ABASYN
UNIVERSITY
Passion To Design Futures

---

***Name:Yashfa Javed***

***Registration#9066***

***Assignment#03***

***Subject:Stic(Special Topics In Computing)***

***Submitted To:Ma,am Sadaf***

ABASYN UNIVERSITY

---

**ISLAMABAD CAMPUS**

**Batch:Spring-2024**

**Submitted By :Yashfa Javed**

# ASSIGNMENT#03

# LEAF DISEASE CLASSIFICATION

# -:CNN FUNDAMENTALS:-

## ➢ OBJECTIVES:

**Student should be able to explain**

- forward pass
- Backpropagation
- Gradient descent

**We have a work of modifying the code by using Following changes:**

- Layers: 1
- Epochs: 1
- Seed: 1
- Dense layer: 8×8×32
- Conv2D: (8,8,2)
- Batch size: 32
- Image size: 8×8

---

**After Modification we have to answer the questions which are as given:**

## ➢ QUESTION#01:

1. Create separate files for each disease & healthy images in train-1 & test-1 ( 2 marks)

We modify the code by using Train_1 and Test_1 Folders as per requirement so:

### i. Train_1 Data Set:(Folder)

#### ➢ Functionality:

The TRAIN-1 dataset is used to teach the neural network. The model iteratively adjusts its internal weights and biases using this data to minimize prediction errors, allowing it to learn the features and patterns associated with each disease class.

This folder contains the images the model will learn from

- Healthy

• Early Blight
• Leaf Mold
• Late Blight
• Septoria
• Mosaic Virus

We have an 1 image  in each folder which we get from the Train and Test Folders

## ii. Test_1Data Set:(Folder)

This folder contains the images the model will learn from

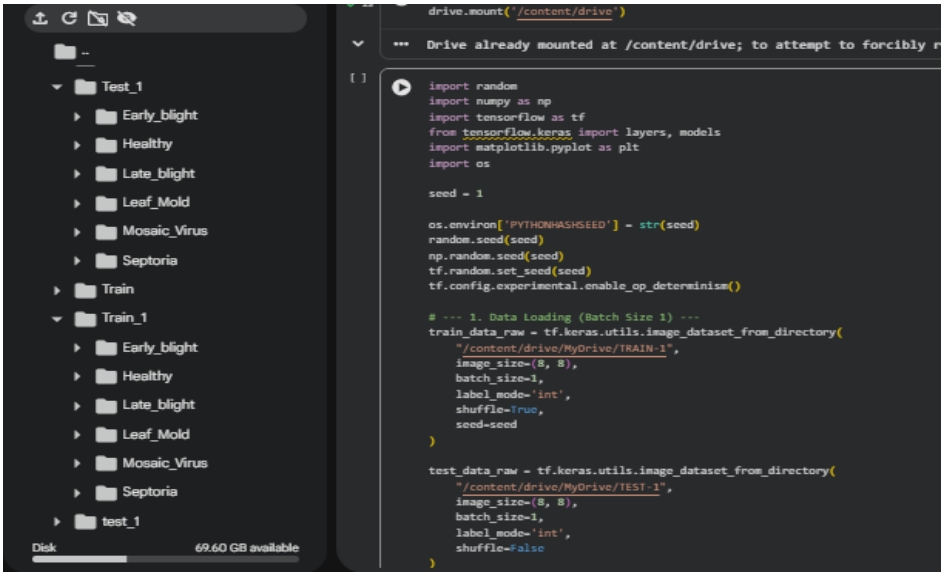
- **Functionality:**

The Test_1 dataset is used to evaluate the final performance of the trained model. Since the model has never seen this data during training, it provides an honest, unbiased measure of how well the network can generalize its learned patterns to new, unseen images.

- Healthy
- Early Blight
- Leaf Mold
- Late Blight
- Septoria
- Mosaic Virus

We have an 1 image of  in each folder which we get from the Train and Test Folders

It would be shown as:



## ➢ QUESTION#02:

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? ( 8 marks)

### 1. Normalization:

Firstly we have to apply normalization so:

How it Works in our Code:

In your code, the raw pixel values for an 8x8 image range from 0 (black) to 255 (white). Normalization is performed using this simple formula:

### 2. Functionality:

- **Stability:** It prevents some input features (or color channels) from having a disproportionately large influence on the model's loss calculation.
- **Speed:** It ensures that the Gradient Descent algorithm (used to update weights) converges (finds the minimum loss) much faster and more efficiently.

## Code Overview:

```
)

# Check first batch BEFORE normalization
np.set_printoptions(precision=2, suppress=True)

for images, labels in train_data_raw.take(1):
    print("Before normalization:")
    print(images.numpy())

# Store class names before mapping
class_names = train_data_raw.class_names

# --- 2. Normalization Mapping ---
train_data = train_data_raw.map(lambda x, y: (x / 255.0, y))
test_data = test_data_raw.map(lambda x, y: (x / 255.0, y))

# --- 3. Model Definition ---
model = models.Sequential([
    layers.Input(shape=(8, 8, 3)),
    layers.Conv2D(8, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(8, activation='relu'),
    layers.Dropout(0.3, seed=seed), # fixed randomness
    layers.Dense(len(class_names), activation='softmax')
])
```

**Output of before and after normalization:**

**Before Normalization:**

```
Found 5977 files belonging to 6 classes.
Found 6 files belonging to 6 classes.
Before normalization:
[[[[147.5  141.5  144.5 ]
   [134.   125.   128.5 ]
   [117.   114.75 112.25]
   [ 75.25 106.75  96.75]
   [ 74.25 108.75  90.5 ]
   [ 56.75  90.75  65.75]
   [ 86.25 129.75  93.25]
   [ 70.    62.25  59.75]]

  [[ 39.25  62.    56.25]
   [ 49.5   72.25  67.5 ]
   [115.25 105.25 104.25]
```

**After Normalization:**

```
--- NORMALIZATION CHECK (AFTER MOVING) ---
After normalization:
[[[[0.66 0.62 0.68]
   [0.66 0.62 0.68]
   [0.63 0.59 0.65]
   [0.56 0.52 0.57]
   [0.64 0.6  0.65]
   [0.63 0.58 0.63]
   [0.6  0.56 0.6 ]
   [0.57 0.53 0.58]]

  [[0.66 0.62 0.67]
   [0.69 0.65 0.7 ]
   [0.69 0.64 0.69]
```

---

- **Convolution:**

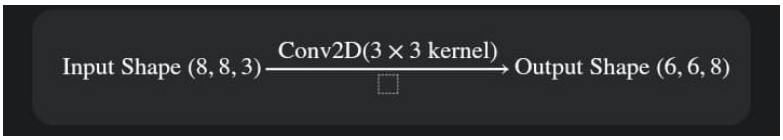
Convolution is the fundamental operation within the Conv2D layer—the core component of your CNN model. Its main role is to automatically detect relevant features in your small 8 \times 8 input images.

**The Role of Convolution in the Network:**

The primary function of the Convolutional layer is Feature Extraction. It learns patterns that are essential for classification, such as edges, corners, and textures specific to the different plant diseases in your dataset.

**Connection to our Code:**

The Conv2D layer transforms our data:




The values we observe in the === BEFORE MAXPOOLING (Conv2D Output) === block of your code are the numerical results (the activated feature maps) of this Convolution operation.

**From Output:**

```
=== BEFORE MAXPOOLING (Conv2D Output) ===
[[[[0.   0.   0.53 0.   0.09 1.32 0.   0.25]
   [0.   0.   0.28 0.15 0.24 0.92 0.   0.12]
   [0.   0.1  0.55 0.08 0.41 0.68 0.14 0.12]]

  [[0.   0.   0.31 0.23 0.13 0.97 0.   0.05]
   [0.   0.   0.23 0.14 0.33 0.88 0.   0.2 ]
   [0.   0.27 0.58 0.44 0.32 0.72 0.05 0.3 ]]

  [[0.   0.06 1.18 0.01 0.31 1.42 0.   0.39]
   [0.   0.02 0.69 0.   0.23 1.16 0.02 0.25]
   [0.   0.08 0.9  0.08 0.34 0.92 0.   0.24]]]]
```

---

- **Maxpooling:**

Max Pooling s hrinks the image/feature map by taking the maximum value from small regions (usually 2×2 blocks).

**Code Overview:**

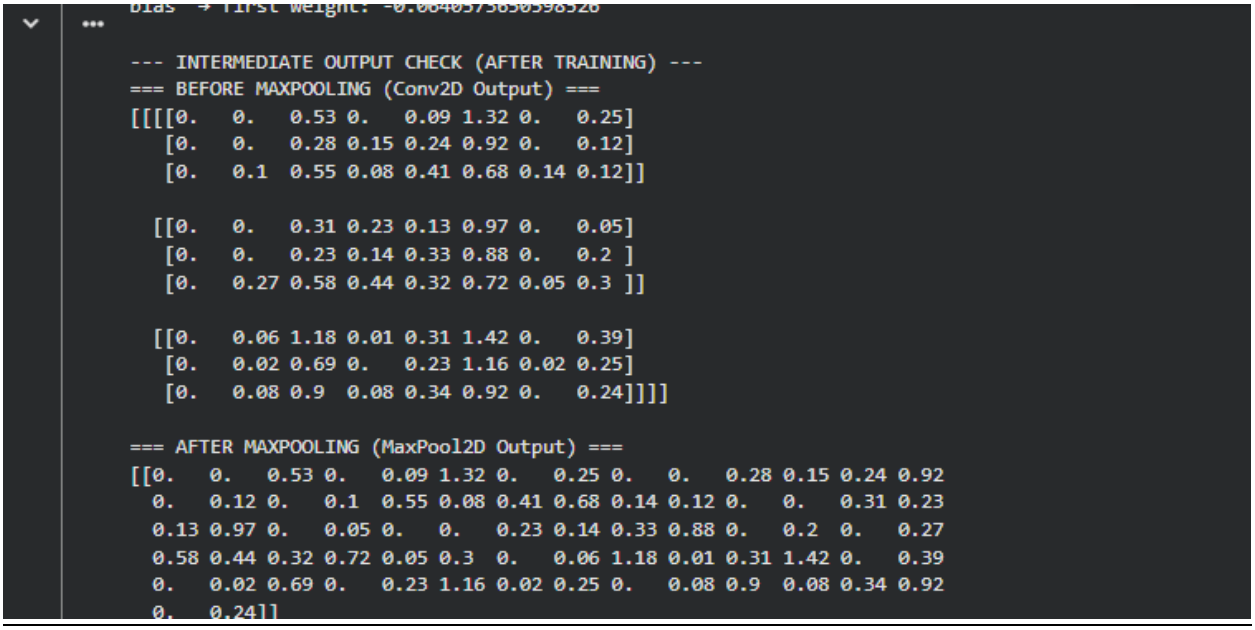
```
[17]
✓ 48s
        del tape

        print("\n=== FORWARD VALUES ===")
        stats("Before MaxPool (Conv2D output)", conv_out.numpy())
        stats("After MaxPool (MaxPool output)", maxpool_out.numpy())

        print("\n=== BACKWARD GRADIENTS ===")
        stats("Gradient dL/d(Conv2D output)", grad_conv.numpy())
        stats("Gradient dL/d(MaxPool output)", grad_pool.numpy())

        print("\n=== TRAINABLE VARIABLE GRADIENTS ===")
        for var, g in zip(model.trainable_variables, grad_vars):
            print(f"{var.name}  grad shape:{g.shape}  min:{tf.reduce_min(g):.6f}  max:{tf.reduce_max(g):.6f}")
        break
    # ---------- End of backprop inspection ----------


    # --- 6. Training ---
    print("\n--- STARTING TRAINING ---")
    history = model.fit(
        train_data,
        validation_data=test_data,
        epochs=1,
        verbose=1
    )
```

## Max Pooling Before and After:

The MaxPooling2D layer's function is to reduce the size of the data and achieve translational invariance (recognizing features regardless of minor shifts).

| Shape | Data Structre | Output Shape | Role/Significance | Proof from Code Output |
|---|---|---|---|---|
| 1. Before Max Pooling | Output of the Conv2D layer (Feature Maps). | (6, 6, 8) | Contains detailed, raw features extracted by the 8 filters. | name:Before MaxPool (Conv2D output) shape:(1, 6, 6, 8) \| |
| 2. After Max Pooling | Max pooling operation (using a 2 \times 2 windows | (3, 3, 8) | Downsampled map. It keeps only the strongest feature signal from every 2 \times 2 region, reducing the data by 75% (6 \times 6 \rightarrow 3 \times 3). | name:After MaxPool (MaxPool output) shape:(1, 72) (The 3 \times 3 \times 8 is flattened to 72) |

## Output:



```
Bias -> First Weight: -0.08405736505998526

--- INTERMEDIATE OUTPUT CHECK (AFTER TRAINING) ---
=== BEFORE MAXPOOLING (Conv2D Output) ===
[[[[0.   0.   0.53 0.   0.09 1.32 0.   0.25]
   [0.   0.   0.28 0.15 0.24 0.92 0.   0.12]
   [0.   0.1  0.55 0.08 0.41 0.68 0.14 0.12]]

  [[0.   0.   0.31 0.23 0.13 0.97 0.   0.05]
   [0.   0.   0.23 0.14 0.33 0.88 0.   0.2 ]
   [0.   0.27 0.58 0.44 0.32 0.72 0.05 0.3 ]]

  [[0.   0.06 1.18 0.01 0.31 1.42 0.   0.39]
   [0.   0.02 0.69 0.   0.23 1.16 0.02 0.25]
   [0.   0.08 0.9  0.08 0.34 0.92 0.   0.24]]]]

=== AFTER MAXPOOLING (MaxPool2D Output) ===
[[0.   0.   0.53 0.   0.09 1.32 0.   0.25 0.   0.   0.28 0.15 0.24 0.92
  0.   0.12 0.   0.1  0.55 0.08 0.41 0.68 0.14 0.12 0.   0.   0.31 0.23
  0.13 0.97 0.   0.05 0.   0.   0.23 0.14 0.33 0.88 0.   0.2  0.   0.27
  0.58 0.44 0.32 0.72 0.05 0.3  0.   0.06 1.18 0.01 0.31 1.42 0.   0.39
  0.   0.02 0.69 0.   0.23 1.16 0.02 0.25 0.   0.08 0.9  0.08 0.34 0.92
  0.   0.24]]
```

# OBJECTIVES EXPLANATION WITH CODE PROOF:

## 1.Forward pass:

The Forward Pass: Prediction Generation

The Forward Pass is the initial phase where the input data travels from the first layer to the last layer to generate a prediction. It is essentially the flow of information through the network.

**Process:** The normalized image first undergoes feature extraction in the Conv2D layer, then dimensionality reduction in the Max Pooling layer, before being passed to the Dense layers for final classification.

**Role:** This step yields the network's output, which is compared to the true label to calculate the Loss (error).

**Code overview:**

```
print("\n=== FORWARD VALUES ===")
stats("Before MaxPool (Conv2D output)", conv_out.numpy())
stats("After MaxPool (MaxPool output)", maxpool_out.numpy())
```

**Output:**

```
=== FORWARD VALUES ===
Before MaxPool (Conv2D output) shape:(1, 3, 3, 8)  min:0.000000  max:0.859276  mean:0.237326
After MaxPool (MaxPool output) shape:(1, 72)  min:0.000000  max:0.859276  mean:0.237326
```

---

## 2.Backpropagation (Error Attribution):

Backpropagation is the crucial step immediately following the calculation of Loss. It is an algorithm that sends the error signal backward through the network to determine how much each parameter contributed to the mistake.

**Process:** The Loss is mathematically broken down across every neuron and connection, calculating the Gradient for each weight and bias.

**Role:** The resulting Gradient is the slope of the loss function, indicating the direction and magnitude needed to change the weight to minimize the error.

**Code overview:**

```
[17]
✓ 48s
# ---------- 5. Backprop inspection block (Moved before model.fit) ----------
print("\n--- BACKPROPAGATION & FORWARD PASS INSPECTION ---")
conv_layer = model.layers[1]   # Conv2D
pool_layer = model.layers[2]   # MaxPool

probe_model = tf.keras.Model(
    inputs=model.inputs[0],
    outputs=[conv_layer.output, pool_layer.output, model.layers[-1].output]
)

def stats(name, t):
    t = np.array(t)
    print(f"{name} shape:{t.shape}  min:{t.min():.6f}  max:{t.max():.6f}  mean:{t.mean():.6f}")

for images, labels in train_data.take(1):
    images = tf.cast(images, tf.float32)
    labels = tf.cast(labels, tf.int32)

    with tf.GradientTape(persistent=True) as tape:
        conv_out, maxpool_out, preds = probe_model(images, training=False)
        loss = tf.reduce_mean(
            tf.keras.losses.sparse_categorical_crossentropy(labels, preds)
        )

    tape.watch(conv_out)
    tape.watch(maxpool_out)
```

**Output:**

```
--- BACKPROPAGATION & FORWARD PASS INSPECTION ---

=== FORWARD VALUES ===
Before MaxPool (Conv2D output) shape:(1, 3, 3, 8)  min:0.000000  max:0.859276  mean:0.237326
After MaxPool (MaxPool output) shape:(1, 72)  min:0.000000  max:0.859276  mean:0.237326

=== BACKWARD GRADIENTS ===
Gradient dL/d(Conv2D output) shape:(1, 3, 3, 8)  min:-0.217208  max:0.254713  mean:-0.004727
Gradient dL/d(MaxPool output) shape:(1, 72)  min:-0.217208  max:0.254713  mean:-0.004727

=== TRAINABLE VARIABLE GRADIENTS ===
kernel  grad shape:(3, 3, 3, 8)  min:-0.311763  max:0.318924
bias  grad shape:(8,)  min:-0.391561  max:0.249434
kernel  grad shape:(72, 8)  min:-0.369071  max:0.440967
bias  grad shape:(8,)  min:-0.429514  max:0.513184
kernel  grad shape:(8, 6)  min:-0.433956  max:0.094885
bias  grad shape:(6,)  min:-0.815090  max:0.178221
```

---

## 3.Gradient Descent (Parameter Optimization):

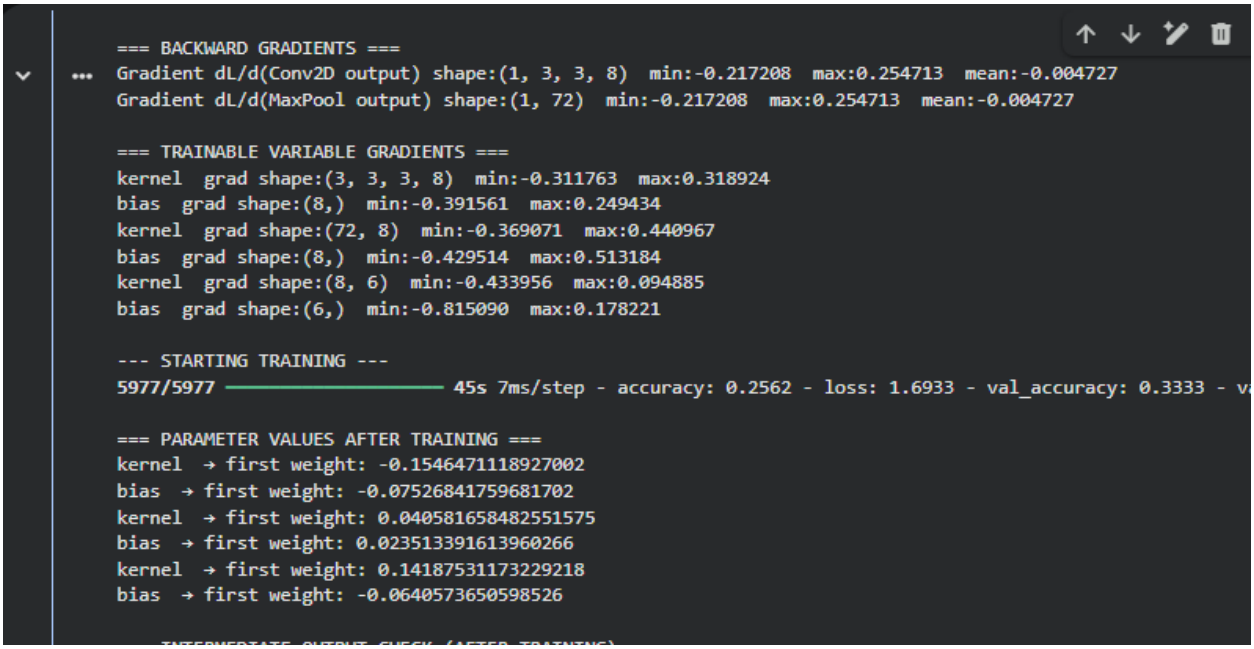
Gradient Descent (handled by the Adam optimizer in your code) is the final mechanism that uses the gradients to execute the learning process.

**Process:** Weights and biases are adjusted in the opposite direction of the gradient—like walking downhill—to continually decrease the Loss.

**Rule:** The update is calculated as: \text{New Weight} = \text{Old Weight} - (\text{Learning Rate} \times \text{Gradient}).

**Role:** This adjustment completes the learning cycle, making the model slightly more accurate for the next epoch.

**Output:**



```
=== BACKWARD GRADIENTS ===
Gradient dL/d(Conv2D output) shape:(1, 3, 3, 8)  min:-0.217208  max:0.254713  mean:-0.004727
Gradient dL/d(MaxPool output) shape:(1, 72)  min:-0.217208  max:0.254713  mean:-0.004727

=== TRAINABLE VARIABLE GRADIENTS ===
kernel  grad shape:(3, 3, 3, 8)  min:-0.311763  max:0.318924
bias  grad shape:(8,)  min:-0.391561  max:0.249434
kernel  grad shape:(72, 8)  min:-0.369071  max:0.440967
bias  grad shape:(8,)  min:-0.429514  max:0.513184
kernel  grad shape:(8, 6)  min:-0.433956  max:0.094885
bias  grad shape:(6,)  min:-0.815090  max:0.178221

--- STARTING TRAINING ---
5977/5977 ━━━━━━━━━━━━━━━━━━━━ 45s 7ms/step - accuracy: 0.2562 - loss: 1.6933 - val_accuracy: 0.3333 - v

=== PARAMETER VALUES AFTER TRAINING ===
kernel  → first weight: -0.1546471118927002
bias  → first weight: -0.07526841759681702
kernel  → first weight: 0.040581658482551575
bias  → first weight: 0.023513391613960266
kernel  → first weight: 0.14187531173229218
bias  → first weight: -0.0640573650598526
```

**THE END**