(\text{net}_{mn}^l) \right] \\ &= \text{poolExpand} \left(\delta^{l+1} \right) \cdot (1) \cdot [(1 - \text{net}_{mn}^l) \cdot \text{net}_{mn}^l] = \text{poolExpand} \left(\delta^{l+1} \right) \cdot (1 - \text{net}_{mn}^l) \cdot \text{net}_{mn}^l \end{aligned}$$

Adjustment of weights and parameters of CNN

The change value of weights and bias can be calculated as Eq. (20) to Eq. (21):

$$\Delta w^l = \frac{\partial L}{\partial w^l} = \frac{\partial L}{\partial \text{net}^l} \times \frac{\partial \text{net}^l}{\partial w^l} = \delta^l \cdot O^{l-1} \quad (20)$$

$$\Delta b^l = \frac{\partial L}{\partial b^l} = \frac{\partial L}{\partial \text{net}^l} \times \frac{\partial \text{net}^l}{\partial b^l} = \delta^l \quad (21)$$

$$\Delta w_{ij}^{-1} = \frac{\partial L}{\partial w_{ij}^{-1}} = \frac{\partial L}{\partial \text{net}_j^{-1}} \times \frac{\partial \text{net}_j^{-1}}{\partial w_{ij}^{-1}} = \delta_j^{-1} \cdot O_i^{-2} \quad (22)$$

$$\Delta b_j^{-1} = \frac{\partial L}{\partial b_j^{-1}} = \frac{\partial L}{\partial \text{net}_j^{-1}} \times \frac{\partial \text{net}_j^{-1}}{\partial b_j^{-1}} = \frac{1}{\text{size}^l} \sum_{j=1}^{\text{size}^{-1}} \delta_j^{-1} \quad (23)$$

The updated value of weights and bias can be calculated as Eq. (24) to Eq. (27). η_CPNN is the learning rate of CNN:

$$w^l(t+1) = w^l(t) - \eta_{CPNN} \times \Delta w^l \quad (24)$$

$$b^l(t+1) = b^l(t) - \eta_{CPNN} \times \Delta b^l \quad (25)$$

$$w_{ij}^{-1}(t+1) = w_{ij}^{-1}(t) - \eta_{CPNN} \times \Delta w_{ij}^{-1} \quad (26)$$

$$b^{-1}(t) = b^{-1}(t) - \eta_{CPNN} \times \Delta b_j^{-1} \quad (27)$$

3. Wavelet transform

Wavelet transform (WT) is an ideal method to process details of signals. WT provides a Time-Frequency Window which can capture higher and lower resolution of details of signals. The problem of Fourier transform (FT) is that the window size cannot be changed when the frequency is changed. This problem can be solved by WT. The $\psi(a, b)$ is called wavelet generating function, which can be expressed as :

$$\psi(a, b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} f(t) * \varphi\left(\frac{t-b}{a}\right)$$

where a and b are the scale parameters which control the extension and translation of function.

$\psi(a, b)$ is designed according to the following conditions

(1) Only in a very small domain, the function value is not 0, and other domains are 0. In other words, translating the signal in timeline is same as adding a window on the original signal.

(2) The integral value of function in the x axis must be 0.

(3) The transform must be reversible. **There are many wavelet generating functions such as:** (1) haar wavelet, (2) db wavelet, (3) sym wavelet, (4) coif series wavelet, etc. The wavelet function of this study is:

$$\varphi(x) = \cos(1.75t) * e^{-\frac{t^2}{2}}.$$

The processes of wavelet transform are visualized as follows:

In Fig. 2, the error is the difference between the signal of wavelet inverse transformation and the original signal. The scale is the parameter of wavelet function, which controls the extension of wavelet function. When the scale parameter is changed, the wavelet transform's ability of information extraction to original signal is changed.

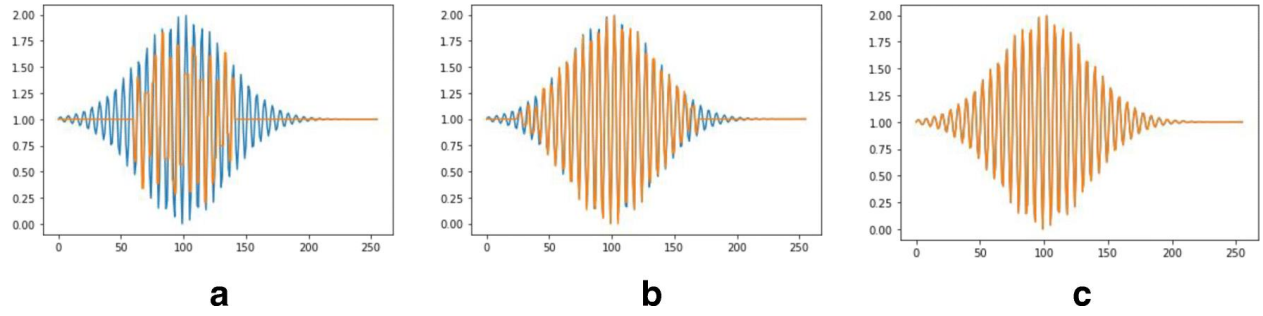


Fig 6: Different effects of WT. **a.** scale = 20, error = 2.08. **b.** scale = 100, error = 0.044. **c.** scale = 200, error = 0.00045.

In summary, by adjusting the scale and translation, wavelet transform can learn the different feature. So, richer features can be learned by adding the wavelet transformation into the CNN.

4. Experiment

Instructions on the experiment

The experiment is to classify images of various celebrities using convolutional neural networks. The neural networks setup is achieved using TensorFlow and CUDA distributed computed framework. The experiment can be summarized into two parts namely:

1. Data Cleaning
2. Training & Testing the CNN model

Data Cleaning

The dataset used is a set of 736 high quality images of collected from the internet, mostly from google. The images comprise of the photographs of various celebrities famous in the world. The data consists of the faces of the following persons.

1. BillGates,
2. Xi Jinping,
3. Kim jong un,
4. Fidel Castro,
5. Barack Obama,
6. Vladimir putin,
7. Donald Trump,
8. Narendra Modi,
9. Nelson mandela,
10. Joe Biden

The idea to choose only politicians is for the following reasons:

1. For just convenience so that all the classes for classification belong to category and
2. It is easy to get the data online.
3. We have decided to use HD images of faces instead of going for already available image datasets online

During the process of cleaning data all the images of the corresponding classes are put in their respective folders and each image is read one after the other. Each image is then used to identify if it has a face and two eyes using OpenCV face cascading techniques. If the image passes the test of at least one face and two eyes, it is then cropped to extract the face and then stored in another folder named after its class. This process is followed, and every image is filtered.

The outcome of the process is the list of folders named after their classes. With each folder containing the cropped faces of all the classes i.e., persons.

```
[1]: cropped_image_dirs = ['./dataset/cropped/BillGates', './dataset/cropped/jin_
    ping', './dataset/cropped/Kim jong un', './dataset/cropped/Fidel Castro',
    './dataset/cropped/Obama', './dataset/cropped/putin', './dataset/cropped/
    Trump', './dataset/cropped/Modi',
    './dataset/cropped/Nelson mandela', './dataset/cropped/joe biden']
```

The above image describes the folder structure of the images after filtering the data.

What is OpenCV and why did we use it?

OpenCV is an open-source computer vision and machine learning software library. It's widely used for various tasks in image processing, including facial recognition and detection. One of the essential tools within OpenCV for facial detection is the concept of cascading classifiers.

Cascading classifiers, particularly Haar cascades, are a way to identify objects within an image. They work by using a series of classifiers arranged in a hierarchy (cascade) to efficiently detect objects. In the case of facial features, these classifiers are trained to recognize specific patterns that represent various parts of a face, such as eyes, nose, mouth, etc.

```
[4]: '''  
face_cascade = cv2.CascadeClassifier('./opencv/haarcascades/  
↳haarcascade_frontalface_default.xml')  
eye_cascade = cv2.CascadeClassifier('./opencv/haarcascades/haarcascade_eye.xml')
```

Original Image

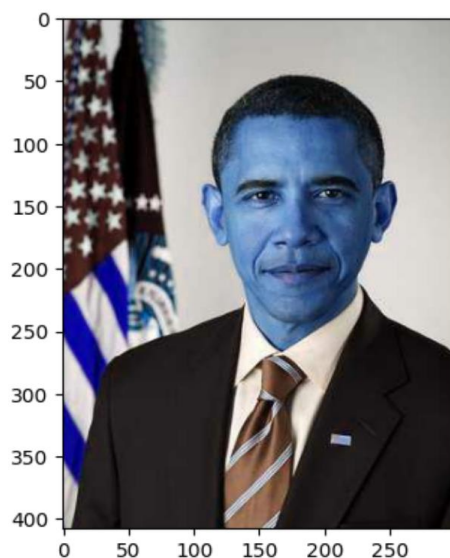
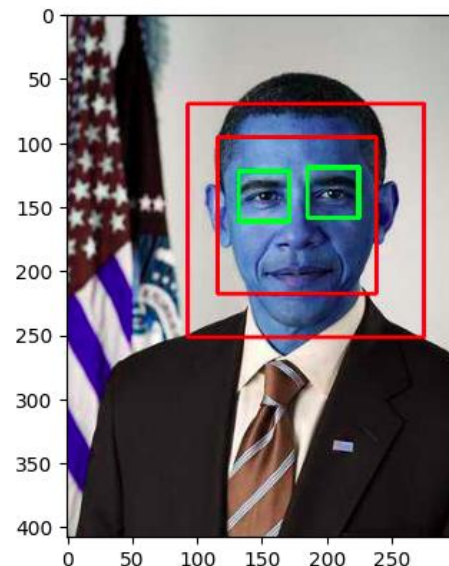


Image with Face and eyes detected.



The above images represent the OpenCV face and eyes detection. After detecting and cropping the faces we will train the model using neural networks.

Training and testing the CNN model:

This step involves two steps.

1. Wavelet transformation of image
2. CNN model building

Wavelet transformation: We then perform wavelet transformation of the images to extract the sharp features of the faces of the people. This step is necessary because that faces, unlike other material objects are very similar to each other. To perform this, we used **Pywavelets** package in python. The below method essentially performs a wavelet

decomposition of the input image, manipulates the wavelet coefficients (in this case, zeroing out the approximation coefficients), and reconstructs the modified coefficients to obtain the transformed image.

For the current experiment we used Haar wavelet transformation function for extracting the coefficients.

Wavelet transform

```
[5]: '''
import numpy as np
import pywt
import cv2

def w2d(img, mode='haar', level=1):
    imArray = img
    #Datatype conversions
    #convert to grayscale
    imArray = cv2.cvtColor( imArray,cv2.COLOR_RGB2GRAY )
    #convert to float
    imArray = np.float32(imArray)
    imArray /= 255;
    # compute coefficients
    coeffs=pywt.wavedec2(imArray, mode, level=level)

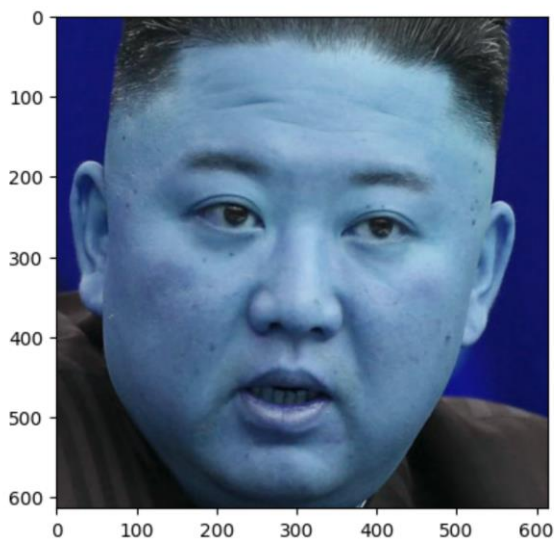
    #Process Coefficients
    coeffs_H=list(coeffs)
    # print(coeffs_H.shape)
    # print(len(coeffs_H))

    # coeffs_H[0] *= 0;
    coeffs_H[0] *= 0;

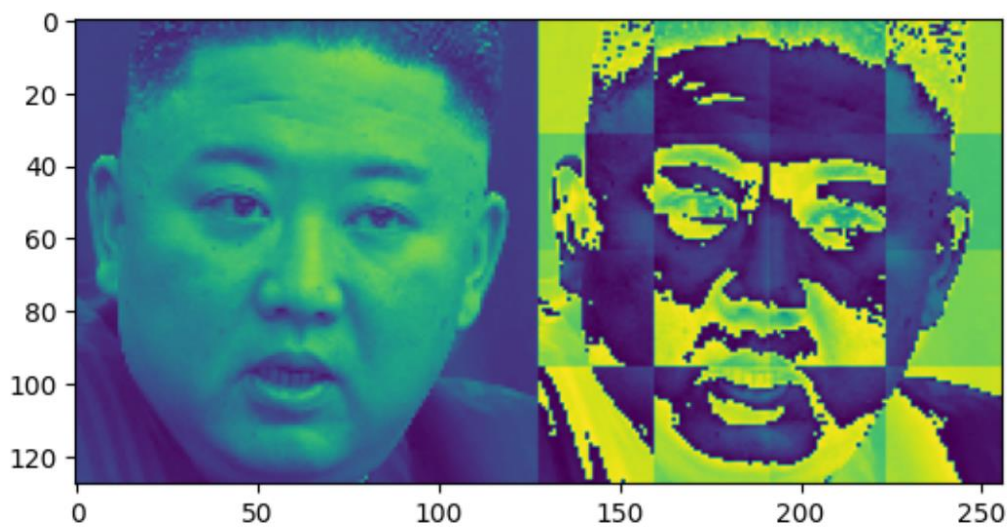
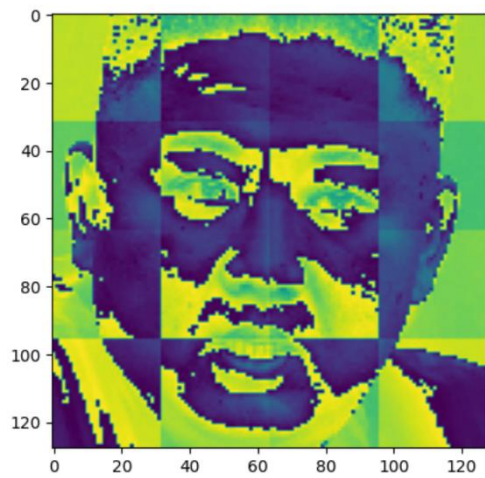
    # reconstruction
    imArray_H=pywt.waverec2(coeffs_H, mode);
    imArray_H *= 255;
    imArray_H = np.uint8(imArray_H)

    return imArray_H
```

Original Image:



Wavelet Transformed image of the above image.



horizontally stacked training Image

Each image in the dataset undergoes the following filters.

1. Scaled to 128*128 pixels.
2. Converted into its wavelet transform with zeroing the approximation coefficients.
3. Original image and wavelet transformed image are horizontally stacked to get training image.
4. Each image is of the size (128, 256, 1).

Train and Test Data Dimensions

- The training data set is of the size (552, 128, 256, 1)
- The testing data set is of the size (184, 128, 256, 1)
- Total data set is of the size (736, 128, 256, 1)

Model Design

1. **Input Shape:** The variable `input_shape` represents the shape of the input data. In this case, it's a grayscale image with dimensions 128x256 and a single channel (1 for grayscale).

2. Model Architecture:

Sequential Model: This is a linear stack of layers in the neural network.

Convolutional Layers: The model adds three convolutional layers successively:

1. **The first layer** (Conv2D) has 32 filters of size 3x3, using ReLU activation.
2. **The second layer** (Conv2D) has 64 filters of size 3x3, also using ReLU activation.
3. **The third layer** (Conv2D) has 128 filters of size 3x3, with ReLU activation.
4. Each convolutional layer applies a set of filters to the input image, extracting features through convolutions.

MaxPooling Layers: After each convolutional layer, a max-pooling layer (MaxPooling2D) with a window size of 2x2 is applied. Max-pooling reduces the spatial dimensions, reducing computation and extracting dominant features.

Flatten Layer: This layer flattens the output from the convolutional layers into a 1D array to prepare for the fully connected layers (Dense layers).

Fully Connected (Dense) Layers: The flattened output is fed into dense layers.

- **Three dense layers** with 3000, 1000, and 500 neurons respectively, ReLU activation functions.
- **Dropout layers** (Dropout (0.2)) are added after each dense layer with a dropout rate of 0.2. Dropout helps prevent overfitting by randomly dropping a fraction of neurons during training.

Output Layer:

The final dense layer has the number of neurons equal to the number of classes in the classification task (represented by `len(class_dict)`), using a softmax activation function. Softmax ensures the output represents class probabilities, enabling multi-class classification.

Hyperparameters

Optimizer is **'SGD'** or stochastic gradient descent,

loss is **'sparse_categorical_crossentropy'**,

metrics list is **['accuracy']**

number of epoches is 30.

The optimizer chosen is Stochastic Gradient Descent ('SGD'). SGD is a classic optimization algorithm used for training neural networks. It updates the network's parameters in the direction that minimizes the loss function. The specified loss function is 'sparse categorical entropy'. This loss function is commonly used for multi-class classification problems where the labels are integers (not one-hot encoded).

```
[18]: '''  
print('1 -----')  
# optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)  
ann.compile(optimizer='SGD',  
            loss='sparse_categorical_crossentropy',  
            metrics=['accuracy'])  
print('2 -----')
```

```
[19]: '''  
print(X_train.shape, y_train.shape)  
ann.fit(X_train, y_train, epochs=30)  
print('3 -----')
```

5. Result Analysis

The model is then converged in 30 epoches with the following the metrics.

Epoch 30/30

18/18 [=====] - 2s 87ms/step - loss: 0.0257 - accuracy: 0.9982

Loss = 0.0257

Accuracy = 0.9982

Confusion matrix using the test data is:

Support indicates the spread of the data of each class. Class 8 and 5 are little under sampled but the rest are almost of the same range.

6/6 [=====] - 1s 81ms/step

Classification Report:

	precision	recall	f1-score	support
0.0	1.00	0.89	0.94	19
1.0	1.00	1.00	1.00	15
2.0	1.00	1.00	1.00	18
3.0	1.00	1.00	1.00	17
4.0	1.00	1.00	1.00	29
5.0	1.00	1.00	1.00	11
6.0	0.91	1.00	0.95	20
7.0	1.00	1.00	1.00	21
8.0	1.00	1.00	1.00	10
9.0	1.00	1.00	1.00	24
accuracy			0.99	184
macro avg	0.99	0.99	0.99	184
weighted avg	0.99	0.99	0.99	184

In the above image the classification indexes correspond to the following names of the persons

'BillGates': 0,
'Jin ping': 1,
'Kim Jong un': 2,
'Fidel Castro': 3,
'Obama': 4,
'Putin': 5,
'Trump': 6,
'Modi': 7,
'Nelson Mandela': 8,
'Joe Biden': 9

The F1 score is a metric that combines the precision and recall of a model into a single value. It's especially useful when dealing with imbalanced classes (when the number of samples in different classes varies widely).

The F1 scores are also very good. 1 for all except 0.95 for class 6 and 0.94 for class 0

precision of class 6 is 0.91 and rest are just 1.

recall of class 0 is 0.89 and rest are just 1.

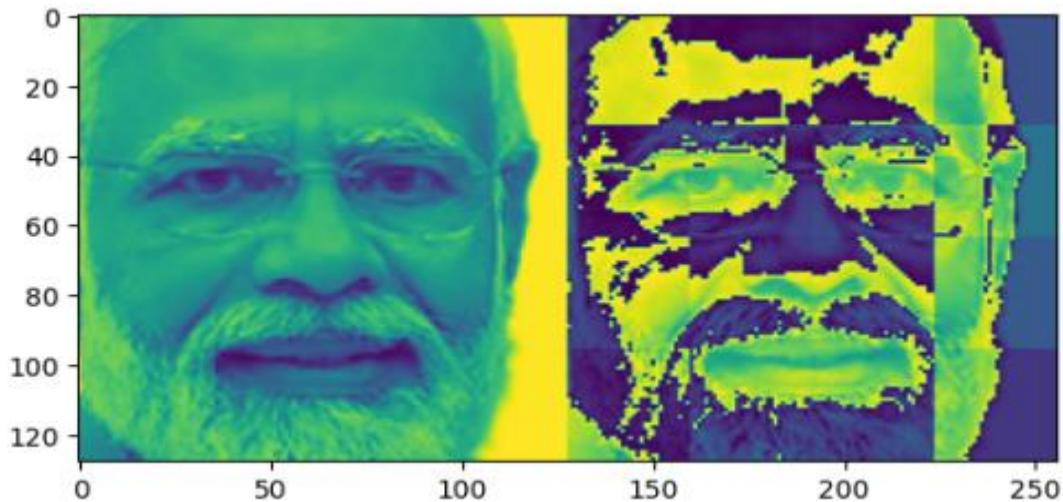
```
face = getFace('./politiciantest/mot.jpg')  
print(face)
```

The above figure shows that reading the image in the format of training data.

Input image:



Output:



We read the image of the person named Narendra Modi and convert the image into wavelet transformed image by zeroing out approximate coefficients and then stacked the original and wavelet image horizontally as shown in the above figure.

```
1 (128, 256)
(1, 128, 256)
(1, 128, 256, 1)
1/1 [=====] - 0s 24ms/step
[[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]]
done
Modi
```












The dimensions of the image are listed above as (128, 256) pixels and we feed this image to the above trained CNN model in the format of (1, 128, 256, 1). We now predict the output of the model for this data to get an output of 10 probabilities. The above results we got is [0, 0, 0, 0, 0, 0, 0, 1, 0, 0].

This result can be interpreted as list of probabilities for each class. So, the maximum probability can be interpreted as the index of the label predicted by the model. From the above result we can say that the maximum probability is given by index 7 which is the nothing but Narendra Modi.

Additional resource:

<https://drive.google.com/drive/folders/1d-Y-qo2hsQuWflcDNtEqsnHio7RuggQ0?usp=sharing>

The above link has the dataset we used, trained model, notebooks for data cleaning and model building.

Name	Owner
 dataset	 me
 poitician_classification_CNN_algorithm.ipynb 	 me
 poitician_classification_CNNsuccess_Data_cleaning.ip... 	 me
 politician.h5 	 me

Plots:

Gradient Descent

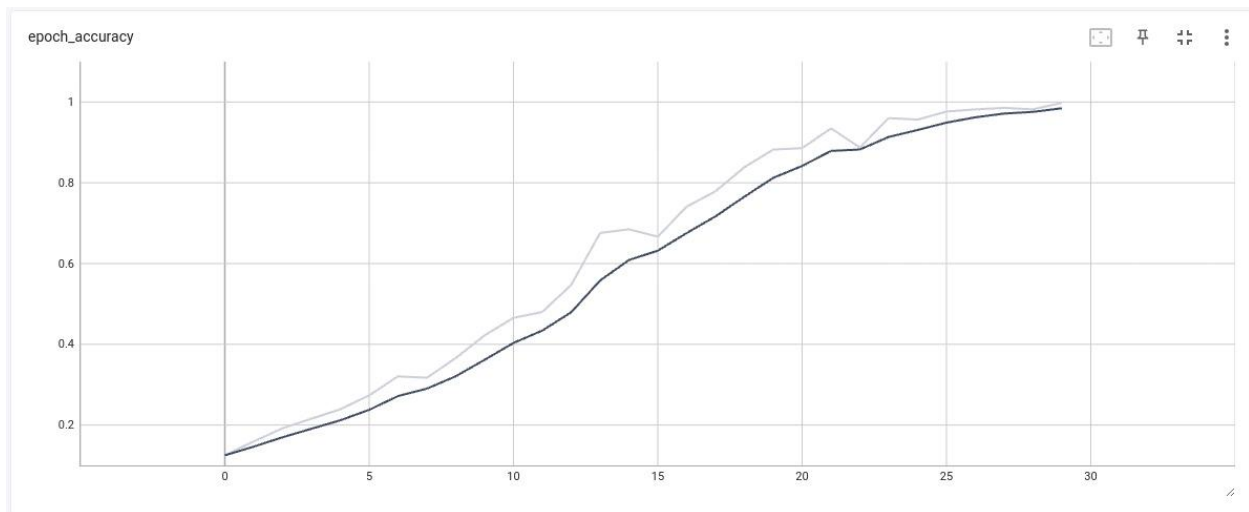


Fig: Increase in accuracy of the model as epoches increases

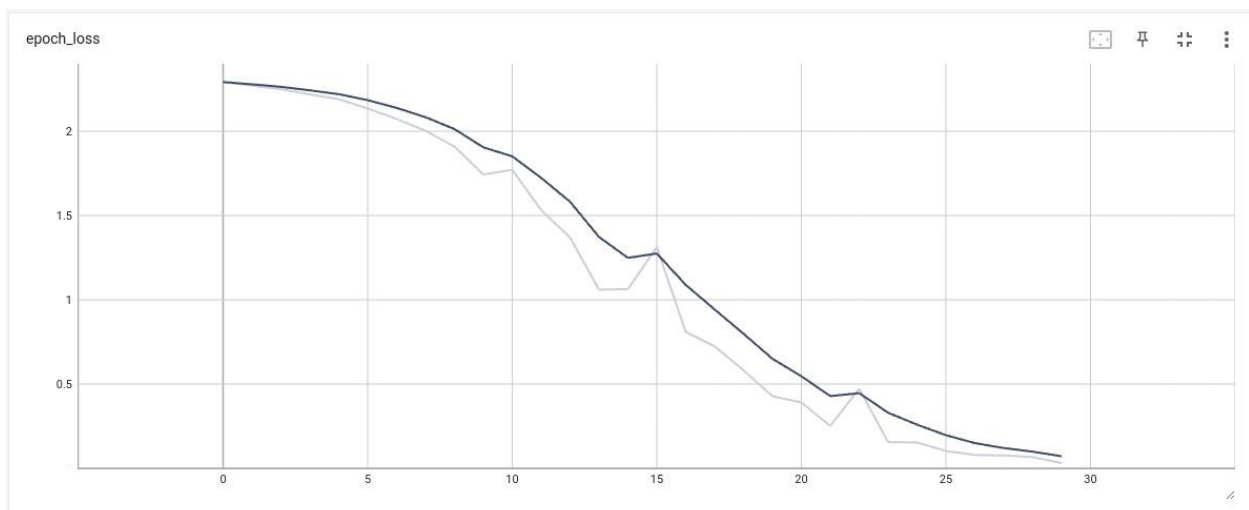


Fig: Decrease in loss of the model as epoches increases

Convolution Layers:

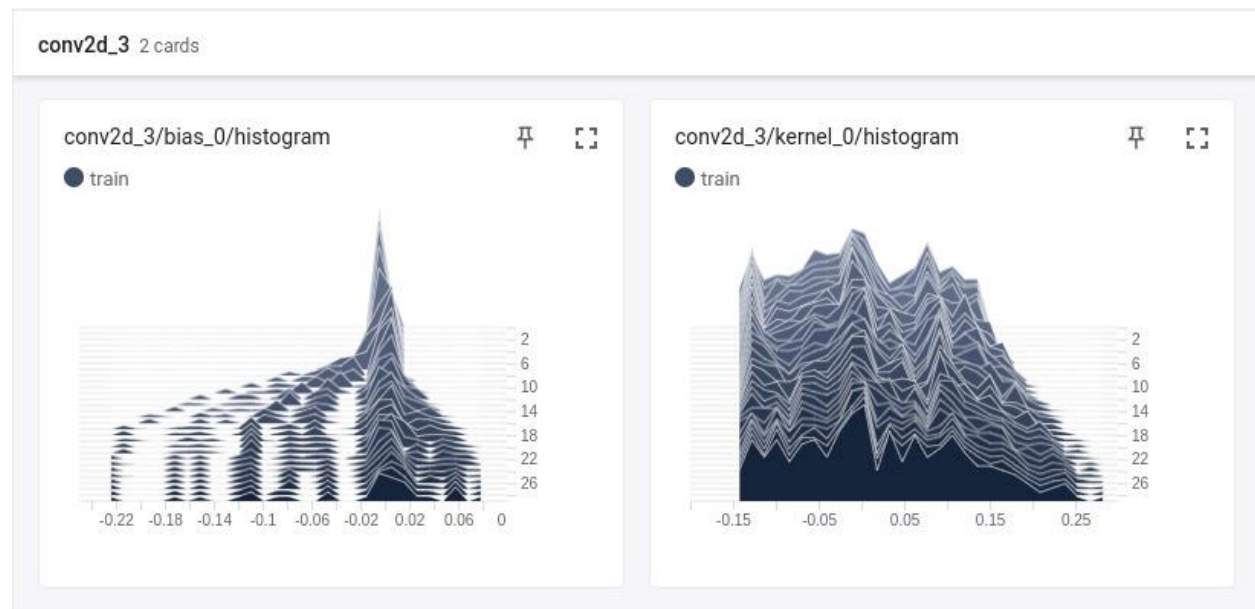


Fig: The graph consists of two histograms, one for the bias and one for the kernel of the conv2d_3 layer.

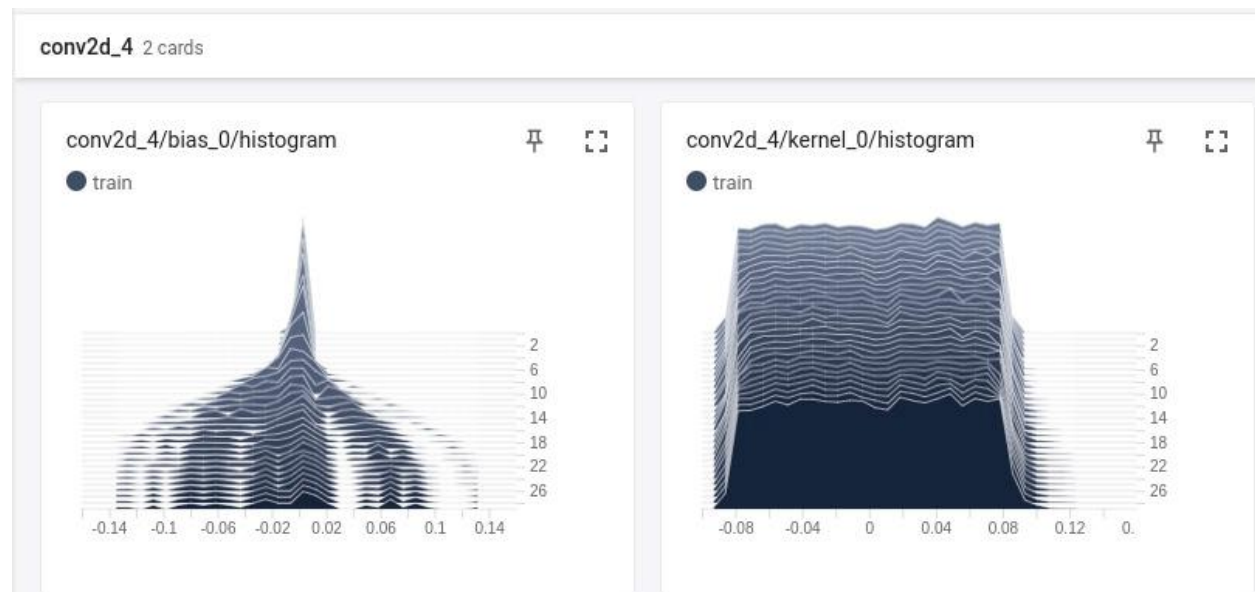


Fig: The graph consists of two histograms, one for the bias and one for the kernel of the conv2d_4 layer.

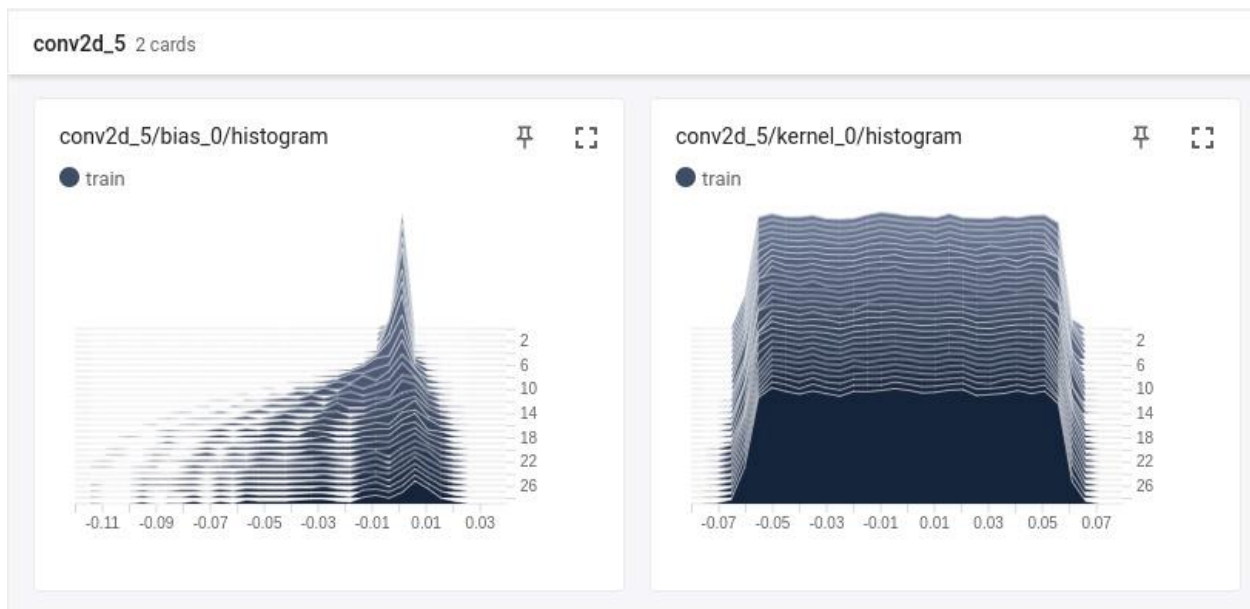


Fig: The graph consists of two histograms, one for the bias and one for the kernel of the conv2d_5 layer.

Neural Network Layers:

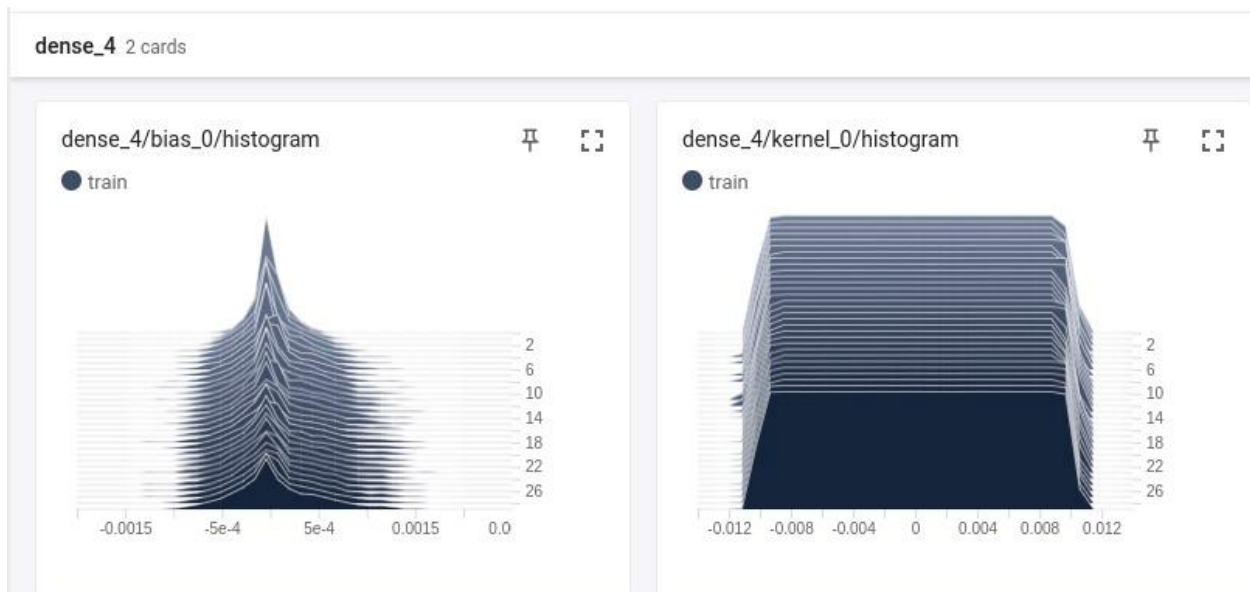


Fig: The graph consists of two histograms, one for the bias and one for the kernel of the dense_4 layer.

dense_5 2 cards

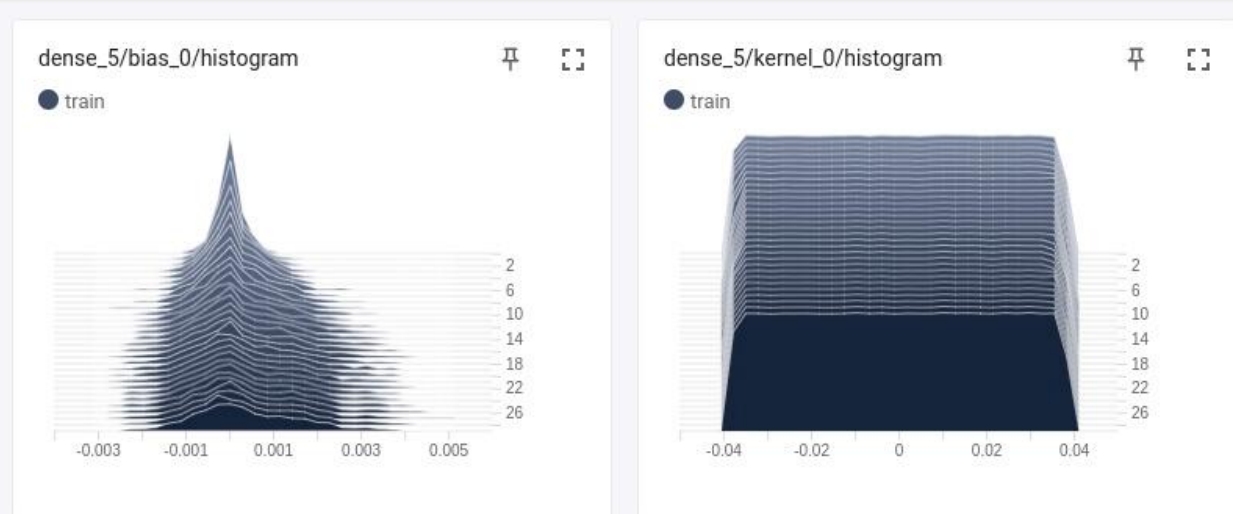


Fig: graph consists of two histograms, one for the bias and one for the kernel of the dense_5 layer.

dense_6 2 cards

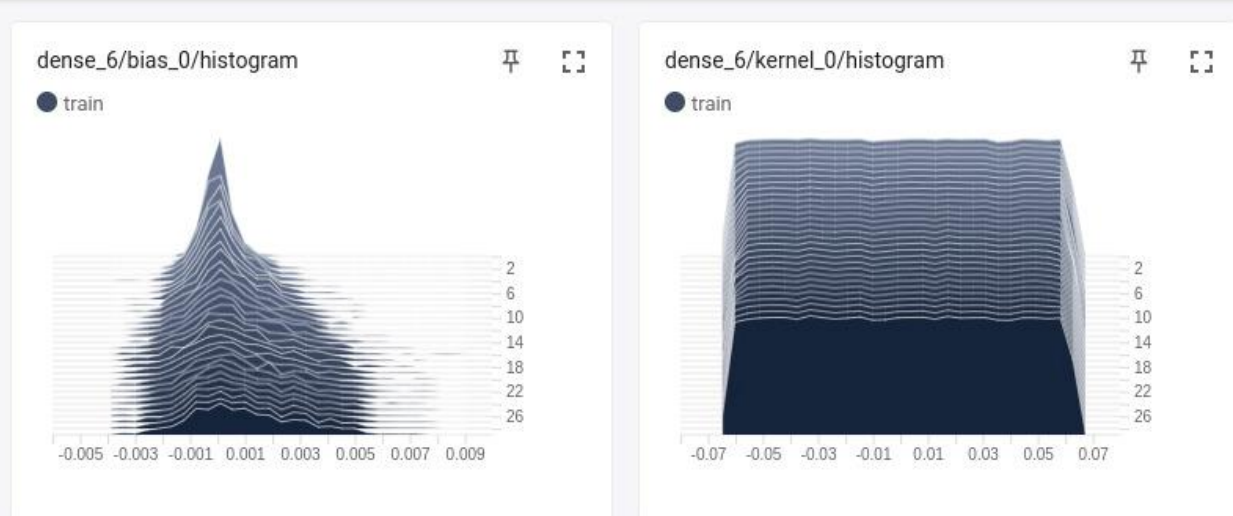


Fig: The graph consists of two histograms, one for the bias and one for the kernel of the dense_6 layer.

dense_7 2 cards

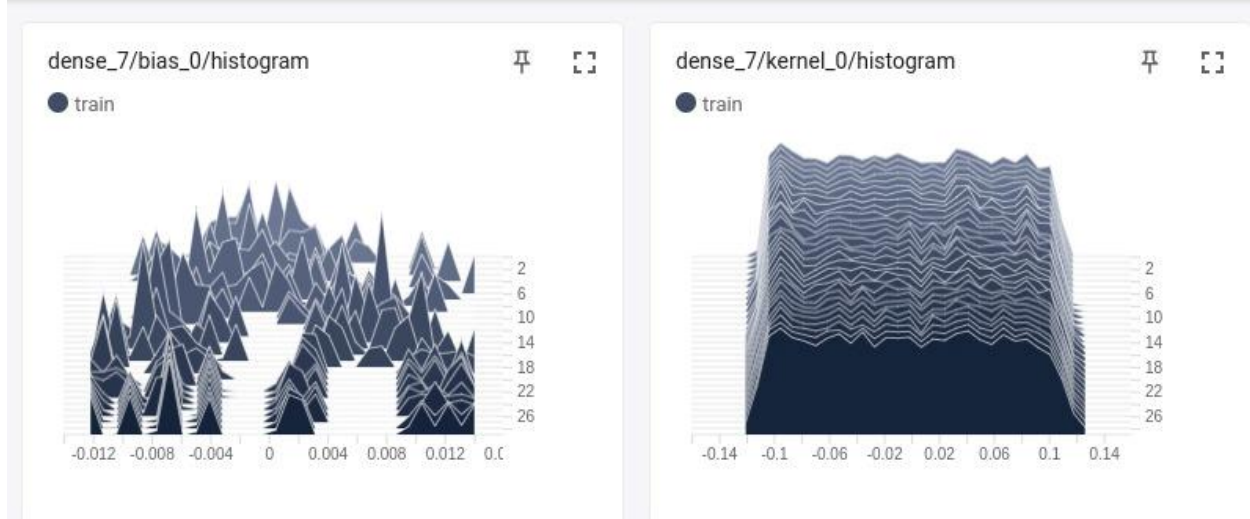


Fig: The graph consists of two histograms, one for the bias and one for the kernel of the dense_7 layer, which is the last layer of the network.

The above graphs show how a convolutional neural network (CNN) learns to classify images by adjusting the weights and biases in its layers. The weights and biases are the parameters that determine the output of the network. The graph consists of two histograms, one for the bias and one for the kernel of the dense layer (there are 4 dense layers). The histograms show the distribution of the values of the bias and kernel along a range of values, and how the distribution changes over time as the network trains on more data. The histograms are color-coded from dark blue to light blue, where dark blue represents the earliest step and light blue represents the latest step. The histograms show that as the network trains on more data, the distribution of the bias and kernel values becomes more stable and less noisy, which means that the network is converging to a set of optimal parameters that minimize the loss function and maximize the accuracy on the task of image classification.

References

- [1] Viola and Jones, "Rapid object detection using boosted cascade of simple features", Computer Vision and Pattern Recognition, 2001
 - [2] Papageorgiou, Oren and Poggio, "A general framework for object detection", International Conference on Computer Vision, 1998
 - [3] Papageorgiou, C, & Poggio, T. (2000). "A trainable system for object detection". International Journal of Computer Vision, 38 (1)
 - [4] Dalal, N. (2006). Finding people in images and videos [unpublished doctoral dissertation]. Institute National Polytechnique Grenoble.
 - [5] Liu P, Zhang H, Lian W, Zuo W (2019) Multi-level wavelet convolutional neural networks. IEEE Access 7:74973–74985
 - [6] De Silva D, Vithanage H, Fernando K, Piyatilake I (2020) Multi-path learnable wavelet neural network for image classification. In: Twelfth International Conference on Machine Vision (ICMV 2019), vol. 11433, p. 114331O. International Society for Optics and Photonics Res improv wavelet convolutional wavelet neural netw 35
 - [7] Wang F, Yu Y, Zhang Z, Li J, Zhen Z, Li K (2018) Wavelet decomposition and convolutional lstm networks based improved deep learning model for solar irradiance forecasting. Appl Sci 8(8):1286 36 Jing-Wei LIU1,2 et al.
 - [8] Kiskin I, Orozco BP, Windebank T, Zilli D, Sinka M, Willis K, Roberts S (2017) Mosquito detection with neural networks: the buzz of deep learning. arXiv preprint arXiv:1705.05180
 - [9] Pati YC, Krishnaprasad PS (1993) Analysis and synthesis of feedforward neural networks using discrete affine wavelet transformations. IEEE Trans Neural Netw 4(1):73–85
 - [10] Sifuzzaman M, Islam M, Ali M (2009) Application of wavelet transform and its advantages compared to fourier transform
- https://docs.opencv.org/4.x/d2/d99/tutorial_js_face_detection.html
- <https://pywavelets.readthedocs.io/en/0.2.2/ref/dwt-discrete-wavelet-transform.html#multilevel-decomposition-using-wavedec>
- https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html?highlight=haar
- <https://www.kaggle.com/code/sifboudjellal/face-recognition-using-cnn>
- <https://www.semanticscholar.org/paper/Face-Recognition-Based-on-Haar-Wavelet-Transform-Alwakeel-Arabia/1eea2c9c252e047cc3af88972c8109ff93e473da>

<https://www.slideserve.com/kendall/edge-detection-and-wavelet-transform>

<https://www.section.io/engineering-education/face-recognition-using-wavelet-features/>

[https://www.researchgate.net/publication/283211538 Face Image Resolution Enhancement based on Weighted Fusion of Wavelet Decomposition](https://www.researchgate.net/publication/283211538_Face_Image_Resolution_Enhancement_based_on_Weighted_Fusion_of_Wavelet_Decomposition)