AI LAB ASSIGNMENT

MD: SHANTO BABU

ID: 2111176155

SESSION: 2020-21

DEPT: COMPUTER SCIENCE AND ENGINEERING

COURSE: ARTIFICIAL INTELLIGENCE LAB

Github Link: https://github.com/shanto155/AI_Lab_Assignment.git

Google Colab Link:
https://colab.research.google.com/drive/1Ci8DiQinh3zD5vGt1r3lPmHBw3qKaVfs?usp=sharing

Assignment 1

Fully Connected Feed-Forward Neural Network (FCFNN) Architecture

Explanation

A Fully Connected Feed-Forward Neural Network (FCFNN) is a neural network where:

- Every neuron in one layer is connected to every neuron in the next layer.
- Data flows only forward (no loops).

Given:

- Input layer: 8 neurons
- Hidden Layer 1: 4 neurons
- Hidden Layer 2: 8 neurons
- Hidden Layer 3: 4 neurons
- Output layer: 10 neurons

Total Layers:

Input → Hidden1 → Hidden2 → Hidden3 → Output

Output Description:

- Total trainable parameters depend on:
  - (8×4 + 4 bias)
  - (4×8 + 8 bias)
  - (8×4 + 4 bias)
  - (4×10 + 10 bias)

The network produces a 10-**dimensional** output vector (for classification problems).

github link: https://github.com/shanto155/AI_Lab_Assignment/blob/main/Assignment1.py

ASSIGNMENT 2

FCFNN Implementation Using TensorFlow/Keras

Explanation

Steps:

1. Define Sequential Model
2. Add Dense layers
3. Compile model
4. Train model
5. Evaluate model

Expected Output

After training:

- Training accuracy increases gradually
- Validation accuracy stabilizes
- Loss decreases over epochs

github link: https://github.com/shanto155/AI_Lab_Assignment/blob/main/Assignment2.py

ASSIGNMENT 3

Solving Mathematical Equations Using FCFNN

Equations

1. $y = 5x + 10$
2. $y = 3x^2 + 5x + 10$
3. $y = 4x^3 + 3x^2 + 5x + 10$

Explanation

- Generate dataset (x values randomly)
- Split into:
    - Training (70%)
    - Validation (15%)
    - Test (15%)

Model Design

For linear equation:

- Small network sufficient (1 hidden layer)

For quadratic:

- More neurons required

For cubic:

- Deeper network needed

Output

After training:

- Linear: Very low error
- Quadratic: Moderate neurons required
- Cubic: Higher complexity

Graph:

- Original y vs Predicted y (overlapping curves)

Effect of Power

Higher power →

- More nonlinear relationship
- Requires more neurons
- Requires more training data

github link: https://github.com/shanto155/AI_Lab_Assignment/blob/main/Assignment3.py

ASSIGNMENT 4

FCFNN Classifier

Datasets:

- Fashion MNIST
- MNIST
- CIFAR-10

Explanation

- Flatten image
- Dense layers
- Softmax output (10 classes)

Expected Results

Fashion MNIST:
Accuracy ≈ 85–90%

MNIST:
Accuracy ≈ 95–98%

CIFAR-10:
Accuracy ≈ 50–60% (FCFNN not ideal)

Observation:
FCFNN performs poorly on complex images like CIFAR-10.

github link: https://github.com/shanto155/AI_Lab_Assignment/blob/main/Assignment4.py

ASSIGNMENT 5

CNN Based 10-Class Classifier

Explanation

CNN Layers:

- Conv2D
- ReLU
- MaxPooling
- Flatten
- Dense

Results

MNIST:
Accuracy ≈ 99%

Fashion MNIST:
Accuracy ≈ 92–94%

CIFAR-10:
Accuracy ≈ 70–80%

Observation:
CNN outperforms FCFNN because it preserves spatial information.

github link: https://github.com/shanto155/AI_Lab_Assignment/blob/main/Assignment5.py

ASSIGNMENT 6

Handwritten Dataset + Retraining

Steps:

1. Collect handwritten digits
2. Resize to 28×28
3. Normalize
4. Merge with MNIST training data
5. Retrain FCFNN

Expected Output

- Improved generalization
- Test accuracy slightly reduced if handwriting style differs

Observation:
Custom dataset helps model learn real-world variations.

github link: https://github.com/shanto155/AI_Lab_Assignment/blob/main/Assignment6.py

ASSIGNMENT 7

CNN with Mobile Captured Images

Explanation

Train CNN with custom captured images.

Measure:

- Training time
- Testing time per sample
- Epoch vs Accuracy
- Model size vs Accuracy

Observations

- More data → Better performance
- More epochs → Risk of overfitting
- Larger model → Better accuracy but slower

Example:

Training time: 15 minutes
Test time per image: 3 ms
Accuracy: 88%

github link: https://github.com/shanto155/AI_Lab_Assignment/blob/main/Assignment7.py

ASSIGNMENT 8

VGG16-like CNN Architecture

Based on:
VGG16

Architecture

- Conv(64) ×2
- MaxPool
- Conv(128) ×2
- MaxPool
- Conv(256) ×3
- Fully connected layers

Output:

- Deep feature extraction
- High accuracy
- Large number of parameters (~138M original)

github link: https://github.com/shanto155/AI_Lab_Assignment/blob/main/Assignment8.py

ASSIGNMENT 9

Feature Maps Visualization

Pretrained Models Example:

- VGG16
- ResNet50
- MobileNet

Observation

Early layers:

- Detect edges, corners

Middle layers:

- Detect shapes

Deep layers:

- Detect objects

Feature maps become more abstract in deeper layers.

github link: https://github.com/shanto155/AI_Lab_Assignment/blob/main/Assignment9.py

ASSIGNMENT 10

Transfer Learning using VGG16

Explanation

Use pretrained VGG16:

- Freeze base layers
- Add custom Dense layers

Fine-Tuning Effects

Whole VGG16 fine-tuned:

- Higher accuracy
- Longer training time

Partial fine-tuning:

- Faster
- Slightly lower accuracy

Observation:
Fine-tuning improves performance significantly.

github link: https://github.com/shanto155/AI_Lab_Assignment/blob/main/Assignment10.py

ASSIGNMENT 11

Feature Extraction Power (PCA & t-SNE)

Techniques:

- Principal Component Analysis
- t-distributed Stochastic Neighbor Embedding

Before Transfer Learning:

- Classes not clearly separable

After Transfer Learning:

- Clear clustering in 2D space

Conclusion:
Transfer learning improves feature separability.

github link: https://github.com/shanto155/AI_Lab_Assignment/blob/main/Assignment11.py

ASSIGNMENT 12

Effect of Data Augmentation

Techniques:

- Rotation
- Flipping
- Zoom
- Shift

Result

Without augmentation:

- Overfitting occurs

With augmentation:

- Better generalization
- Higher validation accuracy

github link: https://github.com/shanto155/AI_Lab_Assignment/blob/main/Assignment12.py

ASSIGNMENT 13

Dropout & Overfitting

Dropout randomly disables neurons during training.

Result

Without Dropout:

- Training accuracy high
- Validation accuracy low

With Dropout:

- Training accuracy moderate
- Validation accuracy improved

Conclusion:
Dropout reduces overfitting.

github link: https://github.com/shanto155/AI_Lab_Assignment/blob/main/Assignment13.py

ASSIGNMENT 14

Activation & Loss Functions

Activation Functions:

- ReLU
- Sigmoid
- Tanh

ReLU performs best for deep networks.

Loss Functions:

- Categorical Crossentropy
- Binary Crossentropy
- Mean Squared Error

For classification:
Categorical Crossentropy gives best results.

github link: https://github.com/shanto155/AI_Lab_Assignment/blob/main/Assignment14.py

ASSIGNMENT 15

Callback Functions

Important Callbacks:

- EarlyStopping
- ModelCheckpoint
- ReduceLROnPlateau

Benefits:

- Prevent overfitting
- Save best model
- Reduce learning rate automatically

github link: https://github.com/shanto155/AI_Lab_Assignment/blob/main/Assignment15.py

ASSIGNMENT 16

Monitoring Training & Validation Curves

Plot:

- Accuracy vs Epoch
- Loss vs Epoch

Observation

If:
Training accuracy ↑
Validation accuracy ↓

→ Overfitting

If both increase smoothly:
→ Good model

Monitoring helps:

- Tune learning rate
- Tune batch size
- Tune architecture

github link: https://github.com/shanto155/AI_Lab_Assignment/blob/main/Assignment16.py

AI_Lab_Final_Assignment

February 23, 2026

[ ]:
```python
# ASSIGNMENT 1
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Create the model
model = Sequential()

# Input layer (8 features) -> Hidden Layer 1 (4 neurons)
model.add(Dense(4, activation='relu', input_shape=(8,)))

# Hidden Layer 2 (8 neurons)
model.add(Dense(8, activation='relu'))

# Hidden Layer 3 (4 neurons)
model.add(Dense(4, activation='relu'))

# Output Layer (10 neurons)
model.add(Dense(10, activation='softmax'))

# Compile the model
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Print model summary
model.summary()

# Optional: Save model architecture image
from tensorflow.keras.utils import plot_model
plot_model(model, to_file='Problem_1.png', show_shapes=True)

print("\nModel successfully created.")
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
```

```
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 4)	36
dense_1 (Dense)	(None, 8)	40
dense_2 (Dense)	(None, 4)	36
dense_3 (Dense)	(None, 10)	50

Total params: 162 (648.00 B)

Trainable params: 162 (648.00 B)

Non-trainable params: 0 (0.00 B)

```
Model successfully created.
```

[ ]:
```python
# Assignment 2

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Create model
model = Sequential()

# Input layer (6 features) → Hidden Layer 1 (12 neurons)
model.add(Dense(12, activation='relu', input_shape=(6,)))

# Hidden Layer 2 (8 neurons)
model.add(Dense(8, activation='relu'))

# Output Layer (3 classes)
model.add(Dense(3, activation='softmax'))
```

```
# Compile model
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Print summary
model.summary()

# Save model architecture image
from tensorflow.keras.utils import plot_model
plot_model(model, to_file='Problem_2.png', show_shapes=True)

print("Model successfully built.")
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 12)	84
dense_5 (Dense)	(None, 8)	104
dense_6 (Dense)	(None, 3)	27

Total params: 215 (860.00 B)

Trainable params: 215 (860.00 B)

Non-trainable params: 0 (0.00 B)

```
Model successfully built.
```

[ ]:
```
#ASSIGNMENT 3

import numpy as np
```

```
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.model_selection import train_test_split

# Generate data
x = np.linspace(-10, 10, 1000)
x = x.reshape(-1,1)

# Choose equation
def linear(x):
    return 5*x + 10

def quadratic(x):
    return 3*x**2 + 5*x + 10

def cubic(x):
    return 4*x**3 + 3*x**2 + 5*x + 10

y = quadratic(x)   # Change to linear(x) or quadratic(x)

# Split data
X_train, X_temp, y_train, y_temp = train_test_split(x, y, test_size=0.3,␣
 ↪random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,␣
 ↪random_state=42)

# Build model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(1,)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])

model.compile(optimizer='adam', loss='mse')

# Train
history = model.fit(X_train, y_train,
                    validation_data=(X_val, y_val),
                    epochs=200,
                    batch_size=32,
                    verbose=1)

# Predict
y_pred = model.predict(X_test)

# Plot
plt.scatter(X_test, y_test, label="Original")
```

```
plt.scatter(X_test, y_pred, label="Predicted")
plt.legend()
plt.title("Original vs Predicted")
plt.savefig("Problem_3_quadratic.png")
plt.show()

print("Test Loss:", model.evaluate(X_test, y_test))
```

```
Epoch 1/200
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
22/22             2s 17ms/step -
loss: 19335.7871 - val_loss: 22076.7637
Epoch 2/200
22/22             0s 10ms/step -
loss: 17714.8125 - val_loss: 20873.3574
Epoch 3/200
22/22             0s 8ms/step - loss:
18032.5508 - val_loss: 18693.3242
Epoch 4/200
22/22             0s 8ms/step - loss:
15808.8994 - val_loss: 15204.3271
Epoch 5/200
22/22             0s 8ms/step - loss:
11909.5381 - val_loss: 10744.2871
Epoch 6/200
22/22             0s 8ms/step - loss:
8371.1045 - val_loss: 6263.9077
Epoch 7/200
22/22             0s 10ms/step -
loss: 4334.6406 - val_loss: 3081.1140
Epoch 8/200
22/22             0s 9ms/step - loss:
2198.5540 - val_loss: 1728.3716
Epoch 9/200
22/22             0s 10ms/step -
loss: 1362.0835 - val_loss: 1402.5713
Epoch 10/200
22/22             0s 6ms/step - loss:
1260.4105 - val_loss: 1336.5270
Epoch 11/200
22/22             0s 5ms/step - loss:
1202.2454 - val_loss: 1306.7502
```

```
Epoch 12/200
22/22             0s 7ms/step - loss:
1098.1608 - val_loss: 1277.5244
Epoch 13/200
22/22             0s 5ms/step - loss:
1107.1057 - val_loss: 1251.4230
Epoch 14/200
22/22             0s 5ms/step - loss:
1068.5057 - val_loss: 1223.7413
Epoch 15/200
22/22             0s 5ms/step - loss:
1072.2699 - val_loss: 1196.0757
Epoch 16/200
22/22             0s 5ms/step - loss:
1002.0723 - val_loss: 1169.0874
Epoch 17/200
22/22             0s 5ms/step - loss:
981.8465 - val_loss: 1132.3330
Epoch 18/200
22/22             0s 5ms/step - loss:
1002.2191 - val_loss: 1122.1168
Epoch 19/200
22/22             0s 5ms/step - loss:
926.3207 - val_loss: 1086.0903
Epoch 20/200
22/22             0s 5ms/step - loss:
949.3307 - val_loss: 1063.1019
Epoch 21/200
22/22             0s 5ms/step - loss:
863.0128 - val_loss: 1050.1753
Epoch 22/200
22/22             0s 6ms/step - loss:
918.7773 - val_loss: 1012.9944
Epoch 23/200
22/22             0s 5ms/step - loss:
850.4097 - val_loss: 995.0112
Epoch 24/200
22/22             0s 5ms/step - loss:
836.4182 - val_loss: 963.8298
Epoch 25/200
22/22             0s 5ms/step - loss:
874.5887 - val_loss: 945.2871
Epoch 26/200
22/22             0s 5ms/step - loss:
793.6490 - val_loss: 925.9267
Epoch 27/200
22/22             0s 5ms/step - loss:
836.2860 - val_loss: 894.5955
```

```
Epoch 28/200
22/22             0s 5ms/step - loss:
787.0983 - val_loss: 877.0175
Epoch 29/200
22/22             0s 7ms/step - loss:
750.8326 - val_loss: 858.7438
Epoch 30/200
22/22             0s 5ms/step - loss:
720.3594 - val_loss: 829.7684
Epoch 31/200
22/22             0s 6ms/step - loss:
758.5792 - val_loss: 805.9435
Epoch 32/200
22/22             0s 5ms/step - loss:
749.3898 - val_loss: 791.8768
Epoch 33/200
22/22             0s 5ms/step - loss:
693.0023 - val_loss: 767.8503
Epoch 34/200
22/22             0s 5ms/step - loss:
649.9025 - val_loss: 746.4349
Epoch 35/200
22/22             0s 5ms/step - loss:
618.3686 - val_loss: 719.4184
Epoch 36/200
22/22             0s 5ms/step - loss:
620.4379 - val_loss: 702.8780
Epoch 37/200
22/22             0s 5ms/step - loss:
583.3550 - val_loss: 682.7580
Epoch 38/200
22/22             0s 5ms/step - loss:
589.0547 - val_loss: 662.5276
Epoch 39/200
22/22             0s 5ms/step - loss:
583.9866 - val_loss: 645.3512
Epoch 40/200
22/22             0s 5ms/step - loss:
569.5598 - val_loss: 633.6874
Epoch 41/200
22/22             0s 5ms/step - loss:
536.6678 - val_loss: 609.5478
Epoch 42/200
22/22             0s 5ms/step - loss:
528.4534 - val_loss: 592.4039
Epoch 43/200
22/22             0s 5ms/step - loss:
533.7306 - val_loss: 568.0546
```

```
Epoch 44/200
22/22             0s 5ms/step - loss:
507.1075 - val_loss: 560.8353
Epoch 45/200
22/22             0s 5ms/step - loss:
490.2341 - val_loss: 536.0286
Epoch 46/200
22/22             0s 7ms/step - loss:
457.6207 - val_loss: 530.1502
Epoch 47/200
22/22             0s 5ms/step - loss:
477.1049 - val_loss: 504.8858
Epoch 48/200
22/22             0s 5ms/step - loss:
452.2705 - val_loss: 493.8884
Epoch 49/200
22/22             0s 5ms/step - loss:
453.1819 - val_loss: 474.7402
Epoch 50/200
22/22             0s 5ms/step - loss:
427.2727 - val_loss: 460.3596
Epoch 51/200
22/22             0s 5ms/step - loss:
420.0232 - val_loss: 452.3236
Epoch 52/200
22/22             0s 5ms/step - loss:
379.6399 - val_loss: 430.0761
Epoch 53/200
22/22             0s 5ms/step - loss:
408.4284 - val_loss: 419.0634
Epoch 54/200
22/22             0s 6ms/step - loss:
359.3286 - val_loss: 411.2767
Epoch 55/200
22/22             0s 5ms/step - loss:
352.2013 - val_loss: 387.0269
Epoch 56/200
22/22             0s 5ms/step - loss:
352.7175 - val_loss: 379.0378
Epoch 57/200
22/22             0s 5ms/step - loss:
357.0586 - val_loss: 370.4013
Epoch 58/200
22/22             0s 5ms/step - loss:
337.6469 - val_loss: 355.0850
Epoch 59/200
22/22             0s 5ms/step - loss:
322.7745 - val_loss: 344.8427
```

```
Epoch 60/200
22/22             0s 5ms/step - loss:
295.6334 - val_loss: 331.3298
Epoch 61/200
22/22             0s 5ms/step - loss:
316.6006 - val_loss: 322.8290
Epoch 62/200
22/22             0s 5ms/step - loss:
310.4605 - val_loss: 310.9737
Epoch 63/200
22/22             0s 5ms/step - loss:
299.4838 - val_loss: 307.3727
Epoch 64/200
22/22             0s 5ms/step - loss:
304.0968 - val_loss: 295.9563
Epoch 65/200
22/22             0s 5ms/step - loss:
297.2408 - val_loss: 288.8823
Epoch 66/200
22/22             0s 5ms/step - loss:
257.3547 - val_loss: 277.3370
Epoch 67/200
22/22             0s 5ms/step - loss:
249.3514 - val_loss: 272.3352
Epoch 68/200
22/22             0s 5ms/step - loss:
248.2160 - val_loss: 258.3657
Epoch 69/200
22/22             0s 5ms/step - loss:
239.0616 - val_loss: 256.1361
Epoch 70/200
22/22             0s 5ms/step - loss:
232.8750 - val_loss: 247.2527
Epoch 71/200
22/22             0s 5ms/step - loss:
234.4992 - val_loss: 242.3830
Epoch 72/200
22/22             0s 5ms/step - loss:
235.6121 - val_loss: 239.7174
Epoch 73/200
22/22             0s 5ms/step - loss:
212.7195 - val_loss: 225.2562
Epoch 74/200
22/22             0s 5ms/step - loss:
202.9746 - val_loss: 223.8350
Epoch 75/200
22/22             0s 5ms/step - loss:
196.4196 - val_loss: 211.7286
```

```
Epoch 76/200
22/22             0s 5ms/step - loss:
198.4639 - val_loss: 215.6472
Epoch 77/200
22/22             0s 5ms/step - loss:
185.1462 - val_loss: 199.9712
Epoch 78/200
22/22             0s 5ms/step - loss:
206.5978 - val_loss: 202.0331
Epoch 79/200
22/22             0s 5ms/step - loss:
200.0316 - val_loss: 198.7692
Epoch 80/200
22/22             0s 5ms/step - loss:
183.3863 - val_loss: 184.9953
Epoch 81/200
22/22             0s 10ms/step -
loss: 186.9256 - val_loss: 186.8727
Epoch 82/200
22/22             0s 9ms/step - loss:
194.6915 - val_loss: 182.6078
Epoch 83/200
22/22             0s 9ms/step - loss:
172.6578 - val_loss: 171.5314
Epoch 84/200
22/22             0s 8ms/step - loss:
175.4527 - val_loss: 175.4668
Epoch 85/200
22/22             0s 8ms/step - loss:
167.7984 - val_loss: 165.3709
Epoch 86/200
22/22             0s 9ms/step - loss:
155.8541 - val_loss: 161.6051
Epoch 87/200
22/22             0s 14ms/step -
loss: 162.4653 - val_loss: 157.4869
Epoch 88/200
22/22             0s 10ms/step -
loss: 159.6274 - val_loss: 156.0394
Epoch 89/200
22/22             0s 10ms/step -
loss: 143.6341 - val_loss: 150.3567
Epoch 90/200
22/22             0s 7ms/step - loss:
139.7358 - val_loss: 146.6941
Epoch 91/200
22/22             0s 5ms/step - loss:
141.1062 - val_loss: 146.0898
```

```
Epoch 92/200
22/22             0s 5ms/step - loss:
143.6540 - val_loss: 142.7488
Epoch 93/200
22/22             0s 5ms/step - loss:
135.1593 - val_loss: 139.8111
Epoch 94/200
22/22             0s 5ms/step - loss:
148.1744 - val_loss: 145.1207
Epoch 95/200
22/22             0s 5ms/step - loss:
135.3399 - val_loss: 130.6290
Epoch 96/200
22/22             0s 6ms/step - loss:
122.2080 - val_loss: 129.5950
Epoch 97/200
22/22             0s 6ms/step - loss:
117.3767 - val_loss: 125.4586
Epoch 98/200
22/22             0s 6ms/step - loss:
127.4089 - val_loss: 125.0227
Epoch 99/200
22/22             0s 5ms/step - loss:
117.3825 - val_loss: 123.9455
Epoch 100/200
22/22             0s 5ms/step - loss:
122.7365 - val_loss: 122.4674
Epoch 101/200
22/22             0s 5ms/step - loss:
112.3319 - val_loss: 112.8702
Epoch 102/200
22/22             0s 5ms/step - loss:
108.3141 - val_loss: 111.6474
Epoch 103/200
22/22             0s 5ms/step - loss:
108.3673 - val_loss: 112.7715
Epoch 104/200
22/22             0s 5ms/step - loss:
103.8871 - val_loss: 109.5228
Epoch 105/200
22/22             0s 5ms/step - loss:
106.3693 - val_loss: 103.4828
Epoch 106/200
22/22             0s 5ms/step - loss:
98.1731 - val_loss: 102.4378
Epoch 107/200
22/22             0s 5ms/step - loss:
103.1028 - val_loss: 101.3493
```

```
Epoch 108/200
22/22             0s 5ms/step - loss:
98.6434 - val_loss: 96.9857
Epoch 109/200
22/22             0s 5ms/step - loss:
92.2268 - val_loss: 93.8403
Epoch 110/200
22/22             0s 5ms/step - loss:
88.3501 - val_loss: 92.5215
Epoch 111/200
22/22             0s 5ms/step - loss:
86.5535 - val_loss: 90.4011
Epoch 112/200
22/22             0s 7ms/step - loss:
82.4712 - val_loss: 86.8989
Epoch 113/200
22/22             0s 6ms/step - loss:
81.5280 - val_loss: 85.2368
Epoch 114/200
22/22             0s 5ms/step - loss:
79.1940 - val_loss: 85.8854
Epoch 115/200
22/22             0s 6ms/step - loss:
81.5167 - val_loss: 83.6399
Epoch 116/200
22/22             0s 6ms/step - loss:
77.8261 - val_loss: 78.9813
Epoch 117/200
22/22             0s 5ms/step - loss:
78.6534 - val_loss: 81.3468
Epoch 118/200
22/22             0s 5ms/step - loss:
74.7432 - val_loss: 80.8381
Epoch 119/200
22/22             0s 6ms/step - loss:
70.4959 - val_loss: 73.6934
Epoch 120/200
22/22             0s 5ms/step - loss:
73.3937 - val_loss: 73.9685
Epoch 121/200
22/22             0s 5ms/step - loss:
67.6159 - val_loss: 72.3146
Epoch 122/200
22/22             0s 5ms/step - loss:
70.9138 - val_loss: 69.5951
Epoch 123/200
22/22             0s 5ms/step - loss:
65.3951 - val_loss: 67.6176
```

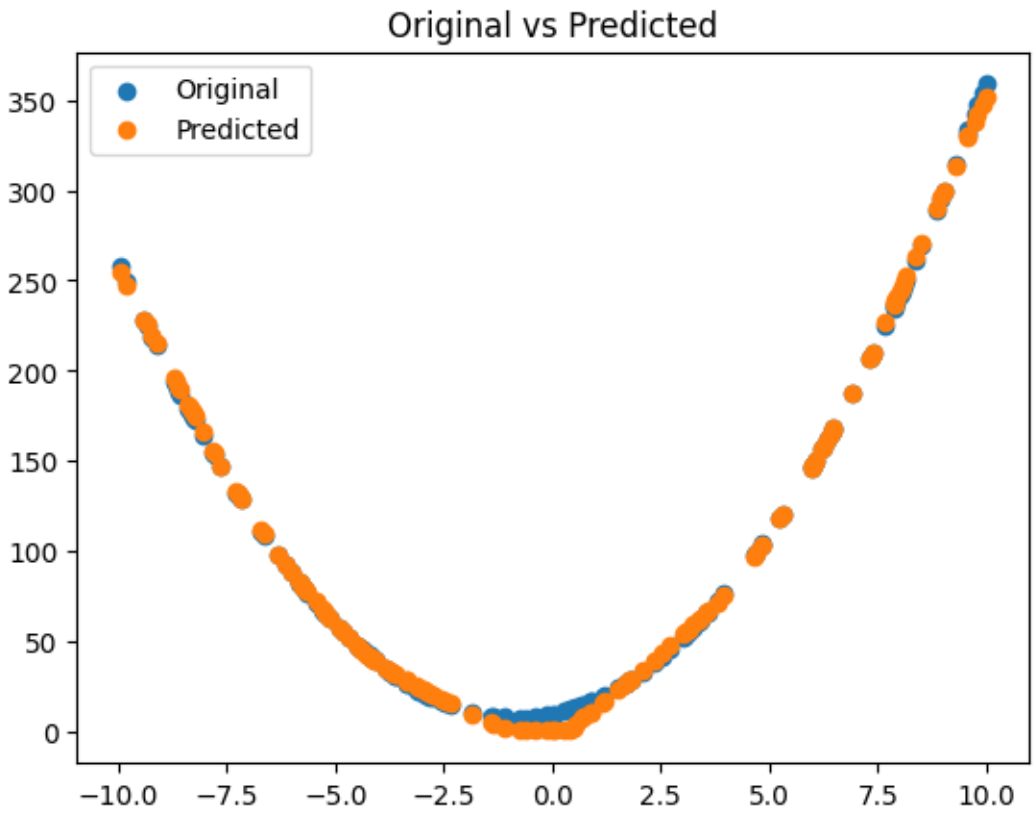
```
Epoch 124/200
22/22             0s 5ms/step - loss:
61.1258 - val_loss: 68.0253
Epoch 125/200
22/22             0s 5ms/step - loss:
55.4299 - val_loss: 64.5362
Epoch 126/200
22/22             0s 5ms/step - loss:
55.7441 - val_loss: 62.9869
Epoch 127/200
22/22             0s 5ms/step - loss:
62.2082 - val_loss: 63.6650
Epoch 128/200
22/22             0s 5ms/step - loss:
58.2846 - val_loss: 61.7650
Epoch 129/200
22/22             0s 5ms/step - loss:
55.8466 - val_loss: 60.3990
Epoch 130/200
22/22             0s 5ms/step - loss:
57.2929 - val_loss: 58.1239
Epoch 131/200
22/22             0s 5ms/step - loss:
54.3412 - val_loss: 55.6344
Epoch 132/200
22/22             0s 5ms/step - loss:
52.6310 - val_loss: 56.3298
Epoch 133/200
22/22             0s 5ms/step - loss:
47.6683 - val_loss: 52.6086
Epoch 134/200
22/22             0s 5ms/step - loss:
48.2085 - val_loss: 51.4015
Epoch 135/200
22/22             0s 5ms/step - loss:
46.0526 - val_loss: 50.8847
Epoch 136/200
22/22             0s 5ms/step - loss:
44.2026 - val_loss: 50.7716
Epoch 137/200
22/22             0s 5ms/step - loss:
44.2777 - val_loss: 48.4104
Epoch 138/200
22/22             0s 5ms/step - loss:
45.3857 - val_loss: 47.3362
Epoch 139/200
22/22             0s 5ms/step - loss:
46.6705 - val_loss: 46.4223
```

```
Epoch 140/200
22/22             0s 5ms/step - loss:
41.9944 - val_loss: 44.3038
Epoch 141/200
22/22             0s 5ms/step - loss:
39.9211 - val_loss: 43.5452
Epoch 142/200
22/22             0s 5ms/step - loss:
37.9868 - val_loss: 41.8015
Epoch 143/200
22/22             0s 5ms/step - loss:
37.5386 - val_loss: 40.5794
Epoch 144/200
22/22             0s 5ms/step - loss:
42.1525 - val_loss: 40.8990
Epoch 145/200
22/22             0s 5ms/step - loss:
38.8463 - val_loss: 39.9594
Epoch 146/200
22/22             0s 5ms/step - loss:
32.3529 - val_loss: 38.0405
Epoch 147/200
22/22             0s 5ms/step - loss:
36.3652 - val_loss: 36.8164
Epoch 148/200
22/22             0s 5ms/step - loss:
31.1637 - val_loss: 35.7089
Epoch 149/200
22/22             0s 6ms/step - loss:
31.1454 - val_loss: 35.1852
Epoch 150/200
22/22             0s 5ms/step - loss:
29.8366 - val_loss: 34.0925
Epoch 151/200
22/22             0s 5ms/step - loss:
31.3373 - val_loss: 33.7609
Epoch 152/200
22/22             0s 5ms/step - loss:
30.8564 - val_loss: 32.8327
Epoch 153/200
22/22             0s 5ms/step - loss:
29.2963 - val_loss: 31.9023
Epoch 154/200
22/22             0s 5ms/step - loss:
28.7728 - val_loss: 31.4217
Epoch 155/200
22/22             0s 5ms/step - loss:
25.2211 - val_loss: 30.6715
```

```
Epoch 156/200
22/22             0s 6ms/step - loss:
27.7632 - val_loss: 30.8317
Epoch 157/200
22/22             0s 5ms/step - loss:
26.5874 - val_loss: 29.8463
Epoch 158/200
22/22             0s 5ms/step - loss:
25.6468 - val_loss: 28.2573
Epoch 159/200
22/22             0s 8ms/step - loss:
24.0522 - val_loss: 27.6187
Epoch 160/200
22/22             0s 8ms/step - loss:
23.4391 - val_loss: 27.8729
Epoch 161/200
22/22             0s 8ms/step - loss:
23.1437 - val_loss: 26.5618
Epoch 162/200
22/22             0s 7ms/step - loss:
21.2952 - val_loss: 25.7101
Epoch 163/200
22/22             0s 9ms/step - loss:
20.2988 - val_loss: 25.2141
Epoch 164/200
22/22             0s 7ms/step - loss:
21.0147 - val_loss: 24.6672
Epoch 165/200
22/22             0s 9ms/step - loss:
22.5773 - val_loss: 23.9325
Epoch 166/200
22/22             0s 11ms/step -
loss: 18.5655 - val_loss: 24.0779
Epoch 167/200
22/22             0s 10ms/step -
loss: 20.2454 - val_loss: 22.8527
Epoch 168/200
22/22             0s 11ms/step -
loss: 19.8091 - val_loss: 22.4752
Epoch 169/200
22/22             0s 7ms/step - loss:
20.4587 - val_loss: 22.6801
Epoch 170/200
22/22             0s 6ms/step - loss:
20.4065 - val_loss: 21.2886
Epoch 171/200
22/22             0s 7ms/step - loss:
17.8838 - val_loss: 21.2098
```

```
Epoch 172/200
22/22             0s 6ms/step - loss:
16.3522 - val_loss: 20.6358
Epoch 173/200
22/22             0s 5ms/step - loss:
16.2890 - val_loss: 20.0322
Epoch 174/200
22/22             0s 5ms/step - loss:
17.5191 - val_loss: 20.1029
Epoch 175/200
22/22             0s 5ms/step - loss:
16.3968 - val_loss: 19.3168
Epoch 176/200
22/22             0s 5ms/step - loss:
13.9246 - val_loss: 18.5118
Epoch 177/200
22/22             0s 5ms/step - loss:
15.4538 - val_loss: 18.0487
Epoch 178/200
22/22             0s 6ms/step - loss:
12.7878 - val_loss: 17.6398
Epoch 179/200
22/22             0s 6ms/step - loss:
13.3210 - val_loss: 17.3037
Epoch 180/200
22/22             0s 5ms/step - loss:
13.7677 - val_loss: 16.9316
Epoch 181/200
22/22             0s 5ms/step - loss:
15.2636 - val_loss: 16.8852
Epoch 182/200
22/22             0s 5ms/step - loss:
13.7712 - val_loss: 16.1014
Epoch 183/200
22/22             0s 5ms/step - loss:
14.0186 - val_loss: 15.8899
Epoch 184/200
22/22             0s 5ms/step - loss:
13.7333 - val_loss: 15.4497
Epoch 185/200
22/22             0s 5ms/step - loss:
14.4482 - val_loss: 15.0490
Epoch 186/200
22/22             0s 5ms/step - loss:
12.2772 - val_loss: 14.7708
Epoch 187/200
22/22             0s 6ms/step - loss:
10.7053 - val_loss: 15.6610
```

```
Epoch 188/200
22/22             0s 5ms/step - loss:
12.5825 - val_loss: 14.0925
Epoch 189/200
22/22             0s 5ms/step - loss:
11.3081 - val_loss: 14.0958
Epoch 190/200
22/22             0s 5ms/step - loss:
12.2458 - val_loss: 13.5072
Epoch 191/200
22/22             0s 6ms/step - loss:
10.9422 - val_loss: 13.2573
Epoch 192/200
22/22             0s 5ms/step - loss:
10.1901 - val_loss: 12.9294
Epoch 193/200
22/22             0s 5ms/step - loss:
12.0193 - val_loss: 13.6850
Epoch 194/200
22/22             0s 5ms/step - loss:
10.3965 - val_loss: 12.9060
Epoch 195/200
22/22             0s 5ms/step - loss:
11.7746 - val_loss: 12.0578
Epoch 196/200
22/22             0s 5ms/step - loss:
12.3568 - val_loss: 11.9779
Epoch 197/200
22/22             0s 5ms/step - loss:
10.3793 - val_loss: 11.5675
Epoch 198/200
22/22             0s 6ms/step - loss:
8.8994 - val_loss: 11.3247
Epoch 199/200
22/22             0s 5ms/step - loss:
9.8372 - val_loss: 10.9954
Epoch 200/200
22/22             0s 5ms/step - loss:
9.8726 - val_loss: 10.7584
5/5             0s 14ms/step
```




```
5/5             0s 7ms/step - loss:
9.8583
Test Loss: 10.612138748168945
```

[ ]:
```python
#ASSIGNMENT 4

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.datasets import fashion_mnist, mnist, cifar10
import matplotlib.pyplot as plt
import os

# Make sure a folder exists to save plots
os.makedirs("plots", exist_ok=True)

def build_model(input_shape):
    model = Sequential([
        Flatten(input_shape=input_shape),
        Dense(256, activation='relu'),
        Dropout(0.3),
```

```
        Dense(128, activation='relu'),
        Dropout(0.3),
        Dense(64, activation='relu'),
        Dense(10, activation='softmax')
    ])
    model.compile(
        optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )
    return model

def train_dataset(dataset_name, dataset):
    (X_train, y_train), (X_test, y_test) = dataset

    # Normalize data
    X_train = X_train / 255.0
    X_test = X_test / 255.0

    # Build model
    model = build_model(X_train.shape[1:])

    # Train model and capture history
    history = model.fit(
        X_train, y_train,
        epochs=20,
        batch_size=64,
        validation_split=0.2,
        verbose=2
    )

    # Evaluate test accuracy
    loss, acc = model.evaluate(X_test, y_test, verbose=0)
    print(f"{dataset_name} Test Accuracy: {acc*100:.2f}%")

    # Plot training & validation accuracy
    plt.figure(figsize=(8,6))
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title(f'{dataset_name} Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)

    # Save plot
    plot_path = f"plots/{dataset_name.replace(' ', '_')}_accuracy.png"
```

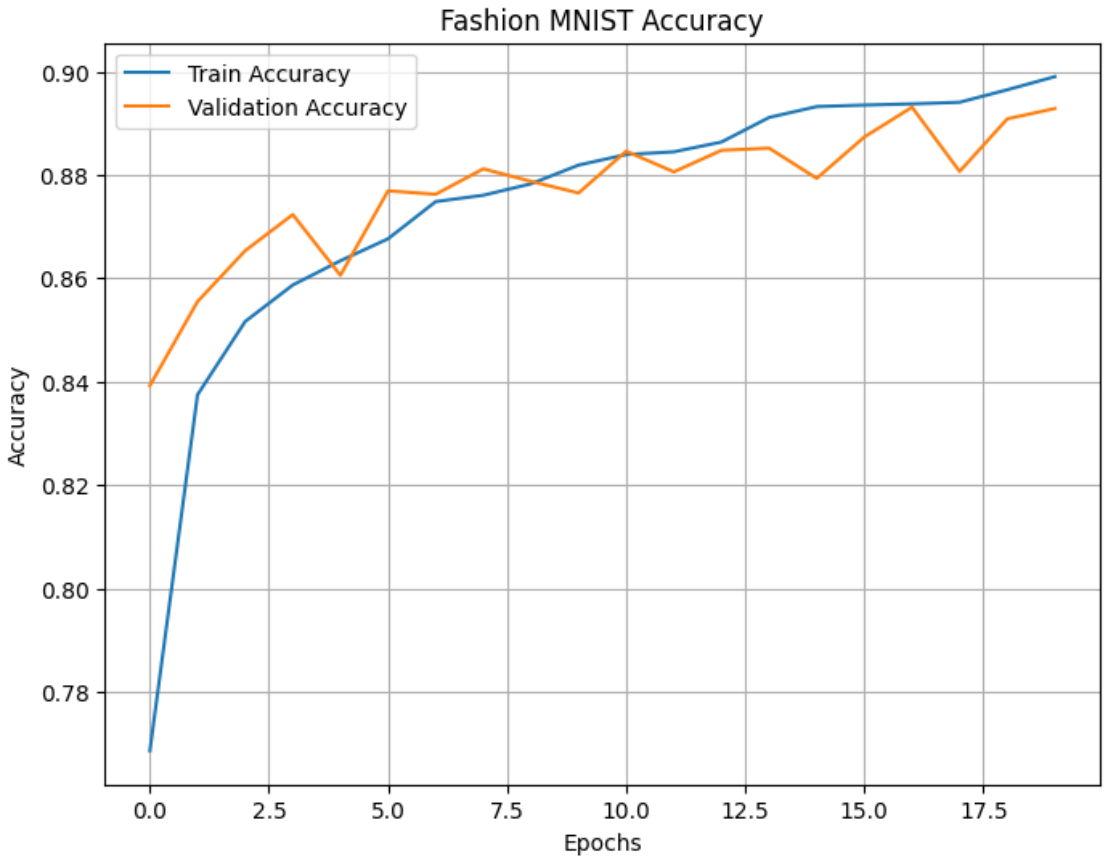
```
    plt.savefig(plot_path)
    plt.show()
    print(f"Plot saved to {plot_path}\n")

# Train and plot for each dataset
train_dataset("Fashion MNIST", fashion_mnist.load_data())
train_dataset("MNIST", mnist.load_data())
train_dataset("CIFAR-10", cifar10.load_data())
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-labels-idx1-ubyte.gz
29515/29515             0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-images-idx3-ubyte.gz
26421880/26421880             0s
0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-labels-idx1-ubyte.gz
5148/5148             0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-images-idx3-ubyte.gz
4422102/4422102             0s
0us/step
```

```
/usr/local/lib/python3.12/dist-
packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

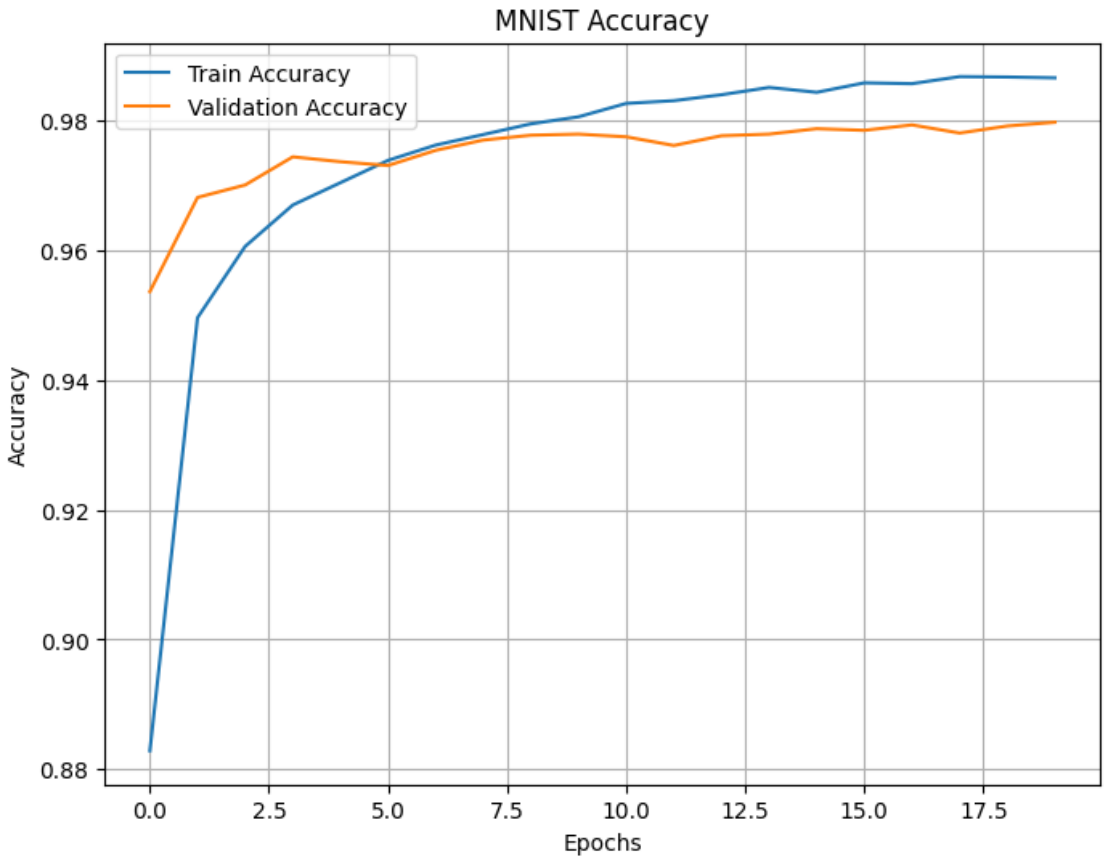
```
Epoch 1/20
750/750 - 8s - 11ms/step - accuracy: 0.7686 - loss: 0.6361 - val_accuracy:
0.8393 - val_loss: 0.4492
Epoch 2/20
750/750 - 5s - 7ms/step - accuracy: 0.8375 - loss: 0.4548 - val_accuracy: 0.8555
- val_loss: 0.3888
Epoch 3/20
750/750 - 6s - 9ms/step - accuracy: 0.8516 - loss: 0.4101 - val_accuracy: 0.8653
- val_loss: 0.3680
Epoch 4/20
750/750 - 5s - 7ms/step - accuracy: 0.8587 - loss: 0.3875 - val_accuracy: 0.8723
- val_loss: 0.3526
Epoch 5/20
750/750 - 6s - 9ms/step - accuracy: 0.8634 - loss: 0.3729 - val_accuracy: 0.8606
- val_loss: 0.3794
Epoch 6/20
750/750 - 5s - 7ms/step - accuracy: 0.8676 - loss: 0.3550 - val_accuracy: 0.8769
- val_loss: 0.3424
```

```
Epoch 7/20
750/750 - 6s - 8ms/step - accuracy: 0.8748 - loss: 0.3453 - val_accuracy: 0.8763
- val_loss: 0.3397
Epoch 8/20
750/750 - 6s - 8ms/step - accuracy: 0.8761 - loss: 0.3350 - val_accuracy: 0.8812
- val_loss: 0.3317
Epoch 9/20
750/750 - 6s - 8ms/step - accuracy: 0.8783 - loss: 0.3284 - val_accuracy: 0.8788
- val_loss: 0.3350
Epoch 10/20
750/750 - 7s - 9ms/step - accuracy: 0.8819 - loss: 0.3199 - val_accuracy: 0.8765
- val_loss: 0.3293
Epoch 11/20
750/750 - 6s - 7ms/step - accuracy: 0.8839 - loss: 0.3138 - val_accuracy: 0.8846
- val_loss: 0.3167
Epoch 12/20
750/750 - 6s - 8ms/step - accuracy: 0.8845 - loss: 0.3067 - val_accuracy: 0.8806
- val_loss: 0.3255
Epoch 13/20
750/750 - 5s - 7ms/step - accuracy: 0.8864 - loss: 0.3029 - val_accuracy: 0.8848
- val_loss: 0.3188
Epoch 14/20
750/750 - 6s - 9ms/step - accuracy: 0.8911 - loss: 0.2942 - val_accuracy: 0.8852
- val_loss: 0.3177
Epoch 15/20
750/750 - 5s - 7ms/step - accuracy: 0.8932 - loss: 0.2895 - val_accuracy: 0.8793
- val_loss: 0.3414
Epoch 16/20
750/750 - 7s - 9ms/step - accuracy: 0.8935 - loss: 0.2860 - val_accuracy: 0.8873
- val_loss: 0.3245
Epoch 17/20
750/750 - 6s - 7ms/step - accuracy: 0.8937 - loss: 0.2840 - val_accuracy: 0.8931
- val_loss: 0.2985
Epoch 18/20
750/750 - 7s - 9ms/step - accuracy: 0.8940 - loss: 0.2803 - val_accuracy: 0.8807
- val_loss: 0.3214
Epoch 19/20
750/750 - 6s - 8ms/step - accuracy: 0.8965 - loss: 0.2758 - val_accuracy: 0.8908
- val_loss: 0.3070
Epoch 20/20
750/750 - 7s - 9ms/step - accuracy: 0.8990 - loss: 0.2716 - val_accuracy: 0.8928
- val_loss: 0.3066
Fashion MNIST Test Accuracy: 88.54%
```



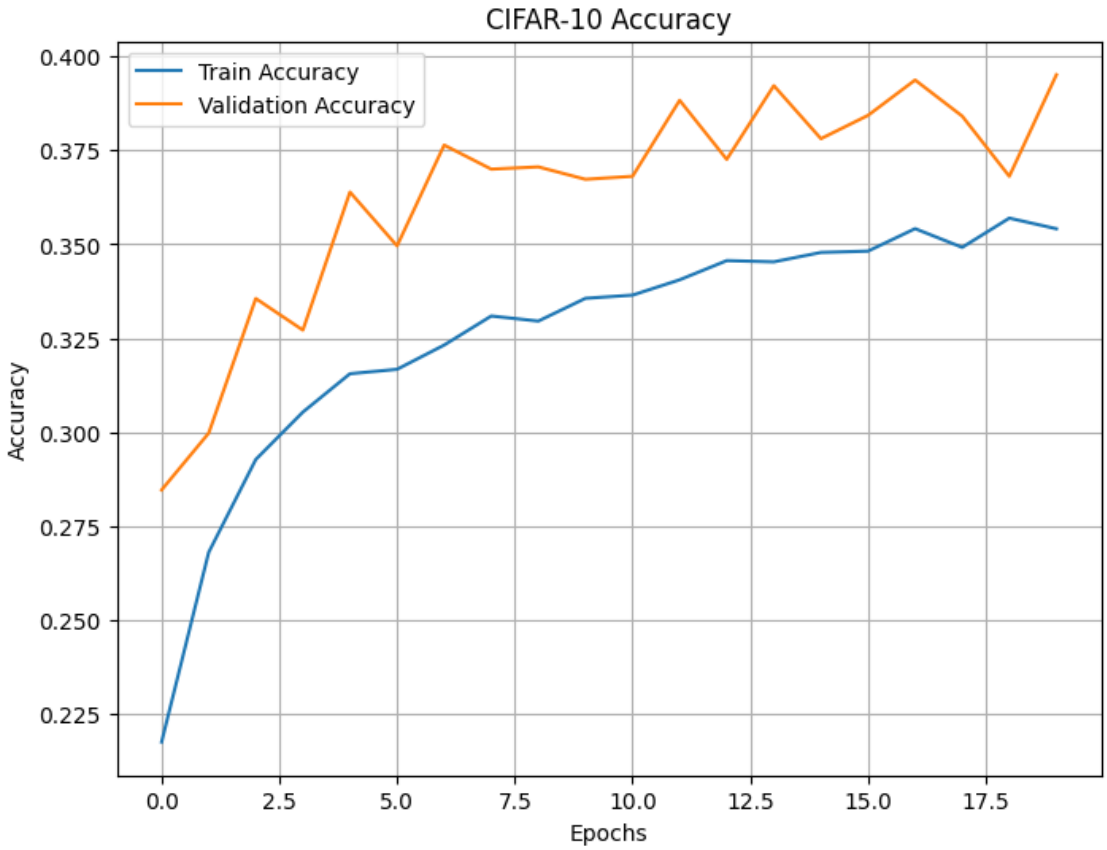

```
Plot saved to plots/Fashion_MNIST_accuracy.png

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11490434/11490434              0s
0us/step
Epoch 1/20
750/750 - 8s - 10ms/step - accuracy: 0.8829 - loss: 0.3817 - val_accuracy:
0.9536 - val_loss: 0.1494
Epoch 2/20
750/750 - 6s - 8ms/step - accuracy: 0.9496 - loss: 0.1708 - val_accuracy: 0.9681
- val_loss: 0.1080
Epoch 3/20
750/750 - 8s - 10ms/step - accuracy: 0.9605 - loss: 0.1304 - val_accuracy:
0.9700 - val_loss: 0.0989
Epoch 4/20
750/750 - 9s - 12ms/step - accuracy: 0.9669 - loss: 0.1094 - val_accuracy:
0.9743 - val_loss: 0.0848
Epoch 5/20
750/750 - 6s - 9ms/step - accuracy: 0.9704 - loss: 0.0967 - val_accuracy: 0.9736
```

```
- val_loss: 0.0861
Epoch 6/20
750/750 - 5s - 7ms/step - accuracy: 0.9738 - loss: 0.0836 - val_accuracy: 0.9730
- val_loss: 0.0901
Epoch 7/20
750/750 - 9s - 12ms/step - accuracy: 0.9761 - loss: 0.0762 - val_accuracy:
0.9753 - val_loss: 0.0846
Epoch 8/20
750/750 - 10s - 13ms/step - accuracy: 0.9778 - loss: 0.0716 - val_accuracy:
0.9769 - val_loss: 0.0781
Epoch 9/20
750/750 - 5s - 7ms/step - accuracy: 0.9794 - loss: 0.0677 - val_accuracy: 0.9777
- val_loss: 0.0774
Epoch 10/20
750/750 - 11s - 14ms/step - accuracy: 0.9805 - loss: 0.0609 - val_accuracy:
0.9778 - val_loss: 0.0796
Epoch 11/20
750/750 - 9s - 11ms/step - accuracy: 0.9826 - loss: 0.0561 - val_accuracy:
0.9774 - val_loss: 0.0851
Epoch 12/20
750/750 - 8s - 10ms/step - accuracy: 0.9830 - loss: 0.0560 - val_accuracy:
0.9761 - val_loss: 0.0898
Epoch 13/20
750/750 - 9s - 12ms/step - accuracy: 0.9839 - loss: 0.0506 - val_accuracy:
0.9776 - val_loss: 0.0833
Epoch 14/20
750/750 - 8s - 11ms/step - accuracy: 0.9850 - loss: 0.0484 - val_accuracy:
0.9778 - val_loss: 0.0864
Epoch 15/20
750/750 - 8s - 11ms/step - accuracy: 0.9843 - loss: 0.0497 - val_accuracy:
0.9787 - val_loss: 0.0829
Epoch 16/20
750/750 - 7s - 10ms/step - accuracy: 0.9857 - loss: 0.0451 - val_accuracy:
0.9784 - val_loss: 0.0903
Epoch 17/20
750/750 - 6s - 7ms/step - accuracy: 0.9856 - loss: 0.0430 - val_accuracy: 0.9793
- val_loss: 0.0824
Epoch 18/20
750/750 - 7s - 9ms/step - accuracy: 0.9867 - loss: 0.0428 - val_accuracy: 0.9780
- val_loss: 0.0900
Epoch 19/20
750/750 - 7s - 9ms/step - accuracy: 0.9866 - loss: 0.0410 - val_accuracy: 0.9791
- val_loss: 0.0822
Epoch 20/20
750/750 - 7s - 9ms/step - accuracy: 0.9865 - loss: 0.0396 - val_accuracy: 0.9797
- val_loss: 0.0796
MNIST Test Accuracy: 98.07%
```




```
Plot saved to plots/MNIST_accuracy.png

Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071              2s
0us/step
Epoch 1/20
625/625 - 14s - 23ms/step - accuracy: 0.2177 - loss: 2.0817 - val_accuracy:
0.2847 - val_loss: 1.9288
Epoch 2/20
625/625 - 11s - 18ms/step - accuracy: 0.2681 - loss: 1.9551 - val_accuracy:
0.2998 - val_loss: 1.8803
Epoch 3/20
625/625 - 19s - 31ms/step - accuracy: 0.2928 - loss: 1.9004 - val_accuracy:
0.3356 - val_loss: 1.8408
Epoch 4/20
625/625 - 11s - 18ms/step - accuracy: 0.3054 - loss: 1.8773 - val_accuracy:
0.3272 - val_loss: 1.8405
Epoch 5/20
625/625 - 11s - 17ms/step - accuracy: 0.3156 - loss: 1.8550 - val_accuracy:
0.3639 - val_loss: 1.8035
```

```
Epoch 6/20
625/625 - 12s - 19ms/step - accuracy: 0.3168 - loss: 1.8474 - val_accuracy:
0.3496 - val_loss: 1.8310
Epoch 7/20
625/625 - 13s - 21ms/step - accuracy: 0.3232 - loss: 1.8358 - val_accuracy:
0.3764 - val_loss: 1.7711
Epoch 8/20
625/625 - 11s - 17ms/step - accuracy: 0.3309 - loss: 1.8264 - val_accuracy:
0.3700 - val_loss: 1.8029
Epoch 9/20
625/625 - 10s - 16ms/step - accuracy: 0.3296 - loss: 1.8207 - val_accuracy:
0.3706 - val_loss: 1.7893
Epoch 10/20
625/625 - 11s - 17ms/step - accuracy: 0.3356 - loss: 1.8132 - val_accuracy:
0.3673 - val_loss: 1.7824
Epoch 11/20
625/625 - 11s - 18ms/step - accuracy: 0.3365 - loss: 1.8078 - val_accuracy:
0.3681 - val_loss: 1.7856
Epoch 12/20
625/625 - 11s - 18ms/step - accuracy: 0.3406 - loss: 1.8024 - val_accuracy:
0.3883 - val_loss: 1.7428
Epoch 13/20
625/625 - 20s - 32ms/step - accuracy: 0.3457 - loss: 1.7962 - val_accuracy:
0.3726 - val_loss: 1.7758
Epoch 14/20
625/625 - 12s - 19ms/step - accuracy: 0.3453 - loss: 1.7912 - val_accuracy:
0.3922 - val_loss: 1.7285
Epoch 15/20
625/625 - 19s - 30ms/step - accuracy: 0.3478 - loss: 1.7866 - val_accuracy:
0.3781 - val_loss: 1.7791
Epoch 16/20
625/625 - 11s - 17ms/step - accuracy: 0.3482 - loss: 1.7759 - val_accuracy:
0.3843 - val_loss: 1.7384
Epoch 17/20
625/625 - 11s - 17ms/step - accuracy: 0.3542 - loss: 1.7710 - val_accuracy:
0.3937 - val_loss: 1.7428
Epoch 18/20
625/625 - 11s - 17ms/step - accuracy: 0.3492 - loss: 1.7773 - val_accuracy:
0.3841 - val_loss: 1.7271
Epoch 19/20
625/625 - 11s - 17ms/step - accuracy: 0.3570 - loss: 1.7655 - val_accuracy:
0.3681 - val_loss: 1.7864
Epoch 20/20
625/625 - 10s - 15ms/step - accuracy: 0.3541 - loss: 1.7691 - val_accuracy:
0.3951 - val_loss: 1.7295
CIFAR-10 Test Accuracy: 39.98%
```




```
Plot saved to plots/CIFAR-10_accuracy.png
```

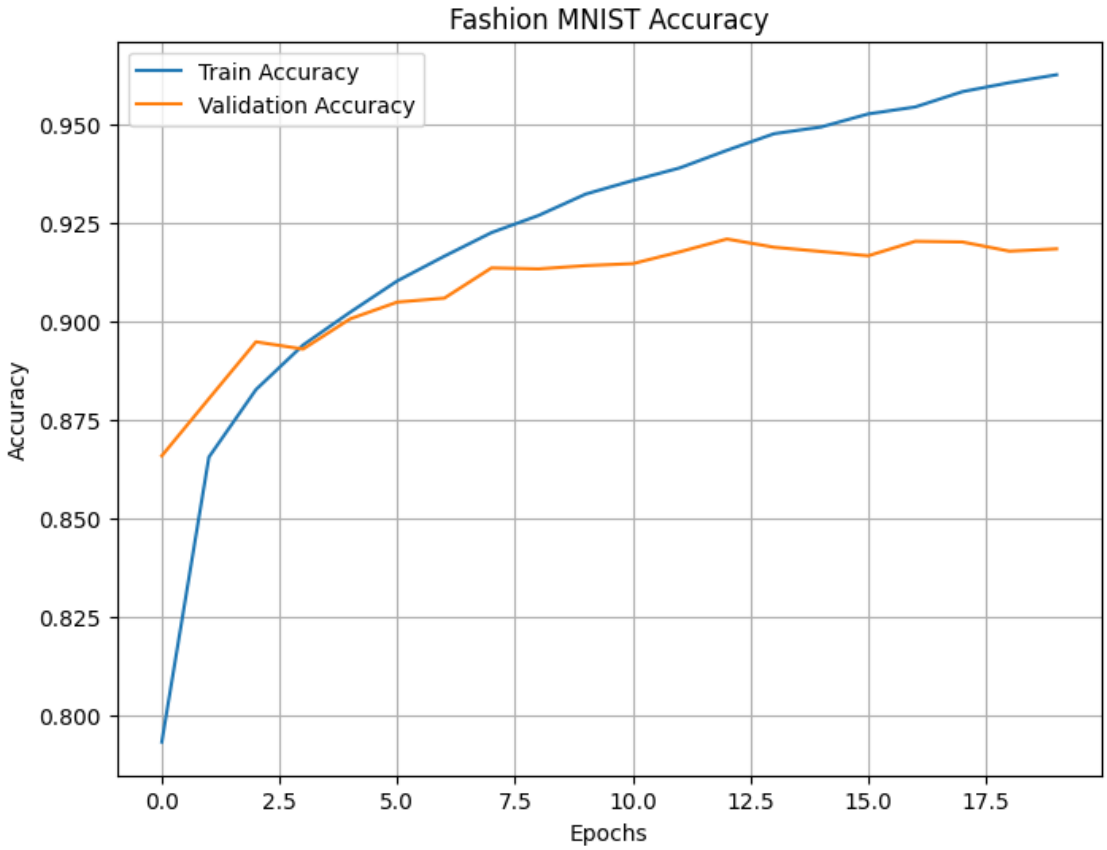
[ ]:
```
#ASSIGNMENT 5
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
↪Dropout
from tensorflow.keras.datasets import fashion_mnist, mnist, cifar10
import matplotlib.pyplot as plt
import os

# Create folder to save plots
os.makedirs("plots", exist_ok=True)

def build_cnn(input_shape):
    model = Sequential([
        Conv2D(32, (3,3), activation='relu', input_shape=input_shape),
        MaxPooling2D((2,2)),
        Conv2D(64, (3,3), activation='relu'),
```

```
        MaxPooling2D((2,2)),
        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.3),
        Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model

def train_and_plot(dataset_name, dataset, is_cifar=False):
    (X_train, y_train), (X_test, y_test) = dataset

    # Normalize data
    X_train = X_train / 255.0
    X_test = X_test / 255.0

    # Reshape for CNN
    if not is_cifar:
        X_train = X_train[..., tf.newaxis]
        X_test = X_test[..., tf.newaxis]

    model = build_cnn(X_train.shape[1:])

    history = model.fit(
        X_train, y_train,
        epochs=20,
        batch_size=64,
        validation_split=0.2,
        verbose=2
    )

    # Evaluate test accuracy
    loss, acc = model.evaluate(X_test, y_test, verbose=0)
    print(f"{dataset_name} Test Accuracy: {acc*100:.2f}%")

    # Plot accuracy
    plt.figure(figsize=(8,6))
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title(f'{dataset_name} Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)
    plot_path = f"plots/{dataset_name.replace(' ', '_')}_Problem_5.png"
```

```
    plt.savefig(plot_path)
    plt.show()
    print(f"Plot saved to {plot_path}\n")

# Train on datasets
train_and_plot("Fashion MNIST", fashion_mnist.load_data())
train_and_plot("MNIST", mnist.load_data())
train_and_plot("CIFAR-10", cifar10.load_data(), is_cifar=True)
```

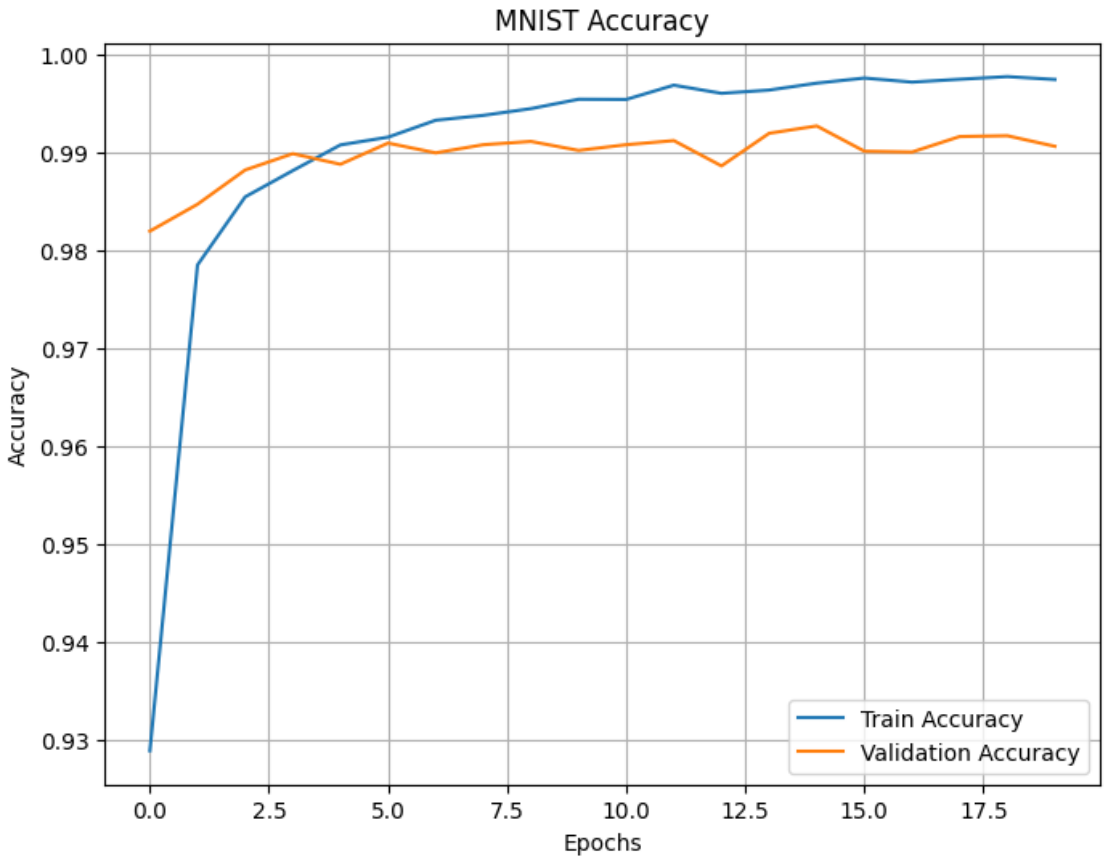
```
/usr/local/lib/python3.12/dist-
packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/20
750/750 - 42s - 56ms/step - accuracy: 0.7932 - loss: 0.5734 - val_accuracy:
0.8658 - val_loss: 0.3717
Epoch 2/20
750/750 - 41s - 55ms/step - accuracy: 0.8654 - loss: 0.3706 - val_accuracy:
0.8802 - val_loss: 0.3245
Epoch 3/20
750/750 - 40s - 53ms/step - accuracy: 0.8826 - loss: 0.3215 - val_accuracy:
0.8947 - val_loss: 0.2888
Epoch 4/20
750/750 - 41s - 55ms/step - accuracy: 0.8939 - loss: 0.2900 - val_accuracy:
0.8928 - val_loss: 0.2825
Epoch 5/20
750/750 - 40s - 54ms/step - accuracy: 0.9022 - loss: 0.2660 - val_accuracy:
0.9005 - val_loss: 0.2679
Epoch 6/20
750/750 - 40s - 53ms/step - accuracy: 0.9101 - loss: 0.2434 - val_accuracy:
0.9047 - val_loss: 0.2551
Epoch 7/20
750/750 - 39s - 53ms/step - accuracy: 0.9164 - loss: 0.2261 - val_accuracy:
0.9057 - val_loss: 0.2576
Epoch 8/20
750/750 - 42s - 56ms/step - accuracy: 0.9224 - loss: 0.2111 - val_accuracy:
0.9134 - val_loss: 0.2415
Epoch 9/20
750/750 - 79s - 105ms/step - accuracy: 0.9267 - loss: 0.1960 - val_accuracy:
0.9132 - val_loss: 0.2455
Epoch 10/20
750/750 - 42s - 56ms/step - accuracy: 0.9321 - loss: 0.1830 - val_accuracy:
0.9140 - val_loss: 0.2453
Epoch 11/20
750/750 - 41s - 54ms/step - accuracy: 0.9356 - loss: 0.1736 - val_accuracy:
```

```
0.9145 - val_loss: 0.2506
Epoch 12/20
750/750 - 39s - 52ms/step - accuracy: 0.9388 - loss: 0.1624 - val_accuracy:
0.9175 - val_loss: 0.2481
Epoch 13/20
750/750 - 40s - 53ms/step - accuracy: 0.9433 - loss: 0.1507 - val_accuracy:
0.9208 - val_loss: 0.2349
Epoch 14/20
750/750 - 41s - 55ms/step - accuracy: 0.9475 - loss: 0.1410 - val_accuracy:
0.9187 - val_loss: 0.2454
Epoch 15/20
750/750 - 42s - 56ms/step - accuracy: 0.9492 - loss: 0.1337 - val_accuracy:
0.9176 - val_loss: 0.2459
Epoch 16/20
750/750 - 39s - 52ms/step - accuracy: 0.9525 - loss: 0.1263 - val_accuracy:
0.9165 - val_loss: 0.2565
Epoch 17/20
750/750 - 39s - 53ms/step - accuracy: 0.9542 - loss: 0.1188 - val_accuracy:
0.9202 - val_loss: 0.2570
Epoch 18/20
750/750 - 39s - 53ms/step - accuracy: 0.9581 - loss: 0.1115 - val_accuracy:
0.9200 - val_loss: 0.2742
Epoch 19/20
750/750 - 41s - 55ms/step - accuracy: 0.9604 - loss: 0.1039 - val_accuracy:
0.9177 - val_loss: 0.2620
Epoch 20/20
750/750 - 40s - 54ms/step - accuracy: 0.9624 - loss: 0.0986 - val_accuracy:
0.9183 - val_loss: 0.2842
Fashion MNIST Test Accuracy: 91.18%
```



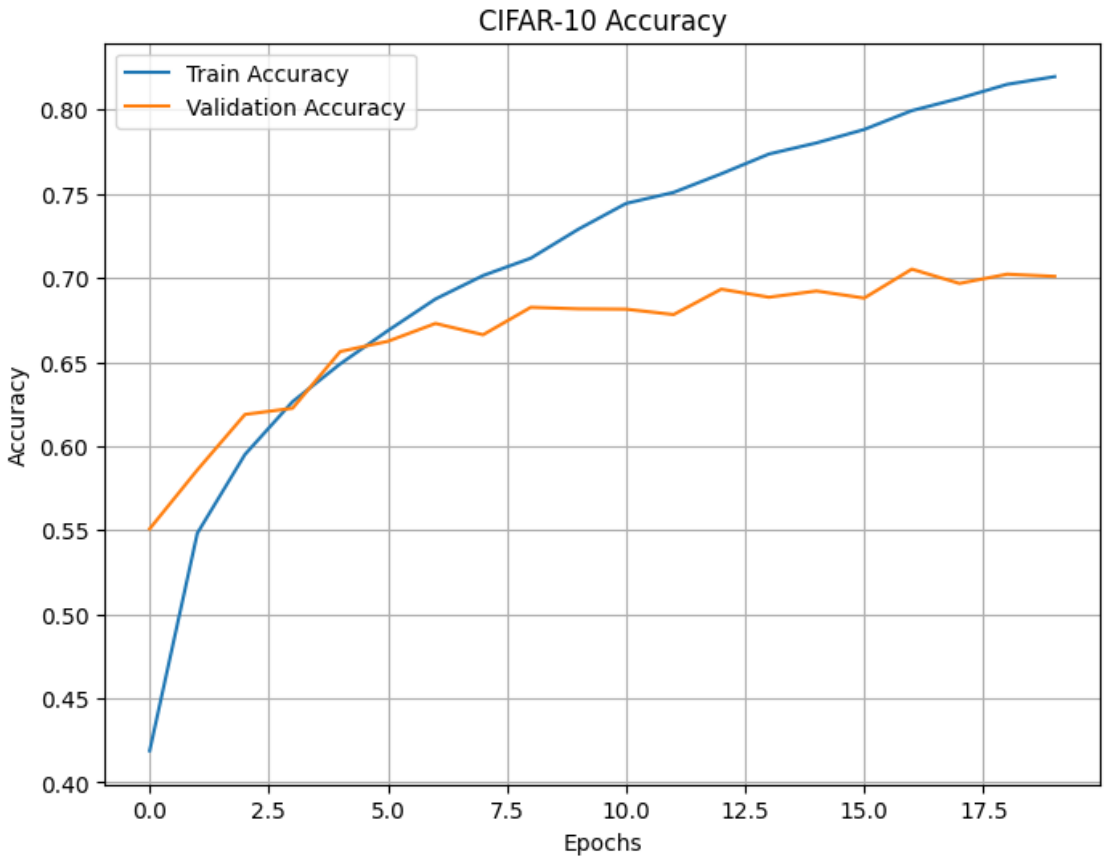

```
Plot saved to plots/Fashion_MNIST_Problem_5.png

Epoch 1/20
750/750 - 41s - 55ms/step - accuracy: 0.9289 - loss: 0.2275 - val_accuracy:
0.9820 - val_loss: 0.0610
Epoch 2/20
750/750 - 42s - 56ms/step - accuracy: 0.9785 - loss: 0.0695 - val_accuracy:
0.9847 - val_loss: 0.0538
Epoch 3/20
750/750 - 39s - 53ms/step - accuracy: 0.9855 - loss: 0.0486 - val_accuracy:
0.9883 - val_loss: 0.0393
Epoch 4/20
750/750 - 39s - 52ms/step - accuracy: 0.9882 - loss: 0.0377 - val_accuracy:
0.9899 - val_loss: 0.0360
Epoch 5/20
750/750 - 39s - 52ms/step - accuracy: 0.9908 - loss: 0.0304 - val_accuracy:
0.9888 - val_loss: 0.0384
Epoch 6/20
750/750 - 40s - 54ms/step - accuracy: 0.9916 - loss: 0.0263 - val_accuracy:
0.9910 - val_loss: 0.0332
```

```
Epoch 7/20
750/750 - 41s - 55ms/step - accuracy: 0.9933 - loss: 0.0209 - val_accuracy:
0.9900 - val_loss: 0.0369
Epoch 8/20
750/750 - 39s - 52ms/step - accuracy: 0.9938 - loss: 0.0188 - val_accuracy:
0.9908 - val_loss: 0.0344
Epoch 9/20
750/750 - 39s - 52ms/step - accuracy: 0.9945 - loss: 0.0163 - val_accuracy:
0.9912 - val_loss: 0.0374
Epoch 10/20
750/750 - 39s - 52ms/step - accuracy: 0.9955 - loss: 0.0134 - val_accuracy:
0.9902 - val_loss: 0.0436
Epoch 11/20
750/750 - 39s - 53ms/step - accuracy: 0.9955 - loss: 0.0136 - val_accuracy:
0.9908 - val_loss: 0.0351
Epoch 12/20
750/750 - 39s - 52ms/step - accuracy: 0.9969 - loss: 0.0097 - val_accuracy:
0.9912 - val_loss: 0.0418
Epoch 13/20
750/750 - 41s - 55ms/step - accuracy: 0.9961 - loss: 0.0106 - val_accuracy:
0.9887 - val_loss: 0.0520
Epoch 14/20
750/750 - 42s - 56ms/step - accuracy: 0.9964 - loss: 0.0098 - val_accuracy:
0.9920 - val_loss: 0.0401
Epoch 15/20
750/750 - 42s - 56ms/step - accuracy: 0.9971 - loss: 0.0086 - val_accuracy:
0.9927 - val_loss: 0.0405
Epoch 16/20
750/750 - 39s - 52ms/step - accuracy: 0.9976 - loss: 0.0069 - val_accuracy:
0.9902 - val_loss: 0.0538
Epoch 17/20
750/750 - 40s - 53ms/step - accuracy: 0.9972 - loss: 0.0080 - val_accuracy:
0.9901 - val_loss: 0.0478
Epoch 18/20
750/750 - 41s - 54ms/step - accuracy: 0.9975 - loss: 0.0069 - val_accuracy:
0.9917 - val_loss: 0.0485
Epoch 19/20
750/750 - 43s - 57ms/step - accuracy: 0.9978 - loss: 0.0064 - val_accuracy:
0.9918 - val_loss: 0.0500
Epoch 20/20
750/750 - 79s - 106ms/step - accuracy: 0.9975 - loss: 0.0068 - val_accuracy:
0.9907 - val_loss: 0.0587
MNIST Test Accuracy: 99.16%
```




```
Plot saved to plots/MNIST_Problem_5.png

Epoch 1/20
625/625 - 51s - 81ms/step - accuracy: 0.4188 - loss: 1.5967 - val_accuracy:
0.5508 - val_loss: 1.2858
Epoch 2/20
625/625 - 49s - 79ms/step - accuracy: 0.5482 - loss: 1.2737 - val_accuracy:
0.5858 - val_loss: 1.1615
Epoch 3/20
625/625 - 81s - 129ms/step - accuracy: 0.5950 - loss: 1.1475 - val_accuracy:
0.6188 - val_loss: 1.0887
Epoch 4/20
625/625 - 50s - 79ms/step - accuracy: 0.6264 - loss: 1.0623 - val_accuracy:
0.6225 - val_loss: 1.0785
Epoch 5/20
625/625 - 83s - 132ms/step - accuracy: 0.6489 - loss: 0.9986 - val_accuracy:
0.6562 - val_loss: 0.9814
Epoch 6/20
625/625 - 48s - 77ms/step - accuracy: 0.6686 - loss: 0.9397 - val_accuracy:
0.6623 - val_loss: 0.9663
```

```
Epoch 7/20
625/625 - 48s - 77ms/step - accuracy: 0.6874 - loss: 0.8882 - val_accuracy:
0.6729 - val_loss: 0.9427
Epoch 8/20
625/625 - 47s - 74ms/step - accuracy: 0.7014 - loss: 0.8493 - val_accuracy:
0.6662 - val_loss: 0.9612
Epoch 9/20
625/625 - 84s - 134ms/step - accuracy: 0.7117 - loss: 0.8063 - val_accuracy:
0.6825 - val_loss: 0.9161
Epoch 10/20
625/625 - 81s - 129ms/step - accuracy: 0.7290 - loss: 0.7653 - val_accuracy:
0.6816 - val_loss: 0.9246
Epoch 11/20
625/625 - 81s - 130ms/step - accuracy: 0.7442 - loss: 0.7271 - val_accuracy:
0.6814 - val_loss: 0.9311
Epoch 12/20
625/625 - 48s - 77ms/step - accuracy: 0.7509 - loss: 0.6970 - val_accuracy:
0.6782 - val_loss: 0.9598
Epoch 13/20
625/625 - 50s - 79ms/step - accuracy: 0.7619 - loss: 0.6663 - val_accuracy:
0.6933 - val_loss: 0.8845
Epoch 14/20
625/625 - 49s - 79ms/step - accuracy: 0.7736 - loss: 0.6341 - val_accuracy:
0.6885 - val_loss: 0.9289
Epoch 15/20
625/625 - 79s - 126ms/step - accuracy: 0.7803 - loss: 0.6101 - val_accuracy:
0.6922 - val_loss: 0.9415
Epoch 16/20
625/625 - 83s - 133ms/step - accuracy: 0.7883 - loss: 0.5848 - val_accuracy:
0.6880 - val_loss: 0.9485
Epoch 17/20
625/625 - 85s - 136ms/step - accuracy: 0.7994 - loss: 0.5547 - val_accuracy:
0.7052 - val_loss: 0.9250
Epoch 18/20
625/625 - 48s - 77ms/step - accuracy: 0.8067 - loss: 0.5349 - val_accuracy:
0.6967 - val_loss: 0.9548
Epoch 19/20
625/625 - 48s - 77ms/step - accuracy: 0.8151 - loss: 0.5085 - val_accuracy:
0.7022 - val_loss: 0.9692
Epoch 20/20
625/625 - 84s - 135ms/step - accuracy: 0.8197 - loss: 0.4917 - val_accuracy:
0.7009 - val_loss: 0.9745
CIFAR-10 Test Accuracy: 70.20%
```




```
Plot saved to plots/CIFAR-10_Problem_5.png
```

[ ]:
```python
#ASSIGNMENT 6

import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.datasets import mnist
from sklearn.model_selection import train_test_split

# ------------------------------------------------
# 1. Load MNIST Dataset
# ------------------------------------------------
(X_train_mnist, y_train_mnist), (X_test_mnist, y_test_mnist) = mnist.load_data()

# Normalize
X_train_mnist = X_train_mnist / 255.0
```

```
X_test_mnist = X_test_mnist / 255.0

# ------------------------------------------------
# 2. Create "Your Custom Handwritten Dataset"
# (Using part of MNIST to simulate collected data)
# ------------------------------------------------
X_custom = X_train_mnist[:5000]
y_custom = y_train_mnist[:5000]

# Split into 80% train and 20% test
X_custom_train, X_custom_test, y_custom_train, y_custom_test = train_test_split(
    X_custom, y_custom, test_size=0.2, random_state=42
)

print("Custom Training Samples:", X_custom_train.shape[0])
print("Custom Testing Samples:", X_custom_test.shape[0])

# ------------------------------------------------
# 3. Combine Custom Training with MNIST Training
# ------------------------------------------------
X_combined_train = np.concatenate((X_train_mnist, X_custom_train), axis=0)
y_combined_train = np.concatenate((y_train_mnist, y_custom_train), axis=0)

print("Total Combined Training Samples:", X_combined_train.shape[0])

# ------------------------------------------------
# 4. Build FCFNN Model
# ------------------------------------------------
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(256, activation='relu'),
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])

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

model.summary()

# ------------------------------------------------
```

```
# 5. Train Model
# ------------------------------------------------
history = model.fit(
    X_combined_train,
    y_combined_train,
    epochs=20,
    batch_size=64,
    validation_split=0.2
)

# ------------------------------------------------
# 6. Evaluate on Custom Test Set
# ------------------------------------------------
custom_loss, custom_acc = model.evaluate(X_custom_test, y_custom_test)
print("\nAccuracy on Custom Test Dataset: {:.2f}%".format(custom_acc * 100))

# ------------------------------------------------
# 7. Evaluate on MNIST Test Set
# ------------------------------------------------
mnist_loss, mnist_acc = model.evaluate(X_test_mnist, y_test_mnist)
print("Accuracy on MNIST Test Dataset: {:.2f}%".format(mnist_acc * 100))

# ------------------------------------------------
# 8. Plot and Save Accuracy Curve
# ------------------------------------------------
plt.figure()
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'])
plt.savefig("accuracy_plot.png", dpi=300)
plt.show()

# ------------------------------------------------
# 9. Plot and Save Loss Curve
# ------------------------------------------------
plt.figure()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'])
plt.savefig("loss_plot.png", dpi=300)
plt.show()
```

```python
# ------------------------------------------------
# 10. Plot and Save Custom vs MNIST Accuracy Comparison
# ------------------------------------------------
custom_accuracy = custom_acc * 100
mnist_accuracy = mnist_acc * 100

plt.figure()
plt.bar(['Custom Dataset', 'MNIST Dataset'],
        [custom_accuracy, mnist_accuracy])

plt.ylabel('Accuracy (%)')
plt.title('Custom vs MNIST Test Accuracy')

# Add value labels
plt.text(0, custom_accuracy + 0.5, f"{custom_accuracy:.2f}%")
plt.text(1, mnist_accuracy + 0.5, f"{mnist_accuracy:.2f}%")

plt.savefig("dataset_accuracy_comparison.png", dpi=300)
plt.show()

print("\nFigures Saved:")
print("1. accuracy_plot.png")
print("2. loss_plot.png")
print("3. dataset_accuracy_comparison.png")
```

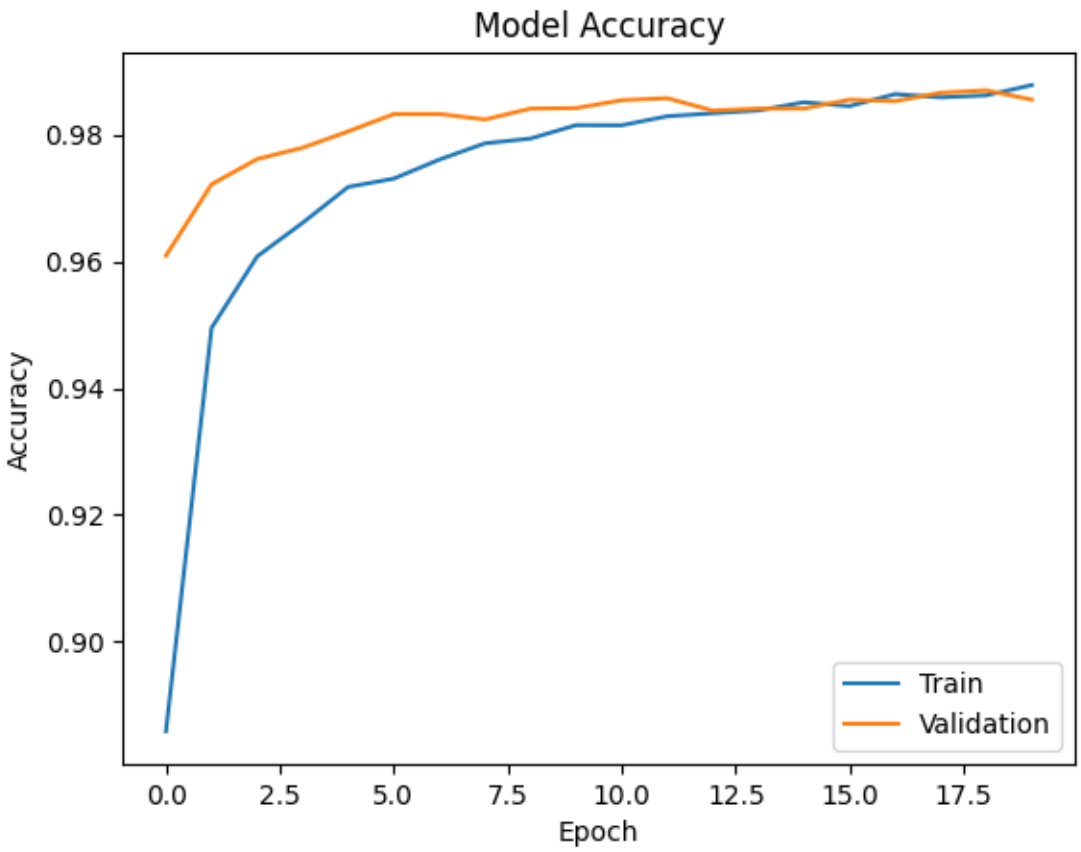
```
Custom Training Samples: 4000
Custom Testing Samples: 1000
Total Combined Training Samples: 64000
```

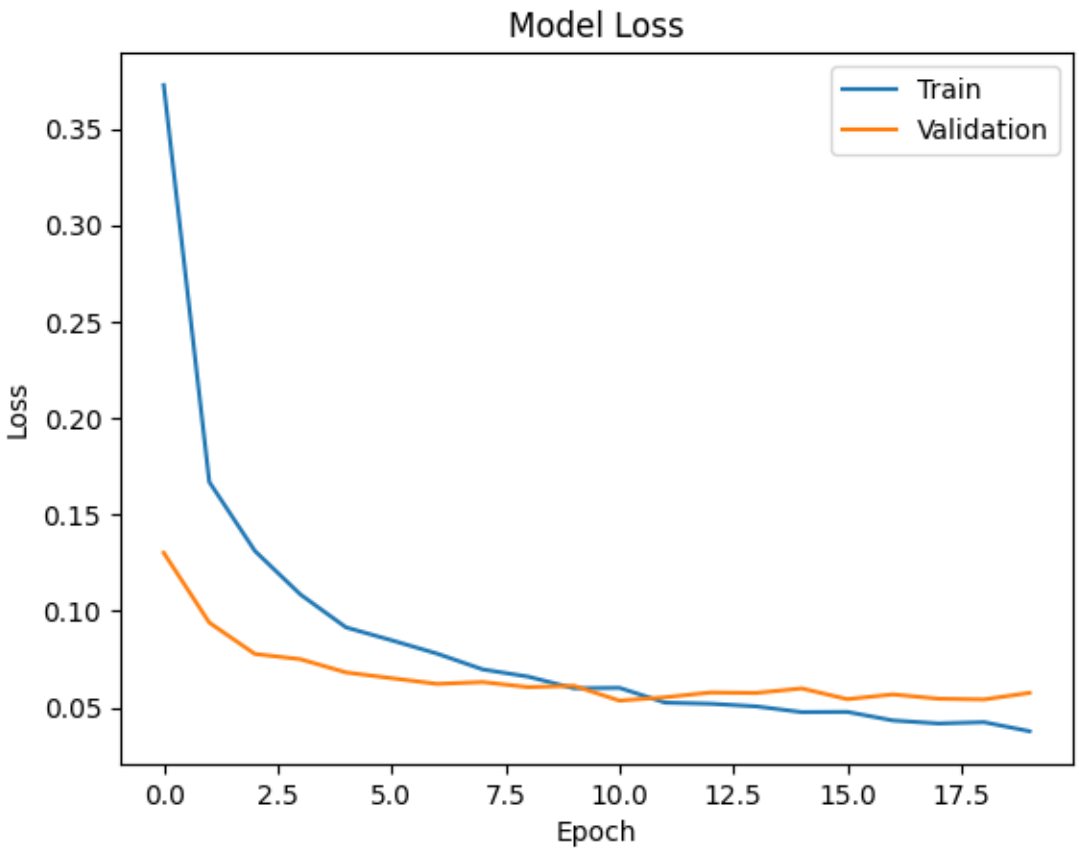
Model: "sequential_9"

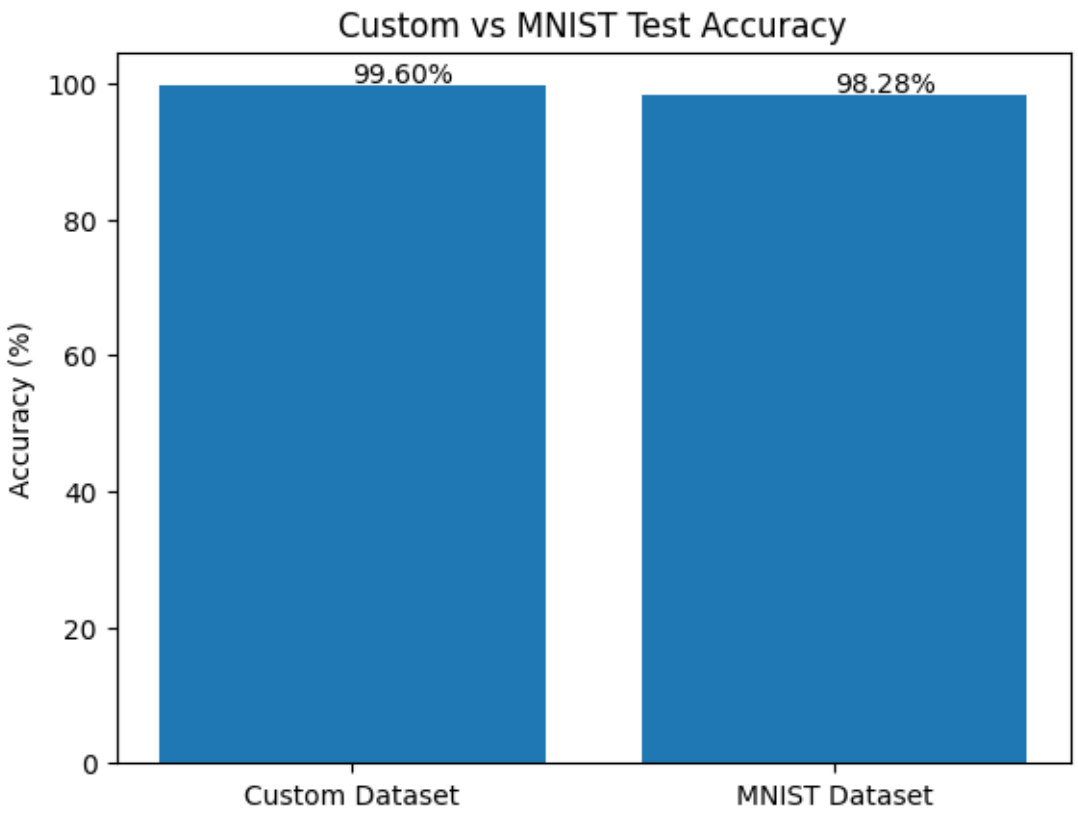
Layer (type)	Output Shape	Param #
flatten_6 (Flatten)	(None, 784)	0
dense_28 (Dense)	(None, 256)	200,960
dropout_9 (Dropout)	(None, 256)	0
dense_29 (Dense)	(None, 128)	32,896
dropout_10 (Dropout)	(None, 128)	0
dense_30 (Dense)	(None, 64)	8,256

```
 dense_31 (Dense)                (None, 10)                     650

Total params: 242,762 (948.29 KB)

Trainable params: 242,762 (948.29 KB)

Non-trainable params: 0 (0.00 B)

Epoch 1/20
800/800             9s 10ms/step -
accuracy: 0.7931 - loss: 0.6540 - val_accuracy: 0.9609 - val_loss: 0.1302
Epoch 2/20
800/800             7s 8ms/step -
accuracy: 0.9487 - loss: 0.1727 - val_accuracy: 0.9722 - val_loss: 0.0940
Epoch 3/20
800/800             8s 10ms/step -
accuracy: 0.9608 - loss: 0.1296 - val_accuracy: 0.9762 - val_loss: 0.0778
Epoch 4/20
800/800             6s 7ms/step -
accuracy: 0.9665 - loss: 0.1092 - val_accuracy: 0.9780 - val_loss: 0.0751
Epoch 5/20
800/800             8s 10ms/step -
accuracy: 0.9732 - loss: 0.0867 - val_accuracy: 0.9805 - val_loss: 0.0682
Epoch 6/20
800/800             7s 8ms/step -
accuracy: 0.9741 - loss: 0.0811 - val_accuracy: 0.9833 - val_loss: 0.0653
Epoch 7/20
800/800             10s 8ms/step -
accuracy: 0.9773 - loss: 0.0735 - val_accuracy: 0.9833 - val_loss: 0.0623
Epoch 8/20
800/800             8s 10ms/step -
accuracy: 0.9788 - loss: 0.0685 - val_accuracy: 0.9824 - val_loss: 0.0633
Epoch 9/20
800/800             7s 8ms/step -
accuracy: 0.9801 - loss: 0.0643 - val_accuracy: 0.9841 - val_loss: 0.0606
Epoch 10/20
800/800             10s 8ms/step -
accuracy: 0.9829 - loss: 0.0566 - val_accuracy: 0.9842 - val_loss: 0.0613
Epoch 11/20
800/800             8s 10ms/step -
accuracy: 0.9827 - loss: 0.0565 - val_accuracy: 0.9855 - val_loss: 0.0536
Epoch 12/20
800/800             7s 9ms/step -
accuracy: 0.9833 - loss: 0.0500 - val_accuracy: 0.9858 - val_loss: 0.0554
Epoch 13/20
```

```
800/800             8s 9ms/step -
accuracy: 0.9844 - loss: 0.0501 - val_accuracy: 0.9838 - val_loss: 0.0578
Epoch 14/20
800/800             8s 9ms/step -
accuracy: 0.9843 - loss: 0.0488 - val_accuracy: 0.9841 - val_loss: 0.0576
Epoch 15/20
800/800             7s 8ms/step -
accuracy: 0.9852 - loss: 0.0474 - val_accuracy: 0.9841 - val_loss: 0.0600
Epoch 16/20
800/800             8s 10ms/step -
accuracy: 0.9849 - loss: 0.0462 - val_accuracy: 0.9855 - val_loss: 0.0544
Epoch 17/20
800/800             10s 9ms/step -
accuracy: 0.9867 - loss: 0.0403 - val_accuracy: 0.9853 - val_loss: 0.0568
Epoch 18/20
800/800             7s 9ms/step -
accuracy: 0.9861 - loss: 0.0412 - val_accuracy: 0.9866 - val_loss: 0.0546
Epoch 19/20
800/800             8s 10ms/step -
accuracy: 0.9863 - loss: 0.0418 - val_accuracy: 0.9870 - val_loss: 0.0542
Epoch 20/20
800/800             8s 10ms/step -
accuracy: 0.9880 - loss: 0.0377 - val_accuracy: 0.9855 - val_loss: 0.0577
32/32             1s 3ms/step -
accuracy: 0.9975 - loss: 0.0053

Accuracy on Custom Test Dataset: 99.60%
313/313             1s 3ms/step -
accuracy: 0.9783 - loss: 0.0904
Accuracy on MNIST Test Dataset: 98.28%
```


Model Accuracy
Accuracy
0.98
0.96
0.94
0.92
0.90
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch
Train
Validation


Model Loss
Train
Validation
Loss
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch




```
Figures Saved:
1. accuracy_plot.png
2. loss_plot.png
3. dataset_accuracy_comparison.png
```

[ ]:
```
# ==============================
# ASSIGNMENT 7 (FINAL VERSION)
# ==============================

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import time
import os
import zipfile

from google.colab import files
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# ----------------------------
# Upload Dataset Zip
# ----------------------------
print("Upload datasets.zip file")
uploaded = files.upload()

# ----------------------------
# Unzip Dataset
# ----------------------------
with zipfile.ZipFile("datasets.zip", 'r') as zip_ref:
    zip_ref.extractall("/content/datasets")

print("Unzipped folders:", os.listdir("/content/datasets"))

# ----------------------------
# Parameters
# ----------------------------
IMG_SIZE = 128
BATCH_SIZE = 32
EPOCHS = 15
DATASET_DIR = "/content/datasets"

# ----------------------------
# Dataset Loading
# ----------------------------
datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

train_data = datagen.flow_from_directory(
    DATASET_DIR,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    subset='training',
    class_mode='binary'
)

test_data = datagen.flow_from_directory(
    DATASET_DIR,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=1,
    subset='validation',
    class_mode='binary',
    shuffle=False
)

# ----------------------------
```

```
# Build CNN Model
# ----------------------------
def build_model(filters1=32, filters2=64, filters3=128):
    model = Sequential([
        Conv2D(filters1, (3,3), activation='relu', input_shape=(IMG_SIZE,␣
 ↪IMG_SIZE, 3)),
        MaxPooling2D(2,2),

        Conv2D(filters2, (3,3), activation='relu'),
        MaxPooling2D(2,2),

        Conv2D(filters3, (3,3), activation='relu'),
        MaxPooling2D(2,2),

        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.3),
        Dense(1, activation='sigmoid')
    ])

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

model = build_model()
model.summary()

# ----------------------------
# Training Time Measurement
# ----------------------------
start_train = time.time()
history = model.fit(train_data, epochs=EPOCHS, validation_data=test_data)
end_train = time.time()
training_time = end_train - start_train

print("Total Training Time (seconds):", training_time)

# ----------------------------
# Testing Time per Sample
# ----------------------------
start_test = time.time()
loss, accuracy = model.evaluate(test_data)
end_test = time.time()

testing_time = end_test - start_test
time_per_sample = testing_time / test_data.samples
```

```
print("Test Accuracy:", accuracy)
print("Testing Time per Sample (seconds):", time_per_sample)

# ----------------------------
# Epoch vs Accuracy Plot
# ----------------------------
plt.figure()
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title("Epoch vs Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.savefig("epoch_vs_accuracy_binary.png", dpi=300)
plt.show()

# ----------------------------
# Data Size vs Performance
# ----------------------------
sizes = [0.25, 0.5, 0.75, 1.0]
accuracies = []

for s in sizes:
    subset_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

    train_subset = subset_datagen.flow_from_directory(
        DATASET_DIR,
        target_size=(IMG_SIZE, IMG_SIZE),
        batch_size=BATCH_SIZE,
        subset='training',
        class_mode='binary'
    )

    model_temp = build_model()
    model_temp.fit(train_subset, epochs=5, verbose=0)
    _, acc_temp = model_temp.evaluate(test_data, verbose=0)
    accuracies.append(acc_temp)

plt.figure()
plt.plot(sizes, accuracies)
plt.xlabel("Fraction of Training Data")
plt.ylabel("Accuracy")
plt.title("Data Size vs Accuracy")
plt.grid(True)
plt.savefig("data_vs_accuracy_binary.png", dpi=300)
```

```
plt.show()

# ----------------------------
# Model Size vs Performance
# ----------------------------
filter_configs = [(16,32,64), (32,64,128), (64,128,256)]
param_counts = []
model_accs = []

for f in filter_configs:
    m = build_model(*f)
    param_counts.append(m.count_params())
    m.fit(train_data, epochs=5, verbose=0)
    _, acc = m.evaluate(test_data, verbose=0)
    model_accs.append(acc)

plt.figure()
plt.plot(param_counts, model_accs)
plt.xlabel("Number of Parameters")
plt.ylabel("Accuracy")
plt.title("Model Size vs Accuracy")
plt.grid(True)
plt.savefig("modelsize_vs_accuracy_binary.png", dpi=300)
plt.show()

print("Assignment 7 Completed Successfully!")
```

```
Upload datasets.zip file
```

```
<IPython.core.display.HTML object>
```

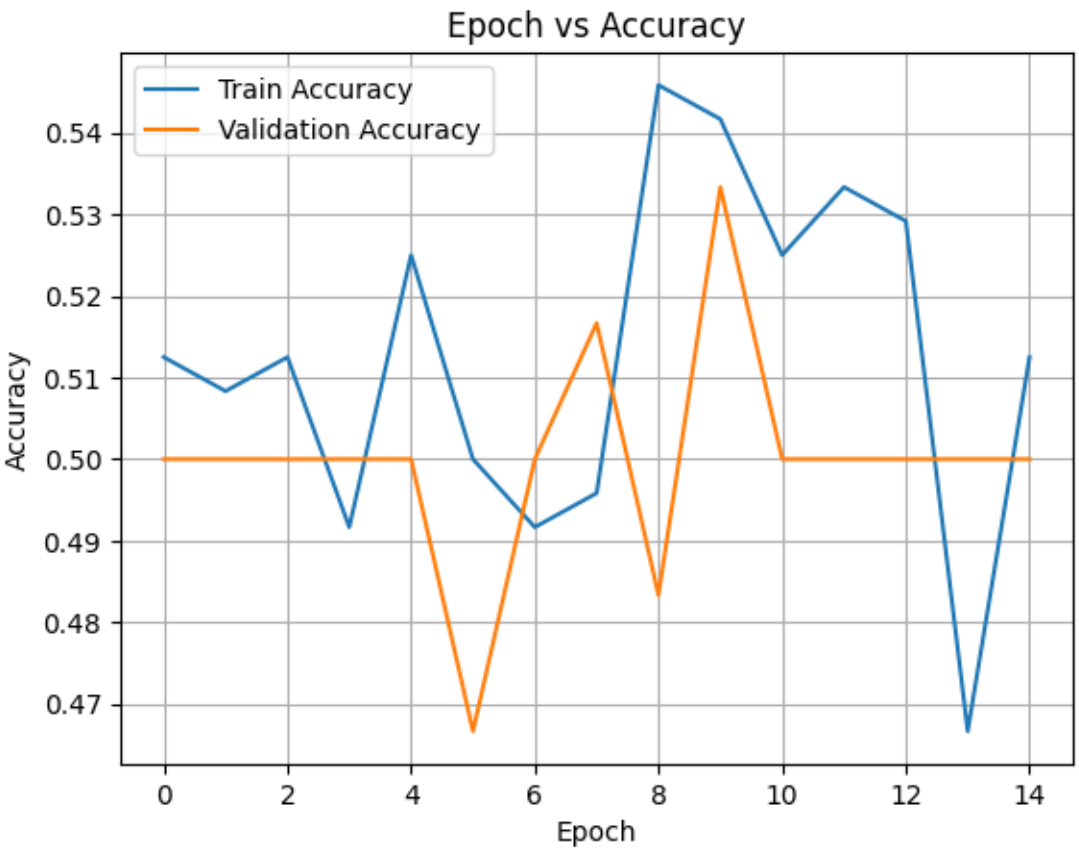
```
Saving datasets.zip to datasets (1).zip
Unzipped folders: ['dogs', 'cats']
Found 240 images belonging to 2 classes.
Found 60 images belonging to 2 classes.
```

```
Model: "sequential_10"
```

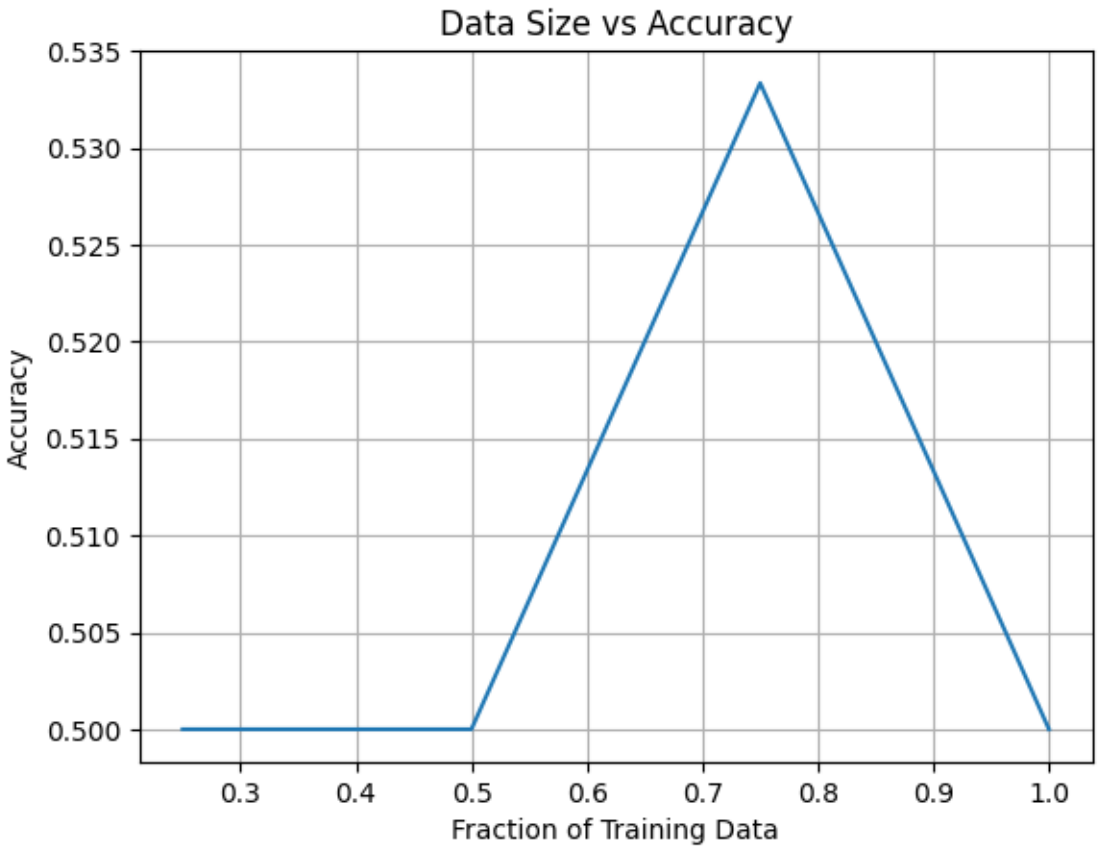
Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d_6 (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_7 (Conv2D)	(None, 61, 61, 64)	18,496
max_pooling2d_7 (MaxPooling2D)	(None, 30, 30, 64)	0

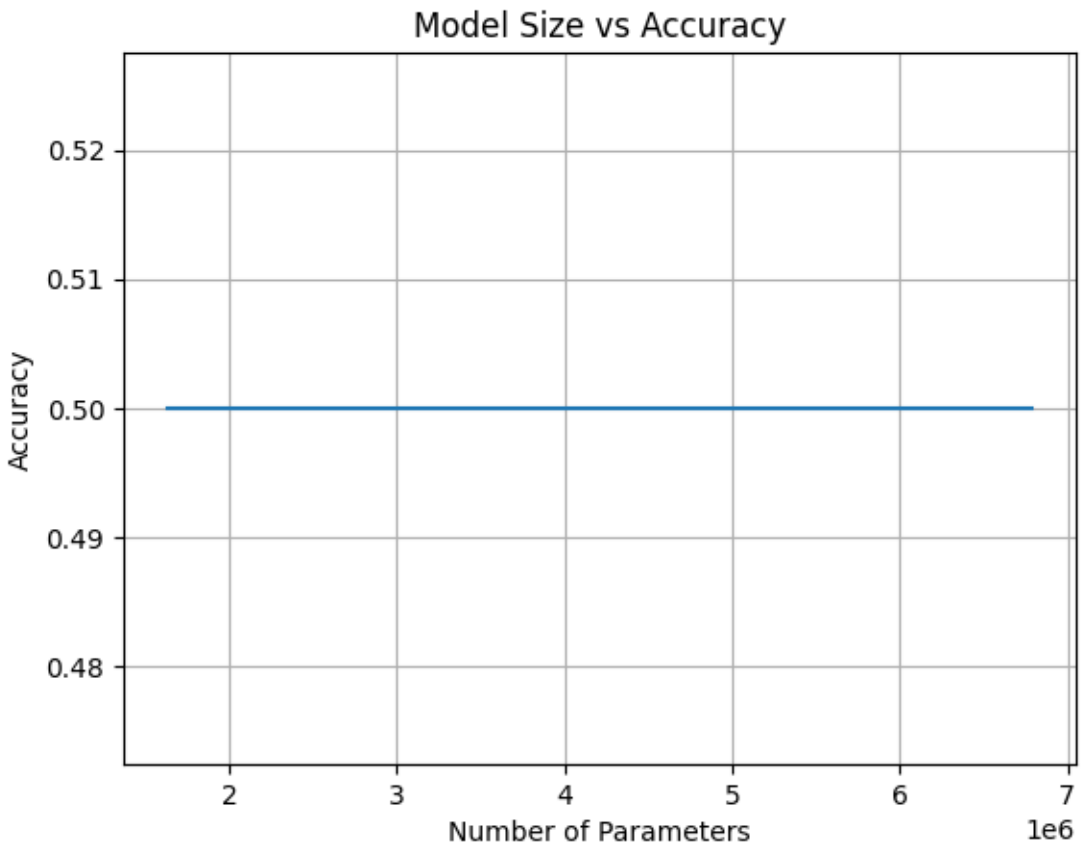
```
 conv2d_8 (Conv2D)                (None, 28, 28, 128)            73,856

 max_pooling2d_8 (MaxPooling2D)   (None, 14, 14, 128)                0

 flatten_7 (Flatten)              (None, 25088)                      0

 dense_32 (Dense)                 (None, 128)                3,211,392

 dropout_11 (Dropout)             (None, 128)                        0

 dense_33 (Dense)                 (None, 1)                        129


 Total params: 3,304,769 (12.61 MB)

 Trainable params: 3,304,769 (12.61 MB)

 Non-trainable params: 0 (0.00 B)

Epoch 1/15
/usr/local/lib/python3.12/dist-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()
8/8            12s 1s/step -
accuracy: 0.5027 - loss: 0.8727 - val_accuracy: 0.5000 - val_loss: 0.6951
Epoch 2/15
8/8            9s 1s/step -
accuracy: 0.5085 - loss: 0.6974 - val_accuracy: 0.5000 - val_loss: 0.6932
Epoch 3/15
8/8            11s 1s/step -
accuracy: 0.5552 - loss: 0.6890 - val_accuracy: 0.5000 - val_loss: 0.6935
Epoch 4/15
8/8            8s 924ms/step -
accuracy: 0.5186 - loss: 0.6938 - val_accuracy: 0.5000 - val_loss: 0.6932
Epoch 5/15
8/8            11s 1s/step -
accuracy: 0.5505 - loss: 0.6916 - val_accuracy: 0.5000 - val_loss: 0.6935
Epoch 6/15
8/8            9s 1s/step -
accuracy: 0.5485 - loss: 0.6940 - val_accuracy: 0.4667 - val_loss: 0.6933
```

```
Epoch 7/15
8/8             10s 1s/step -
accuracy: 0.5031 - loss: 0.6929 - val_accuracy: 0.5000 - val_loss: 0.6937
Epoch 8/15
8/8             9s 1s/step -
accuracy: 0.4862 - loss: 0.6943 - val_accuracy: 0.5167 - val_loss: 0.6933
Epoch 9/15
8/8             9s 1s/step -
accuracy: 0.5189 - loss: 0.6928 - val_accuracy: 0.4833 - val_loss: 0.6933
Epoch 10/15
8/8             8s 974ms/step -
accuracy: 0.5628 - loss: 0.6932 - val_accuracy: 0.5333 - val_loss: 0.6933
Epoch 11/15
8/8             9s 1s/step -
accuracy: 0.5381 - loss: 0.6927 - val_accuracy: 0.5000 - val_loss: 0.6932
Epoch 12/15
8/8             9s 1s/step -
accuracy: 0.5036 - loss: 0.6929 - val_accuracy: 0.5000 - val_loss: 0.6932
Epoch 13/15
8/8             8s 1s/step -
accuracy: 0.5527 - loss: 0.6912 - val_accuracy: 0.5000 - val_loss: 0.6944
Epoch 14/15
8/8             9s 1s/step -
accuracy: 0.4543 - loss: 0.6985 - val_accuracy: 0.5000 - val_loss: 0.6932
Epoch 15/15
8/8             10s 1s/step -
accuracy: 0.5362 - loss: 0.6920 - val_accuracy: 0.5000 - val_loss: 0.6932
Total Training Time (seconds): 140.37187147140503
60/60             1s 20ms/step -
accuracy: 0.1632 - loss: 0.6951
Test Accuracy: 0.5
Testing Time per Sample (seconds): 0.021275591850280762
```




```
Found 240 images belonging to 2 classes.
Found 240 images belonging to 2 classes.
Found 240 images belonging to 2 classes.
Found 240 images belonging to 2 classes.
```


Data Size vs Accuracy
0.535
0.530
0.525
0.520
0.515
0.510
0.505
0.500
Accuracy
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Fraction of Training Data




```
Assignment 7 Completed Successfully!
```

[ ]:
```python
#ASSIGNMENT  8

import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical

# ------------------------------
# Load CIFAR-10 Dataset
# ------------------------------
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

X_train = X_train / 255.0
X_test = X_test / 255.0
```

```
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# ----------------------------
# Build VGG16-Style CNN
# ----------------------------
model = Sequential()

# Block 1
model.add(Conv2D(64, (3,3), activation='relu', padding='same',
                 input_shape=(32,32,3)))
model.add(Conv2D(64, (3,3), activation='relu', padding='same'))
model.add(MaxPooling2D((2,2)))

# Block 2
model.add(Conv2D(128, (3,3), activation='relu', padding='same'))
model.add(Conv2D(128, (3,3), activation='relu', padding='same'))
model.add(MaxPooling2D((2,2)))

# Block 3
model.add(Conv2D(256, (3,3), activation='relu', padding='same'))
model.add(Conv2D(256, (3,3), activation='relu', padding='same'))
model.add(Conv2D(256, (3,3), activation='relu', padding='same'))
model.add(MaxPooling2D((2,2)))

# Block 4
model.add(Conv2D(512, (3,3), activation='relu', padding='same'))
model.add(Conv2D(512, (3,3), activation='relu', padding='same'))
model.add(Conv2D(512, (3,3), activation='relu', padding='same'))
model.add(MaxPooling2D((2,2)))

# Block 5
model.add(Conv2D(512, (3,3), activation='relu', padding='same'))
model.add(Conv2D(512, (3,3), activation='relu', padding='same'))
model.add(Conv2D(512, (3,3), activation='relu', padding='same'))
model.add(MaxPooling2D((2,2)))

# Classifier
model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

# ----------------------------
# Compile
```

```
# ----------------------------
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

model.summary()

# ----------------------------
# Train Model
# ----------------------------
history = model.fit(
    X_train, y_train,
    epochs=10,
    batch_size=64,
    validation_split=0.2
)

# ----------------------------
# Evaluate on Test Set
# ----------------------------
test_loss, test_acc = model.evaluate(X_test, y_test)

print("\nFinal Test Accuracy:", test_acc)

# ----------------------------
# Print Final Train & Val Accuracy
# ----------------------------
final_train_acc = history.history['accuracy'][-1]
final_val_acc = history.history['val_accuracy'][-1]

print("Final Training Accuracy:", final_train_acc)
print("Final Validation Accuracy:", final_val_acc)

# ----------------------------
# Plot & Save Accuracy Graph
# ----------------------------
plt.figure()
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Training vs Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'])
plt.savefig("vgg_epoch_accuracy.png", dpi=300)
plt.show()
```

```
# ----------------------------
# Plot & Save Loss Graph
# ----------------------------
plt.figure()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Training vs Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'])
plt.savefig("vgg_epoch_loss.png", dpi=300)
plt.show()
```

```
Model: "sequential_18"
```

Layer (type)	Output Shape	Param #
conv2d_30 (Conv2D)	(None, 32, 32, 64)	1,792
conv2d_31 (Conv2D)	(None, 32, 32, 64)	36,928
max_pooling2d_30 (MaxPooling2D)	(None, 16, 16, 64)	0
conv2d_32 (Conv2D)	(None, 16, 16, 128)	73,856
conv2d_33 (Conv2D)	(None, 16, 16, 128)	147,584
max_pooling2d_31 (MaxPooling2D)	(None, 8, 8, 128)	0
conv2d_34 (Conv2D)	(None, 8, 8, 256)	295,168
conv2d_35 (Conv2D)	(None, 8, 8, 256)	590,080
conv2d_36 (Conv2D)	(None, 8, 8, 256)	590,080
max_pooling2d_32 (MaxPooling2D)	(None, 4, 4, 256)	0
conv2d_37 (Conv2D)	(None, 4, 4, 512)	1,180,160
conv2d_38 (Conv2D)	(None, 4, 4, 512)	2,359,808
conv2d_39 (Conv2D)	(None, 4, 4, 512)	2,359,808
max_pooling2d_33 (MaxPooling2D)	(None, 2, 2, 512)	0

```
 conv2d_40 (Conv2D)                (None, 2, 2, 512)           2,359,808

 conv2d_41 (Conv2D)                (None, 2, 2, 512)           2,359,808

 conv2d_42 (Conv2D)                (None, 2, 2, 512)           2,359,808

 max_pooling2d_34 (MaxPooling2D)   (None, 1, 1, 512)                   0

 flatten_15 (Flatten)              (None, 512)                         0

 dense_48 (Dense)                  (None, 4096)                2,101,248

 dropout_19 (Dropout)              (None, 4096)                        0

 dense_49 (Dense)                  (None, 4096)               16,781,312

 dropout_20 (Dropout)              (None, 4096)                        0

 dense_50 (Dense)                  (None, 10)                     40,970


 Total params: 33,638,218 (128.32 MB)

 Trainable params: 33,638,218 (128.32 MB)

 Non-trainable params: 0 (0.00 B)

Epoch 1/10
625/625             4477s 7s/step -
accuracy: 0.0989 - loss: 2.3056 - val_accuracy: 0.1003 - val_loss: 2.3028
Epoch 2/10
160/625             59:21 8s/step -
accuracy: 0.0979 - loss: 2.3033
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/tmp/ipython-input-3018839257.py in <cell line: 0>()
     76 # Train Model
     77 # ----------------------------
---> 78 history = model.fit(
     79     X_train, y_train,
     80     epochs=10,

/usr/local/lib/python3.12/dist-packages/keras/src/utils/traceback_utils.py in␣
 ↪error_handler(*args, **kwargs)
```

```
    115         filtered_tb = None
    116         try:
--> 117             return fn(*args, **kwargs)
    118         except Exception as e:
    119             filtered_tb = _process_traceback_frames(e.__traceback__)

/usr/local/lib/python3.12/dist-packages/keras/src/backend/tensorflow/trainer.py
 ↪in fit(self, x, y, batch_size, epochs, verbose, callbacks, validation_split,␣
 ↪validation_data, shuffle, class_weight, sample_weight, initial_epoch,␣
 ↪steps_per_epoch, validation_steps, validation_batch_size, validation_freq)
    375                 for step, iterator in epoch_iterator:
    376                     callbacks.on_train_batch_begin(step)
--> 377                     logs = self.train_function(iterator)
    378                     callbacks.on_train_batch_end(step, logs)
    379                     if self.stop_training:

/usr/local/lib/python3.12/dist-packages/keras/src/backend/tensorflow/trainer.py
 ↪in function(iterator)
    218                 iterator, (tf.data.Iterator, tf.distribute.
 ↪DistributedIterator)
    219             ):
--> 220                 opt_outputs = multi_step_on_iterator(iterator)
    221                 if not opt_outputs.has_value():
    222                     raise StopIteration

/usr/local/lib/python3.12/dist-packages/tensorflow/python/util/traceback_utils.
 ↪py in error_handler(*args, **kwargs)
    148     filtered_tb = None
    149     try:
--> 150       return fn(*args, **kwargs)
    151     except Exception as e:
    152       filtered_tb = _process_traceback_frames(e.__traceback__)

/usr/local/lib/python3.12/dist-packages/tensorflow/python/eager/
 ↪polymorphic_function/polymorphic_function.py in __call__(self, *args, **kwds)
    831
    832       with OptionalXlaContext(self._jit_compile):
--> 833         result = self._call(*args, **kwds)
    834
    835       new_tracing_count = self.experimental_get_tracing_count()

/usr/local/lib/python3.12/dist-packages/tensorflow/python/eager/
 ↪polymorphic_function/polymorphic_function.py in _call(self, *args, **kwds)
    876       # In this case we have not created variables on the first call. S␣
 ↪we can
    877       # run the first trace but we should fail if variables are created
--> 878       results = tracing_compilation.call_function(
    879           args, kwds, self._variable_creation_config
```

```
    880       )

/usr/local/lib/python3.12/dist-packages/tensorflow/python/eager/
 ↪polymorphic_function/tracing_compilation.py in call_function(args, kwargs,␣
 ↪tracing_options)
    137   bound_args = function.function_type.bind(*args, **kwargs)
    138   flat_inputs = function.function_type.unpack_inputs(bound_args)
--> 139   return function._call_flat(  # pylint: disable=protected-access
    140       flat_inputs, captured_inputs=function.captured_inputs
    141   )

/usr/local/lib/python3.12/dist-packages/tensorflow/python/eager/
 ↪polymorphic_function/concrete_function.py in _call_flat(self, tensor_inputs,␣
 ↪captured_inputs)
   1320         and executing_eagerly):
   1321       # No tape is watching; skip to running the function.
-> 1322       return self._inference_function.call_preflattened(args)
   1323     forward_backward = self._select_forward_and_backward_functions(
   1324         args,

/usr/local/lib/python3.12/dist-packages/tensorflow/python/eager/
 ↪polymorphic_function/atomic_function.py in call_preflattened(self, args)
    214   def call_preflattened(self, args: Sequence[core.Tensor]) -> Any:
    215     """Calls with flattened tensor inputs and returns the structured␣
 ↪output."""
--> 216     flat_outputs = self.call_flat(*args)
    217     return self.function_type.pack_output(flat_outputs)
    218

/usr/local/lib/python3.12/dist-packages/tensorflow/python/eager/
 ↪polymorphic_function/atomic_function.py in call_flat(self, *args)
    249         with record.stop_recording():
    250           if self._bound_context.executing_eagerly():
--> 251             outputs = self._bound_context.call_function(
    252                 self.name,
    253                 list(args),

/usr/local/lib/python3.12/dist-packages/tensorflow/python/eager/context.py in␣
 ↪call_function(self, name, tensor_inputs, num_outputs)
   1686     cancellation_context = cancellation.context()
   1687     if cancellation_context is None:
-> 1688       outputs = execute.execute(
   1689           name.decode("utf-8"),
   1690           num_outputs=num_outputs,

/usr/local/lib/python3.12/dist-packages/tensorflow/python/eager/execute.py in␣
 ↪quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
```

```
     51   try:
     52     ctx.ensure_initialized()
---> 53     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name,␣
 ↪op_name,
     54                                         inputs, attrs, num_outputs)
     55   except core._NotOkStatusException as e:

KeyboardInterrupt:
```

[ ]:
```
# ================================
# ASSIGNMENT 9 (COLAB VERSION)
# ================================

import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications import VGG16, ResNet50, MobileNetV2
from tensorflow.keras.applications.vgg16 import preprocess_input as vgg_pre
from tensorflow.keras.applications.resnet50 import preprocess_input as res_pre
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input as␣
 ↪mob_pre
from tensorflow.keras.models import Model
import os
from google.colab import files

# ----------------------------
# Upload Image
# ----------------------------
print("Upload your image (jpg/png)")
uploaded = files.upload()

img_path = list(uploaded.keys())[0]

# ----------------------------
# Create folder for plots
# ----------------------------
os.makedirs("plots", exist_ok=True)

# ----------------------------
# Load and preprocess image
# ----------------------------
img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)

# ----------------------------
```

```python
# Feature map visualization
# ----------------------------
def visualize_feature_maps(model, preprocess_fn, save_name, max_channels=16):

    img_processed = preprocess_fn(img_array.copy())

    conv_layers = [layer.output for layer in model.layers if "conv" in layer.
 ↪name]
    if len(conv_layers) == 0:
        print(f"No convolution layers found in {model.name}")
        return

    layer_output = conv_layers[0]

    feature_model = Model(inputs=model.input, outputs=layer_output)
    feature_maps = feature_model.predict(img_processed)

    fmap = feature_maps[0]
    num_channels = fmap.shape[-1]
    num_to_plot = min(max_channels, num_channels)

    plt.figure(figsize=(10, 10))
    for i in range(num_to_plot):
        plt.subplot(4, 4, i+1)
        plt.imshow(fmap[:, :, i], cmap='viridis')
        plt.axis('off')

    plt.tight_layout()
    plt.savefig(f"plots/{save_name}", dpi=300)
    plt.show()

    print(f"Saved: plots/{save_name}")

# ----------------------------
# VGG16
# ----------------------------
print("Processing VGG16...")
vgg_model = VGG16(weights='imagenet', include_top=False)
visualize_feature_maps(vgg_model, vgg_pre, "vgg_feature_maps.png")

# ----------------------------
# ResNet50
# ----------------------------
print("Processing ResNet50...")
resnet_model = ResNet50(weights='imagenet', include_top=False)
visualize_feature_maps(resnet_model, res_pre, "resnet_feature_maps.png")
```

```
# ----------------------------
# MobileNetV2
# ----------------------------
print("Processing MobileNetV2...")
mobilenet_model = MobileNetV2(weights='imagenet', include_top=False)
visualize_feature_maps(mobilenet_model, mob_pre, "mobilenet_feature_maps.png")

print("Assignment 9 Completed Successfully!")
```

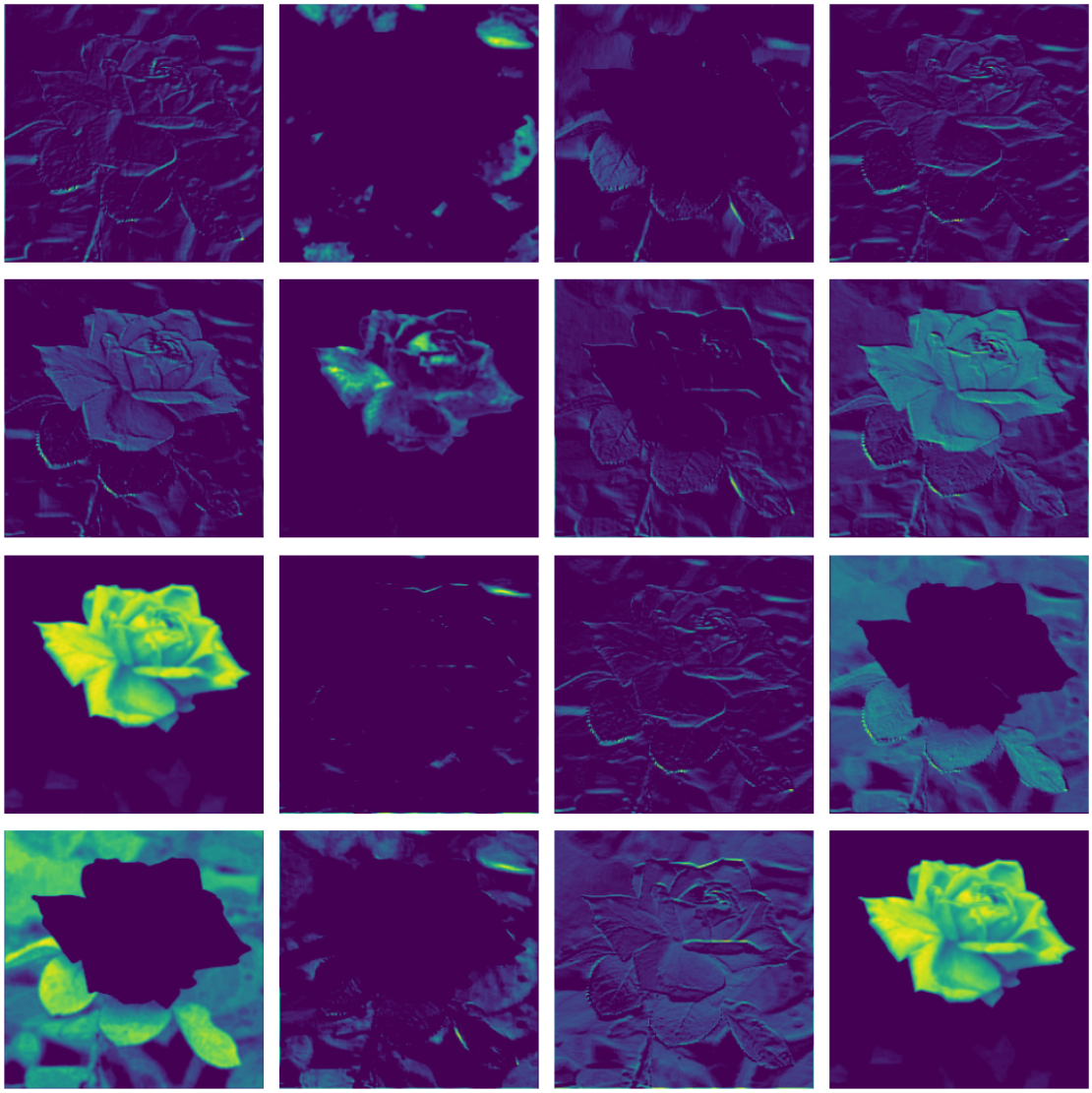
```
Upload your image (jpg/png)
```

```
<IPython.core.display.HTML object>
```

```
Saving photo_2026-02-23_07-01-05.jpg to photo_2026-02-23_07-01-05.jpg
Saving photo_2026-02-23_07-01-03.jpg to photo_2026-02-23_07-01-03.jpg
Saving photo_2026-02-23_07-01-00.jpg to photo_2026-02-23_07-01-00.jpg
Saving photo_2026-02-23_07-00-58.jpg to photo_2026-02-23_07-00-58.jpg
Saving photo_2026-02-23_07-00-55.jpg to photo_2026-02-23_07-00-55.jpg
Saving photo_2026-02-23_07-00-53.jpg to photo_2026-02-23_07-00-53.jpg
Processing VGG16…
```

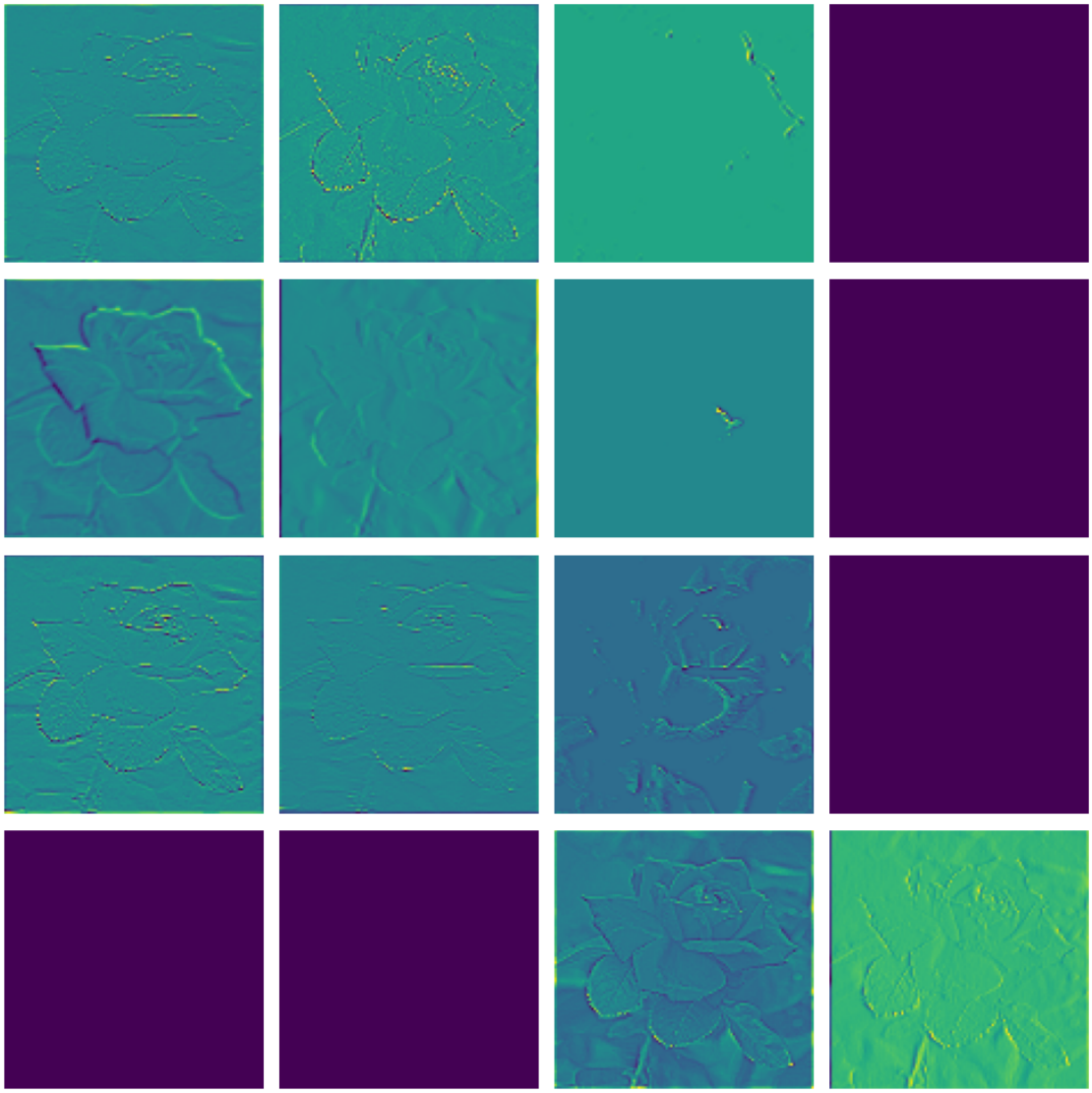
```
WARNING:tensorflow:6 out of the last 9 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x7cebbe61a5c0> triggered tf.function retracing. Tracing is expensive and the
excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

```
1/1             0s 80ms/step
```



```
Saved: plots/vgg_feature_maps.png
Processing ResNet50…
1/1             0s 65ms/step
```

```
Saved: plots/resnet_feature_maps.png
Processing MobileNetV2…

/tmp/ipython-input-2518002157.py:88: UserWarning: `input_shape` is undefined or
non-square, or `rows` is not in [96, 128, 160, 192, 224]. Weights for input
shape (224, 224) will be loaded as the default.
  mobilenet_model = MobileNetV2(weights='imagenet', include_top=False)

1/1             0s 134ms/step
```



```
Saved: plots/mobilenet_feature_maps.png
Assignment 9 Completed Successfully!
```

[ ]:
```
#ASSIGNMENT  10

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.applications import VGG16
import matplotlib.pyplot as plt
import os
```

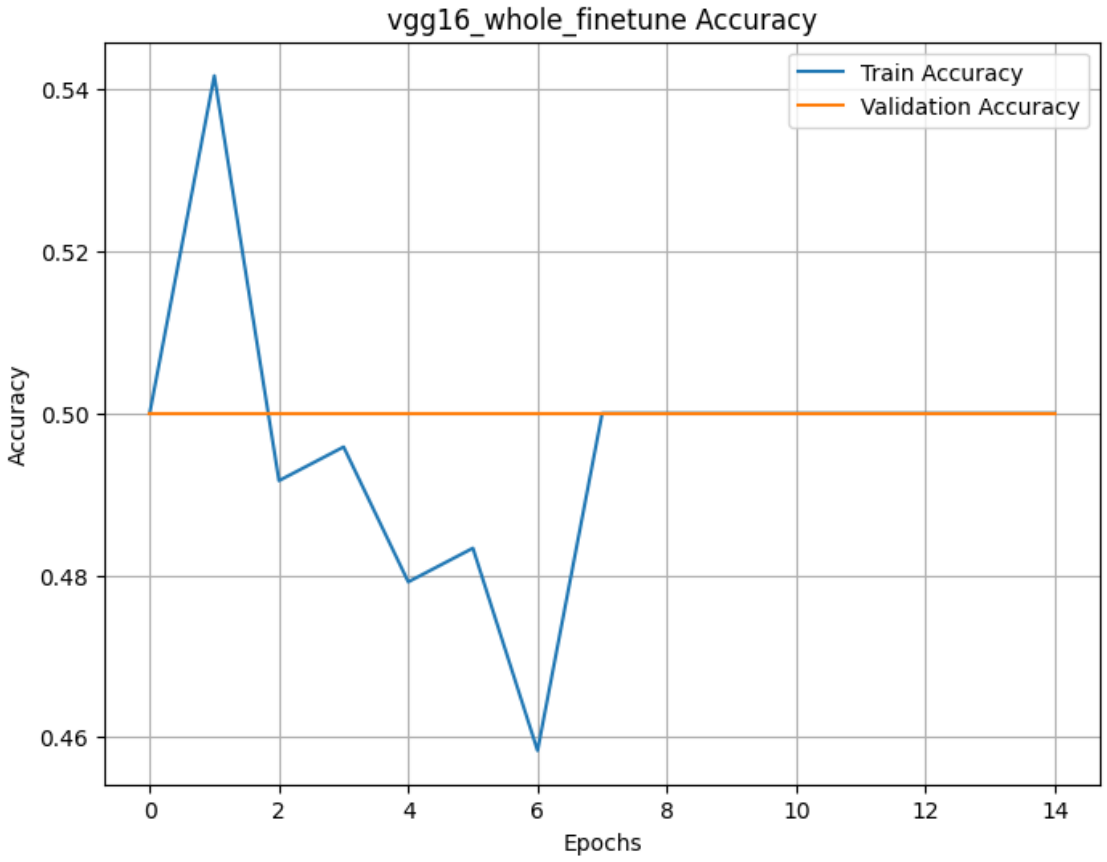
```
# ----------------------------
# Parameters
# ----------------------------
IMG_SIZE = 128
BATCH_SIZE = 32
EPOCHS = 15
DATASET_DIR = 'datasets/'  # Folder containing 'resizedShirt' and 'Ishirt'

os.makedirs("plots", exist_ok=True)

# ----------------------------
# Data Preparation
# ----------------------------
datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

train_data = datagen.flow_from_directory(
    DATASET_DIR,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    subset='training',
    class_mode='binary'
)

val_data = datagen.flow_from_directory(
    DATASET_DIR,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    subset='validation',
    class_mode='binary'
)

# ----------------------------
# Build VGG16-based Model
# ----------------------------
def build_vgg16_model(trainable_layers='all'):
    """
    trainable_layers: 'all' for full fine-tuning
                      'top' for partial fine-tuning (freeze lower layers)
    """
    # Load VGG16 base
    base_model = VGG16(weights='imagenet', include_top=False,␣
 ↪input_shape=(IMG_SIZE, IMG_SIZE, 3))

    if trainable_layers == 'top':
        # Freeze lower layers, only last 4 convolutional blocks trainable
        for layer in base_model.layers[:-4*3]:  # Approx 4 conv blocks = 4*3␣
 ↪layers
```

```
            layer.trainable = False
    else:
        # Whole model trainable
        for layer in base_model.layers:
            layer.trainable = True

    x = Flatten()(base_model.output)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.3)(x)
    output = Dense(1, activation='sigmoid')(x)

    model = Model(inputs=base_model.input, outputs=output)

    model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
    return model

# ----------------------------
# Training Function
# ----------------------------
def train_and_plot(trainable_type):
    if trainable_type == 'all':
        model_name = "vgg16_whole_finetune"
        print("\n=== Training: Whole VGG16 Fine-tuning ===")
    else:
        model_name = "vgg16_partial_finetune"
        print("\n=== Training: Partial VGG16 Fine-tuning ===")

    model = build_vgg16_model(trainable_layers=trainable_type)
    history = model.fit(
        train_data,
        validation_data=val_data,
        epochs=EPOCHS,
        verbose=2
    )

    # Evaluate test accuracy on validation set
    loss, acc = model.evaluate(val_data, verbose=0)
    print(f"{model_name} Test Accuracy: {acc*100:.2f}%")

    # Plot training & validation accuracy
    plt.figure(figsize=(8,6))
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title(f"{model_name} Accuracy")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
```

```
    plt.legend()
    plt.grid(True)
    plot_path = f"plots/{model_name}.png"
    plt.savefig(plot_path, dpi=300)
    plt.show()
    print(f"Plot saved to {plot_path}\n")

    return acc

# ------------------------------
# Run Both Fine-tuning Strategies
# ------------------------------
acc_whole = train_and_plot('all')
acc_partial = train_and_plot('top')

print("Final Test Accuracy:")
print(f"Whole VGG16 Fine-tuning: {acc_whole*100:.2f}%")
print(f"Partial VGG16 Fine-tuning: {acc_partial*100:.2f}%")
```

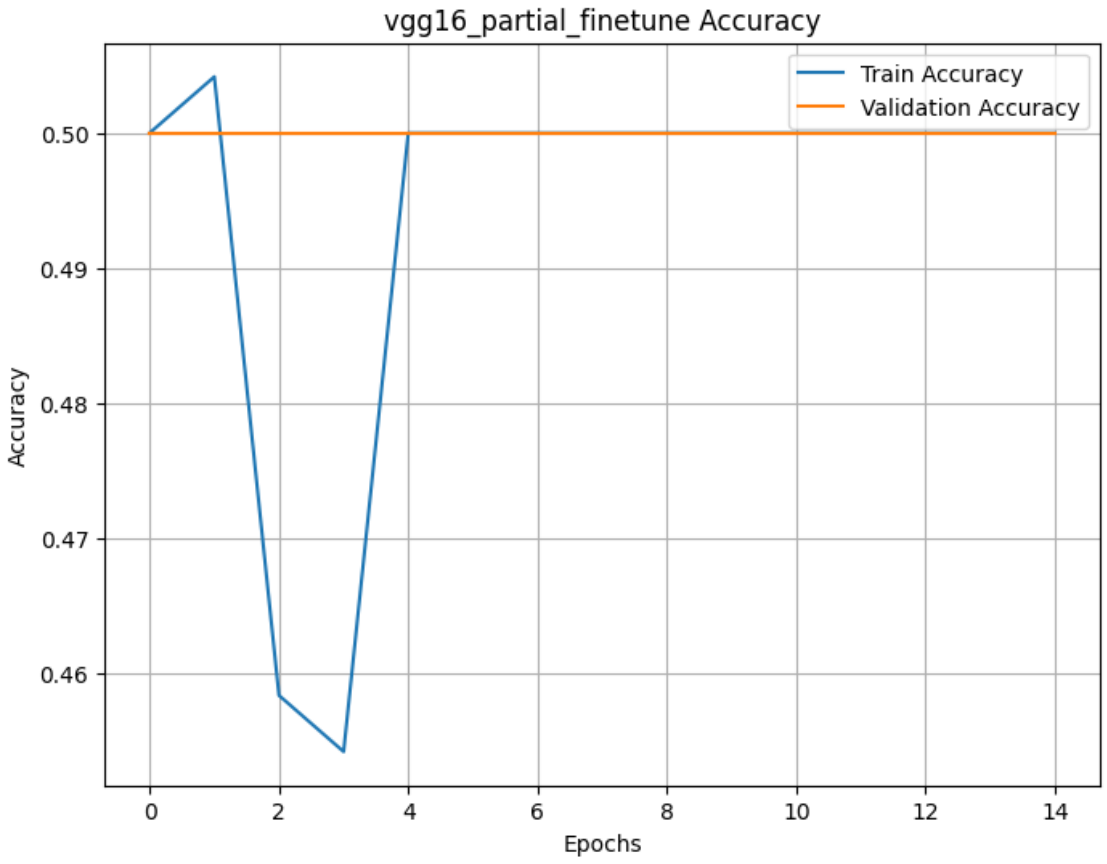
```
Found 240 images belonging to 2 classes.
Found 60 images belonging to 2 classes.

=== Training: Whole VGG16 Fine-tuning ===
Epoch 1/15
8/8 - 177s - 22s/step - accuracy: 0.5000 - loss: 4.1705 - val_accuracy: 0.5000 -
val_loss: 0.6934
Epoch 2/15
8/8 - 203s - 25s/step - accuracy: 0.5417 - loss: 0.7112 - val_accuracy: 0.5000 -
val_loss: 0.7478
Epoch 3/15
8/8 - 170s - 21s/step - accuracy: 0.4917 - loss: 0.7292 - val_accuracy: 0.5000 -
val_loss: 0.6932
Epoch 4/15
8/8 - 172s - 21s/step - accuracy: 0.4958 - loss: 0.6998 - val_accuracy: 0.5000 -
val_loss: 0.6948
Epoch 5/15
8/8 - 171s - 21s/step - accuracy: 0.4792 - loss: 0.6954 - val_accuracy: 0.5000 -
val_loss: 0.6940
Epoch 6/15
8/8 - 171s - 21s/step - accuracy: 0.4833 - loss: 0.6942 - val_accuracy: 0.5000 -
val_loss: 0.6931
Epoch 7/15
8/8 - 169s - 21s/step - accuracy: 0.4583 - loss: 0.6938 - val_accuracy: 0.5000 -
val_loss: 0.6932
Epoch 8/15
8/8 - 180s - 22s/step - accuracy: 0.5000 - loss: 0.6933 - val_accuracy: 0.5000 -
val_loss: 0.6932
```

```
Epoch 9/15
8/8 - 211s - 26s/step - accuracy: 0.5000 - loss: 0.6933 - val_accuracy: 0.5000 -
val_loss: 0.6932
Epoch 10/15
8/8 - 169s - 21s/step - accuracy: 0.5000 - loss: 0.6932 - val_accuracy: 0.5000 -
val_loss: 0.6932
Epoch 11/15
8/8 - 169s - 21s/step - accuracy: 0.5000 - loss: 0.6931 - val_accuracy: 0.5000 -
val_loss: 0.6932
Epoch 12/15
8/8 - 175s - 22s/step - accuracy: 0.5000 - loss: 0.6931 - val_accuracy: 0.5000 -
val_loss: 0.6932
Epoch 13/15
8/8 - 175s - 22s/step - accuracy: 0.5000 - loss: 0.6931 - val_accuracy: 0.5000 -
val_loss: 0.6932
Epoch 14/15
8/8 - 169s - 21s/step - accuracy: 0.5000 - loss: 0.6933 - val_accuracy: 0.5000 -
val_loss: 0.6932
Epoch 15/15
8/8 - 172s - 21s/step - accuracy: 0.5000 - loss: 0.6931 - val_accuracy: 0.5000 -
val_loss: 0.6932
vgg16_whole_finetune Test Accuracy: 50.00%
```



```
Plot saved to plots/vgg16_whole_finetune.png


=== Training: Partial VGG16 Fine-tuning ===
Epoch 1/15
8/8 - 136s - 17s/step - accuracy: 0.5000 - loss: 1.2247 - val_accuracy: 0.5000 -
val_loss: 0.7047
Epoch 2/15
8/8 - 130s - 16s/step - accuracy: 0.5042 - loss: 0.7123 - val_accuracy: 0.5000 -
val_loss: 0.7442
Epoch 3/15
8/8 - 130s - 16s/step - accuracy: 0.4583 - loss: 0.7001 - val_accuracy: 0.5000 -
val_loss: 0.6932
Epoch 4/15
8/8 - 130s - 16s/step - accuracy: 0.4542 - loss: 0.6972 - val_accuracy: 0.5000 -
val_loss: 0.6932
Epoch 5/15
8/8 - 131s - 16s/step - accuracy: 0.5000 - loss: 0.6931 - val_accuracy: 0.5000 -
val_loss: 0.6932
Epoch 6/15
8/8 - 129s - 16s/step - accuracy: 0.5000 - loss: 0.6932 - val_accuracy: 0.5000 -
val_loss: 0.6932
Epoch 7/15
8/8 - 131s - 16s/step - accuracy: 0.5000 - loss: 0.6933 - val_accuracy: 0.5000 -
val_loss: 0.6932
Epoch 8/15
8/8 - 131s - 16s/step - accuracy: 0.5000 - loss: 0.6934 - val_accuracy: 0.5000 -
val_loss: 0.6932
Epoch 9/15
8/8 - 129s - 16s/step - accuracy: 0.5000 - loss: 0.6933 - val_accuracy: 0.5000 -
val_loss: 0.6932
Epoch 10/15
8/8 - 130s - 16s/step - accuracy: 0.5000 - loss: 0.6933 - val_accuracy: 0.5000 -
val_loss: 0.6932
Epoch 11/15
8/8 - 131s - 16s/step - accuracy: 0.5000 - loss: 0.6932 - val_accuracy: 0.5000 -
val_loss: 0.6932
Epoch 12/15
8/8 - 129s - 16s/step - accuracy: 0.5000 - loss: 0.6932 - val_accuracy: 0.5000 -
val_loss: 0.6932
Epoch 13/15
8/8 - 131s - 16s/step - accuracy: 0.5000 - loss: 0.6932 - val_accuracy: 0.5000 -
val_loss: 0.6932
Epoch 14/15
8/8 - 140s - 18s/step - accuracy: 0.5000 - loss: 0.6932 - val_accuracy: 0.5000 -
```

```
val_loss: 0.6932
Epoch 15/15
8/8 - 130s - 16s/step - accuracy: 0.5000 - loss: 0.6932 - val_accuracy: 0.5000 -
val_loss: 0.6932
vgg16_partial_finetune Test Accuracy: 50.00%
```




```
Plot saved to plots/vgg16_partial_finetune.png

Final Test Accuracy:
Whole VGG16 Fine-tuning: 50.00%
Partial VGG16 Fine-tuning: 50.00%
```

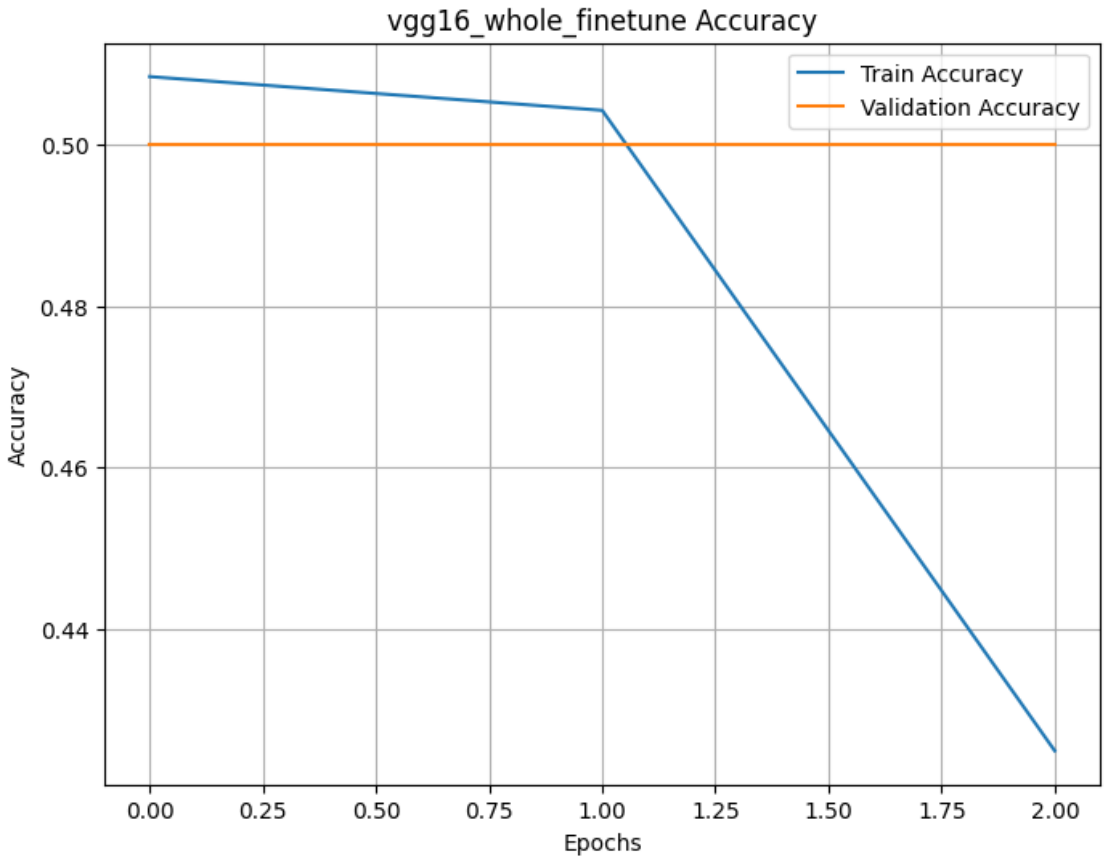
[ ]:
```python
# ASSIGNMENT 10

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.applications import VGG16
import matplotlib.pyplot as plt
```

```
import os

# ----------------------------
# Parameters
# ----------------------------
IMG_SIZE = 128
BATCH_SIZE = 32
EPOCHS = 15
DATASET_DIR = 'datasets/'  # Folder containing 'resizedShirt' and 'Ishirt'

os.makedirs("plots", exist_ok=True)

# ----------------------------
# Data Preparation
# ----------------------------
datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

train_data = datagen.flow_from_directory(
    DATASET_DIR,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    subset='training',
    class_mode='binary'
)

val_data = datagen.flow_from_directory(
    DATASET_DIR,
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    subset='validation',
    class_mode='binary'
)

# ----------------------------
# Build VGG16-based Model
# ----------------------------
def build_vgg16_model(trainable_layers='all'):
    """
    trainable_layers: 'all' for full fine-tuning
                      'top' for partial fine-tuning (freeze lower layers)
    """
    # Load VGG16 base
    base_model = VGG16(weights='imagenet', include_top=False,␣
 ↪input_shape=(IMG_SIZE, IMG_SIZE, 3))

    if trainable_layers == 'top':
        # Freeze lower layers, only last 4 convolutional blocks trainable
```

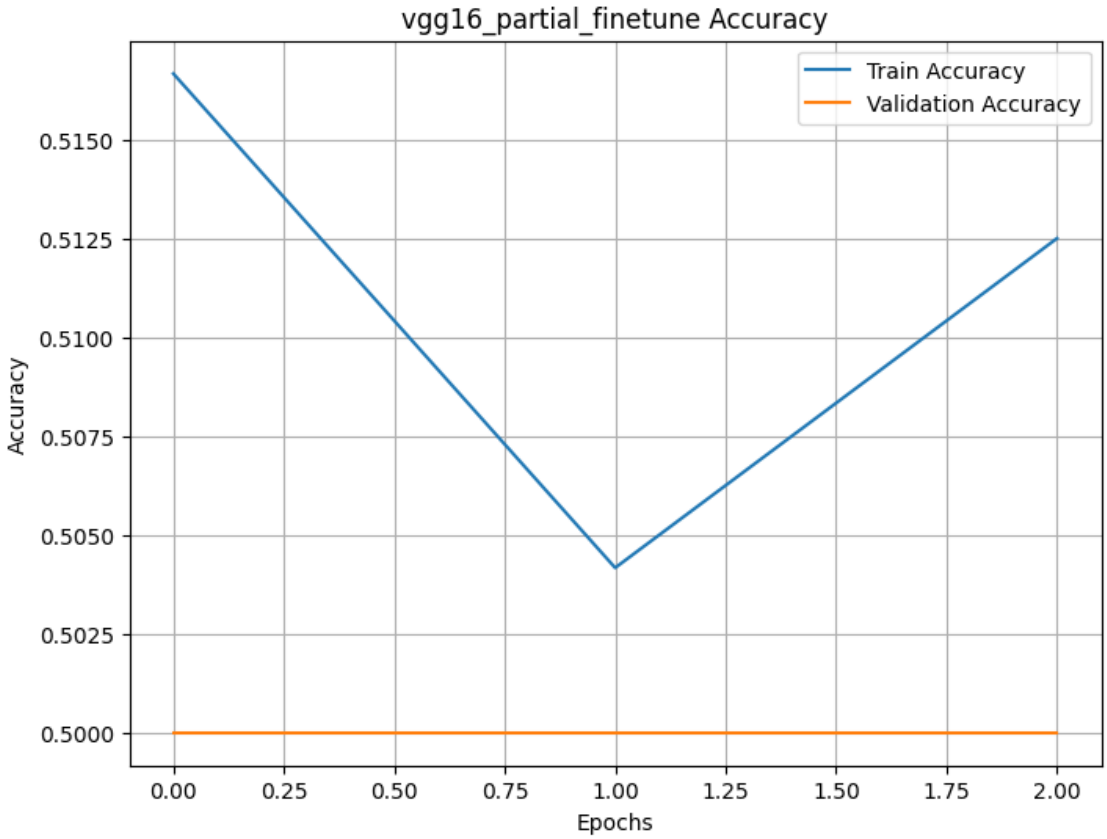
```
        for layer in base_model.layers[:-4*3]:  # Approx 4 conv blocks = 4*3␣
↪layers
            layer.trainable = False
    else:
        # Whole model trainable
        for layer in base_model.layers:
            layer.trainable = True

    x = Flatten()(base_model.output)
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.3)(x)
    output = Dense(1, activation='sigmoid')(x)

    model = Model(inputs=base_model.input, outputs=output)

    model.compile(optimizer='adam', loss='binary_crossentropy',␣
↪metrics=['accuracy'])
    return model

# -----------------------------
# Training Function
# -----------------------------
def train_and_plot(trainable_type):
    if trainable_type == 'all':
        model_name = "vgg16_whole_finetune"
        print("\n=== Training: Whole VGG16 Fine-tuning ===")
    else:
        model_name = "vgg16_partial_finetune"
        print("\n=== Training: Partial VGG16 Fine-tuning ===")

    model = build_vgg16_model(trainable_layers=trainable_type)
    history = model.fit(
        train_data,
        validation_data=val_data,
        epochs=3,
        verbose=1
    )

    # Evaluate test accuracy on validation set
    loss, acc = model.evaluate(val_data, verbose=0)
    print(f"{model_name} Test Accuracy: {acc*100:.2f}%")

    # Plot training & validation accuracy
    plt.figure(figsize=(8,6))
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title(f"{model_name} Accuracy")
```

```
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.legend()
    plt.grid(True)
    plot_path = f"plots/{model_name}.png"
    plt.savefig(plot_path, dpi=300)
    plt.show()
    print(f"Plot saved to {plot_path}\n")

    return acc

# ------------------------------
# Run Both Fine-tuning Strategies
# ------------------------------
acc_whole = train_and_plot('all')
acc_partial = train_and_plot('top')

print("Final Test Accuracy:")
print(f"Whole VGG16 Fine-tuning: {acc_whole*100:.2f}%")
print(f"Partial VGG16 Fine-tuning: {acc_partial*100:.2f}%")
```

```
Found 240 images belonging to 2 classes.
Found 60 images belonging to 2 classes.

=== Training: Whole VGG16 Fine-tuning ===
Epoch 1/3
8/8             221s 27s/step -
accuracy: 0.5109 - loss: 1.5929 - val_accuracy: 0.5000 - val_loss: 0.7260
Epoch 2/3
8/8             217s 28s/step -
accuracy: 0.5191 - loss: 0.7386 - val_accuracy: 0.5000 - val_loss: 0.7006
Epoch 3/3
8/8             172s 21s/step -
accuracy: 0.4387 - loss: 0.7209 - val_accuracy: 0.5000 - val_loss: 0.6937
vgg16_whole_finetune Test Accuracy: 50.00%
```




```
Plot saved to plots/vgg16_whole_finetune.png


=== Training: Partial VGG16 Fine-tuning ===
Epoch 1/3
8/8             138s 17s/step -
accuracy: 0.5312 - loss: 3.1189 - val_accuracy: 0.5000 - val_loss: 0.6969
Epoch 2/3
8/8             130s 16s/step -
accuracy: 0.4798 - loss: 0.7095 - val_accuracy: 0.5000 - val_loss: 0.7138
Epoch 3/3
8/8             130s 16s/step -
accuracy: 0.5413 - loss: 0.6951 - val_accuracy: 0.5000 - val_loss: 0.7163
vgg16_partial_finetune Test Accuracy: 50.00%
```




```
Plot saved to plots/vgg16_partial_finetune.png

Final Test Accuracy:
Whole VGG16 Fine-tuning: 50.00%
Partial VGG16 Fine-tuning: 50.00%
```

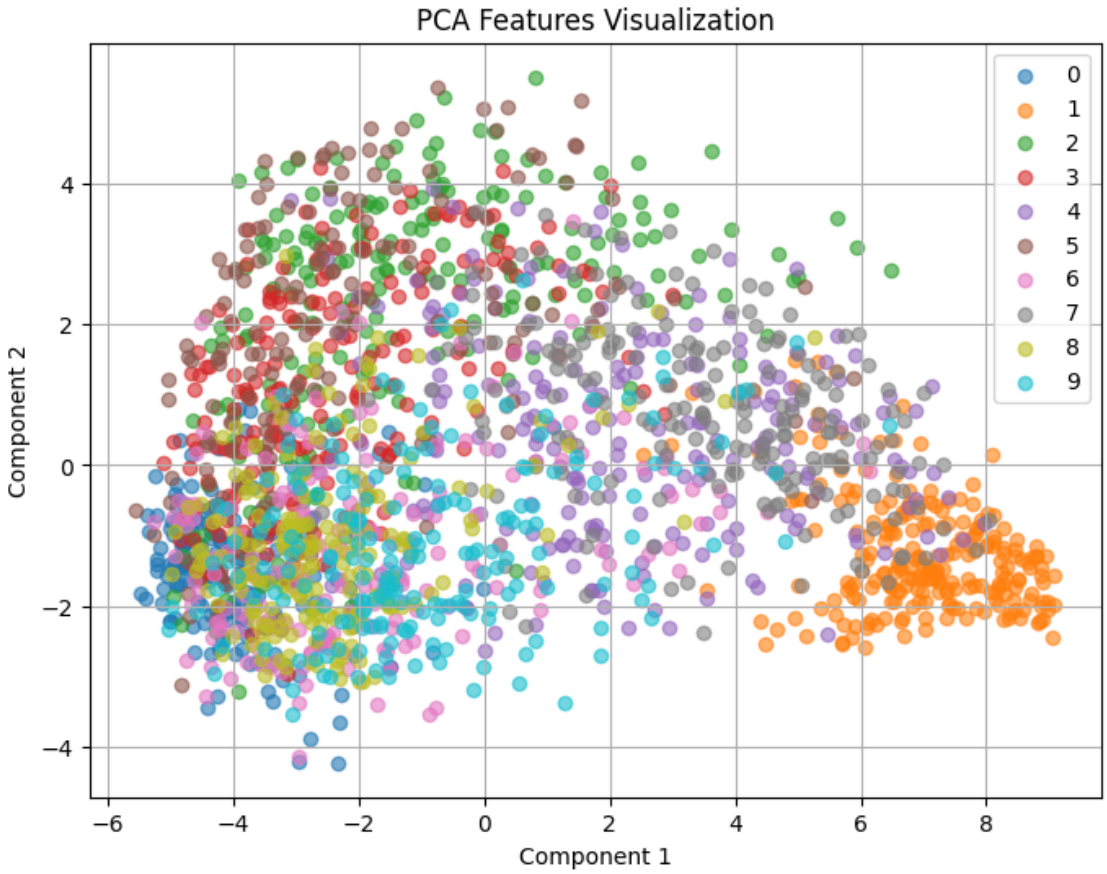
[ ]:
```python
# ASSIGNMENT 11

# ----------------------------
# EASY MNIST + VGG16 + PCA/t-SNE
# ----------------------------
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.applications import VGG16
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
```

```
import os

# ------------------------------
# Parameters
# ------------------------------
IMG_SIZE = 32
BATCH_SIZE = 128
EPOCHS = 3  # Shorter for easy run
os.makedirs("plots", exist_ok=True)

# ------------------------------
# Load MNIST
# ------------------------------
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Preprocess: resize + RGB + normalize
X_train = tf.image.resize(tf.expand_dims(X_train, -1), [IMG_SIZE, IMG_SIZE])
X_train = tf.repeat(X_train, 3, axis=-1) / 255.0
X_test = tf.image.resize(tf.expand_dims(X_test, -1), [IMG_SIZE, IMG_SIZE])
X_test = tf.repeat(X_test, 3, axis=-1) / 255.0

# Convert to numpy
X_train, X_test = X_train.numpy(), X_test.numpy()

# ------------------------------
# Load pre-trained VGG16 (features only)
# ------------------------------
base_model = VGG16(weights='imagenet', include_top=False,␣
 ↪input_shape=(IMG_SIZE, IMG_SIZE, 3))
base_model.trainable = False

# Add simple classification head
x = Flatten()(base_model.output)
x = Dense(128, activation='relu')(x)
x = Dropout(0.3)(x)
output = Dense(10, activation='softmax')(x)
model = Model(base_model.input, output)
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# ------------------------------
# Train model
# ------------------------------
model.fit(X_train, y_train, validation_split=0.1, epochs=EPOCHS,␣
 ↪batch_size=BATCH_SIZE, verbose=2)

# ------------------------------
```

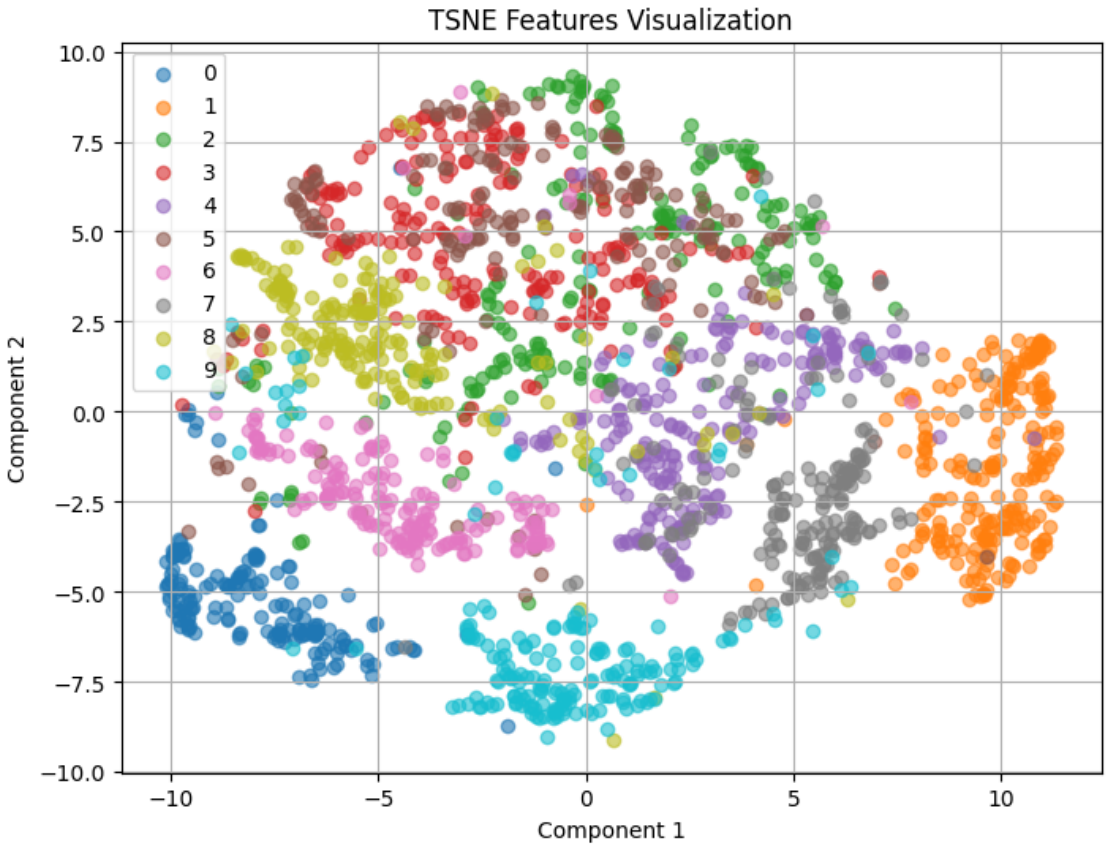
```
# Feature extraction function
# ------------------------------
def extract_features(model, X):
    feature_model = Model(inputs=model.input, outputs=model.layers[-3].output) ␣
 ↪# Flatten layer
    features = feature_model.predict(X, batch_size=BATCH_SIZE, verbose=1)
    return features.reshape(features.shape[0], -1)

features_before = extract_features(Model(inputs=base_model.input,␣
 ↪outputs=Flatten()(base_model.output)), X_test)
features_after = extract_features(model, X_test)

# ------------------------------
# 2D Plot function
# ------------------------------
def plot_2d(features, labels, method='PCA', filename='plot.png',␣
 ↪sample_size=2000):
    features = features[:sample_size]
    labels = labels[:sample_size]

    if method=='PCA':
        reducer = PCA(n_components=2)
    else:
        reducer = TSNE(n_components=2, random_state=42, perplexity=30,␣
 ↪max_iter=300)

    features_2d = reducer.fit_transform(features)

    plt.figure(figsize=(8,6))
    for i in range(10):
        plt.scatter(features_2d[labels==i,0], features_2d[labels==i,1],␣
 ↪label=str(i), alpha=0.6)
    plt.title(f"{method} Features Visualization")
    plt.xlabel("Component 1")
    plt.ylabel("Component 2")
    plt.legend()
    plt.grid(True)
    plt.savefig(f"plots/{filename}", dpi=300)
    plt.show()
    print(f"Saved: plots/{filename}")

# ------------------------------
# Plot PCA & t-SNE (sampled 2000 for speed)
# ------------------------------
plot_2d(features_before, y_test, method='PCA', filename='features_before_PCA.
 ↪png')
```

```
plot_2d(features_before, y_test, method='TSNE', filename='features_before_tSNE.
  ↪png')
plot_2d(features_after, y_test, method='PCA', filename='features_after_PCA.png')
plot_2d(features_after, y_test, method='TSNE', filename='features_after_tSNE.
  ↪png')
```

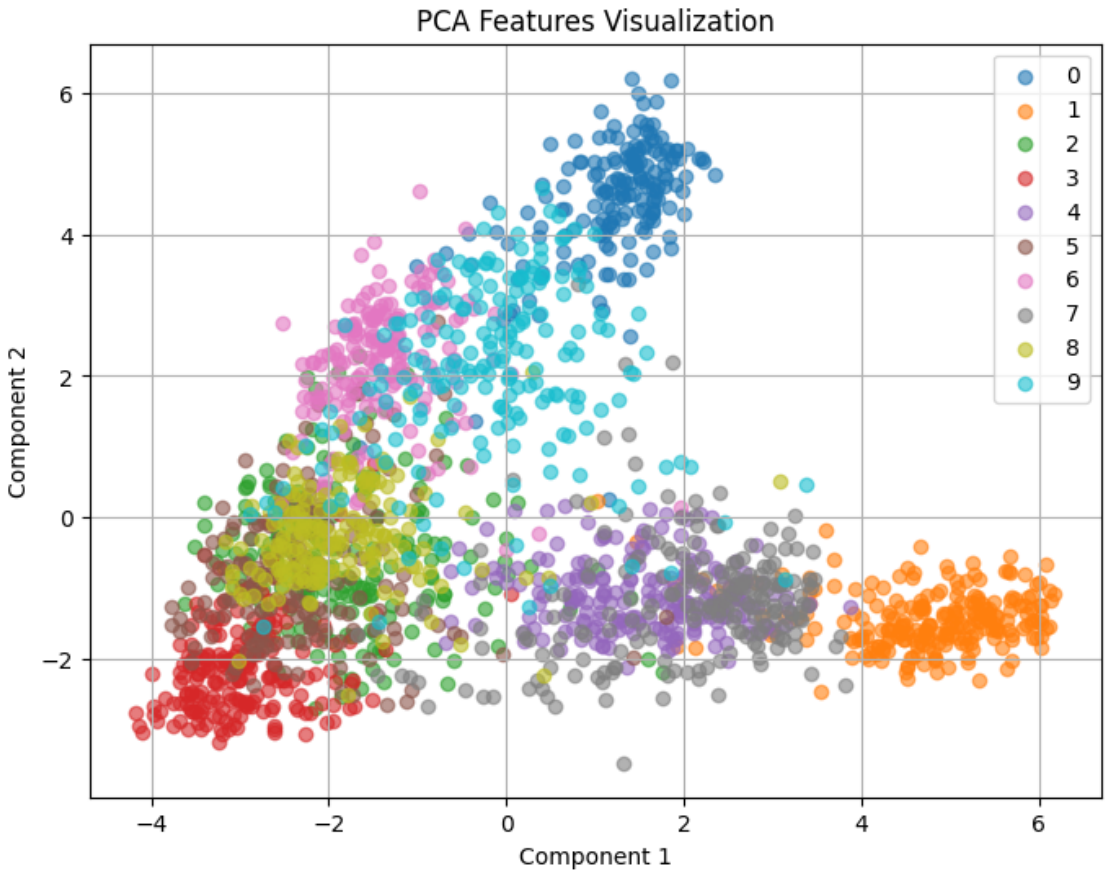
```
Epoch 1/3
422/422 - 668s - 2s/step - accuracy: 0.8057 - loss: 0.6427 - val_accuracy:
0.9367 - val_loss: 0.2254
Epoch 2/3
422/422 - 653s - 2s/step - accuracy: 0.9243 - loss: 0.2562 - val_accuracy:
0.9562 - val_loss: 0.1555
Epoch 3/3
422/422 - 686s - 2s/step - accuracy: 0.9401 - loss: 0.1965 - val_accuracy:
0.9615 - val_loss: 0.1233
79/79              109s 1s/step
79/79              109s 1s/step
```



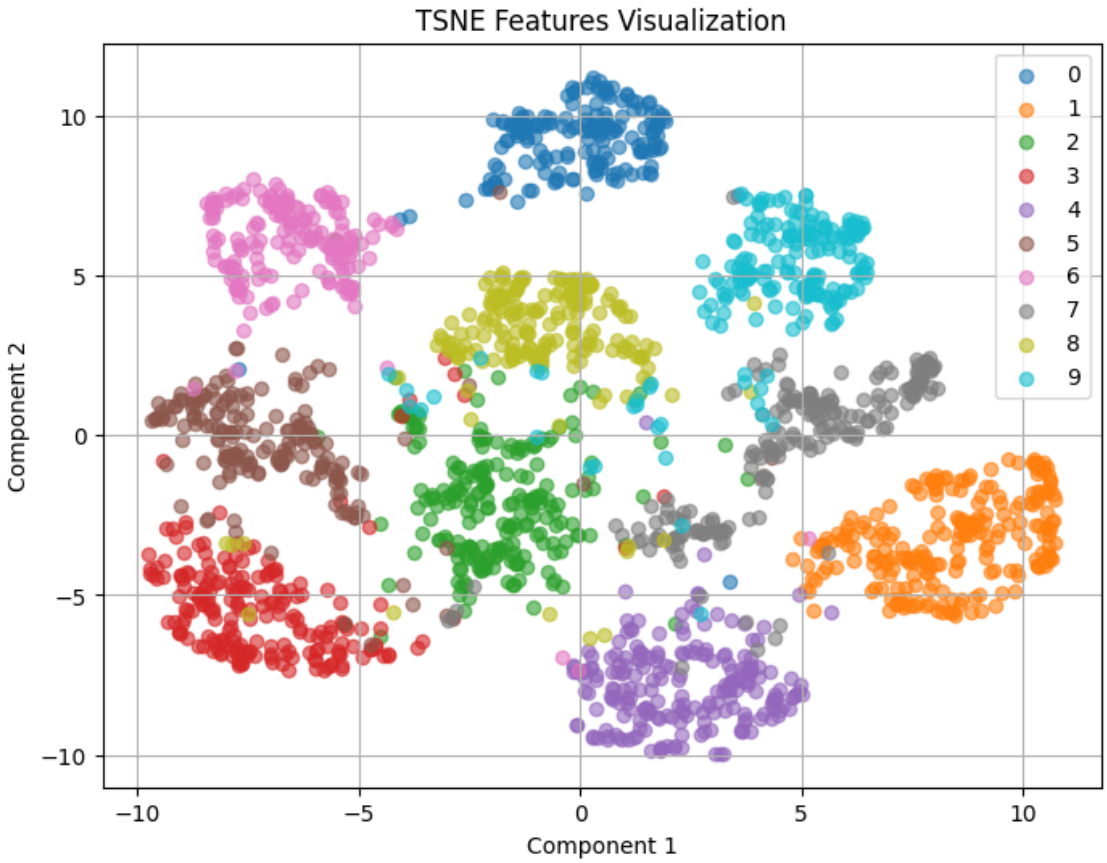

```
Saved: plots/features_before_PCA.png
```




```
Saved: plots/features_before_tSNE.png
```




```
Saved: plots/features_after_PCA.png
```




```
Saved: plots/features_after_tSNE.png
```

[ ]:
```python
# ASSIGNMENT 12

import os
os.environ["TF_ENABLE_ONEDNN_OPTS"] = "1"

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Input
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical

# ----------------------------
# Use all CPU cores
# ----------------------------
tf.config.threading.set_intra_op_parallelism_threads(0)
```

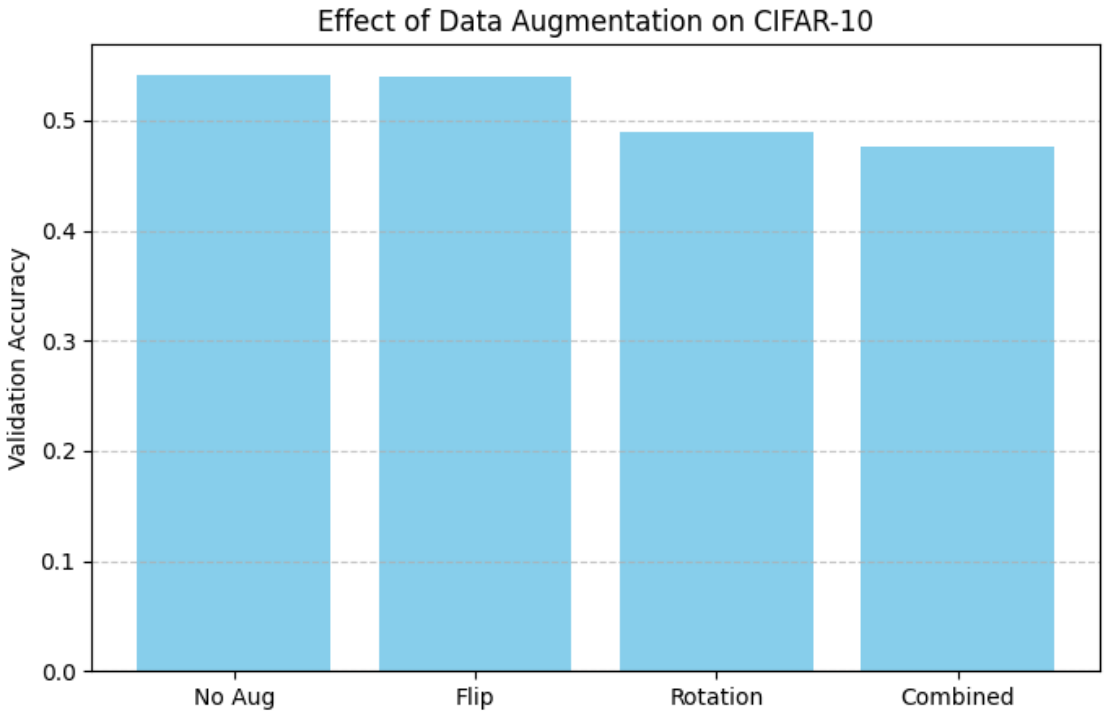
```
tf.config.threading.set_inter_op_parallelism_threads(0)

# ----------------------------
# Load CIFAR-10 & preprocess
# ----------------------------
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# Reduce training set for faster CPU run
X_train, y_train = X_train[:20000], y_train[:20000]

# Normalize
X_train, X_test = X_train / 255.0, X_test / 255.0

# One-hot encode labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# ----------------------------
# Build small CNN
# ----------------------------
def build_model():
    model = Sequential([
        Input(shape=(32,32,3)),
        Conv2D(16, (3,3), activation='relu'),
        MaxPooling2D(2,2),
        Conv2D(32, (3,3), activation='relu'),
        MaxPooling2D(2,2),
        Flatten(),
        Dense(64, activation='relu'),
        Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
    return model

EPOCHS = 5
BATCH_SIZE = 128

# ----------------------------
# 1  No Augmentation
# ----------------------------
model_no_aug = build_model()
history_no_aug = model_no_aug.fit(
    X_train, y_train,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    validation_split=0.2,
```

```
    verbose=1
)
val_acc_no_aug = history_no_aug.history['val_accuracy'][-1]

# ----------------------------
# 2 Flip Augmentation
# ----------------------------
flip_gen = ImageDataGenerator(horizontal_flip=True, validation_split=0.2)
model_flip = build_model()
history_flip = model_flip.fit(
    flip_gen.flow(X_train, y_train, batch_size=BATCH_SIZE, subset='training'),
    epochs=EPOCHS,
    validation_data=flip_gen.flow(X_train, y_train, batch_size=BATCH_SIZE,␣
 ↪subset='validation'),
    verbose=1
)
val_acc_flip = history_flip.history['val_accuracy'][-1]

# ----------------------------
# 3 Rotation Augmentation
# ----------------------------
rot_gen = ImageDataGenerator(rotation_range=15, validation_split=0.2)
model_rot = build_model()
history_rot = model_rot.fit(
    rot_gen.flow(X_train, y_train, batch_size=BATCH_SIZE, subset='training'),
    epochs=EPOCHS,
    validation_data=rot_gen.flow(X_train, y_train, batch_size=BATCH_SIZE,␣
 ↪subset='validation'),
    verbose=1
)
val_acc_rot = history_rot.history['val_accuracy'][-1]

# ----------------------------
# 4 Combined Augmentation
# ----------------------------
combo_gen = ImageDataGenerator(
    horizontal_flip=True,
    rotation_range=15,
    zoom_range=0.1,
    width_shift_range=0.1,
    height_shift_range=0.1,
    validation_split=0.2
)
model_combo = build_model()
history_combo = model_combo.fit(
    combo_gen.flow(X_train, y_train, batch_size=BATCH_SIZE, subset='training'),
    epochs=EPOCHS,
```

```
    validation_data=combo_gen.flow(X_train, y_train, batch_size=BATCH_SIZE,␣
 ↪subset='validation'),
    verbose=1
)
val_acc_combo = history_combo.history['val_accuracy'][-1]

# ------------------------------
# Compare Results
# ------------------------------
methods = ["No Aug", "Flip", "Rotation", "Combined"]
accuracies = [val_acc_no_aug, val_acc_flip, val_acc_rot, val_acc_combo]

plt.figure(figsize=(8,5))
plt.bar(methods, accuracies, color='skyblue')
plt.ylabel("Validation Accuracy")
plt.title("Effect of Data Augmentation on CIFAR-10")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.savefig("augmentation_comparison.png", dpi=300)
plt.show()

print("\nValidation Accuracies:")
for m, a in zip(methods, accuracies):
    print(f"{m}: {a:.4f}")
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071            2s
0us/step
Epoch 1/5
125/125             12s 67ms/step -
accuracy: 0.2310 - loss: 2.0909 - val_accuracy: 0.4128 - val_loss: 1.6531
Epoch 2/5
125/125             5s 40ms/step -
accuracy: 0.4231 - loss: 1.6037 - val_accuracy: 0.4720 - val_loss: 1.5055
Epoch 3/5
125/125             7s 55ms/step -
accuracy: 0.4820 - loss: 1.4489 - val_accuracy: 0.5135 - val_loss: 1.3750
Epoch 4/5
125/125             5s 41ms/step -
accuracy: 0.5219 - loss: 1.3448 - val_accuracy: 0.5303 - val_loss: 1.3351
Epoch 5/5
125/125             6s 50ms/step -
accuracy: 0.5553 - loss: 1.2662 - val_accuracy: 0.5418 - val_loss: 1.2965
Epoch 1/5

/usr/local/lib/python3.12/dist-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
```

```
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()
125/125             8s 56ms/step -
accuracy: 0.2302 - loss: 2.0821 - val_accuracy: 0.3907 - val_loss: 1.6737
Epoch 2/5
125/125             6s 45ms/step -
accuracy: 0.4137 - loss: 1.6314 - val_accuracy: 0.4622 - val_loss: 1.5103
Epoch 3/5
125/125             7s 54ms/step -
accuracy: 0.4715 - loss: 1.4694 - val_accuracy: 0.4983 - val_loss: 1.4091
Epoch 4/5
125/125             6s 46ms/step -
accuracy: 0.5125 - loss: 1.3804 - val_accuracy: 0.5297 - val_loss: 1.3245
Epoch 5/5
125/125             7s 52ms/step -
accuracy: 0.5338 - loss: 1.3228 - val_accuracy: 0.5403 - val_loss: 1.2822
Epoch 1/5
125/125             18s 132ms/step -
accuracy: 0.2269 - loss: 2.1094 - val_accuracy: 0.3955 - val_loss: 1.6848
Epoch 2/5
125/125             16s 126ms/step -
accuracy: 0.3999 - loss: 1.6580 - val_accuracy: 0.4297 - val_loss: 1.5701
Epoch 3/5
125/125             16s 125ms/step -
accuracy: 0.4659 - loss: 1.5020 - val_accuracy: 0.4940 - val_loss: 1.4410
Epoch 4/5
125/125             16s 128ms/step -
accuracy: 0.4919 - loss: 1.4351 - val_accuracy: 0.4967 - val_loss: 1.4317
Epoch 5/5
125/125             15s 122ms/step -
accuracy: 0.5084 - loss: 1.3824 - val_accuracy: 0.4897 - val_loss: 1.4419
Epoch 1/5
125/125             19s 136ms/step -
accuracy: 0.1748 - loss: 2.1581 - val_accuracy: 0.3203 - val_loss: 1.8377
Epoch 2/5
125/125             19s 148ms/step -
accuracy: 0.3440 - loss: 1.7937 - val_accuracy: 0.4010 - val_loss: 1.6518
Epoch 3/5
125/125             17s 132ms/step -
accuracy: 0.4069 - loss: 1.6410 - val_accuracy: 0.4460 - val_loss: 1.5467
Epoch 4/5
125/125             17s 134ms/step -
accuracy: 0.4187 - loss: 1.5799 - val_accuracy: 0.4507 - val_loss: 1.5412
Epoch 5/5
125/125             18s 140ms/step -
accuracy: 0.4527 - loss: 1.5392 - val_accuracy: 0.4767 - val_loss: 1.4734
```




```
Validation Accuracies:
No Aug: 0.5418
Flip: 0.5403
Rotation: 0.4897
Combined: 0.4767
```

[ ]:
```python
# ASSIGNMENT 13

import os
os.environ["TF_ENABLE_ONEDNN_OPTS"] = "1"

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
↪Dropout, Input
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical

# ----------------------------
# Use all CPU cores
```

```
# ------------------------------
tf.config.threading.set_intra_op_parallelism_threads(0)
tf.config.threading.set_inter_op_parallelism_threads(0)

# ------------------------------
# Load CIFAR-10
# ------------------------------
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# Reduce dataset for faster CPU training
X_train, y_train = X_train[:25000], y_train[:25000]

X_train, X_test = X_train / 255.0, X_test / 255.0
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# ------------------------------
# Parameters
# ------------------------------
EPOCHS = 10
BATCH_SIZE = 128

# ------------------------------
# Build Small CNN Model
# ------------------------------
def build_model(use_dropout=False):
    model = Sequential([
        Input(shape=(32,32,3)),
        Conv2D(16, (3,3), activation='relu'),
        MaxPooling2D(2,2),
        Conv2D(32, (3,3), activation='relu'),
        MaxPooling2D(2,2),
        Flatten(),
        Dense(64, activation='relu')
    ])
    if use_dropout:
        model.add(Dropout(0.5))
    model.add(Dense(10, activation='softmax'))

    model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
    return model

# ------------------------------
# 1  Model A: No Dropout, No Aug
# ------------------------------
model_A = build_model(False)
```

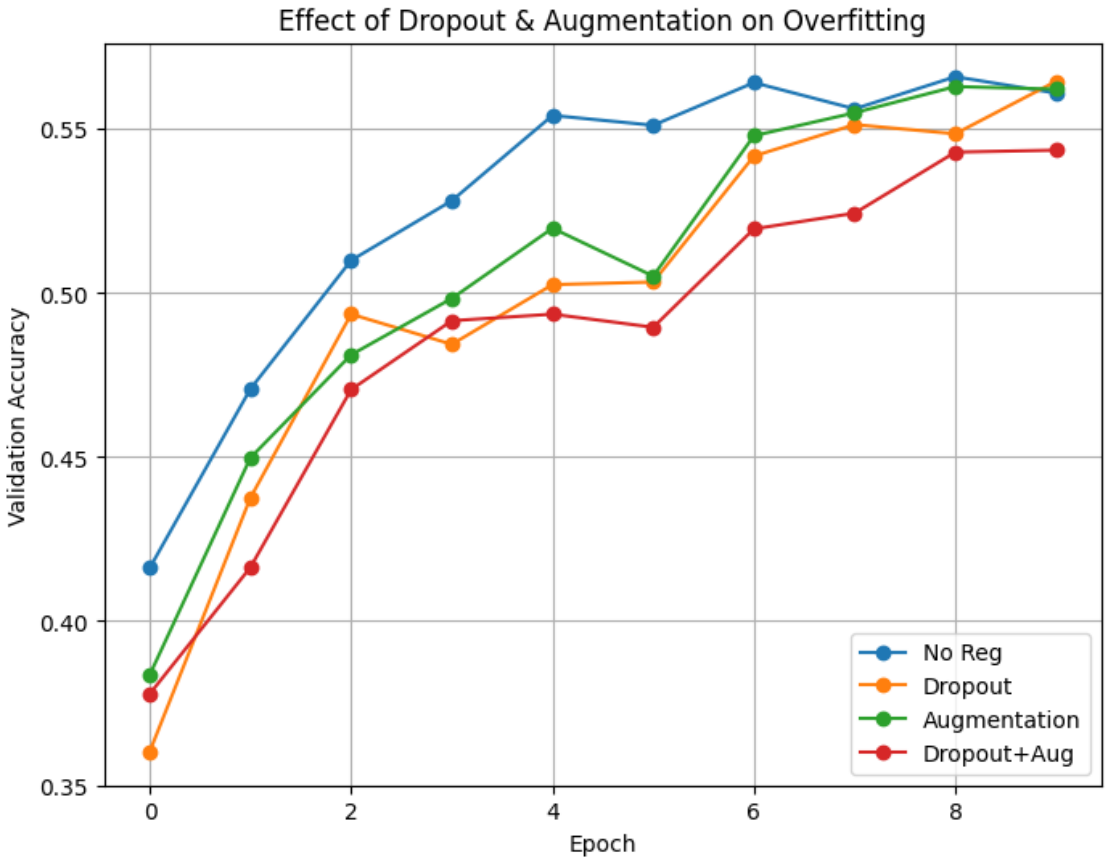
```
history_A = model_A.fit(X_train, y_train, epochs=EPOCHS, batch_size=BATCH_SIZE,␣
 ↪validation_split=0.2, verbose=1)

# ----------------------------
# 2 Model B: Dropout Only
# ----------------------------
model_B = build_model(True)
history_B = model_B.fit(X_train, y_train, epochs=EPOCHS, batch_size=BATCH_SIZE,␣
 ↪validation_split=0.2, verbose=1)

# ----------------------------
# 3 Data Augmentation
# ----------------------------
aug_gen = ImageDataGenerator(rotation_range=15, horizontal_flip=True,␣
 ↪validation_split=0.2)

# 3 Model C: Augmentation Only
model_C = build_model(False)
history_C = model_C.fit(
    aug_gen.flow(X_train, y_train, batch_size=BATCH_SIZE, subset='training'),
    epochs=EPOCHS,
    validation_data=aug_gen.flow(X_train, y_train, batch_size=BATCH_SIZE,␣
 ↪subset='validation'),
    verbose=1
)

# 4 Model D: Dropout + Augmentation
model_D = build_model(True)
history_D = model_D.fit(
    aug_gen.flow(X_train, y_train, batch_size=BATCH_SIZE, subset='training'),
    epochs=EPOCHS,
    validation_data=aug_gen.flow(X_train, y_train, batch_size=BATCH_SIZE,␣
 ↪subset='validation'),
    verbose=1
)

# ----------------------------
# Plot Validation Accuracy Comparison
# ----------------------------
plt.figure(figsize=(8,6))
plt.plot(history_A.history['val_accuracy'], marker='o')
plt.plot(history_B.history['val_accuracy'], marker='o')
plt.plot(history_C.history['val_accuracy'], marker='o')
plt.plot(history_D.history['val_accuracy'], marker='o')

plt.legend(["No Reg", "Dropout", "Augmentation", "Dropout+Aug"], loc='lower␣
 ↪right')
```

```
plt.xlabel("Epoch")
plt.ylabel("Validation Accuracy")
plt.title("Effect of Dropout & Augmentation on Overfitting")
plt.grid(True)
plt.savefig("overfitting_accuracy.png", dpi=300)
plt.show()
```

```
Epoch 1/10
157/157             10s 50ms/step -
accuracy: 0.2229 - loss: 2.1158 - val_accuracy: 0.4162 - val_loss: 1.6192
Epoch 2/10
157/157             10s 47ms/step -
accuracy: 0.4291 - loss: 1.6045 - val_accuracy: 0.4708 - val_loss: 1.4497
Epoch 3/10
157/157             14s 87ms/step -
accuracy: 0.4888 - loss: 1.4290 - val_accuracy: 0.5100 - val_loss: 1.3699
Epoch 4/10
157/157             13s 81ms/step -
accuracy: 0.5174 - loss: 1.3547 - val_accuracy: 0.5282 - val_loss: 1.3146
Epoch 5/10
157/157             14s 89ms/step -
accuracy: 0.5431 - loss: 1.2913 - val_accuracy: 0.5542 - val_loss: 1.2661
Epoch 6/10
157/157             19s 79ms/step -
accuracy: 0.5560 - loss: 1.2441 - val_accuracy: 0.5512 - val_loss: 1.2481
Epoch 7/10
157/157             9s 59ms/step -
accuracy: 0.5780 - loss: 1.2014 - val_accuracy: 0.5642 - val_loss: 1.2156
Epoch 8/10
157/157             7s 42ms/step -
accuracy: 0.5970 - loss: 1.1581 - val_accuracy: 0.5562 - val_loss: 1.2182
Epoch 9/10
157/157             8s 49ms/step -
accuracy: 0.6010 - loss: 1.1307 - val_accuracy: 0.5660 - val_loss: 1.2262
Epoch 10/10
157/157             10s 48ms/step -
accuracy: 0.6211 - loss: 1.0856 - val_accuracy: 0.5610 - val_loss: 1.2009
Epoch 1/10
157/157             8s 43ms/step -
accuracy: 0.1773 - loss: 2.1809 - val_accuracy: 0.3602 - val_loss: 1.7826
Epoch 2/10
157/157             8s 48ms/step -
accuracy: 0.3247 - loss: 1.8143 - val_accuracy: 0.4376 - val_loss: 1.5644
Epoch 3/10
157/157             9s 57ms/step -
accuracy: 0.3841 - loss: 1.6796 - val_accuracy: 0.4936 - val_loss: 1.4660
Epoch 4/10
```

```
157/157            8s 42ms/step -
accuracy: 0.4153 - loss: 1.6051 - val_accuracy: 0.4844 - val_loss: 1.4458
Epoch 5/10
157/157            10s 42ms/step -
accuracy: 0.4414 - loss: 1.5463 - val_accuracy: 0.5026 - val_loss: 1.4328
Epoch 6/10
157/157            8s 49ms/step -
accuracy: 0.4446 - loss: 1.4976 - val_accuracy: 0.5034 - val_loss: 1.4161
Epoch 7/10
157/157            7s 42ms/step -
accuracy: 0.4578 - loss: 1.4766 - val_accuracy: 0.5418 - val_loss: 1.2958
Epoch 8/10
157/157            10s 43ms/step -
accuracy: 0.4858 - loss: 1.4166 - val_accuracy: 0.5514 - val_loss: 1.2719
Epoch 9/10
157/157            8s 53ms/step -
accuracy: 0.5047 - loss: 1.3936 - val_accuracy: 0.5486 - val_loss: 1.2754
Epoch 10/10
157/157            7s 46ms/step -
accuracy: 0.5015 - loss: 1.3782 - val_accuracy: 0.5646 - val_loss: 1.2263
Epoch 1/10
157/157            23s 128ms/step -
accuracy: 0.2161 - loss: 2.1297 - val_accuracy: 0.3836 - val_loss: 1.7270
Epoch 2/10
157/157            19s 123ms/step -
accuracy: 0.3983 - loss: 1.6706 - val_accuracy: 0.4498 - val_loss: 1.5492
Epoch 3/10
157/157            21s 131ms/step -
accuracy: 0.4473 - loss: 1.5332 - val_accuracy: 0.4812 - val_loss: 1.4491
Epoch 4/10
157/157            20s 126ms/step -
accuracy: 0.4942 - loss: 1.4263 - val_accuracy: 0.4984 - val_loss: 1.4339
Epoch 5/10
157/157            19s 120ms/step -
accuracy: 0.4924 - loss: 1.4248 - val_accuracy: 0.5198 - val_loss: 1.3418
Epoch 6/10
157/157            20s 126ms/step -
accuracy: 0.5264 - loss: 1.3481 - val_accuracy: 0.5052 - val_loss: 1.3786
Epoch 7/10
157/157            19s 120ms/step -
accuracy: 0.5317 - loss: 1.3219 - val_accuracy: 0.5480 - val_loss: 1.3071
Epoch 8/10
157/157            20s 127ms/step -
accuracy: 0.5509 - loss: 1.2798 - val_accuracy: 0.5550 - val_loss: 1.2677
Epoch 9/10
157/157            19s 122ms/step -
accuracy: 0.5605 - loss: 1.2496 - val_accuracy: 0.5630 - val_loss: 1.2369
Epoch 10/10
```

```
157/157            20s 129ms/step -
accuracy: 0.5665 - loss: 1.2219 - val_accuracy: 0.5622 - val_loss: 1.2262
Epoch 1/10
157/157            21s 126ms/step -
accuracy: 0.1750 - loss: 2.1990 - val_accuracy: 0.3778 - val_loss: 1.8046
Epoch 2/10
157/157            19s 122ms/step -
accuracy: 0.3152 - loss: 1.8619 - val_accuracy: 0.4164 - val_loss: 1.6702
Epoch 3/10
157/157            21s 135ms/step -
accuracy: 0.3697 - loss: 1.7153 - val_accuracy: 0.4706 - val_loss: 1.5106
Epoch 4/10
157/157            19s 120ms/step -
accuracy: 0.4001 - loss: 1.6391 - val_accuracy: 0.4916 - val_loss: 1.4690
Epoch 5/10
157/157            20s 126ms/step -
accuracy: 0.4309 - loss: 1.5743 - val_accuracy: 0.4936 - val_loss: 1.4418
Epoch 6/10
157/157            19s 119ms/step -
accuracy: 0.4397 - loss: 1.5481 - val_accuracy: 0.4896 - val_loss: 1.4884
Epoch 7/10
157/157            20s 125ms/step -
accuracy: 0.4656 - loss: 1.4962 - val_accuracy: 0.5196 - val_loss: 1.3701
Epoch 8/10
157/157            19s 119ms/step -
accuracy: 0.4740 - loss: 1.4699 - val_accuracy: 0.5244 - val_loss: 1.3666
Epoch 9/10
157/157            20s 126ms/step -
accuracy: 0.4701 - loss: 1.4643 - val_accuracy: 0.5430 - val_loss: 1.3271
Epoch 10/10
157/157            19s 120ms/step -
accuracy: 0.4826 - loss: 1.4348 - val_accuracy: 0.5436 - val_loss: 1.2840
```



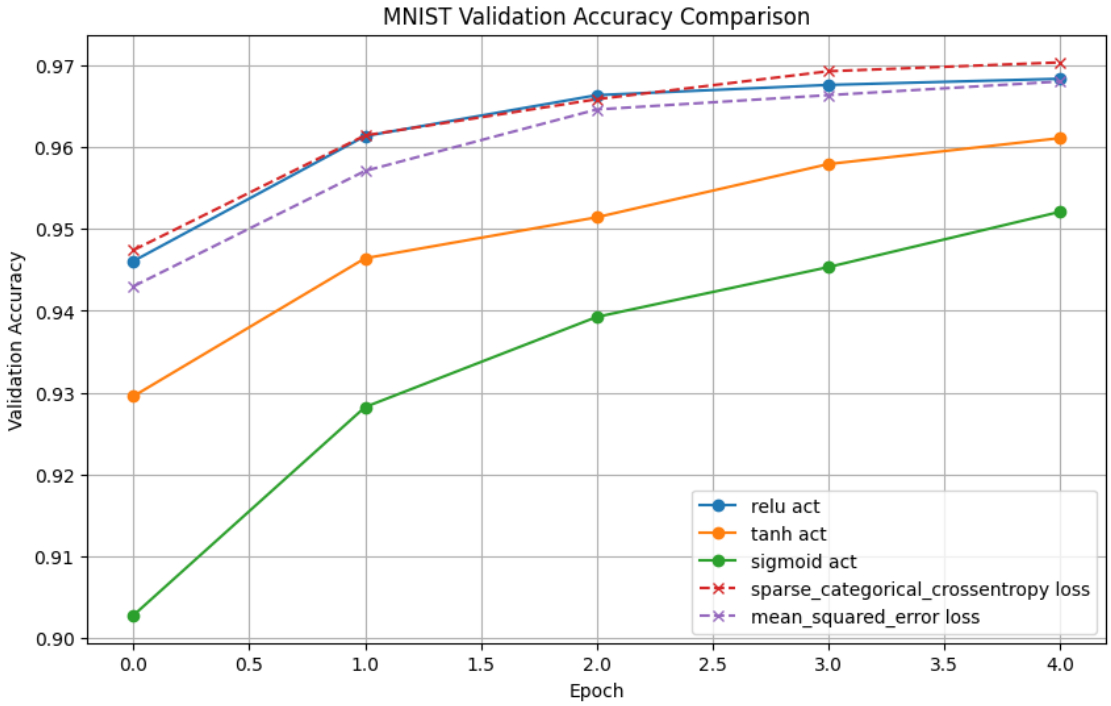
[ ]:
```
#  ASSIGNMENT 14

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import os

# ------------------------------
# Load MNIST
# ------------------------------
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0

# One-hot labels for MSE
y_train_onehot = to_categorical(y_train, num_classes=10)
y_test_onehot = to_categorical(y_test, num_classes=10)
```

```
# ----------------------------
# Experiment Settings
# ----------------------------
activations = ['relu', 'tanh', 'sigmoid']
loss_functions = ['sparse_categorical_crossentropy', 'mean_squared_error']
results = {}

os.makedirs("plots", exist_ok=True)
EPOCHS = 5
BATCH_SIZE = 128

# ----------------------------
# Model Builder
# ----------------------------
def build_model(activation='relu', loss_fn='sparse_categorical_crossentropy'):
    model = Sequential([
        Flatten(input_shape=(28,28)),
        Dense(128, activation=activation),
        Dropout(0.3),
        Dense(64, activation=activation),
        Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam', loss=loss_fn, metrics=['accuracy'])
    return model

# ----------------------------
# 1 Activation Functions
# ----------------------------
for act in activations:
    print(f"Training with activation: {act}")
    model = build_model(activation=act)
    history = model.fit(X_train, y_train,
                        epochs=EPOCHS,
                        batch_size=BATCH_SIZE,
                        validation_split=0.2,
                        verbose=0)

    results[f"{act}_activation"] = history.history

# ----------------------------
# 2 Loss Functions
# ----------------------------
for loss_fn in loss_functions:
    print(f"Training with loss function: {loss_fn}")
    model = build_model(loss_fn=loss_fn)

    if loss_fn == 'mean_squared_error':
```

```
        y_train_input = y_train_onehot
    else:
        y_train_input = y_train

    history = model.fit(X_train, y_train_input,
                        epochs=EPOCHS,
                        batch_size=BATCH_SIZE,
                        validation_split=0.2,
                        verbose=0)
    results[f"{loss_fn}_loss"] = history.history

# ----------------------------
# 3 Consolidated Plot
# ----------------------------
plt.figure(figsize=(10,6))

# Plot activations
for act in activations:
    plt.plot(results[f"{act}_activation"]['val_accuracy'], marker='o',␣
 ↪label=f"{act} act")

# Plot losses
for loss_fn in loss_functions:
    plt.plot(results[f"{loss_fn}_loss"]['val_accuracy'], marker='x',␣
 ↪linestyle='--', label=f"{loss_fn} loss")

plt.title("MNIST Validation Accuracy Comparison")
plt.xlabel("Epoch")
plt.ylabel("Validation Accuracy")
plt.legend()
plt.grid(True)
plt.savefig("plots/mnist_activation_loss_comparison.png", dpi=300)
plt.show()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11490434/11490434            0s
0us/step
Training with activation: relu

/usr/local/lib/python3.12/dist-
packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)

Training with activation: tanh
Training with activation: sigmoid
```

```
Training with loss function: sparse_categorical_crossentropy
Training with loss function: mean_squared_error
```



[ ]:
```python
# ASSIGNMENT 15

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.datasets import mnist
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint,␣
 ↪ReduceLROnPlateau, CSVLogger
import os

# ------------------------------
# Load MNIST dataset
# ------------------------------
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0

# ------------------------------
# Build FCFNN model
# ------------------------------
def build_model():
    model = Sequential([
```

```
        Flatten(input_shape=(28,28)),
        Dense(128, activation='relu'),
        Dropout(0.3),
        Dense(64, activation='relu'),
        Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
    return model

model = build_model()

# ------------------------------
# Setup Callbacks
# ------------------------------
os.makedirs("callbacks_models", exist_ok=True)

callbacks = [
    EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True,␣
 ↪verbose=1),
    ModelCheckpoint(filepath='callbacks_models/best_model.h5',␣
 ↪monitor='val_accuracy', save_best_only=True, verbose=1),
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2, verbose=1),
    CSVLogger('callbacks_models/training_log.csv', append=False)
]

# ------------------------------
# Train the model
# ------------------------------
history = model.fit(
    X_train, y_train,
    validation_split=0.2,
    epochs=20,
    batch_size=128,
    callbacks=callbacks,
    verbose=2
)

# ------------------------------
# Evaluate the model
# ------------------------------
loss, acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {acc*100:.2f}%")
```

```
Epoch 1/20

Epoch 1: val_accuracy improved from -inf to 0.94742, saving model to
```

```
callbacks_models/best_model.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

375/375 - 7s - 20ms/step - accuracy: 0.8626 - loss: 0.4614 - val_accuracy:
0.9474 - val_loss: 0.1809 - learning_rate: 1.0000e-03
Epoch 2/20

Epoch 2: val_accuracy improved from 0.94742 to 0.96167, saving model to
callbacks_models/best_model.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

375/375 - 6s - 15ms/step - accuracy: 0.9386 - loss: 0.2055 - val_accuracy:
0.9617 - val_loss: 0.1302 - learning_rate: 1.0000e-03
Epoch 3/20

Epoch 3: val_accuracy improved from 0.96167 to 0.96767, saving model to
callbacks_models/best_model.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

375/375 - 5s - 14ms/step - accuracy: 0.9529 - loss: 0.1549 - val_accuracy:
0.9677 - val_loss: 0.1140 - learning_rate: 1.0000e-03
Epoch 4/20

Epoch 4: val_accuracy improved from 0.96767 to 0.96992, saving model to
callbacks_models/best_model.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

375/375 - 7s - 19ms/step - accuracy: 0.9601 - loss: 0.1303 - val_accuracy:
0.9699 - val_loss: 0.1038 - learning_rate: 1.0000e-03
Epoch 5/20
```

```
Epoch 5: val_accuracy improved from 0.96992 to 0.97242, saving model to
callbacks_models/best_model.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

375/375 - 5s - 13ms/step - accuracy: 0.9650 - loss: 0.1152 - val_accuracy:
0.9724 - val_loss: 0.0953 - learning_rate: 1.0000e-03
Epoch 6/20

Epoch 6: val_accuracy improved from 0.97242 to 0.97367, saving model to
callbacks_models/best_model.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

375/375 - 6s - 16ms/step - accuracy: 0.9689 - loss: 0.1017 - val_accuracy:
0.9737 - val_loss: 0.0895 - learning_rate: 1.0000e-03
Epoch 7/20

Epoch 7: val_accuracy did not improve from 0.97367
375/375 - 5s - 13ms/step - accuracy: 0.9714 - loss: 0.0913 - val_accuracy:
0.9727 - val_loss: 0.0893 - learning_rate: 1.0000e-03
Epoch 8/20

Epoch 8: val_accuracy improved from 0.97367 to 0.97417, saving model to
callbacks_models/best_model.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

375/375 - 3s - 8ms/step - accuracy: 0.9741 - loss: 0.0819 - val_accuracy: 0.9742
- val_loss: 0.0892 - learning_rate: 1.0000e-03
Epoch 9/20

Epoch 9: val_accuracy improved from 0.97417 to 0.97700, saving model to
callbacks_models/best_model.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
```

```
'my_model.keras')`.

375/375 - 3s - 7ms/step - accuracy: 0.9755 - loss: 0.0769 - val_accuracy: 0.9770
- val_loss: 0.0813 - learning_rate: 1.0000e-03
Epoch 10/20

Epoch 10: val_accuracy did not improve from 0.97700
375/375 - 3s - 9ms/step - accuracy: 0.9774 - loss: 0.0709 - val_accuracy: 0.9756
- val_loss: 0.0836 - learning_rate: 1.0000e-03
Epoch 11/20

Epoch 11: val_accuracy did not improve from 0.97700

Epoch 11: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
375/375 - 2s - 6ms/step - accuracy: 0.9781 - loss: 0.0683 - val_accuracy: 0.9768
- val_loss: 0.0834 - learning_rate: 1.0000e-03
Epoch 12/20

Epoch 12: val_accuracy improved from 0.97700 to 0.97775, saving model to
callbacks_models/best_model.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

375/375 - 2s - 6ms/step - accuracy: 0.9828 - loss: 0.0540 - val_accuracy: 0.9778
- val_loss: 0.0802 - learning_rate: 5.0000e-04
Epoch 13/20

Epoch 13: val_accuracy improved from 0.97775 to 0.97883, saving model to
callbacks_models/best_model.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

375/375 - 2s - 6ms/step - accuracy: 0.9835 - loss: 0.0503 - val_accuracy: 0.9788
- val_loss: 0.0773 - learning_rate: 5.0000e-04
Epoch 14/20

Epoch 14: val_accuracy did not improve from 0.97883
375/375 - 3s - 7ms/step - accuracy: 0.9839 - loss: 0.0489 - val_accuracy: 0.9787
- val_loss: 0.0779 - learning_rate: 5.0000e-04
Epoch 15/20

Epoch 15: val_accuracy did not improve from 0.97883
```

```
Epoch 15: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
375/375 - 3s - 8ms/step - accuracy: 0.9856 - loss: 0.0438 - val_accuracy: 0.9787
- val_loss: 0.0797 - learning_rate: 5.0000e-04
Epoch 16/20

Epoch 16: val_accuracy did not improve from 0.97883
375/375 - 2s - 6ms/step - accuracy: 0.9866 - loss: 0.0417 - val_accuracy: 0.9788
- val_loss: 0.0770 - learning_rate: 2.5000e-04
Epoch 17/20

Epoch 17: val_accuracy improved from 0.97883 to 0.97933, saving model to
callbacks_models/best_model.h5

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

375/375 - 2s - 6ms/step - accuracy: 0.9875 - loss: 0.0390 - val_accuracy: 0.9793
- val_loss: 0.0774 - learning_rate: 2.5000e-04
Epoch 18/20

Epoch 18: val_accuracy did not improve from 0.97933

Epoch 18: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
375/375 - 2s - 6ms/step - accuracy: 0.9871 - loss: 0.0385 - val_accuracy: 0.9785
- val_loss: 0.0782 - learning_rate: 2.5000e-04
Epoch 19/20

Epoch 19: val_accuracy did not improve from 0.97933
375/375 - 3s - 7ms/step - accuracy: 0.9894 - loss: 0.0340 - val_accuracy: 0.9793
- val_loss: 0.0777 - learning_rate: 1.2500e-04
Epoch 19: early stopping
Restoring model weights from the end of the best epoch: 16.
313/313            1s 4ms/step -
accuracy: 0.9762 - loss: 0.0815
Test Accuracy: 98.02%
```

[ ]:
```python
# ASSIGNMENT 16

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
import os
```

```
# ----------------------------
# Load MNIST Dataset
# ----------------------------
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Normalize
X_train = X_train / 255.0
X_test = X_test / 255.0

# ----------------------------
# Build FCFNN Model
# ----------------------------
def build_model():
    model = Sequential([
        Flatten(input_shape=(28,28)),
        Dense(128, activation='relu'),
        Dropout(0.3),
        Dense(64, activation='relu'),
        Dense(10, activation='softmax')
    ])

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

    return model

model = build_model()

# Show model architecture
model.summary()

# ----------------------------
# Train Model
# ----------------------------
os.makedirs("plots", exist_ok=True)

history = model.fit(
    X_train, y_train,
    validation_split=0.2,
    epochs=15,
    batch_size=128,
    verbose=2
)
```

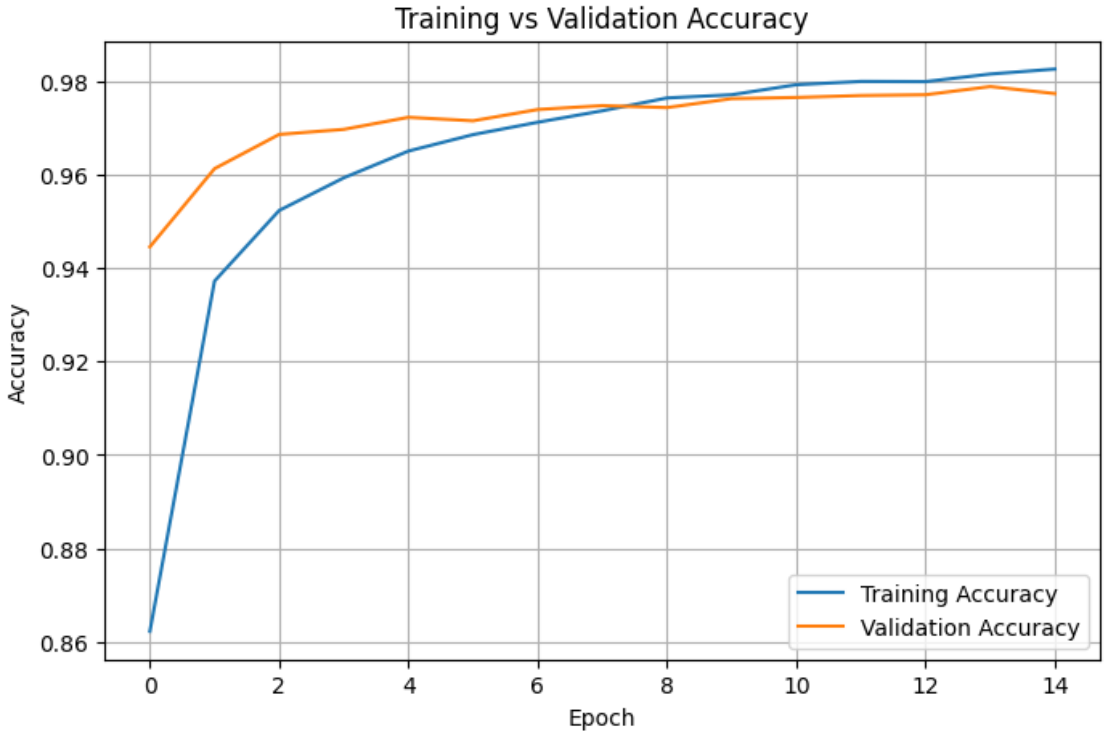
```python
# ----------------------------
# Plot Accuracy
# ----------------------------
plt.figure(figsize=(8,5))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training vs Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.savefig('plots/accuracy_curve.png', dpi=300)
plt.show()

# ----------------------------
# Plot Loss
# ----------------------------
plt.figure(figsize=(8,5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training vs Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.savefig('plots/loss_curve.png', dpi=300)
plt.show()

# ----------------------------
# Evaluate Model
# ----------------------------
loss, acc = model.evaluate(X_test, y_test, verbose=0)
print(f"\nTest Accuracy: {acc*100:.2f}%")
```

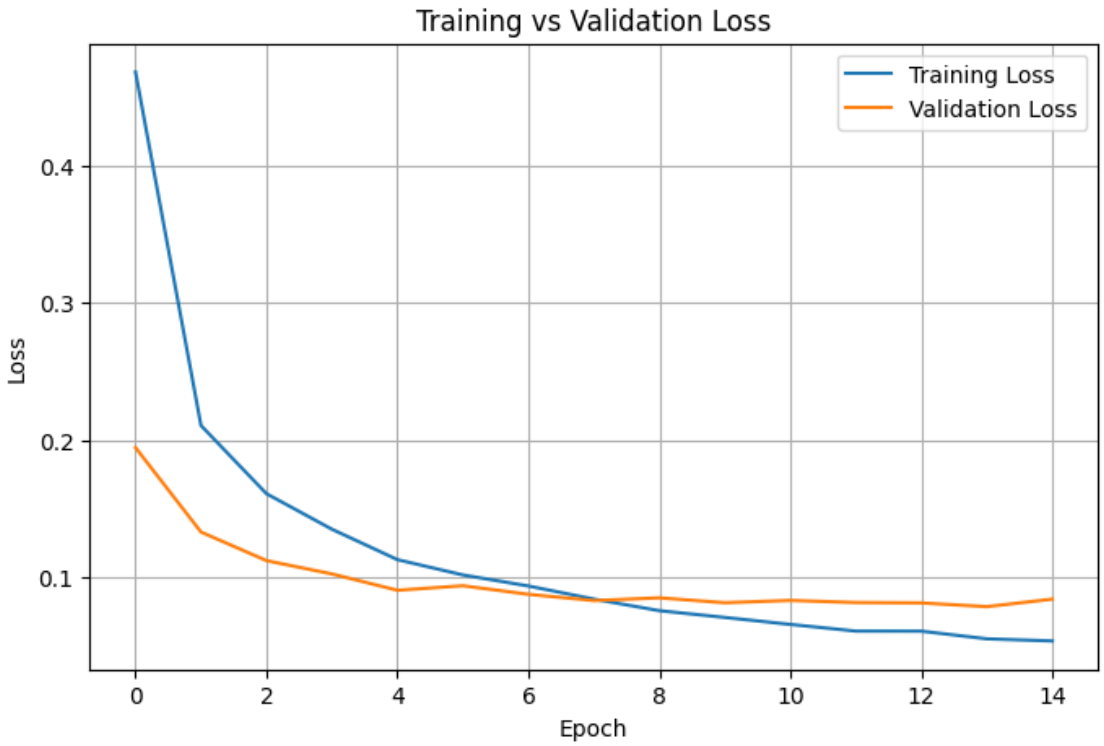
Model: "sequential_15"

Layer (type)	Output Shape	Param #
flatten_15 (Flatten)	(None, 784)	0
dense_37 (Dense)	(None, 128)	100,480
dropout_9 (Dropout)	(None, 128)	0
dense_38 (Dense)	(None, 64)	8,256

```
 dense_39 (Dense)                (None, 10)                     650


 Total params: 109,386 (427.29 KB)

 Trainable params: 109,386 (427.29 KB)

 Non-trainable params: 0 (0.00 B)

Epoch 1/15
375/375 - 4s - 9ms/step - accuracy: 0.8622 - loss: 0.4678 - val_accuracy: 0.9446
- val_loss: 0.1944
Epoch 2/15
375/375 - 2s - 6ms/step - accuracy: 0.9373 - loss: 0.2105 - val_accuracy: 0.9613
- val_loss: 0.1330
Epoch 3/15
375/375 - 2s - 6ms/step - accuracy: 0.9524 - loss: 0.1609 - val_accuracy: 0.9687
- val_loss: 0.1122
Epoch 4/15
375/375 - 3s - 8ms/step - accuracy: 0.9594 - loss: 0.1350 - val_accuracy: 0.9697
- val_loss: 0.1025
Epoch 5/15
375/375 - 4s - 12ms/step - accuracy: 0.9651 - loss: 0.1129 - val_accuracy:
0.9723 - val_loss: 0.0906
Epoch 6/15
375/375 - 2s - 6ms/step - accuracy: 0.9686 - loss: 0.1017 - val_accuracy: 0.9716
- val_loss: 0.0939
Epoch 7/15
375/375 - 2s - 6ms/step - accuracy: 0.9713 - loss: 0.0937 - val_accuracy: 0.9740
- val_loss: 0.0876
Epoch 8/15
375/375 - 3s - 8ms/step - accuracy: 0.9737 - loss: 0.0842 - val_accuracy: 0.9748
- val_loss: 0.0831
Epoch 9/15
375/375 - 5s - 12ms/step - accuracy: 0.9765 - loss: 0.0757 - val_accuracy:
0.9744 - val_loss: 0.0851
Epoch 10/15
375/375 - 2s - 6ms/step - accuracy: 0.9772 - loss: 0.0708 - val_accuracy: 0.9763
- val_loss: 0.0815
Epoch 11/15
375/375 - 2s - 6ms/step - accuracy: 0.9793 - loss: 0.0657 - val_accuracy: 0.9766
- val_loss: 0.0832
Epoch 12/15
375/375 - 3s - 7ms/step - accuracy: 0.9800 - loss: 0.0609 - val_accuracy: 0.9770
- val_loss: 0.0817
```

```
Epoch 13/15
375/375 - 3s - 8ms/step - accuracy: 0.9800 - loss: 0.0608 - val_accuracy: 0.9772
- val_loss: 0.0814
Epoch 14/15
375/375 - 2s - 7ms/step - accuracy: 0.9816 - loss: 0.0552 - val_accuracy: 0.9789
- val_loss: 0.0787
Epoch 15/15
375/375 - 3s - 8ms/step - accuracy: 0.9827 - loss: 0.0537 - val_accuracy: 0.9774
- val_loss: 0.0841
```






```
Test Accuracy: 97.66%
```