



Name: Hamid Saleem

Reg no: 9061

Subject: STIC

Assignment no: 3

FORWARD PASS

So basically, forward pass is the part where the model *guesses*.
When an image goes into the model, layer by layer the model changes it.
First the Conv2D makes small patterns from the image.
Then MaxPool reduces the size.
Then Dense layers try to understand what the image actually is.

In simple words

Forward pass is when the model takes the input and tries to make a prediction.

There is no correction here just a pure guess.

In our code forward pass happens when we run

```
conv_out, maxpool_out, preds = probe_model(images, training=False)
```

This line gives

- conv_out = what Conv2D sees
- maxpool_out = after pooling
- preds = final guess

So this is the forward pass.

BACKPROPAGATION

After the model makes a guess, it checks to see how wrong it was.
This wrong number is called a loss.

In short backpropagation is

Sending the mistake back so that each layer knows how bad it was.

The model figures out gradients that show how much each weight added to the error and how much it should change.

In Code this part is starting here.

```
with tf.GradientTape(persistent=True) as tape:
```

The tape watches everything happening during the forward pass.

Then it calculates the loss and then it finds gradients

```
grad_vars = tape.gradient(loss, model.trainable_variables)
```

These numbers say

How much to change Conv2D

How much to update dense layers

How much to raise or lower bias values

So, backpropagation means that the error goes back.

GRADIENT DESCENT

Now when each weight knows its gradient gradient descent updates it.

Gradient Descent is like telling the model

'Okay, now move your weights a little so next time you make fewer mistakes.'

The optimizer Adam in our case uses gradients and changes the weight values.

CODE

```
import random

import numpy as np

import tensorflow as tf

from tensorflow.keras import layers, models

import matplotlib.pyplot as plt

import os

seed = 1

os.environ['PYTHONHASHSEED'] = str(seed)

random.seed(seed)

np.random.seed(seed)
```

```

tf.random.set_seed(seed)

tf.config.experimental.enable_op_determinism()

train_data_raw = tf.keras.utils.image_dataset_from_directory(
    "/content/drive/MyDrive/Train_1",
    image_size=(8, 8),
    batch_size=1,
    label_mode='int',
    shuffle=True,
    seed=seed
)

test_data_raw = tf.keras.utils.image_dataset_from_directory(
    "/content/drive/MyDrive/Test_1",
    image_size=(8, 8),
    batch_size=1,
    label_mode='int',
    shuffle=False
)

# 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

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))

# Check the same batch AFTER normalization (moved after train_data
definition)
np.set_printoptions(precision=2, suppress=True)

for images, labels in train_data.take(1):
    print("After normalization:")
    print(images.numpy())
    break

```

```

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')
])

model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

print("=== PARAMETER VALUES BEFORE TRAINING ===")
for var in model.trainable_variables:
    print(var.name, " → first weight:", float(var.numpy().flatten()[0]))

# Explicitly call the model with a dummy input to ensure it's built
# and its layers' output tensors are instantiated.
# The input_shape for build is equivalent to the input of the first layer
dummy_input_for_build = tf.zeros(shape=(1, 8, 8, 3))
_ = model(dummy_input_for_build)

# ----- Backprop inspection block -----
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] # Changed model.output to 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: # Made tape persistent
    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)

grad_conv = tape.gradient(loss, conv_out)
grad_pool = tape.gradient(loss, maxpool_out)
grad_vars = tape.gradient(loss, model.trainable_variables)

del tape # Delete the persistent tape to free resources

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 -----

# Create a model that outputs intermediate layer values
conv_layer_model = tf.keras.Model(
    inputs=model.inputs[0],
    outputs=[
        model.layers[1].output, # Conv2D output
        model.layers[2].output # MaxPooling output
    ]
)

# Take one image from test dataset
for images, labels in test_data.take(1):

```

```

conv_out, maxpool_out = conv_layer_model(images)

print("=== BEFORE MAXXPOOLING (Conv2D Output) ===")
print(conv_out.numpy())

print("\n=== AFTER MAXXPOOLING (MaxPool2D Output) ===")
print(maxpool_out.numpy())

history = model.fit(
    train_data,
    validation_data=test_data,
    epochs=1,
    verbose=1
)
print("\n=== PARAMETER VALUES AFTER TRAINING ===")
for var in model.trainable_variables:
    print(var.name, " → first weight:", float(var.numpy().flatten()[0]))

plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('CNN Accuracy for Leaf Disease Classification')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

test_loss, test_acc = model.evaluate(test_data, verbose=0)
print(f" Final Test Accuracy: {test_acc:.4f}")
print("Classess:", class_names)

```

OUTPUT

```

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. ]
 [137. 124. 120.5 ]
 [134.25 123.25 121.25]]

[[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:

```
[[[0.64 0.64 0.66]
  [0.6  0.59 0.61]
  [0.61 0.6  0.62]
  [0.56 0.55 0.58]
  [0.51 0.53 0.51]
  [0.57 0.56 0.58]
  [0.54 0.53 0.55]
  [0.58 0.57 0.59]]]
```

```
[ [0.6  0.59 0.61]
  [0.54 0.53 0.55]
  [0.5  0.5  0.51]
  [0.28 0.35 0.31]
  [0.34 0.49 0.44]
  [0.55 0.55 0.56]
  [0.54 0.54 0.56]
  [0.54 0.53 0.55]]]
```

```
[ [0.59 0.58 0.6 ]
  [0.5  0.49 0.5 ]
  [0.45 0.45 0.46]
  [0.26 0.39 0.32]
  [0.18 0.23 0.18]
  [0.26 0.35 0.2 ]
  [0.38 0.45 0.34]
  [0.48 0.47 0.49]]]
```

```
[ [0.56 0.55 0.55]
  [0.53 0.51 0.52]
  [0.1  0.08 0.05]
  [0.17 0.17 0.16]
  [0.18 0.33 0.2 ]
  [0.26 0.44 0.34]
  [0.13 0.22 0.08]
  [0.47 0.47 0.49]]]
```

```
[ [0.51 0.49 0.5 ]
  [0.47 0.46 0.49]
  [0.24 0.31 0.24]
  [0.26 0.41 0.31]
  [0.25 0.35 0.26]
  [0.29 0.48 0.39]
  [0.35 0.52 0.41]
  [0.42 0.42 0.44]]]
```

```
[ [0.49 0.47 0.48]
  [0.42 0.41 0.4 ]
  [0.26 0.27 0.23]
  [0.12 0.17 0.08]
```

```

[0.23 0.3 0.23]
[0.17 0.34 0.23]
[0.28 0.25 0.23]
[0.45 0.43 0.45]]

[[0.46 0.44 0.45]
[0.15 0.25 0.05]
[0.2 0.3 0.22]
[0.19 0.37 0.17]
[0.24 0.26 0.15]
[0.27 0.22 0.18]
[0.37 0.35 0.35]
[0.45 0.43 0.45]]

[[0.52 0.5 0.51]
[0.47 0.43 0.45]
[0.42 0.36 0.31]
[0.35 0.3 0.28]
[0.42 0.39 0.38]
[0.4 0.36 0.35]
[0.45 0.41 0.42]
[0.48 0.46 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:0.583730
mean:0.151219
After MaxPool (MaxPool output) shape:(1, 72) min:0.000000 max:0.583730
mean:0.151219

=== BACKWARD GRADIENTS ===
Gradient dL/d(Conv2D output) shape:(1, 3, 3, 8) min:-0.347020 max:0.268906
mean:-0.001429
Gradient dL/d(MaxPool output) shape:(1, 72) min:-0.347020 max:0.268906
mean:-0.001429

=== TRAINABLE VARIABLE GRADIENTS ===
kernel grad shape:(3, 3, 3, 8) min:-0.382339 max:0.189027
bias grad shape:(8,) min:-0.748281 max:0.464358
kernel grad shape:(72, 8) min:-0.261841 max:0.330442
bias grad shape:(8,) min:-0.448566 max:0.566088
kernel grad shape:(8, 6) min:-0.228760 max:0.052906
bias grad shape:(6,) min:-0.841885 max:0.194704

=== BEFORE MAXXPOOLING (Conv2D Output) ===
[[[0.25 0. 0.57 0. 0. 0.24 0. 0.47]
[0.12 0. 0.44 0. 0. 0.17 0. 0.18]
[0.18 0. 0.33 0. 0.07 0.18 0. 0.3 ]]

[[0.14 0. 0.41 0. 0.04 0.19 0. 0.32]
[0.06 0. 0.45 0. 0.04 0.18 0. 0.19]
[0.14 0. 0.36 0. 0.08 0.4 0. 0.49]]]

```

```
[[[0.12 0.    0.76 0.    0.03 0.48 0.    0.38]
  [0.2  0.    0.43 0.    0.18 0.4  0.    0.57]
  [0.19 0.    0.59 0.    0.06 0.46 0.    0.51]]]]
```

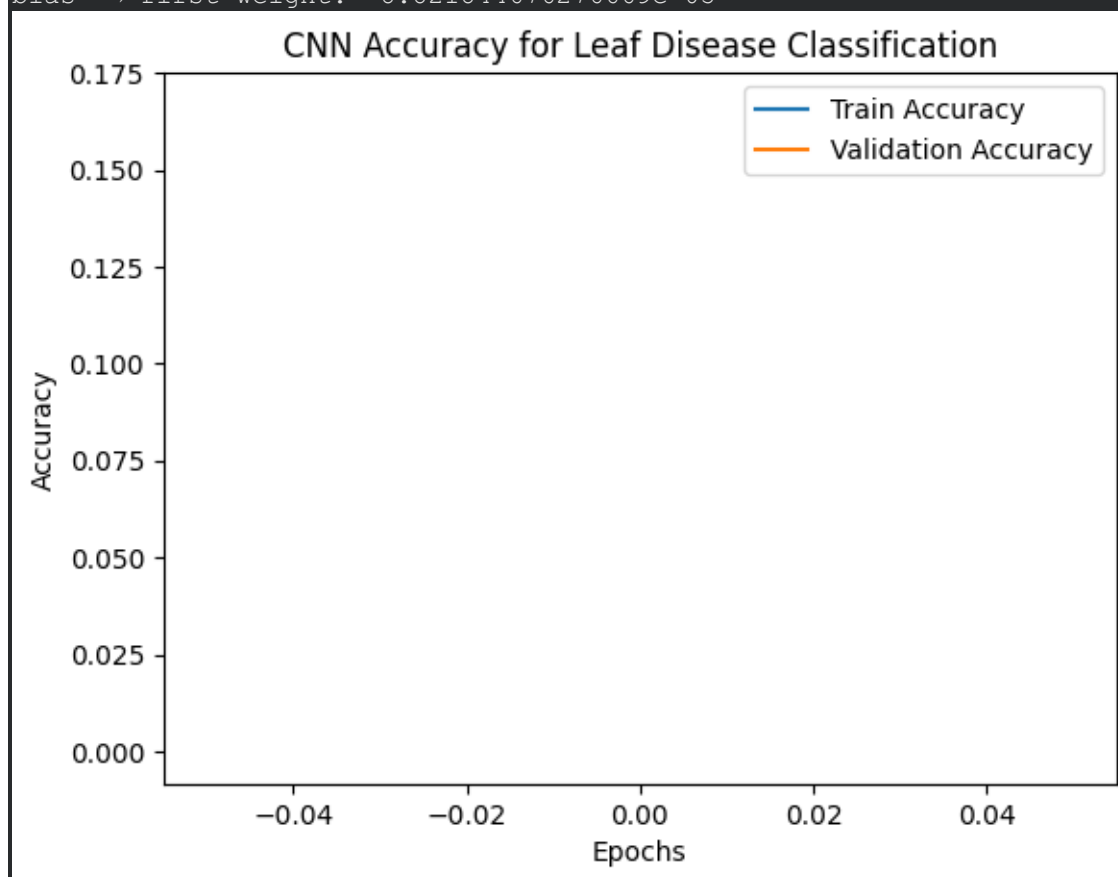
=== AFTER MAXPOOLING (MaxPool2D Output) ===

```
[[[0.25 0.    0.57 0.    0.    0.24 0.    0.47 0.12 0.    0.44 0.    0.    0.17
  0.    0.18 0.18 0.    0.33 0.    0.07 0.18 0.    0.3  0.14 0.    0.41 0.
  0.04 0.19 0.    0.32 0.06 0.    0.45 0.    0.04 0.18 0.    0.19 0.14 0.
  0.36 0.    0.08 0.4  0.    0.49 0.12 0.    0.76 0.    0.03 0.48 0.    0.38
  0.2  0.    0.43 0.    0.18 0.4  0.    0.57 0.19 0.    0.59 0.    0.06 0.46
  0.    0.51]]]
```

6/6 ————— 2s 59ms/step - accuracy: 0.0000e+00 -
loss: 1.8523 - val_accuracy: 0.1667 - val_loss: 1.7861

=== PARAMETER VALUES AFTER TRAINING ===

```
kernel → first weight: -0.11762837320566177
bias   → first weight: -0.0027079577557742596
kernel → first weight: 0.049985647201538086
bias   → first weight: 0.0
kernel → first weight: 0.003199338912963867
bias   → first weight: -8.821844676276669e-05
```



Final Test Accuracy: 0.1667
Classess: ['early_blight', 'healthy', 'late_blight', 'leaf_mold',
'mosaic_virus', 'septoria_spot']