జాతీయ సాంకేతిక విజ్ఞాన సంస్థ వరంగల్
राष्ट्रीय प्रौद्योगिकी संस्थान वरंगल
National Institute of Technology Warangal
An Institute of National Importance

CNN

Machine Learning Algorithms & Applications

(Faculty: Prof. J. Ravi Kumar)

Notes prepared by:
Veludandi Sai Kiran
24ECM1R17
M.Tech in Embedded & Machine Learning Systems,
Department of Electronics & Communication Engineering,
National Institute of Technology,
Warangal.

CONVOLUTIONAL NEURAL NETWORKS (CNN)

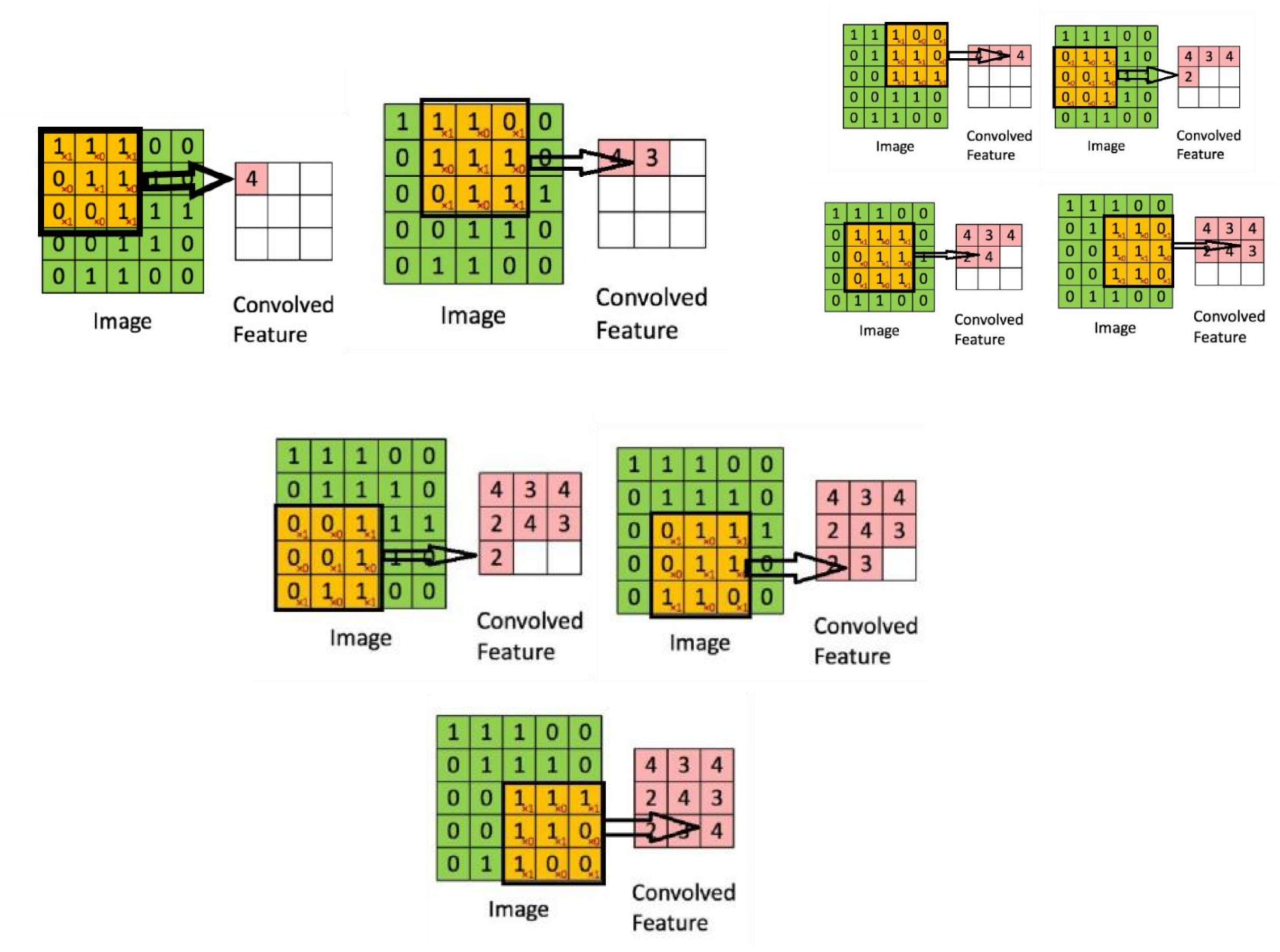
A **Convolutional Neural Network (CNN)** is a type of deep learning model designed for processing structured grid data, such as images. It is widely used for computer vision tasks like image recognition, object detection, and image segmentation. Key components of CNN include:

(i) Convolution Layer

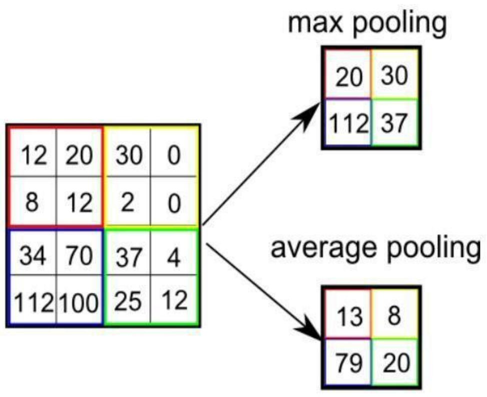
(ii) Pooling Layer

(iii) Fully Connected Layer

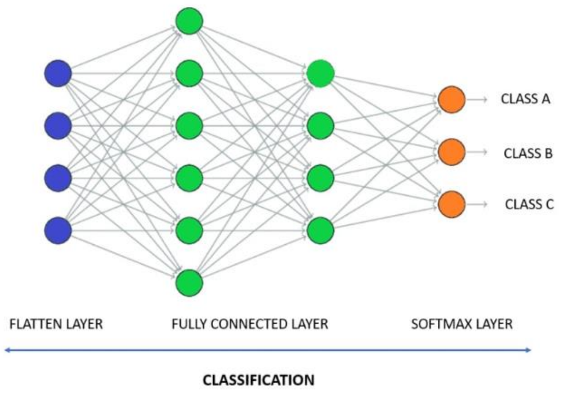
Convolution Layer: The **convolution layer** in a CNN is like a filter that scans an image to detect specific patterns, such as edges, corners, or textures. It uses small grids called kernels to slide over the image, performing mathematical operations to create feature maps. These feature maps highlight important parts of the image while reducing unnecessary details.



Pooling Layer: Same as the Convolutional Layer, Pooling layer is responsible for spatial size reduction of the Convolution Feature. By using this the computational power required to process the information through dimensionality reduction can be decreased. Furthermore, dominant features such as rotational and positional invariant are extracted by pooling layer, thus effective training of the dataset is maintained effectively. There are two different kinds of Pooling known as Average Pooling and Max Pooling. Max Pooling gives higher value from the image portion which is covered by the kernel. On the other side, Average Pooling gives the average of all the values from the image portion covered by Kernel.



Fully Connected Layer: This layer is responsible for learning non-linear function. Now with the help of above layers we have converted the input image into a suitable form for Multi-Level layer, in this we levelled the image into a column vector. The levelled output is given to a neural network and back propagation applied to every step of training process. Combine features for classification or regression tasks.



Aim: The aim of is to develop a comprehensive understanding of the internal computations and architecture of deep learning models, specifically Convolutional Neural Networks (CNNs) and Artificial Neural Networks (ANNs), by performing manual calculations and analyzing the model summary.

(1) Conduct detailed hand calculations using a randomly generated input matrix of size 11×7,applying convolution operations step-by-step with a specified kernel, stride, and padding.

(2) Analyze the CNN model summary to manually compute the total number of parameters in each layer, including convolution, pooling, and fully connected layers, to validate the architecture

(3) Perform hand calculations for a 2 layer fully connected network (2-2-2 architecture).

(4) Compute the forward pass by calculating the weighted sum and applying activation functions at each neuron.

(5) Backpropagate errors to compute gradients for parameter updates.

Problem Statement: To demonstrate a detailed understanding of the mathematical operations involved in deep learning models, this project involves performing hand calculations for a Convolutional Neural Network (CNN) and an Artificial Neural Network (ANN). The CNN calculations are based on a randomly generated input matrix of size 11×7, and the ANN is structured as a 2-layer fully connected network with 2 neurons per layer. Additionally, the model summary of the CNN has been manually analyzed and solved to validate the architecture and parameter computations. This approach ensures a clear understanding of the underlying processes in these models.

Hand Calculations for 2-2-2 Neural Network

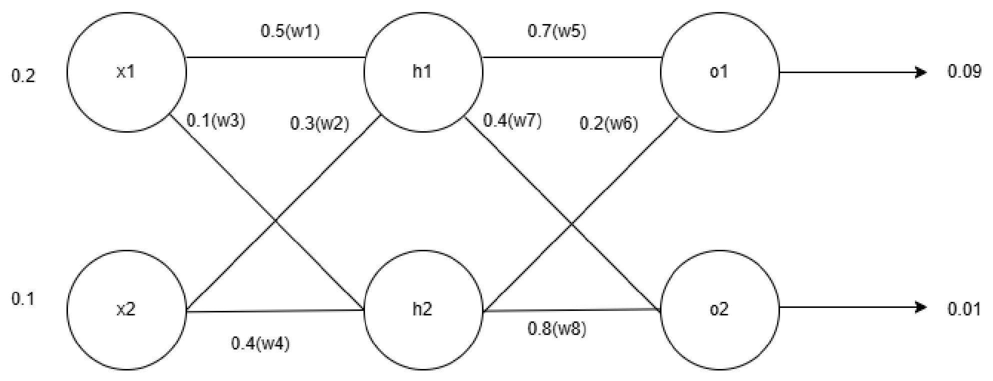


Figure 1: 2-2-2 ANN

Given:

- Inputs: $x_1 = 0.2$, $x_2 = 0.1$
- Outputs: $o_1 = 0.09$, $o_2 = 0.01$
- Weights:
  - $w_1 = 0.5$
  - $w_2 = 0.3$
  - $w_3 = 0.1$
  - $w_4 = 0.4$
  - $w_5 = 0.7$
  - $w_6 = 0.2$
  - $w_7 = 0.4$
  - $w_8 = 0.8$
- Biases:
  - $b_1 = 0.3$ (for hidden layer)
  - $b_2 = 0.5$ (for output layer)
- Activation function: Sigmoid, $f(z) = \frac{1}{1+e^{-z}}$

Step 1: Calculating Hidden Layer Activations

The intermediate hidden layer activations $h_1^{(\text{in})}$ and $h_2^{(\text{in})}$ are calculated as follows:

$$h_1^{(\text{in})} = w_1x_1 + w_2x_2 + b_1 = (0.5)(0.2) + (0.3)(0.1) + 0.3 = 0.43$$
$$h_2^{(\text{in})} = w_3x_1 + w_4x_2 + b_1 = (0.1)(0.2) + (0.4)(0.1) + 0.3 = 0.36$$

Applying the sigmoid activation function to get $h_1^{(\text{out})}$ and $h_2^{(\text{out})}$:

$$h_1^{(\text{out})} = f(h_1^{(\text{in})}) = \frac{1}{1+e^{-0.43}} \approx 0.605$$
$$h_2^{(\text{out})} = f(h_2^{(\text{in})}) = \frac{1}{1+e^{-0.36}} \approx 0.589$$

Step 2: Calculating Output Layer Activations

The intermediate output layer activations $o_1^{(\text{in})}$ and $o_2^{(\text{in})}$ are calculated as follows:

$$o_1^{(\text{in})} = w_5h_1^{(\text{out})} + w_6h_2^{(\text{out})} + b_2 = (0.7)(0.605) + (0.2)(0.589) + 0.5 = 1.0413$$
$$o_2^{(\text{in})} = w_7h_1^{(\text{out})} + w_8h_2^{(\text{out})} + b_2 = (0.4)(0.605) + (0.8)(0.589) + 0.5 = 1.2132$$

Applying the sigmoid activation function to get $o_1^{(\text{out})}$ and $o_2^{(\text{out})}$:

$$o_1^{(\text{out})} = f(o_1^{(\text{in})}) = \frac{1}{1+e^{-1.0413}} \approx 0.739$$
$$o_2^{(\text{out})} = f(o_2^{(\text{in})}) = \frac{1}{1+e^{-1.2132}} \approx 0.770$$

Step 3: Calculating Total Error using Mean Squared Error

The Mean Squared Error (MSE) is calculated using the formula:

$$E = \frac{1}{n}\sum_{i=1}^{n}(o_i^{(\text{out})} - o_i)^2$$

Substituting the given values:

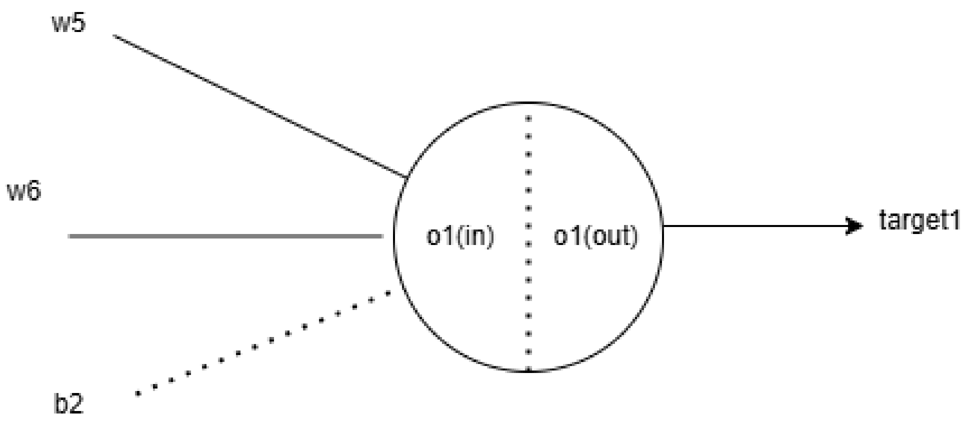
$$E = \frac{1}{2}\left[(0.739 - 0.09)^2 + (0.770 - 0.01)^2\right]$$

$$E \approx 0.498$$

Therefore, the total error is approximately $E \approx 0.498$.

Step 4: Calculating the Gradient for Weight Update (Backpropagation)

To update the weights using backpropagation, we need to calculate the gradient of the total error with respect to each weight. Here, we are specifically calculating the gradient for the weight w_5.



Weight Update for w_5

The chain rule is used to decompose the partial derivative of the total error E_{total} with respect to w_5:

$$\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{total}}}{\partial o_1^{\text{out}}} \cdot \frac{\partial o_1^{\text{out}}}{\partial o_1^{\text{in}}} \cdot \frac{\partial o_1^{\text{in}}}{\partial w_5}$$

$$\frac{\partial E_{\text{total}}}{\partial o_1^{\text{out}}} = o_1^{\text{out}} - \text{target}_1 = 0.739 - 0.09 = 0.649$$

$$\frac{\partial o_1^{\text{out}}}{\partial o_1^{\text{in}}} = o_1^{\text{out}} \cdot (1 - o_1^{\text{out}}) = 0.739 \cdot (1 - 0.739) = 0.739 \cdot 0.261 = 0.192$$

$$\frac{\partial o_1^{\text{in}}}{\partial w_5} = h_1^{\text{out}} = 0.605$$

Now, we multiply the results to find the gradient for w_5:

$$\frac{\partial E_{\text{total}}}{\partial w_5} = 0.649 \cdot 0.192 \cdot 0.605 \approx 0.075$$

- η is the learning rate, $\eta = 0.1$
- $\frac{\partial E_{\text{total}}}{\partial w_5} \approx 0.075$

Substituting the values:

$$w_5^{\text{new}} = 0.7 - 0.1 \cdot 0.075$$
$$w_5^{\text{new}} \approx 0.6925$$

Thus, the updated value of w_5 is approximately 0.6925.

Weight Update for w_6

The chain rule is used to decompose the partial derivative of the total error E_{total} with respect to w_6:

$$\frac{\partial E_{\text{total}}}{\partial w_6} = \frac{\partial E_{\text{total}}}{\partial o_1^{\text{out}}} \cdot \frac{\partial o_1^{\text{out}}}{\partial o_1^{\text{in}}} \cdot \frac{\partial o_1^{\text{in}}}{\partial w_6}$$

$$\frac{\partial E_{\text{total}}}{\partial o_1^{\text{out}}} = o_1^{\text{out}} - \text{target}_1 = 0.739 - 0.09 = 0.649$$

$$\frac{\partial o_1^{\text{out}}}{\partial o_1^{\text{in}}} = o_1^{\text{out}} \cdot (1 - o_1^{\text{out}}) = 0.739 \cdot (1 - 0.739) = 0.739 \cdot 0.261 = 0.192$$

$$\frac{\partial o_1^{\text{in}}}{\partial w_6} = h_2^{\text{out}} = 0.589$$

Now, we multiply the results to find the gradient for w_6:

$$\frac{\partial E_{\text{total}}}{\partial w_6} = 0.649 \cdot 0.192 \cdot 0.589 \approx 0.073$$

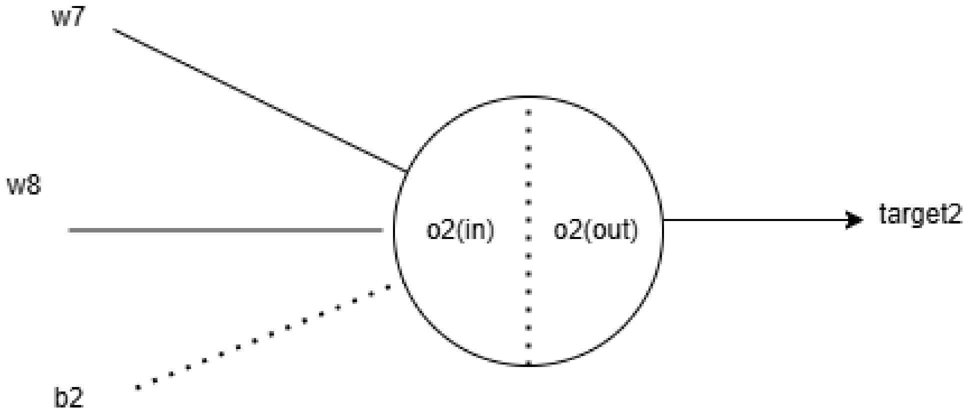
- η is the learning rate, $\eta = 0.1$
- $\frac{\partial E_{\text{total}}}{\partial w_6} \approx 0.073$

Substituting the values:

$$w_6^{\text{new}} = 0.2 - 0.1 \cdot 0.073$$

$$w_6^{\text{new}} \approx 0.1927$$

Thus, the updated value of w_6 is approximately 0.1927.



Weight Update for w_7

The chain rule is used to decompose the partial derivative of the total error E_{total} with respect to w_7:

$$\frac{\partial E_{\text{total}}}{\partial w_7} = \frac{\partial E_{\text{total}}}{\partial o_2^{\text{out}}} \cdot \frac{\partial o_2^{\text{out}}}{\partial o_2^{\text{in}}} \cdot \frac{\partial o_2^{\text{in}}}{\partial w_7}$$

$$\frac{\partial E_{\text{total}}}{\partial o_2^{\text{out}}} = o_2^{\text{out}} - \text{target}_2 = 0.770 - 0.01 = 0.76$$

$$\frac{\partial o_2^{\text{out}}}{\partial o_2^{\text{in}}} = o_2^{\text{out}} \cdot (1 - o_2^{\text{out}}) = 0.770 \cdot (1 - 0.770) = 0.1771$$

$$\frac{\partial o_2^{\text{in}}}{\partial w_7} = h_1^{\text{out}} = 0.605$$

Now, we multiply the results to find the gradient for w_7:

$$\frac{\partial E_{\text{total}}}{\partial w_7} = 0.76 \cdot 0.1771 \cdot 0.605 \approx 0.081$$

- η is the learning rate, $\eta = 0.1$
- $\frac{\partial E_{\text{total}}}{\partial w_7} \approx 0.081$

Substituting the values:

$$w_7^{\text{new}} = 0.4 - 0.1 \cdot 0.081$$
$$w_7^{\text{new}} \approx 0.3919$$

Thus, the updated value of w_7 is approximately 0.3919.

Weight Update for w_8

The chain rule is used to decompose the partial derivative of the total error E_{total} with respect to w_8:

$$\frac{\partial E_{\text{total}}}{\partial w_8} = \frac{\partial E_{\text{total}}}{\partial o_2^{\text{out}}} \cdot \frac{\partial o_2^{\text{out}}}{\partial o_2^{\text{in}}} \cdot \frac{\partial o_2^{\text{in}}}{\partial w_8}$$

$$\frac{\partial E_{\text{total}}}{\partial o_2^{\text{out}}} = o_2^{\text{out}} - \text{target}_2 = 0.770 - 0.01 = 0.76$$

$$\frac{\partial o_2^{\text{out}}}{\partial o_2^{\text{in}}} = o_2^{\text{out}} \cdot (1 - o_2^{\text{out}})0.770 \cdot (1 - 0.770) = 0.1771$$

$$\frac{\partial o_1^{\text{in}}}{\partial w_8} = h_2^{\text{out}} = 0.589$$

Now, we multiply the results to find the gradient for w_8:

$$\frac{\partial E_{\text{total}}}{\partial w_8} = 0.76 \cdot 0.1771 \cdot 0.589 \approx 0.079$$

- η is the learning rate, $\eta = 0.1$
- $\frac{\partial E_{\text{total}}}{\partial w_8} \approx 0.079$

Substituting the values:

$$w_8^{\text{new}} = 0.8 - 0.1 \cdot 0.079$$

$$w_8^{\text{new}} \approx 0.7921$$

Thus, the updated value of w_8 is approximately 0.7921.

Weight Update for w_1

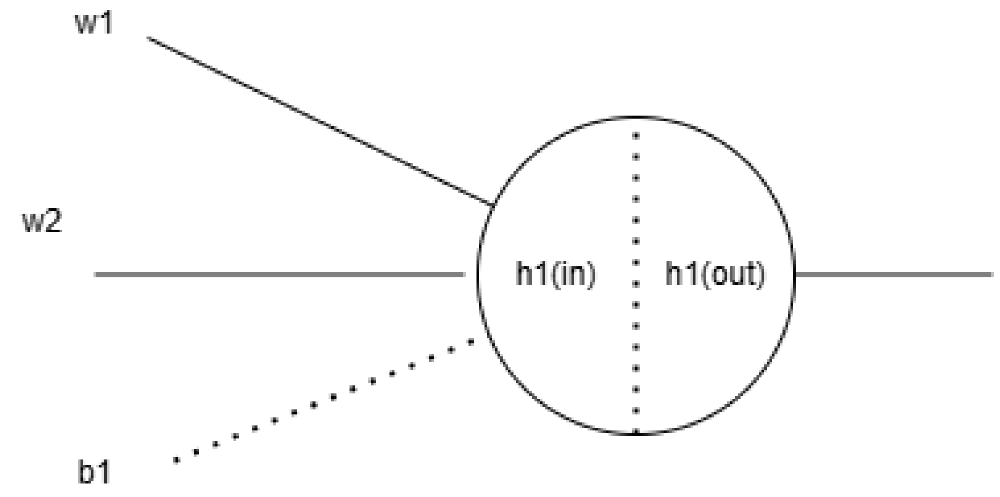
$$\frac{\partial E_{total}}{\partial w1} = \frac{\partial E_{total}}{\partial h1_{out}} \cdot \frac{\partial h1_{out}}{\partial h1_{in}} \cdot \frac{\partial h1_{in}}{\partial w1}$$

$$\frac{\partial E_{total}}{\partial h1_{out}} = \frac{\partial E1}{\partial h1_{out}} + \frac{\partial E2}{\partial h1_{out}}$$

$$\frac{\partial E1}{\partial h1_{out}} = \frac{\partial E1}{\partial o1_{in}} \cdot \frac{\partial o1_{in}}{\partial h1_{out}}$$

$$\frac{\partial E1}{\partial o1_{in}} = \frac{\partial E1}{\partial o1_{out}} \cdot \frac{\partial o1_{out}}{\partial o1_{in}}$$

$$\frac{\partial o1_{in}}{\partial h1_{out}} = w5$$



$$\frac{\partial E2}{\partial h1_{out}} = \frac{\partial E2}{\partial o2_{in}} \cdot \frac{\partial o2_{in}}{\partial h1_{out}}$$

$$\frac{\partial E2}{\partial o2_{in}} = \frac{\partial E2}{\partial o2_{out}} \cdot \frac{\partial o2_{out}}{\partial o2_{in}}$$

$$\frac{\partial o2_{in}}{\partial h1_{out}} = w7$$

$$\frac{\partial E1}{\partial o1_{out}} = o1_{out} - \text{target1} = 0.739 - 0.09 = 0.649$$

$$\frac{\partial o1_{out}}{\partial o1_{in}} = o1_{out} \cdot (1 - o1_{out}) = 0.192$$

$$\frac{\partial E1}{\partial h1_{out}} = 0.649 \cdot 0.192 \cdot 0.7 = 0.0872$$

$$\frac{\partial E2}{\partial o2_{out}} = o2_{out} - \text{target2} = 0.76$$

$$\frac{\partial o2_{out}}{\partial o2_{in}} = o2_{out} \cdot (1 - o2_{out}) = 0.1771$$

$$\frac{\partial E2}{\partial h1_{out}} = 0.76 \cdot 0.1771 \cdot 0.4 = 0.0538$$

$$\frac{\partial E_{\text{total}}}{\partial h_1^{\text{out}}} = \frac{\partial E_1}{\partial h_1^{\text{out}}} + \frac{\partial E_2}{\partial h_1^{\text{out}}} = 0.0872 + 0.0538 = 0.141$$

$$\frac{\partial E_{total}}{\partial w1} = 0.141 \cdot h1_{out} \cdot (1 - h1_{out}) \cdot x1$$

$$\frac{\partial E_{\text{total}}}{\partial w_1} = 0.141 \cdot 0.605 \cdot (1 - 0.605) \cdot 0.2 = 0.0067$$

- η is the learning rate, $\eta = 0.1$
- $\frac{\partial E_{\text{total}}}{\partial w_1} \approx 0.0067$

Substituting the values:

$$w_1^{\text{new}} = w_1 - \eta \cdot \frac{\partial E_{\text{total}}}{\partial w_1}$$

$$w_1^{\text{new}} = 0.5 - 0.1 \cdot 0.0067$$

$$w_1^{\text{new}} = 0.49933$$

Weight Update for w_2

$$\frac{\partial E_{\text{total}}}{\partial w_2} = \frac{\partial E_{\text{total}}}{\partial h_1^{\text{out}}} \cdot \frac{\partial h_1^{\text{out}}}{\partial h_1^{\text{in}}} \cdot \frac{\partial h_1^{\text{in}}}{\partial w_2}$$

$$\frac{\partial h_1^{\text{in}}}{\partial w_2} = x_2$$

$$\frac{\partial E_{\text{total}}}{\partial w_2} = 0.141 \cdot 0.605 \cdot (1 - 0.605) \cdot 0.1 = 0.00336$$

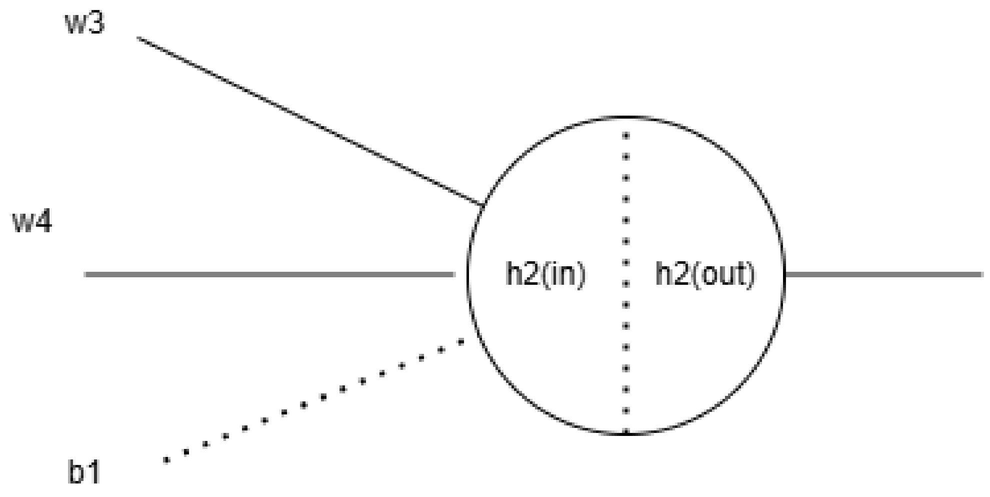
- η is the learning rate, $\eta = 0.1$
- $\frac{\partial E_{\text{total}}}{\partial w_2} \approx 0.00336$

Substituting the values:

$$w_2^{\text{new}} = w_2 - \eta \cdot \frac{\partial E_{\text{total}}}{\partial w_2}$$

$$w_2^{\text{new}} = 0.3 - 0.1 \cdot 0.00336$$

$$w_2^{\text{new}} = 0.2996$$



Weight Update for w_3

$$\frac{\partial E_{\text{total}}}{\partial w_3} = \frac{\partial E_{\text{total}}}{\partial h_{2\text{out}}} \cdot \frac{\partial h_{2\text{out}}}{\partial h_2^{\text{in}}} \cdot \frac{\partial h_2^{\text{in}}}{\partial w_3}$$

$$\frac{\partial E_{\text{total}}}{\partial h_2^{\text{out}}} = \frac{\partial E_1}{\partial h_2^{\text{out}}} + \frac{\partial E_2}{\partial h_2^{\text{out}}}$$

$$\frac{\partial E_1}{\partial h_2^{\text{out}}} = \frac{\partial E_1}{\partial o_1^{\text{in}}} \cdot \frac{\partial o_1^{\text{in}}}{\partial h_2^{\text{out}}}$$

$$\frac{\partial E_1}{\partial o_1^{\text{in}}} = \frac{\partial E_1}{\partial o_1^{\text{out}}} \cdot \frac{\partial o_1^{\text{out}}}{\partial o_1^{\text{in}}}$$

$$\frac{\partial o_1^{\text{in}}}{\partial h_2^{\text{out}}} = w_6$$

$$\frac{\partial E_2}{\partial h_2^{\text{out}}} = \frac{\partial E_2}{\partial o_2^{\text{in}}} \cdot \frac{\partial o_2^{\text{in}}}{\partial h_2^{\text{out}}}$$

$$\frac{\partial E_2}{\partial o_2^{\text{in}}} = \frac{\partial E_2}{\partial o_2^{\text{out}}} \cdot \frac{\partial o_2^{\text{out}}}{\partial o_2^{\text{in}}}$$

$$\frac{\partial o_2^{\text{in}}}{\partial h_2^{\text{out}}} = w_8$$

$$\frac{\partial E_1}{\partial o_1^{\text{out}}} = o_1^{\text{out}} - \text{target}_1 = 0.739 - 0.09 = 0.649$$

$$\frac{\partial o_1^{\text{out}}}{\partial o_1^{\text{in}}} = o_1^{\text{out}}(1 - o_1^{\text{out}}) = 0.192$$

$$\frac{\partial E_1}{\partial h_2^{\text{out}}} = 0.649 \cdot 0.192 \cdot 0.2 = 0.0249$$

$$\frac{\partial E_2}{\partial o_2^{\text{out}}} = o_2^{\text{out}} - \text{target}_2 = 0.76$$

$$\frac{\partial o_2^{\text{out}}}{\partial o_2^{\text{in}}} = o_2^{\text{out}}(1 - o_2^{\text{out}}) = 0.1771$$

$$\frac{\partial E_2}{\partial h_2^{\text{out}}} = 0.76 \cdot 0.1771 \cdot 0.8 = 0.1076$$

$$\frac{\partial E_{\text{total}}}{\partial h_2^{\text{out}}} = \frac{\partial E_1}{\partial h_2^{\text{out}}} + \frac{\partial E_2}{\partial h_2^{\text{out}}} = 0.0249 + 0.1076 = 0.1325$$

$$\frac{\partial E_{\text{total}}}{\partial w_3} = 0.1325 \cdot h_2^{\text{out}} \cdot (1 - h_2^{\text{out}}) \cdot x_1$$

$$\frac{\partial E_{\text{total}}}{\partial w_3} = 0.1325 \cdot 0.589 \cdot (1 - 0.589) \cdot 0.2 = 0.0064$$

- η is the learning rate, $\eta = 0.1$
- $\frac{\partial E_{\text{total}}}{\partial w_3} \approx 0.0064$

Substituting the values:

$$w_3^{\text{new}} = w_3 - \eta \cdot \frac{\partial E_{\text{total}}}{\partial w_3}$$

$$w_3^{\text{new}} = 0.1 - 0.1 \cdot 0.0064$$

$$w_3^{\text{new}} = 0.0993$$

Weight Update for w_4

$$\frac{\partial E_{\text{total}}}{\partial w_4} = \frac{\partial E_{\text{total}}}{\partial h_2^{\text{out}}} \cdot \frac{\partial h_2^{\text{out}}}{\partial h_2^{\text{in}}} \cdot \frac{\partial h_2^{\text{in}}}{\partial w_4}$$

$$\frac{\partial h_2^{\text{in}}}{\partial w_4} = x_2$$

$$\frac{\partial E_{\text{total}}}{\partial w_4} = 0.1325 \cdot 0.589 \cdot (1 - 0.589) \cdot 0.1 = 0.0032$$

- η is the learning rate, $\eta = 0.1$
- $\frac{\partial E_{\text{total}}}{\partial w_4} \approx 0.0032$

Substituting the values:

$$w_4^{\text{new}} = w_4 - \eta \cdot \frac{\partial E_{\text{total}}}{\partial w_4}$$

$$w_4^{\text{new}} = 0.4 - 0.1 \cdot 0.0032$$

$$w_4^{\text{new}} = 0.3996$$

ANN_HAND_CALCULATIONS

Importing Libraries

[1]:
```
import numpy as np
```

Sigmoid Activation Function

[2]:
```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

Derivative of Sigmoid Function

[3]:
```
def sigmoid_derivative(x):
    return x * (1 - x)
```

Assumed Values

[4]:
```
x1, x2 = 0.2, 0.1
o1, o2 = 0.09, 0.01
```

Storing weights in key-value pairs

[5]:
```
weights = {
    "w1": 0.5, "w2": 0.3, "w3": 0.1, "w4": 0.4,
    "w5": 0.7, "w6": 0.2, "w7": 0.4, "w8": 0.8
}
```

Storing bias in key-value pairs

[6]:
```
biases = {
    "b1": 0.3,  # for hidden layer
    "b2": 0.5   # for output layer
}
```

Initializing learning rate

[7]:
```
learning_rate = 0.1
```

Calculating hidden layer activations

[8]:
```
h1_in = weights["w1"] * x1 + weights["w2"] * x2 + biases["b1"]
h2_in = weights["w3"] * x1 + weights["w4"] * x2 + biases["b1"]
```

```
h1_out = sigmoid(h1_in)
h2_out = sigmoid(h2_in)
```

Calculating output layer activations

[9]:
```
o1_in = weights["w5"] * h1_out + weights["w6"] * h2_out + biases["b2"]
o2_in = weights["w7"] * h1_out + weights["w8"] * h2_out + biases["b2"]

o1_out = sigmoid(o1_in)
o2_out = sigmoid(o2_in)
```

Using Mean Squared Error

[10]:
```
error = (1 / 2) * ((o1_out - o1) ** 2 + (o2_out - o2) ** 2)
```

Back Propagation Starts

[11]:
```
error_o1 = o1_out - o1
error_o2 = o2_out - o2
```

Output layer gradients

[12]:
```
dE_dw5 = error_o1 * sigmoid_derivative(o1_out) * h1_out
dE_dw6 = error_o1 * sigmoid_derivative(o1_out) * h2_out
dE_dw7 = error_o2 * sigmoid_derivative(o2_out) * h1_out
dE_dw8 = error_o2 * sigmoid_derivative(o2_out) * h2_out
```

Update weights for the output layer

[13]:
```
weights["w5"] -= learning_rate * dE_dw5
weights["w6"] -= learning_rate * dE_dw6
weights["w7"] -= learning_rate * dE_dw7
weights["w8"] -= learning_rate * dE_dw8
```

Hidden Layer gradients for weights w1,w2,w3,w4

[14]:
```
dE_dh1_out = (error_o1 * sigmoid_derivative(o1_out) * weights["w5"] +
              error_o2 * sigmoid_derivative(o2_out) * weights["w7"])
dE_dh2_out = (error_o1 * sigmoid_derivative(o1_out) * weights["w6"] +
              error_o2 * sigmoid_derivative(o2_out) * weights["w8"])

dE_dw1 = dE_dh1_out * sigmoid_derivative(h1_out) * x1
dE_dw2 = dE_dh1_out * sigmoid_derivative(h1_out) * x2
dE_dw3 = dE_dh2_out * sigmoid_derivative(h2_out) * x1
dE_dw4 = dE_dh2_out * sigmoid_derivative(h2_out) * x2
```

Update weights for the hidden layer

[15]:
```
weights["w1"] -= learning_rate * dE_dw1
weights["w2"] -= learning_rate * dE_dw2
weights["w3"] -= learning_rate * dE_dw3
```

```
weights["w4"] -= learning_rate * dE_dw4
```

[16]:
```
weights
```

[16]:
```
{'w1': 0.4993346585658121,
 'w2': 0.29966732928290607,
 'w3': 0.09936796905620592,
 'w4': 0.399683984528103,
 'w5': 0.6924173351299006,
 'w6': 0.19262800738978184,
 'w7': 0.3918584264618821,
 'w8': 0.7920846271084422}
```

[17]:
```
error
```

[17]:
```
0.5002522842154867
```

[ ]:
```

```

CNN Hand Calculations

Input Matrix (11x7):

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 & 0.5 & 0.6 & 0.7 \\ 0.2 & 0.3 & 0.4 & 0.5 & 0.6 & 0.7 & 0.8 \\ 0.3 & 0.4 & 0.5 & 0.6 & 0.7 & 0.8 & 0.9 \\ 0.4 & 0.5 & 0.6 & 0.7 & 0.8 & 0.9 & 1.0 \\ 0.5 & 0.6 & 0.7 & 0.8 & 0.9 & 1.0 & 1.1 \\ 0.6 & 0.7 & 0.8 & 0.9 & 1.0 & 1.1 & 1.2 \\ 0.7 & 0.8 & 0.9 & 1.0 & 1.1 & 1.2 & 1.3 \\ 0.8 & 0.9 & 1.0 & 1.1 & 1.2 & 1.3 & 1.4 \\ 0.9 & 1.0 & 1.1 & 1.2 & 1.3 & 1.4 & 1.5 \\ 1.0 & 1.1 & 1.2 & 1.3 & 1.4 & 1.5 & 1.6 \\ 1.1 & 1.2 & 1.3 & 1.4 & 1.5 & 1.6 & 1.7 \end{bmatrix}$$

Kernel 1 (2x2):

$$\begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix}$$

After First Convolution (10x6):

$$\begin{bmatrix} 0.23 & 0.33 & 0.43 & 0.53 & 0.63 & 0.73 \\ 0.33 & 0.43 & 0.53 & 0.63 & 0.73 & 0.83 \\ 0.43 & 0.53 & 0.63 & 0.73 & 0.83 & 0.93 \\ 0.53 & 0.63 & 0.73 & 0.83 & 0.93 & 1.03 \\ 0.63 & 0.73 & 0.83 & 0.93 & 1.03 & 1.13 \\ 0.73 & 0.83 & 0.93 & 1.03 & 1.13 & 1.23 \\ 0.83 & 0.93 & 1.03 & 1.13 & 1.23 & 1.33 \\ 0.93 & 1.03 & 1.13 & 1.23 & 1.33 & 1.43 \\ 1.03 & 1.13 & 1.23 & 1.33 & 1.43 & 1.53 \\ 1.13 & 1.23 & 1.33 & 1.43 & 1.53 & 1.63 \end{bmatrix}$$

After First Max Pooling (5x3):

Stride: 2

$$\begin{bmatrix} 0.43 & 0.63 & 0.83 \\ 0.63 & 0.83 & 1.03 \\ 0.83 & 1.03 & 1.23 \\ 1.03 & 1.23 & 1.43 \\ 1.23 & 1.43 & 1.63 \end{bmatrix}$$

Kernel 2 (2x2):

$$\begin{bmatrix} 0.5 & 0.6 \\ 0.7 & 0.8 \end{bmatrix}$$

After Second Convolution (4x2):

$$\begin{bmatrix} 1.698 & 2.218 \\ 2.218 & 2.738 \\ 2.738 & 3.258 \\ 3.258 & 3.778 \end{bmatrix}$$

After Second Max Pooling (2x1):

Stride: 2

$$\begin{bmatrix} 2.738 \\ 3.778 \end{bmatrix}$$

Fully Connected Layer Input:

The 2×1 output is fed to the fully connected layer (ANN).

CNN_HAND_CALCULATIONS

Importing Libraries

```
[1]: import numpy as np
```

Convolution Function

```
[2]: def convolution(input_matrix, kernel):
         """Perform convolution operation."""
         kernel_h, kernel_w = kernel.shape
         input_h, input_w = input_matrix.shape
         output_h = input_h - kernel_h + 1
         output_w = input_w - kernel_w + 1
         output = np.zeros((output_h, output_w))

         for i in range(output_h):
             for j in range(output_w):
                 output[i, j] = np.sum(input_matrix[i:i + kernel_h, j:j + kernel_w]␣
      ↪* kernel)
         return output
```

Max Pooling Function

```
[3]: def max_pooling(input_matrix, pool_size):
         """Perform max pooling operation."""
         pool_h, pool_w = pool_size
         input_h, input_w = input_matrix.shape
         output_h = input_h // pool_h
         output_w = input_w // pool_w
         output = np.zeros((output_h, output_w))

         for i in range(output_h):
             for j in range(output_w):
                 output[i, j] = np.max(input_matrix[i * pool_h:(i + 1) * pool_h, j *␣
      ↪pool_w:(j + 1) * pool_w])
         return output
```

Taking a random matrix

```
[4]: input_matrix = np.array([
         [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7],
         [0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8],
         [0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9],
         [0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
         [0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1],
         [0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2],
         [0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3],
         [0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4],
         [0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5],
         [1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6],
         [1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7]
     ])
```

Taking random Kernels

```
[5]: kernel1 = np.array([[0.1, 0.2], [0.3, 0.4]])
     kernel2 = np.array([[0.5, 0.6], [0.7, 0.8]])
```

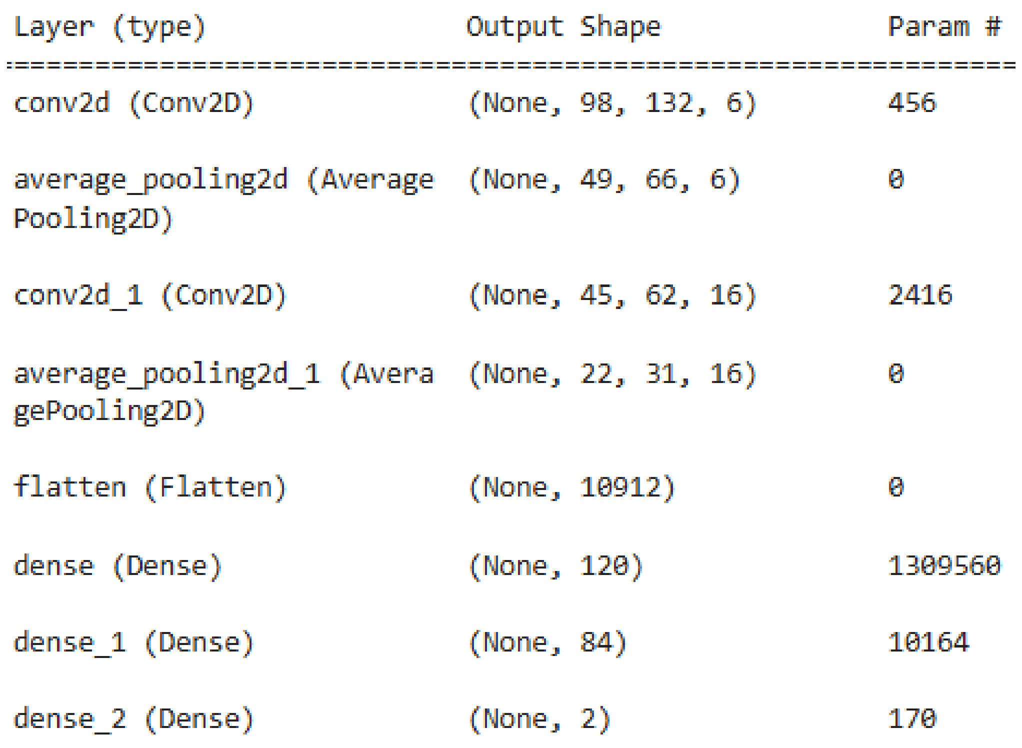
Performing convolution and pooling operations twice

```
[6]: conv1 = convolution(input_matrix, kernel1)
     pool1 = max_pooling(conv1, (2, 2))

     conv2 = convolution(pool1, kernel2)
     pool2 = max_pooling(conv2, (2, 2))
```

```
[7]: print("Input Matrix (11x7):")
     print(input_matrix)
     print("\nKernel1 (2x2):")
     print(kernel1)
     print("\nAfter First Convolution (10x6):")
     print(conv1)
     print("\nAfter First Max Pooling (5x3):")
     print(pool1)
     print("\nKernel2 (2x2):")
     print(kernel2)
     print("\nAfter Second Convolution (4x2):")
     print(conv2)
     print("\nAfter Second Max Pooling (2x1):")
     print(pool2)
```

```
Input Matrix (11x7):
[[0.1 0.2 0.3 0.4 0.5 0.6 0.7]
 [0.2 0.3 0.4 0.5 0.6 0.7 0.8]
 [0.3 0.4 0.5 0.6 0.7 0.8 0.9]
 [0.4 0.5 0.6 0.7 0.8 0.9 1. ]
 [0.5 0.6 0.7 0.8 0.9 1.  1.1]
 [0.6 0.7 0.8 0.9 1.  1.1 1.2]
 [0.7 0.8 0.9 1.  1.1 1.2 1.3]
 [0.8 0.9 1.  1.1 1.2 1.3 1.4]
 [0.9 1.  1.1 1.2 1.3 1.4 1.5]
 [1.  1.1 1.2 1.3 1.4 1.5 1.6]
 [1.1 1.2 1.3 1.4 1.5 1.6 1.7]]

Kernel1 (2x2):
[[0.1 0.2]
 [0.3 0.4]]

After First Convolution (10x6):
[[0.23 0.33 0.43 0.53 0.63 0.73]
 [0.33 0.43 0.53 0.63 0.73 0.83]
 [0.43 0.53 0.63 0.73 0.83 0.93]
 [0.53 0.63 0.73 0.83 0.93 1.03]
 [0.63 0.73 0.83 0.93 1.03 1.13]
 [0.73 0.83 0.93 1.03 1.13 1.23]
 [0.83 0.93 1.03 1.13 1.23 1.33]
 [0.93 1.03 1.13 1.23 1.33 1.43]
 [1.03 1.13 1.23 1.33 1.43 1.53]
 [1.13 1.23 1.33 1.43 1.53 1.63]]

After First Max Pooling (5x3):
[[0.43 0.63 0.83]
 [0.63 0.83 1.03]
 [0.83 1.03 1.23]
 [1.03 1.23 1.43]
 [1.23 1.43 1.63]]

Kernel2 (2x2):
[[0.5 0.6]
 [0.7 0.8]]

After Second Convolution (4x2):
[[1.698 2.218]
 [2.218 2.738]
 [2.738 3.258]
 [3.258 3.778]]

After Second Max Pooling (2x1):
[[2.738]
 [3.778]]
```

The 2X1 output is fed to the Fully Connected Layer(ANN)



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 98, 132, 6)        456

average_pooling2d (Average  (None, 49, 66, 6)         0
Pooling2D)

conv2d_1 (Conv2D)            (None, 45, 62, 16)        2416

average_pooling2d_1 (Avera  (None, 22, 31, 16)        0
gePooling2D)

flatten (Flatten)            (None, 10912)             0

dense (Dense)                (None, 120)               1309560

dense_1 (Dense)              (None, 84)                10164

dense_2 (Dense)              (None, 2)                 170
```

Figure 1: model summary

Layer-wise Parameter Calculation

Layer 1: `conv2d (Conv2D)`

- Input Shape: $(102, 136, 3)$
- Filter Size: 5×5, with 6 filters
- Output Shape: $(98, 132, 6)$

$$\text{Weights per filter} = 5 \times 5 \times 3 = 75$$
$$\text{Total weights} = 75 \times 6 = 450$$
$$\text{Total biases} = 6$$
$$\text{Total parameters} = 450 + 6 = 456$$

Layer 2: `average_pooling2d (AveragePooling2D)`

- Input Shape: $(98, 132, 6)$
- Pool Size: 2×2, Stride $= 2$
- Output Shape: $(49, 66, 6)$
- Parameters: 0 (no trainable parameters)

Layer 3: `conv2d_1 (Conv2D)`

- Input Shape: $(49, 66, 6)$
- Filter Size: 5×5, with 16 filters
- Output Shape: $(45, 62, 16)$

$$\text{Weights per filter} = 5 \times 5 \times 6 = 150$$
$$\text{Total weights} = 150 \times 16 = 2400$$
$$\text{Total biases} = 16$$
$$\text{Total parameters} = 2400 + 16 = 2416$$

Layer 4: `average_pooling2d_1 (AveragePooling2D)`

- Input Shape: $(45, 62, 16)$
- Pool Size: 2×2, Stride $= 2$
- Output Shape: $(22, 31, 16)$
- Parameters: 0 (no trainable parameters)

Layer 5: `flatten (Flatten)`

- Input Shape: $(22, 31, 16)$
- Output Shape: $22 \times 31 \times 16 = 10912$
- Parameters: 0 (no trainable parameters)

Layer 6: `dense (Dense)`

- Input Shape: 10912
- Output Units: 120

$$\text{Weights} = 10912 \times 120 = 1309440$$
$$\text{Biases} = 120$$
$$\text{Total parameters} = 1309440 + 120 = 1309560$$

Layer 7: `dense_1 (Dense)`

- Input Shape: 120
- Output Units: 84

$$\text{Weights} = 120 \times 84 = 10080$$
$$\text{Biases} = 84$$
$$\text{Total parameters} = 10080 + 84 = 10164$$

Layer 8: `dense_2 (Dense)`

- Input Shape: 84
- Output Units: 2

$$\text{Weights} = 84 \times 2 = 168$$
$$\text{Biases} = 2$$
$$\text{Total parameters} = 168 + 2 = 170$$

Summary of Parameters

$$\text{Total Parameters} = 456 + 2416 + 1309560 + 10164 + 170 = 1322766$$