ABASYN UNIVERSITY, ISLAMABAD CAMPUS.

STIC

ASSIGNMENT#03

SUBMITTED TO:

DR.SADAF

SUBMITTED BY:

SULAIMAN ASIIF-8918

BS-(COMPUTER SCIENCE)

Q#01:

Provide in-depth detail of what is happening in convolution, max pooling, activation function, how is loss being calculated? How are weights and biases updated, how does the gradient descent work?

- **Convolution (the "pattern detector")**

Imagine you're looking at a photo. Your eyes don't scan every pixel individually; instead, they notice patterns — edges, curves, textures.

Convolution does the same thing:

A filter (kernel) slides across the image.

At each position, it multiplies the filter values with the pixel values underneath and sums them up.

This produces a feature map that highlights where certain patterns (like vertical edges or corners) appear.

Think of convolution as a magnifying glass that only lights up when it sees the pattern it's trained to detect.

- **Max Pooling (the "summary taker")**

After convolution, we have a detailed map of patterns. But too much detail can overwhelm the system.

Max pooling reduces this by:

Taking small regions (like 2×2 squares).

Keeping only the maximum value in each region.

This is like summarizing a paragraph: instead of every word, you keep the most important point.

It makes the network more efficient and less sensitive to tiny changes (like noise in an image).

- **Activation Function (the "decision maker")**

Once features are extracted, the network needs to decide: is this pattern important or not?

Activation functions introduce non-linearity:

Without them, the network would just be a fancy linear equation.

With them, it can make complex decisions.

Examples:

ReLU (Rectified Linear Unit): outputs 0 if input is negative, otherwise keeps it. Like saying “ignore negative signals, only pass forward positive ones.”

Sigmoid: squashes values between 0 and 1, useful for probabilities.

Think of activation as the brain’s spark — deciding whether a neuron should fire or stay quiet.

- **Loss Calculation (the "error measurement")**

After passing through layers, the network makes a prediction.

Loss is the difference between prediction and reality:

If the network says “cat” but the image is “dog,” the loss is high.

If it says “cat” and it is indeed a cat, the loss is low.

Common loss functions:

Mean Squared Error (MSE): for regression tasks.

Cross-Entropy Loss: for classification tasks.

Loss is like a teacher’s red marks on an exam — showing how far off the answer was.

- **Weights and Biases Update (the "learning process")**

Each filter and neuron has weights (strength of connections) and biases (offsets).

Initially, these are random.

During training:

The network calculates loss.

Then it adjusts weights and biases to reduce that loss.

This is like a student correcting mistakes after feedback.

- **Gradient Descent (the "path to improvement")**

Gradient descent is the method used to update weights:

It calculates the gradient (slope) of the loss function with respect to each weight.

If the slope is steep, it means the weight strongly affects the error.

The algorithm moves weights downhill on the loss curve to minimize error.

Steps:

Compute gradient using backpropagation (chain rule of calculus).

Update weights:

wnew=wold−η⋅∂L∂ww_{\text{new}} = w_{\text{old}} - \eta \cdot \frac{\partial L}{\partial w} where η\eta is the learning rate.

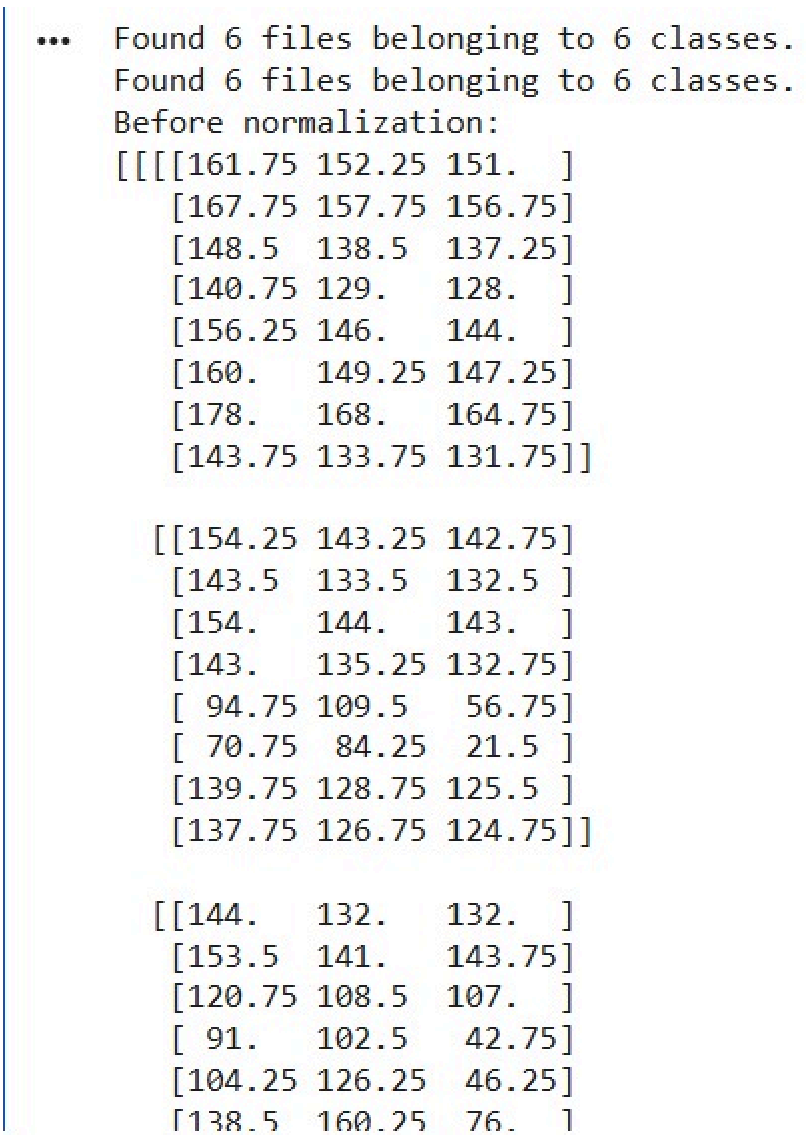
Think of it like hiking down a mountain blindfolded:

You feel the slope under your feet (gradient).

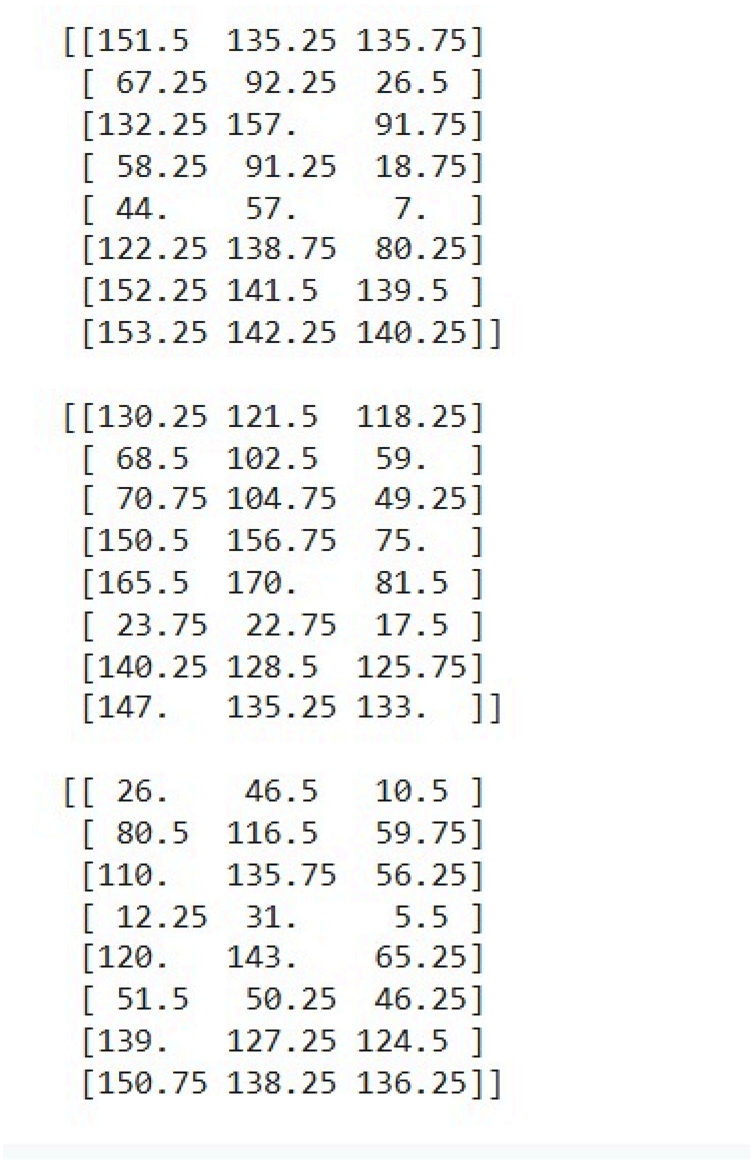
You take small steps downhill (learning rate).

Eventually, you reach the valley (minimum loss).

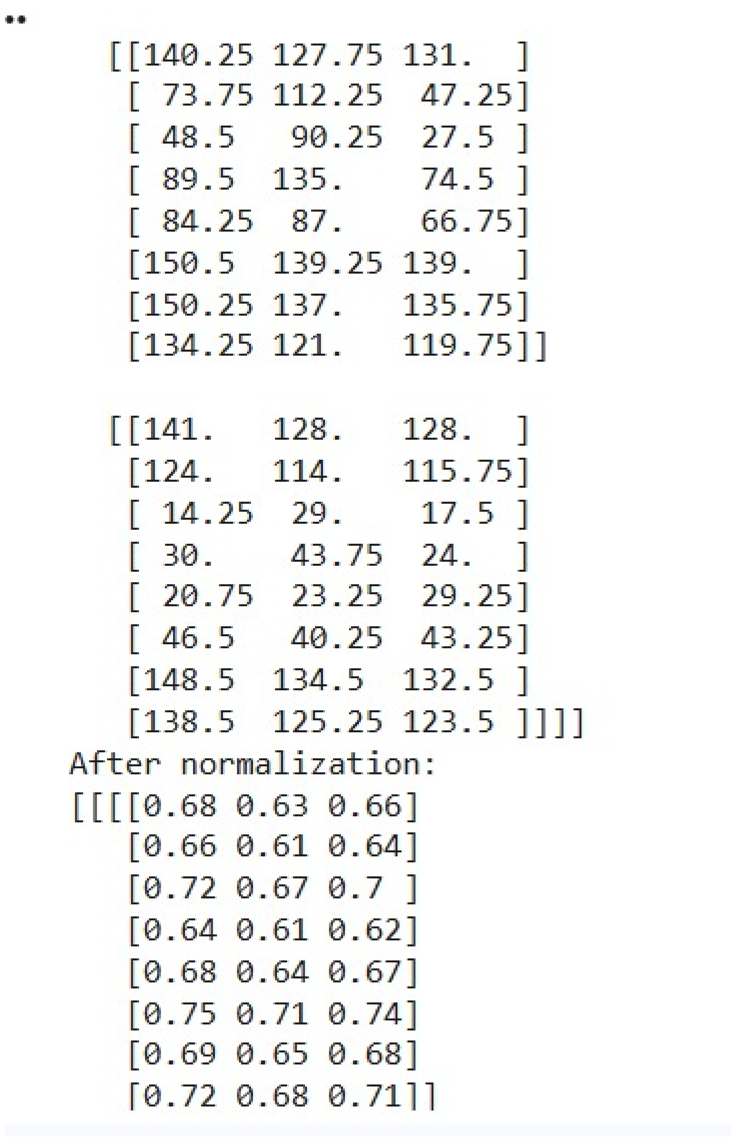
Now the output of given code is given below :



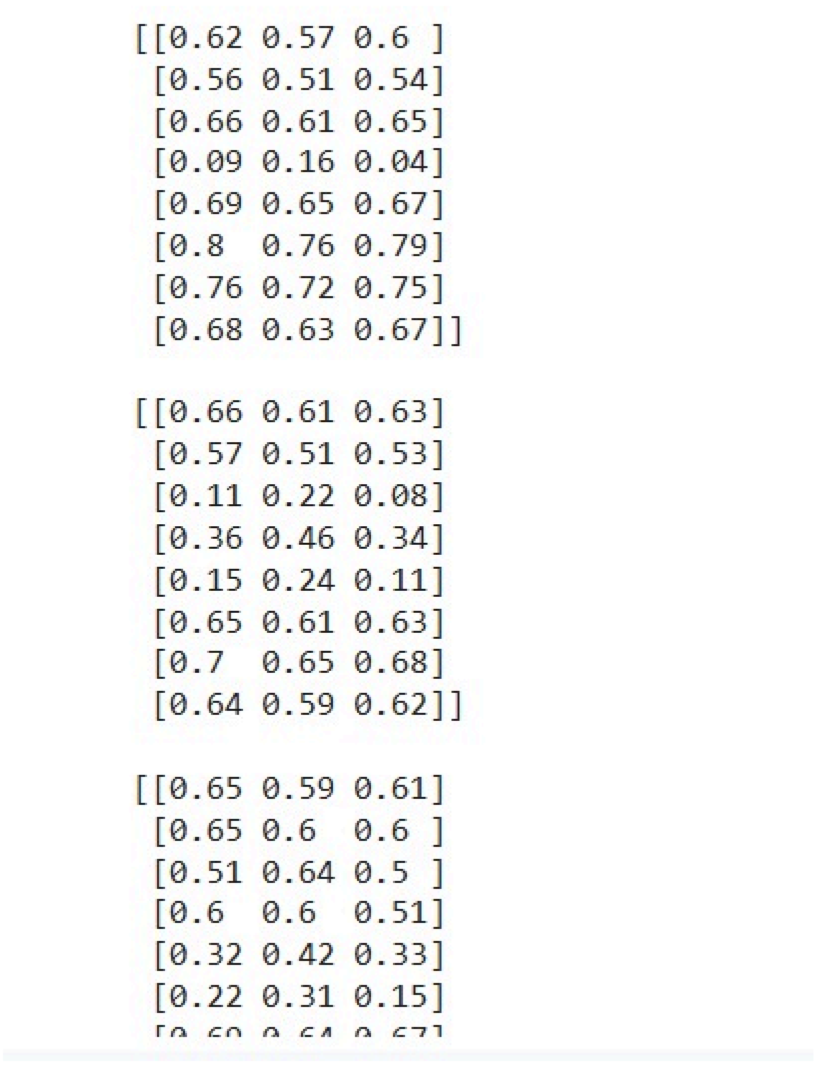
```
...  Found 6 files belonging to 6 classes.
     Found 6 files belonging to 6 classes.
     Before normalization:
     [[[[161.75 152.25 151.  ]
        [167.75 157.75 156.75]
        [148.5  138.5  137.25]
        [140.75 129.   128.  ]
        [156.25 146.   144.  ]
        [160.   149.25 147.25]
        [178.   168.   164.75]
        [143.75 133.75 131.75]]

       [[154.25 143.25 142.75]
        [143.5  133.5  132.5 ]
        [154.   144.   143.  ]
        [143.   135.25 132.75]
        [ 94.75 109.5   56.75]
        [ 70.75  84.25  21.5 ]
        [139.75 128.75 125.5 ]
        [137.75 126.75 124.75]]

       [[144.   132.   132.  ]
        [153.5  141.   143.75]
        [120.75 108.5  107.  ]
        [ 91.   102.5   42.75]
        [104.25 126.25  46.25]
        [138.5  160.25  76.   ]
```



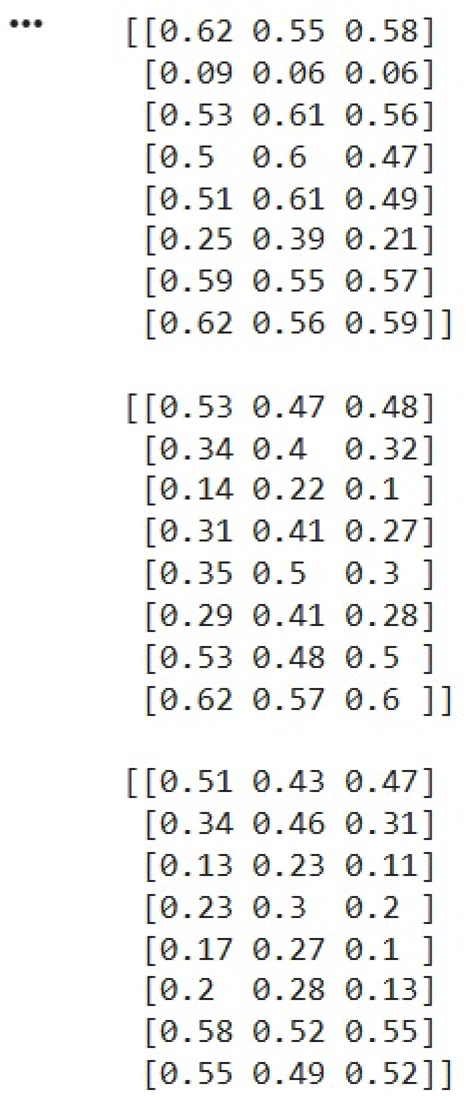
```
[[151.5  135.25 135.75]
 [ 67.25  92.25  26.5 ]
 [132.25 157.    91.75]
 [ 58.25  91.25  18.75]
 [ 44.    57.     7.  ]
 [122.25 138.75  80.25]
 [152.25 141.5  139.5 ]
 [153.25 142.25 140.25]]

[[130.25 121.5  118.25]
 [ 68.5  102.5   59.  ]
 [ 70.75 104.75  49.25]
 [150.5  156.75  75.  ]
 [165.5  170.    81.5 ]
 [ 23.75  22.75  17.5 ]
 [140.25 128.5  125.75]
 [147.   135.25 133.  ]]

[[ 26.    46.5   10.5 ]
 [ 80.5  116.5   59.75]
 [110.   135.75  56.25]
 [ 12.25  31.     5.5 ]
 [120.   143.    65.25]
 [ 51.5   50.25  46.25]
 [139.   127.25 124.5 ]
 [150.75 138.25 136.25]]
```



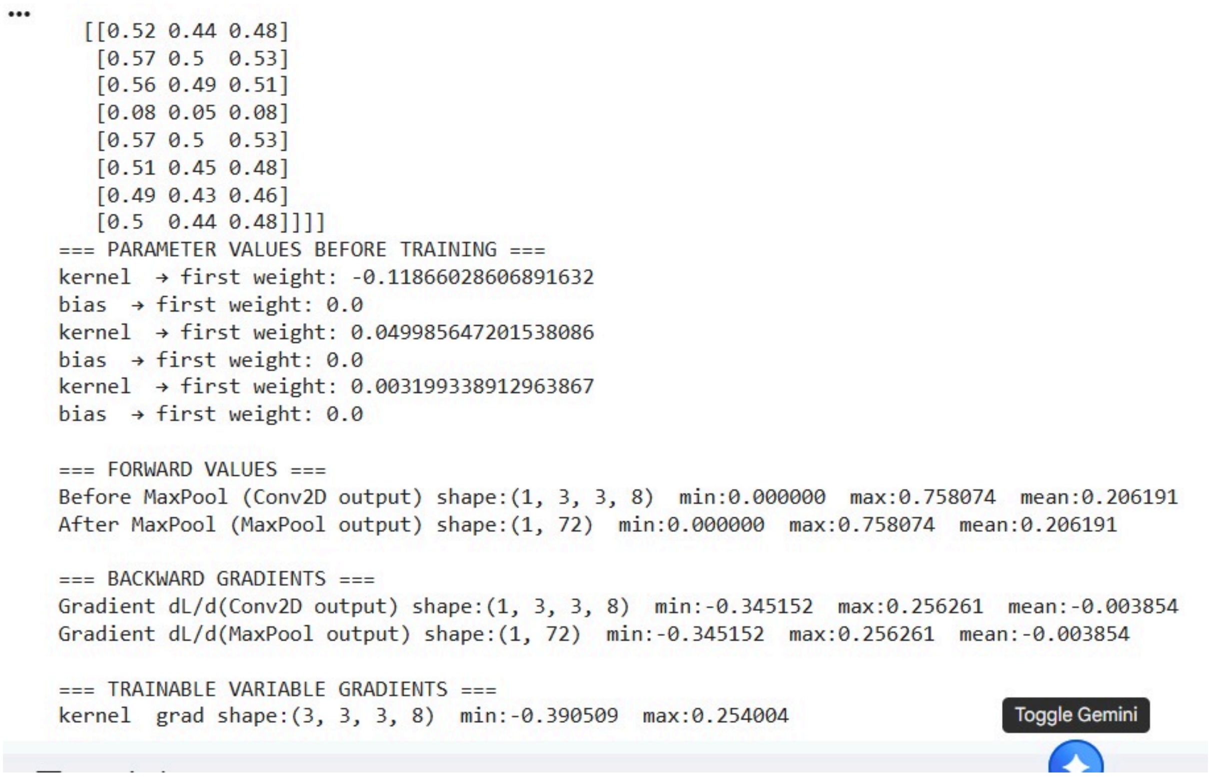
```
••
     [[140.25 127.75 131.  ]
      [ 73.75 112.25  47.25]
      [ 48.5   90.25  27.5 ]
      [ 89.5  135.    74.5 ]
      [ 84.25  87.    66.75]
      [150.5  139.25 139.  ]
      [150.25 137.   135.75]
      [134.25 121.   119.75]]

     [[141.   128.   128.  ]
      [124.   114.   115.75]
      [ 14.25  29.    17.5 ]
      [ 30.    43.75  24.  ]
      [ 20.75  23.25  29.25]
      [ 46.5   40.25  43.25]
      [148.5  134.5  132.5 ]
      [138.5  125.25 123.5 ]]]]
   After normalization:
   [[[[0.68 0.63 0.66]
      [0.66 0.61 0.64]
      [0.72 0.67 0.7 ]
      [0.64 0.61 0.62]
      [0.68 0.64 0.67]
      [0.75 0.71 0.74]
      [0.69 0.65 0.68]
      [0.72 0.68 0.71]]
```



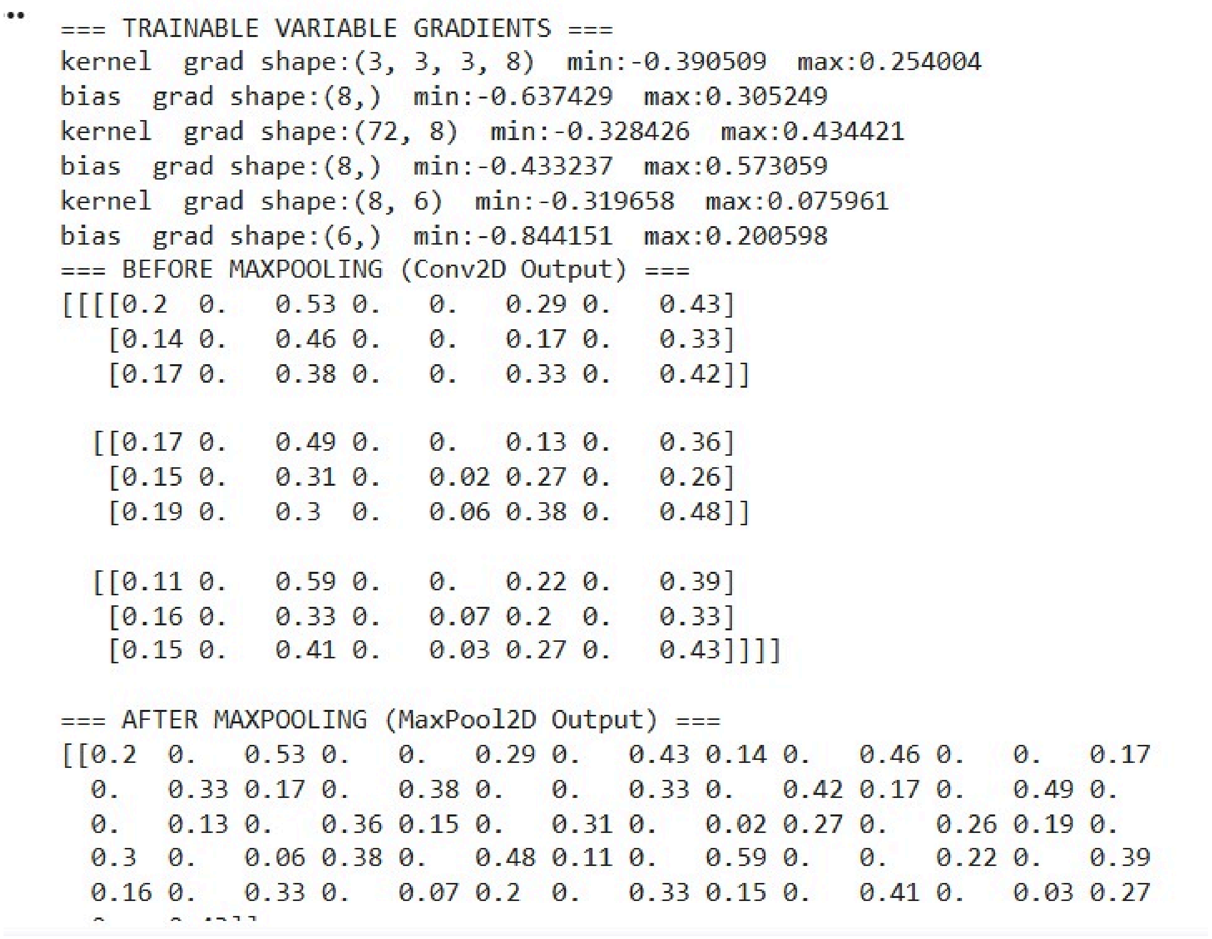
```
[[0.62 0.57 0.6 ]
 [0.56 0.51 0.54]
 [0.66 0.61 0.65]
 [0.09 0.16 0.04]
 [0.69 0.65 0.67]
 [0.8  0.76 0.79]
 [0.76 0.72 0.75]
 [0.68 0.63 0.67]]

[[0.66 0.61 0.63]
 [0.57 0.51 0.53]
 [0.11 0.22 0.08]
 [0.36 0.46 0.34]
 [0.15 0.24 0.11]
 [0.65 0.61 0.63]
 [0.7  0.65 0.68]
 [0.64 0.59 0.62]]

[[0.65 0.59 0.61]
 [0.65 0.6  0.6 ]
 [0.51 0.64 0.5 ]
 [0.6  0.6  0.51]
 [0.32 0.42 0.33]
 [0.22 0.31 0.15]
 [0.69 0.64 0.67]
```



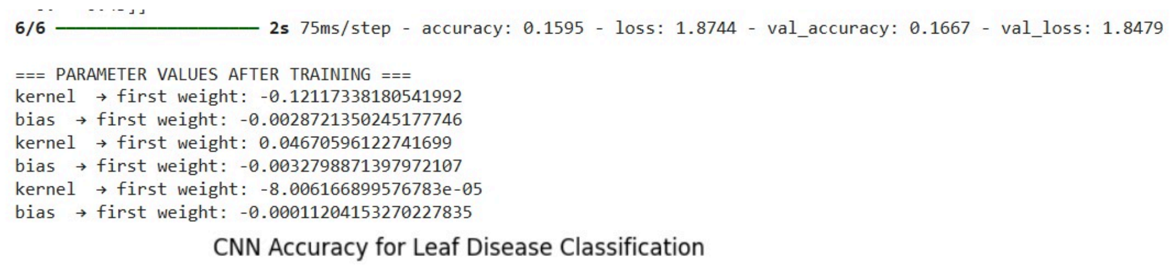
```
[[0.62 0.55 0.58]
 [0.09 0.06 0.06]
 [0.53 0.61 0.56]
 [0.5  0.6  0.47]
 [0.51 0.61 0.49]
 [0.25 0.39 0.21]
 [0.59 0.55 0.57]
 [0.62 0.56 0.59]]

[[0.53 0.47 0.48]
 [0.34 0.4  0.32]
 [0.14 0.22 0.1 ]
 [0.31 0.41 0.27]
 [0.35 0.5  0.3 ]
 [0.29 0.41 0.28]
 [0.53 0.48 0.5 ]
 [0.62 0.57 0.6 ]]

[[0.51 0.43 0.47]
 [0.34 0.46 0.31]
 [0.13 0.23 0.11]
 [0.23 0.3  0.2 ]
 [0.17 0.27 0.1 ]
 [0.2  0.28 0.13]
 [0.58 0.52 0.55]
 [0.55 0.49 0.52]]
```



```
   [[0.52 0.44 0.48]
    [0.57 0.5  0.53]
    [0.56 0.49 0.51]
    [0.08 0.05 0.08]
    [0.57 0.5  0.53]
    [0.51 0.45 0.48]
    [0.49 0.43 0.46]
    [0.5  0.44 0.48]]]]
=== PARAMETER VALUES BEFORE TRAINING ===
kernel  → first weight: -0.11866028606891632
bias  → first weight: 0.0
kernel  → first weight: 0.049985647201538086
bias  → first weight: 0.0
kernel  → first weight: 0.003199338912963867
bias  → first weight: 0.0

=== FORWARD VALUES ===
Before MaxPool (Conv2D output) shape:(1, 3, 3, 8)  min:0.000000  max:0.758074  mean:0.206191
After MaxPool (MaxPool output) shape:(1, 72)  min:0.000000  max:0.758074  mean:0.206191

=== BACKWARD GRADIENTS ===
Gradient dL/d(Conv2D output) shape:(1, 3, 3, 8)  min:-0.345152  max:0.256261  mean:-0.003854
Gradient dL/d(MaxPool output) shape:(1, 72)  min:-0.345152  max:0.256261  mean:-0.003854

=== TRAINABLE VARIABLE GRADIENTS ===
kernel  grad shape:(3, 3, 3, 8)  min:-0.390509  max:0.254004
```

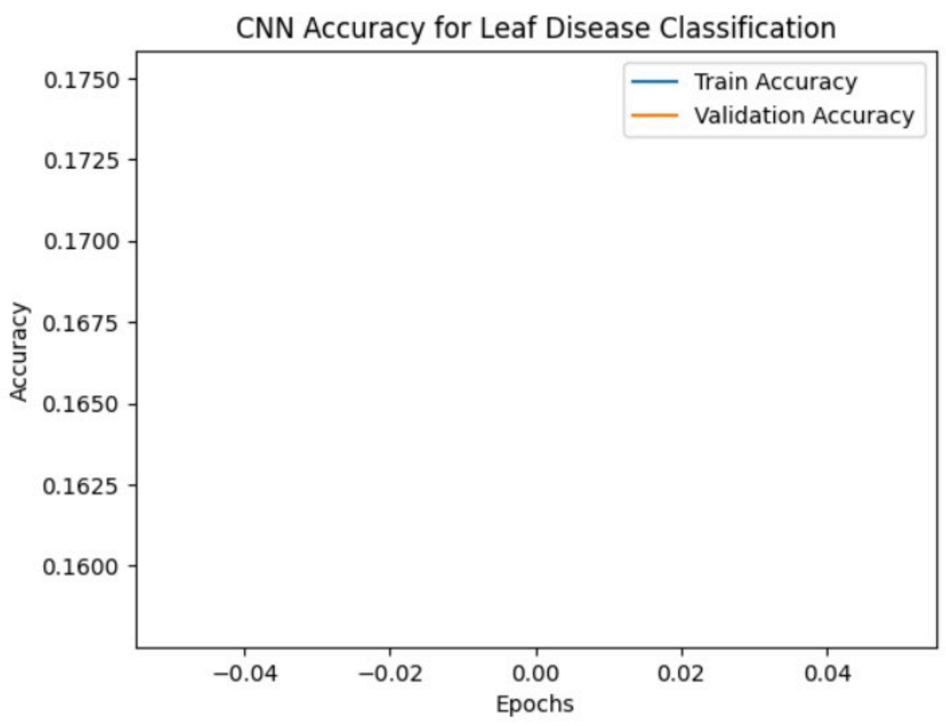


```
=== TRAINABLE VARIABLE GRADIENTS ===
kernel  grad shape:(3, 3, 3, 8)  min:-0.390509  max:0.254004
bias  grad shape:(8,)  min:-0.637429  max:0.305249
kernel  grad shape:(72, 8)  min:-0.328426  max:0.434421
bias  grad shape:(8,)  min:-0.433237  max:0.573059
kernel  grad shape:(8, 6)  min:-0.319658  max:0.075961
bias  grad shape:(6,)  min:-0.844151  max:0.200598
=== BEFORE MAXPOOLING (Conv2D Output) ===
[[[[0.2  0.   0.53 0.   0.   0.29 0.   0.43]
   [0.14 0.   0.46 0.   0.   0.17 0.   0.33]
   [0.17 0.   0.38 0.   0.   0.33 0.   0.42]]

  [[0.17 0.   0.49 0.   0.   0.13 0.   0.36]
   [0.15 0.   0.31 0.   0.02 0.27 0.   0.26]
   [0.19 0.   0.3  0.   0.06 0.38 0.   0.48]]

  [[0.11 0.   0.59 0.   0.   0.22 0.   0.39]
   [0.16 0.   0.33 0.   0.07 0.2  0.   0.33]
   [0.15 0.   0.41 0.   0.03 0.27 0.   0.43]]]]

=== AFTER MAXPOOLING (MaxPool2D Output) ===
[[0.2  0.   0.53 0.   0.   0.29 0.   0.43 0.14 0.   0.46 0.   0.   0.17
  0.   0.33 0.17 0.   0.38 0.   0.   0.33 0.   0.42 0.17 0.   0.49 0.
  0.   0.13 0.   0.36 0.15 0.   0.31 0.   0.02 0.27 0.   0.26 0.19 0.
  0.3  0.   0.06 0.38 0.   0.48 0.11 0.   0.59 0.   0.   0.22 0.   0.39
  0.16 0.   0.33 0.   0.07 0.2  0.   0.33 0.15 0.   0.41 0.   0.03 0.27
  0.   0.43]]
```



```
6/6 ━━━━━━━━━━━━━━━━━━━━ 2s 75ms/step - accuracy: 0.1595 - loss: 1.8744 - val_accuracy: 0.1667 - val_loss: 1.8479

=== PARAMETER VALUES AFTER TRAINING ===
kernel  → first weight: -0.12117338180541992
bias  → first weight: -0.0028721350245177746
kernel  → first weight: 0.04670596122741699
bias  → first weight: -0.0032798871397972107
kernel  → first weight: -8.006166899576783e-05
bias  → first weight: -0.00011204153270227835
```

CNN Accuracy for Leaf Disease Classification



```
Final Test Accuracy: 0.1667
```

This is the output of given code all snap of output is this.