

# AI\_Lab\_Final\_Assignment

February 23, 2026

```
[ ]: # 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.
```

```
[ ]: # 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
```

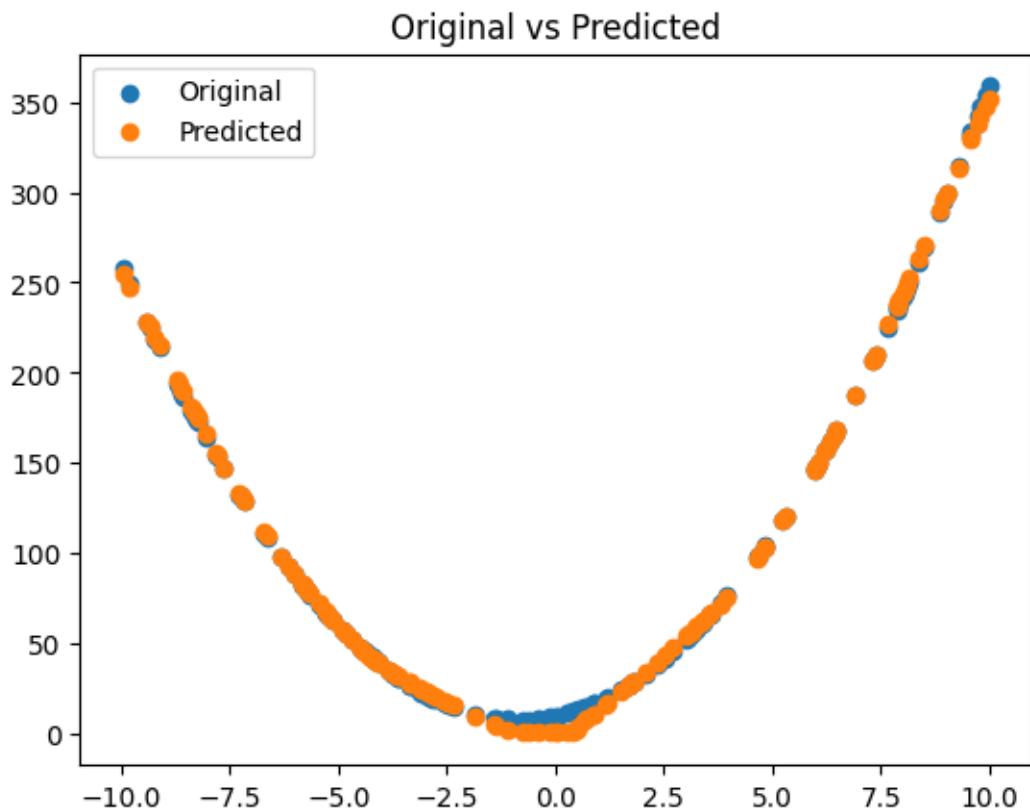
```
Epoch 124/200
22/22          0s 5ms/step - loss:
61.1258 - val_loss: 68.0253
Epoch 125/200
22/22          0s 5ms/step - loss:
55.4299 - val_loss: 64.5362
Epoch 126/200
22/22          0s 5ms/step - loss:
55.7441 - val_loss: 62.9869
Epoch 127/200
22/22          0s 5ms/step - loss:
62.2082 - val_loss: 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

[ ]: #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())

```

```

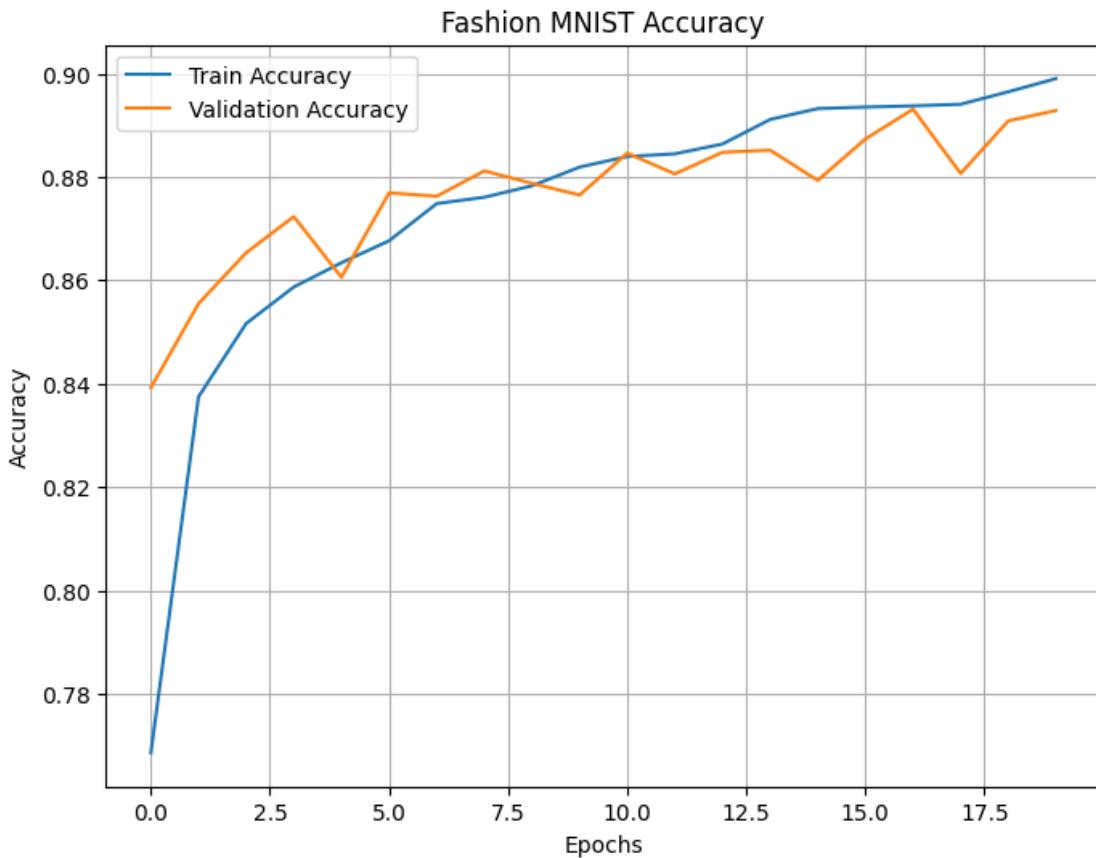
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-labels-idx1-ubyte.gz
29515/29515          0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-images-idx3-ubyte.gz
26421880/26421880      0s
0us/step
Downloading data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



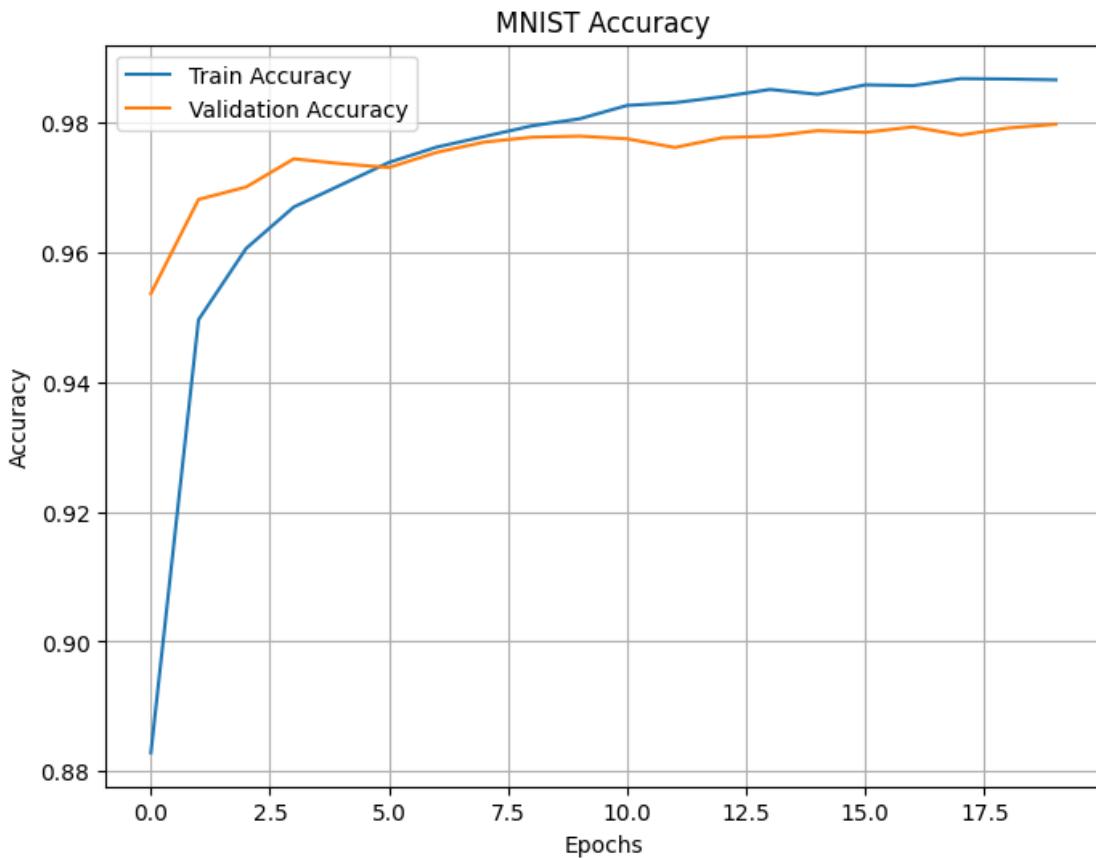
Plot saved to plots/Fashion\_MNIST\_accuracy.png

```
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%

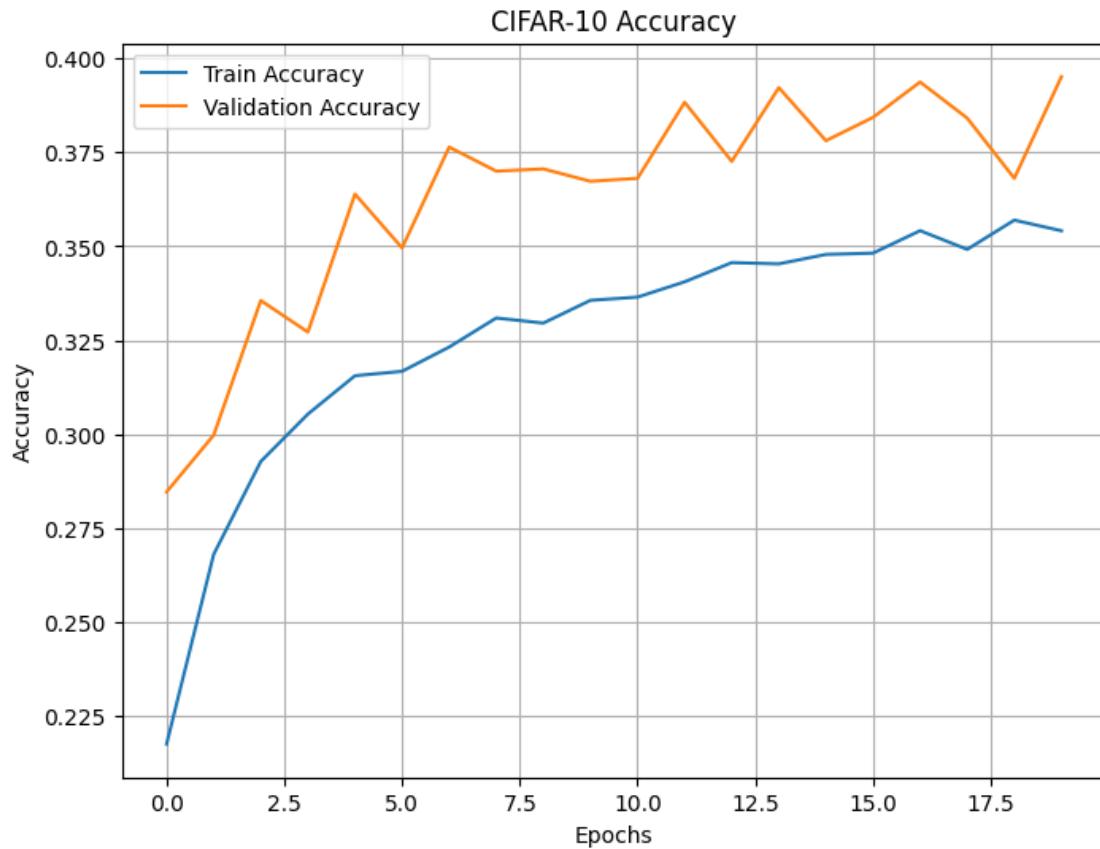
```



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'),
        Flatten(),
        Dense(128, activation='relu'),
        Dense(10, activation='softmax')
    ])
    return model
```

```

        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)

```

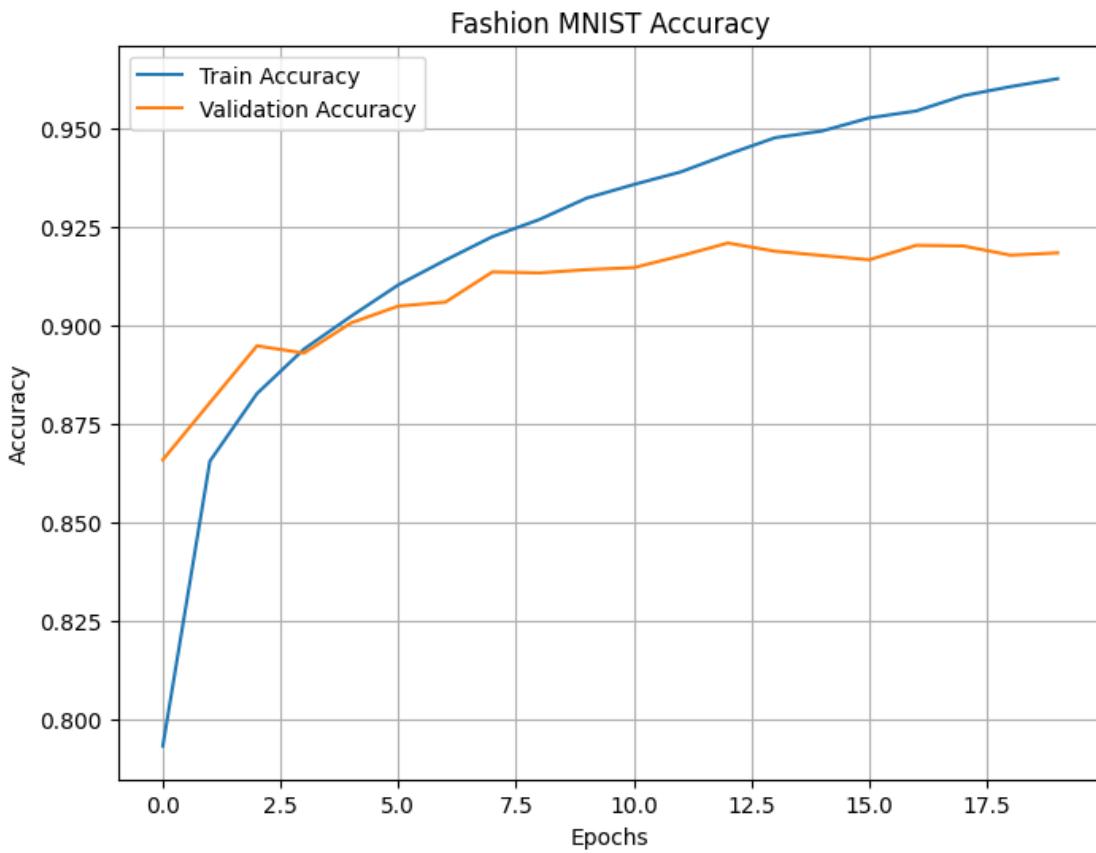
```

/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
```

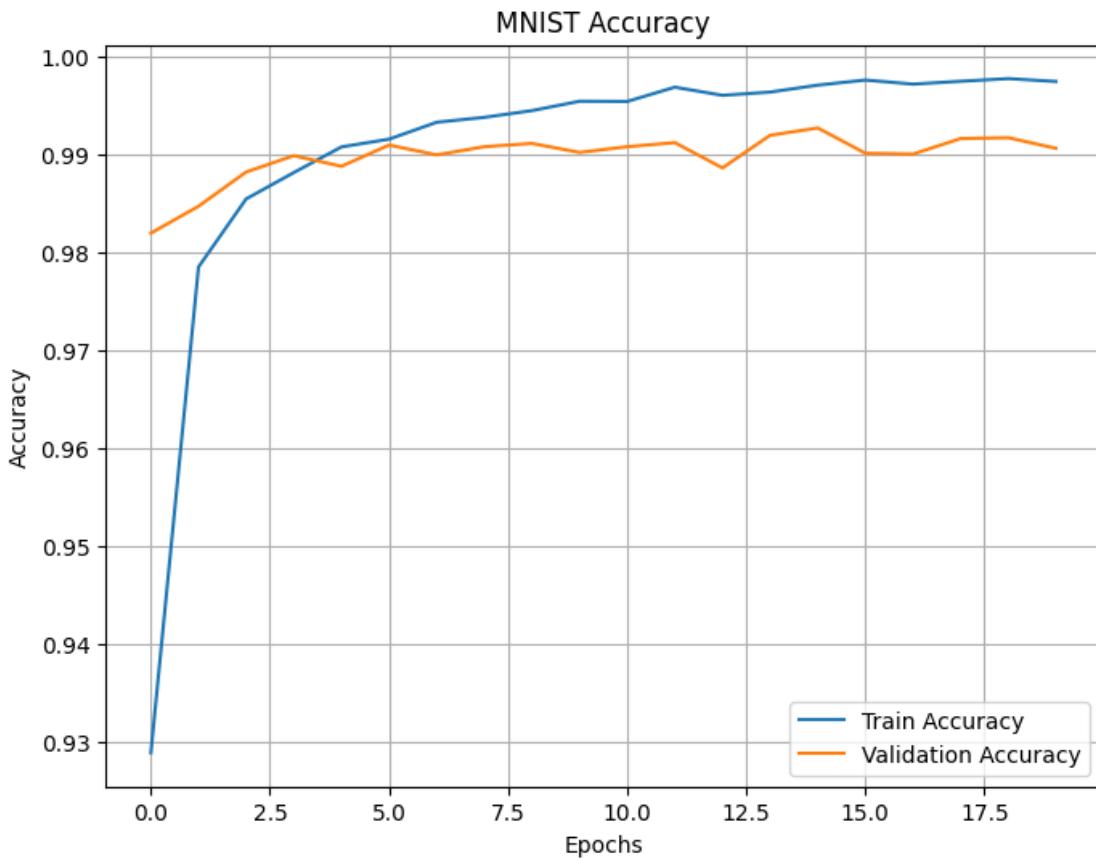
```
Epoch 5/20
```

```
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

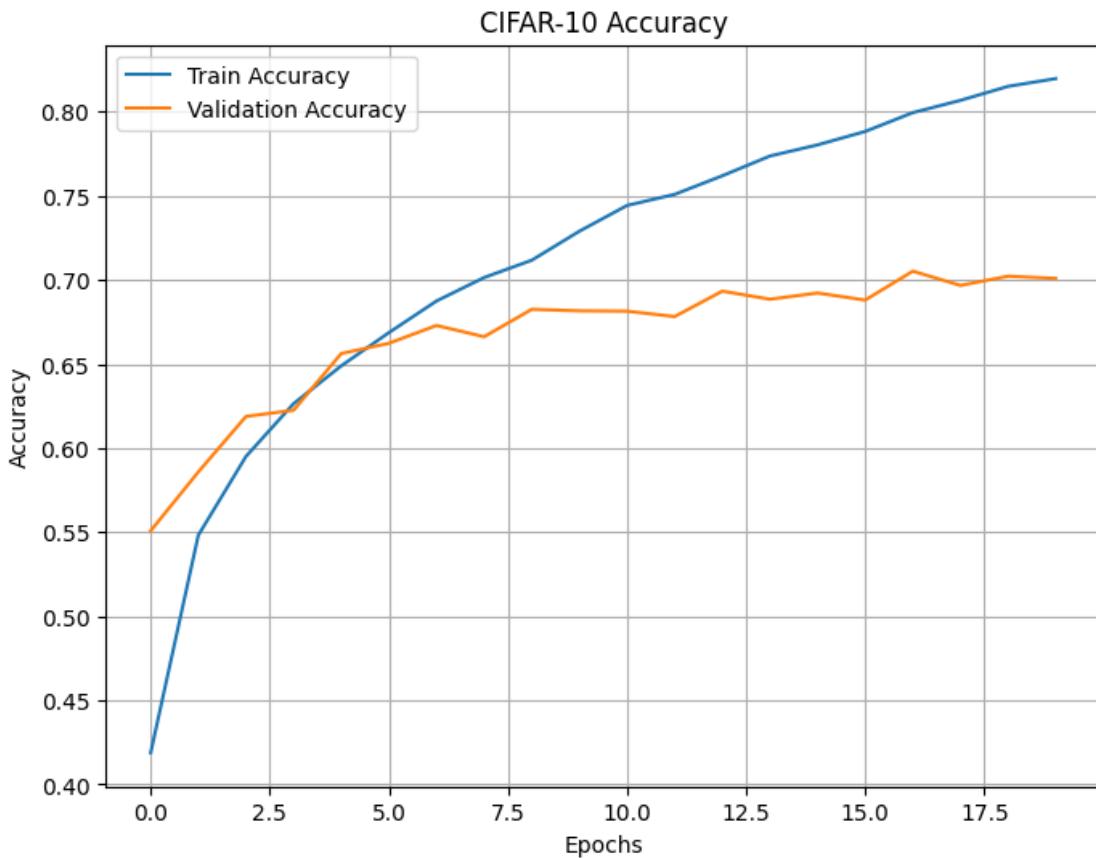
Epoch 5/20

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

[ ]: #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()

```

```

# -----
# 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

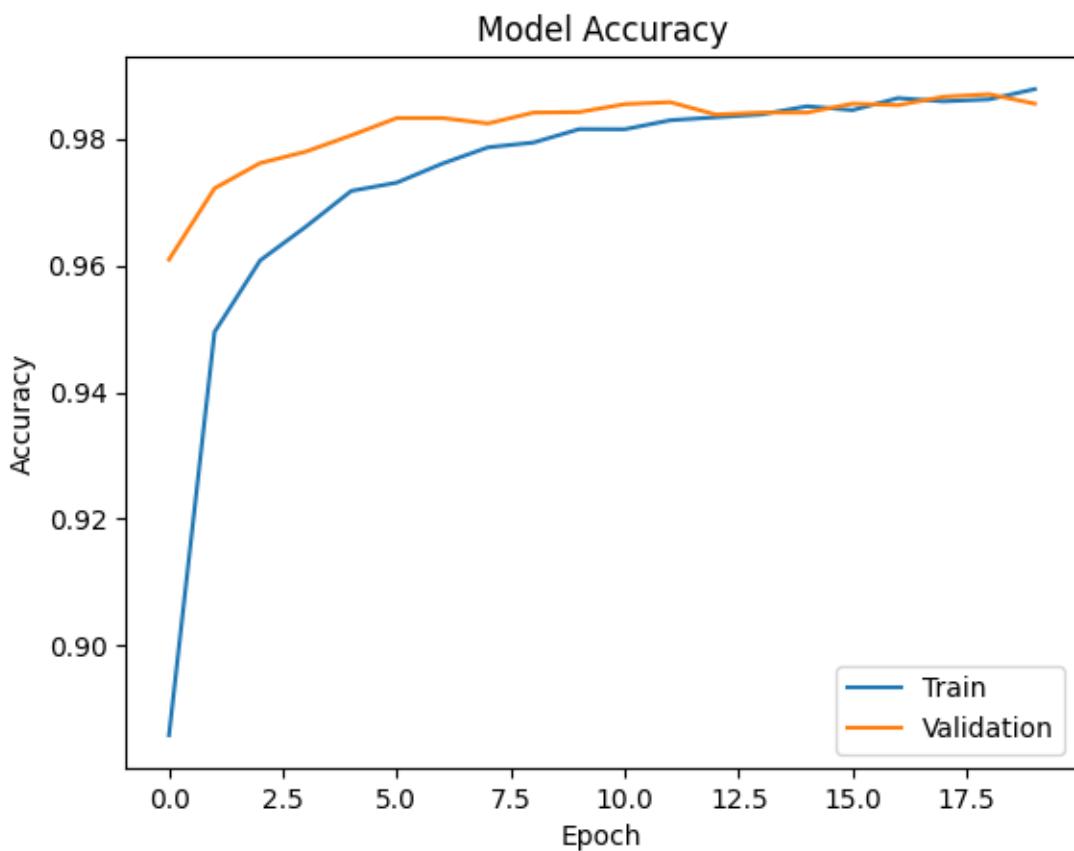
```
800/800           8s 9ms/step -
accuracy: 0.9844 - loss: 0.0501 - val_accuracy: 0.9838 - val_loss: 0.0578
Epoch 14/20
800/800           8s 9ms/step -
accuracy: 0.9843 - loss: 0.0488 - val_accuracy: 0.9841 - val_loss: 0.0576
Epoch 15/20
800/800           7s 8ms/step -
accuracy: 0.9852 - loss: 0.0474 - 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
```

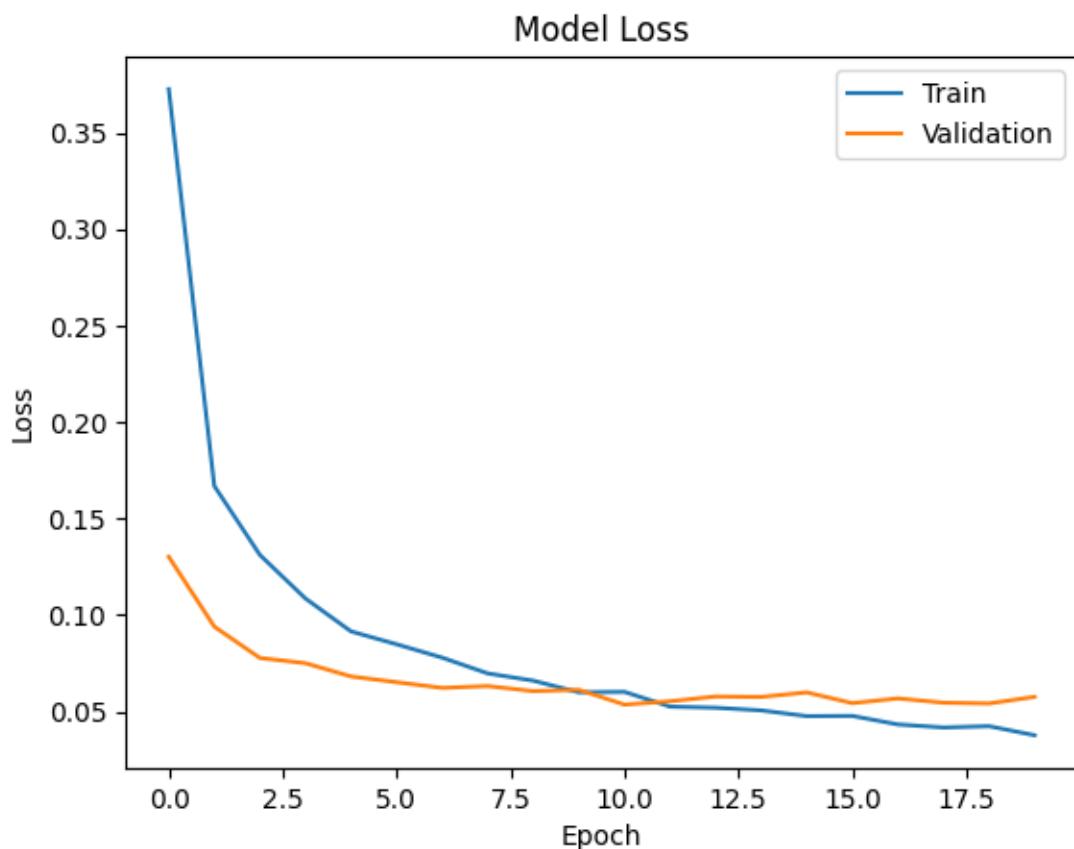
Accuracy on Custom Test Dataset: 99.60%

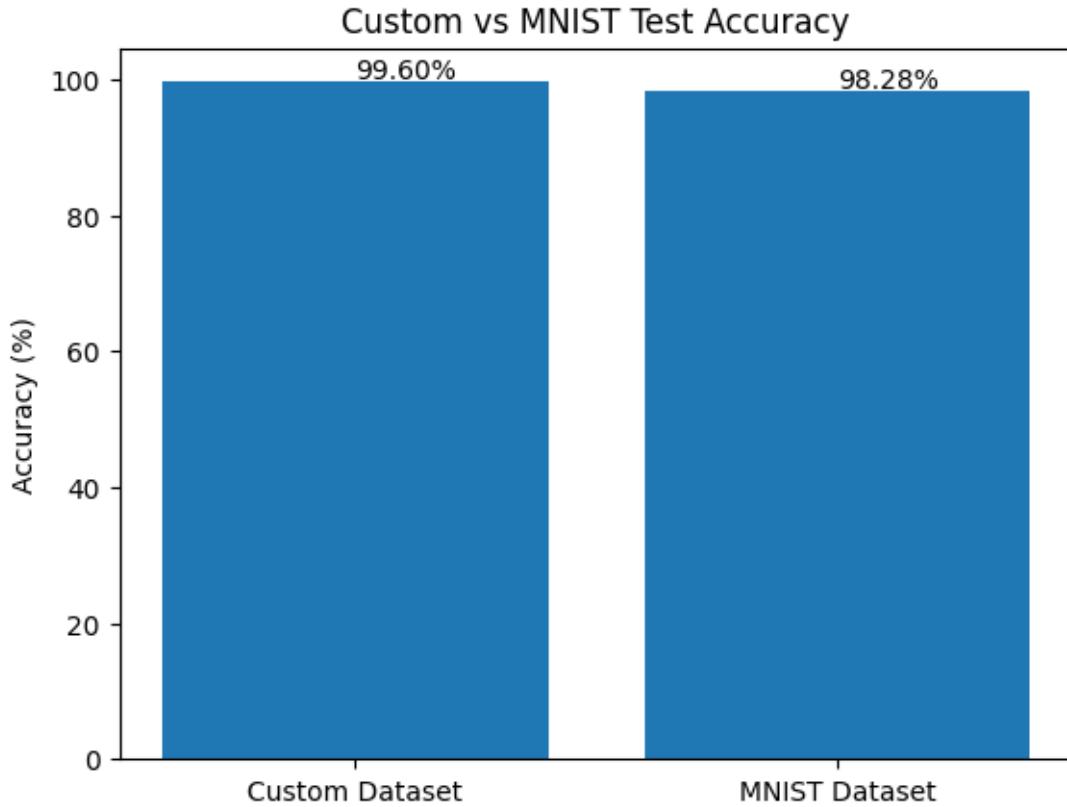
313/313 1s 3ms/step -

accuracy: 0.9783 - loss: 0.0904

Accuracy on MNIST Test Dataset: 98.28%







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("modelsizes_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)

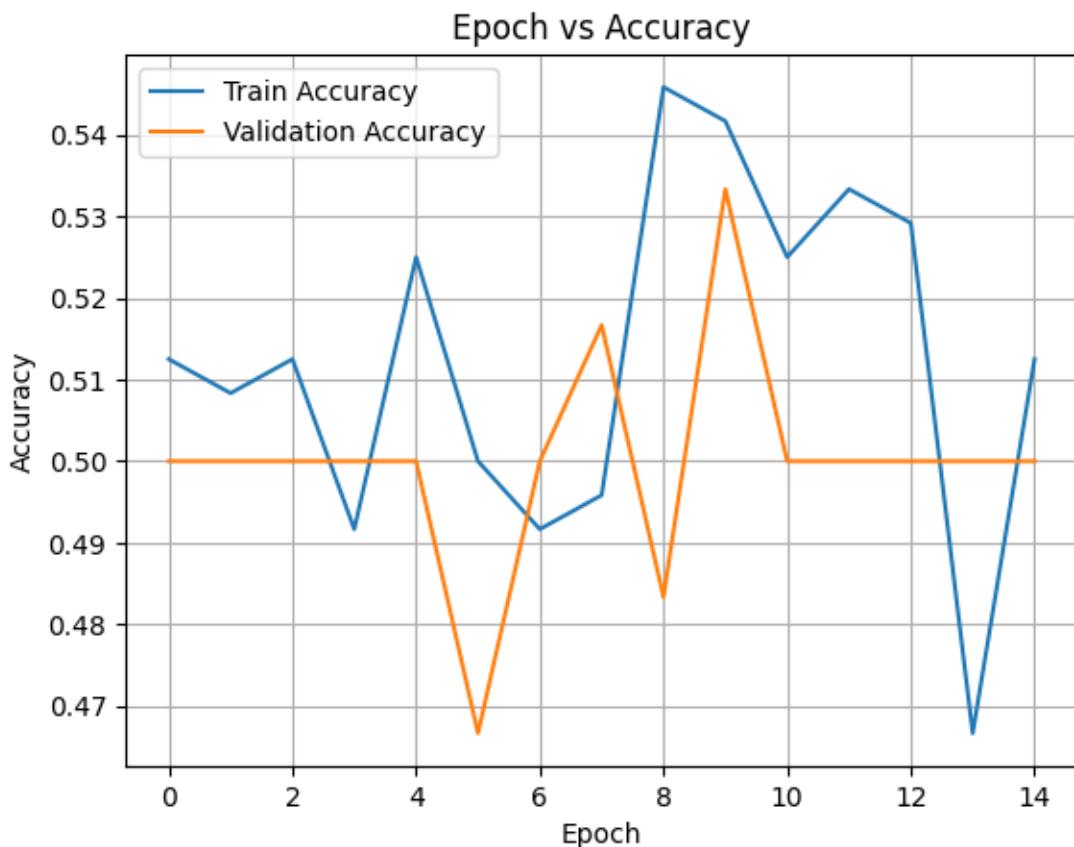
Non-trainable params: 0 (0.00 B)

Epoch 1/15

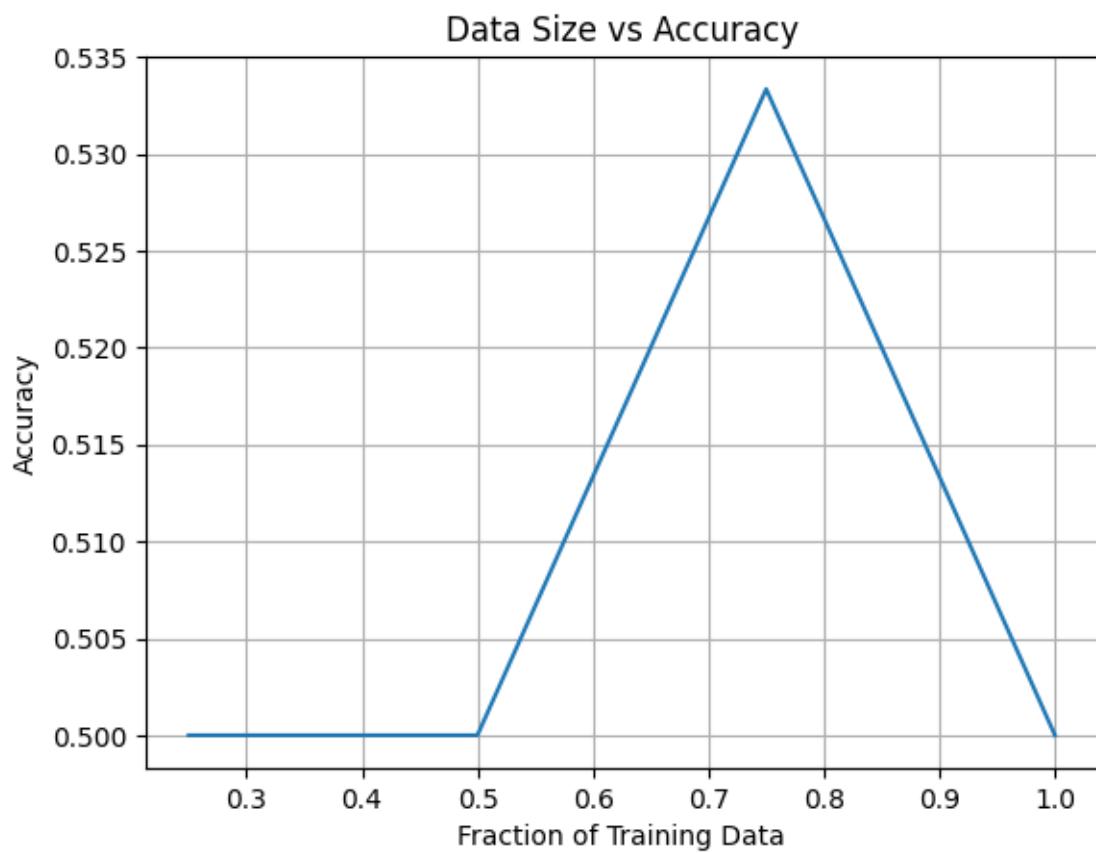
```
/usr/local/lib/python3.12/dist-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
    self._warn_if_super_not_called()

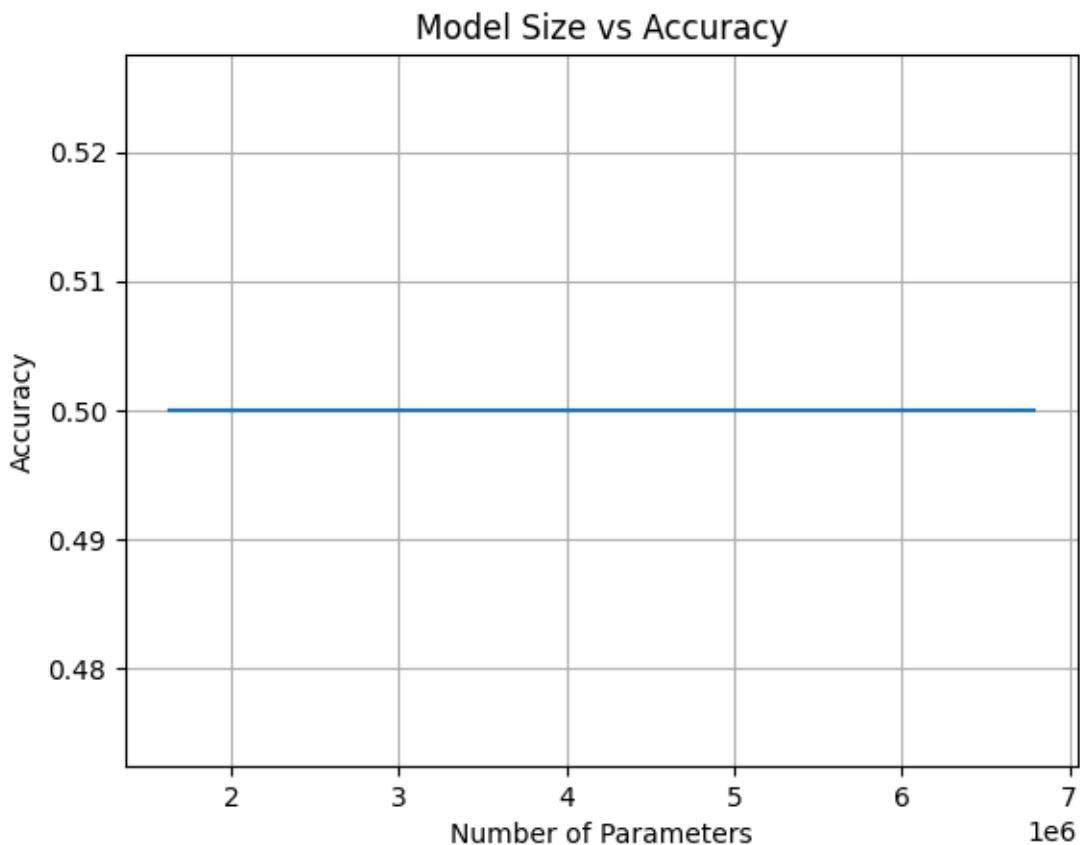
8/8          12s 1s/step -
accuracy: 0.5027 - loss: 0.8727 - val_accuracy: 0.5000 - val_loss: 0.6951
Epoch 2/15
8/8          9s 1s/step -
accuracy: 0.5085 - loss: 0.6974 - val_accuracy: 0.5000 - val_loss: 0.6932
Epoch 3/15
8/8          11s 1s/step -
accuracy: 0.5552 - loss: 0.6890 - val_accuracy: 0.5000 - 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.





Assignment 7 Completed Successfully!

```
[ ]: #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__)
120
121     if filtered_tb is not None:
122         raise e.with_traceback(filtered_tb)
123     else:
124         raise e
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
877
878
879
879
880
881
882
883
884
885
886
887
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2559
2560
2561
2562
2563
2564
2565
2566
2567
25
```



```

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)

# -----

```

```

# 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!")

```

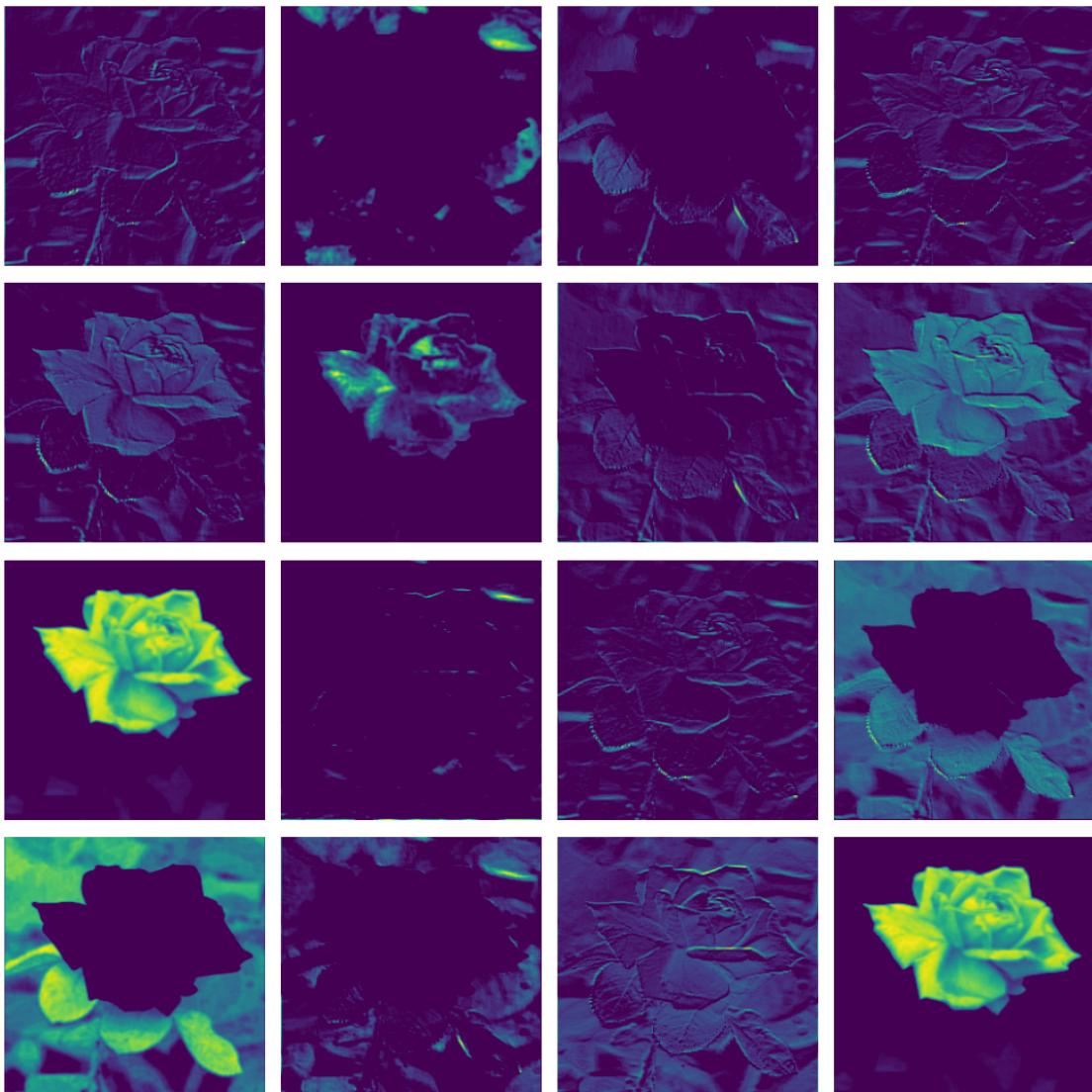
Upload your image (jpg/png)

<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](https://www.tensorflow.org/guide/function#controlling_retracing) and  
[https://www.tensorflow.org/api\\_docs/python/tf/function](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



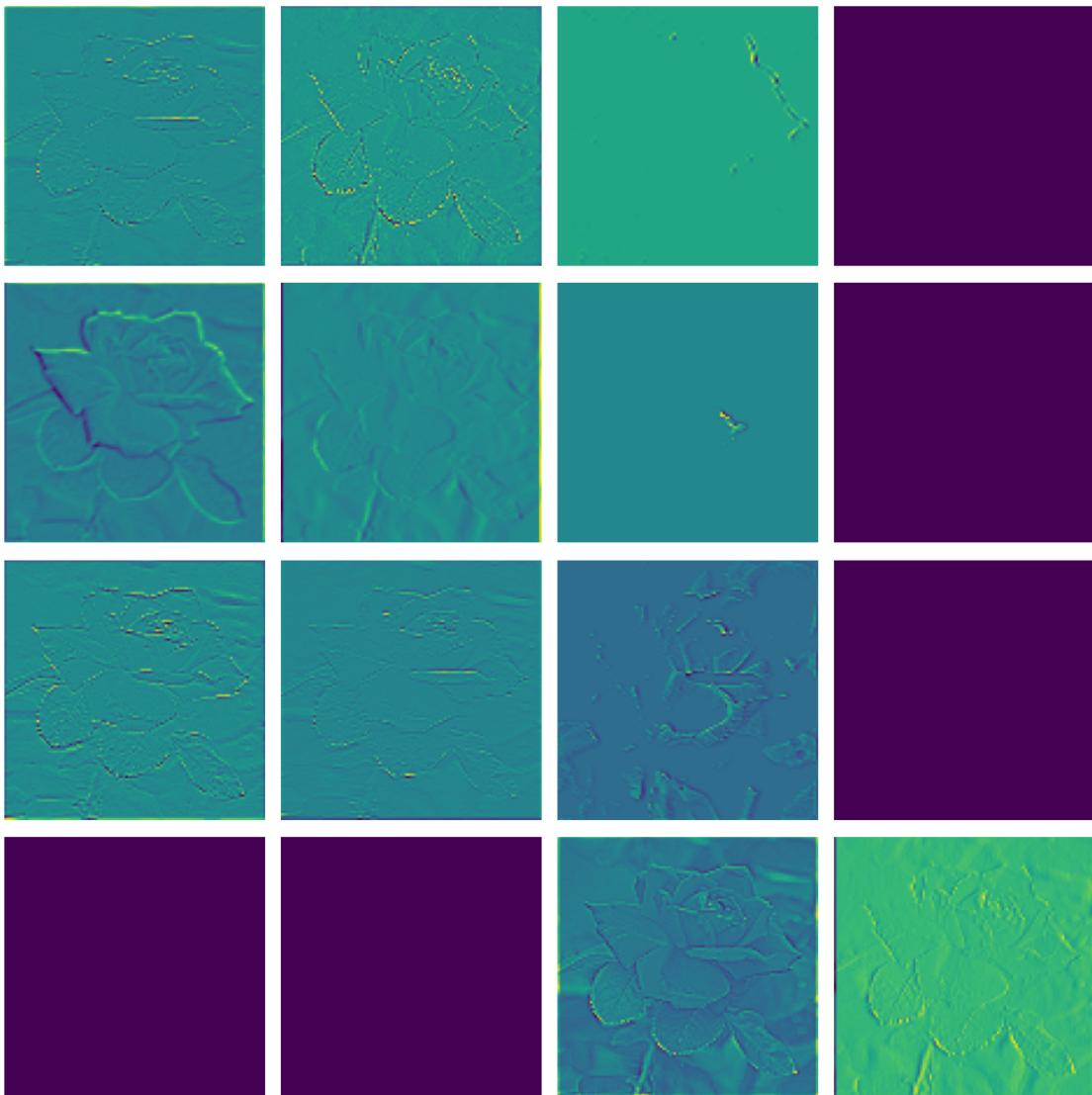
Saved: plots/resnet\_feature\_maps.png

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}%")

```

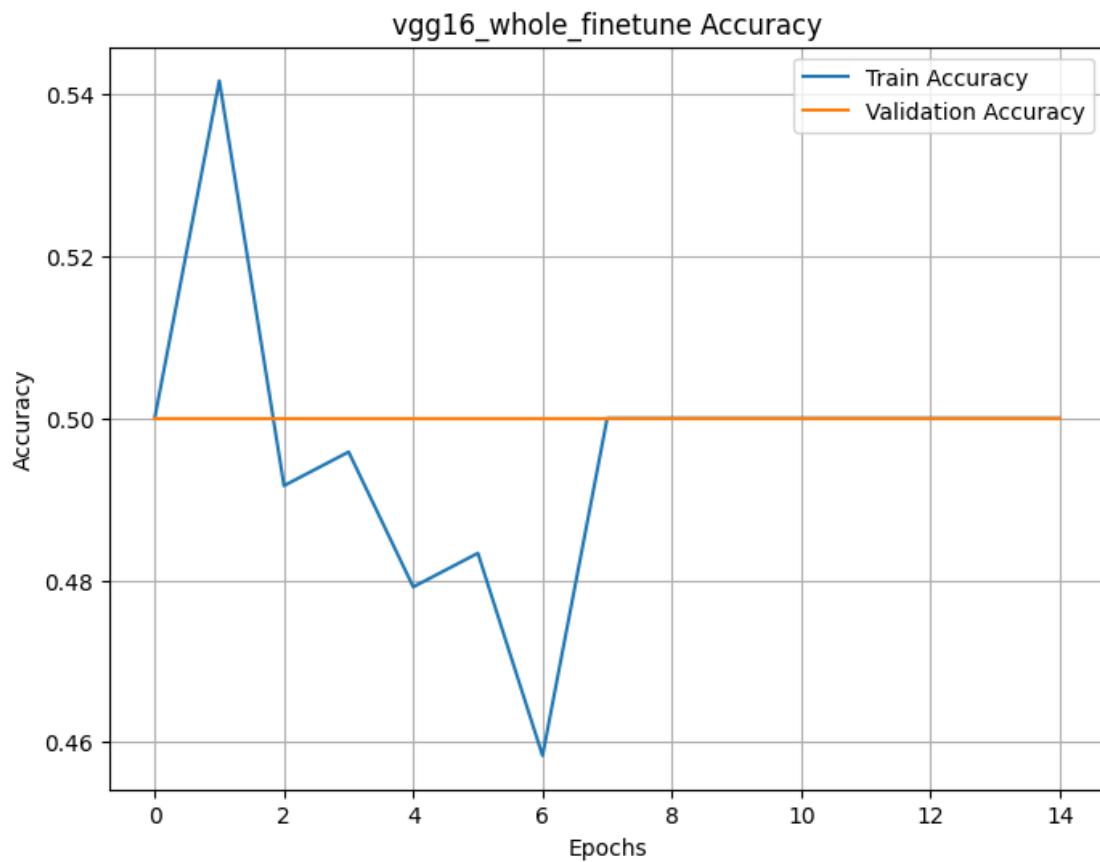
Found 240 images belonging to 2 classes.  
 Found 60 images belonging to 2 classes.

==== 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%

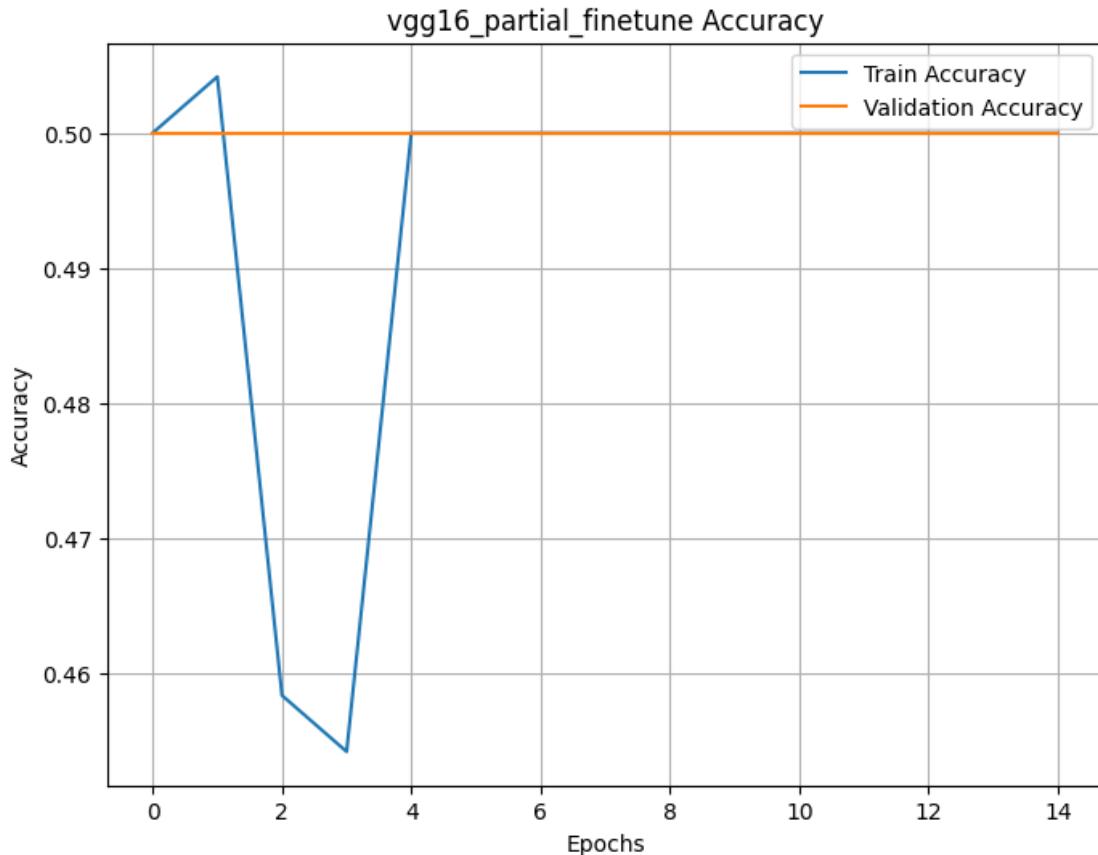
```



```
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/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%
```

```
[ ]: # 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}%")

```

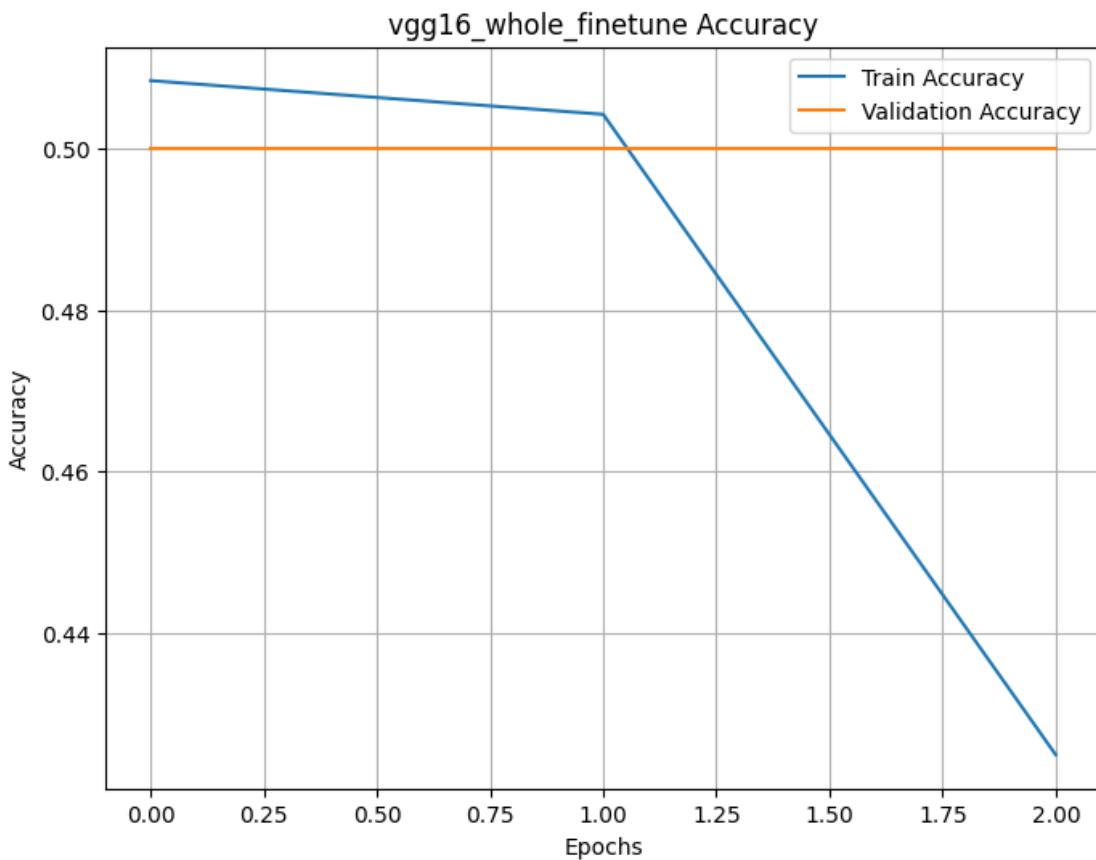
Found 240 images belonging to 2 classes.

Found 60 images belonging to 2 classes.

```

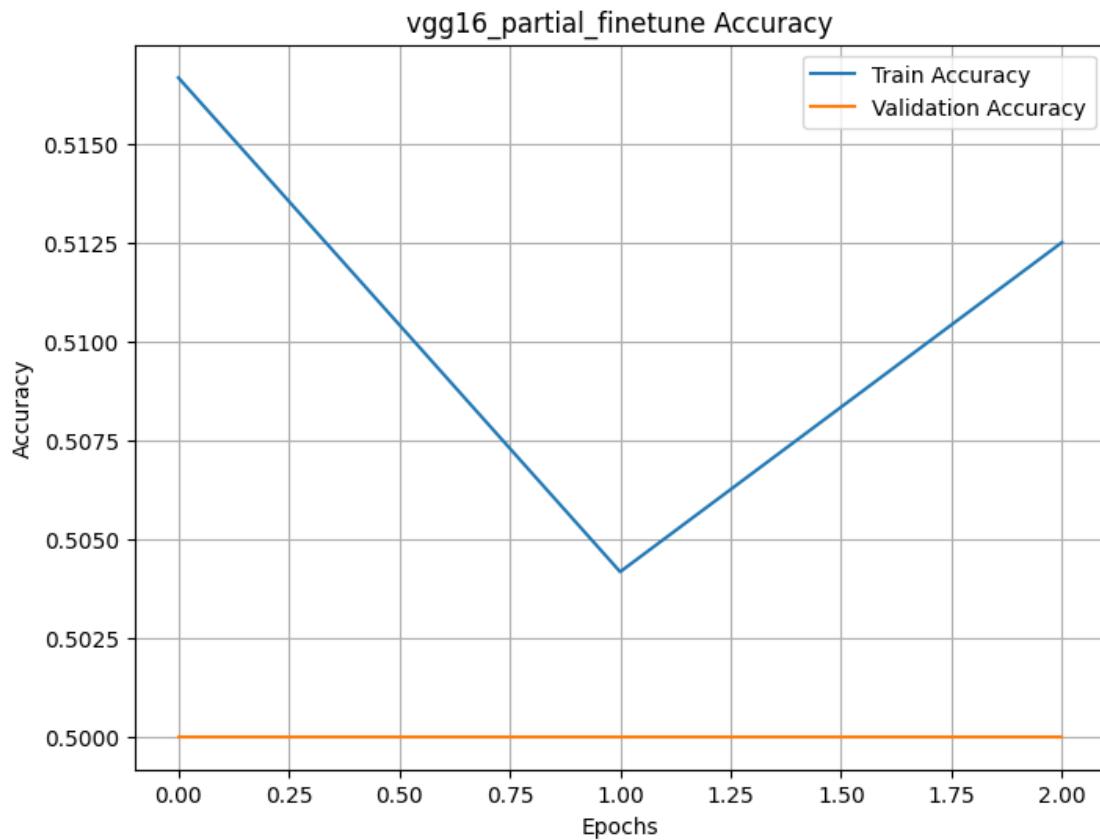
==== Training: Whole VGG16 Fine-tuning ====
Epoch 1/3
8/8          221s 27s/step -
accuracy: 0.5109 - loss: 1.5929 - val_accuracy: 0.5000 - val_loss: 0.7260
Epoch 2/3
8/8          217s 28s/step -
accuracy: 0.5191 - loss: 0.7386 - val_accuracy: 0.5000 - val_loss: 0.7006
Epoch 3/3
8/8          172s 21s/step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

```
[ ]: # 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)

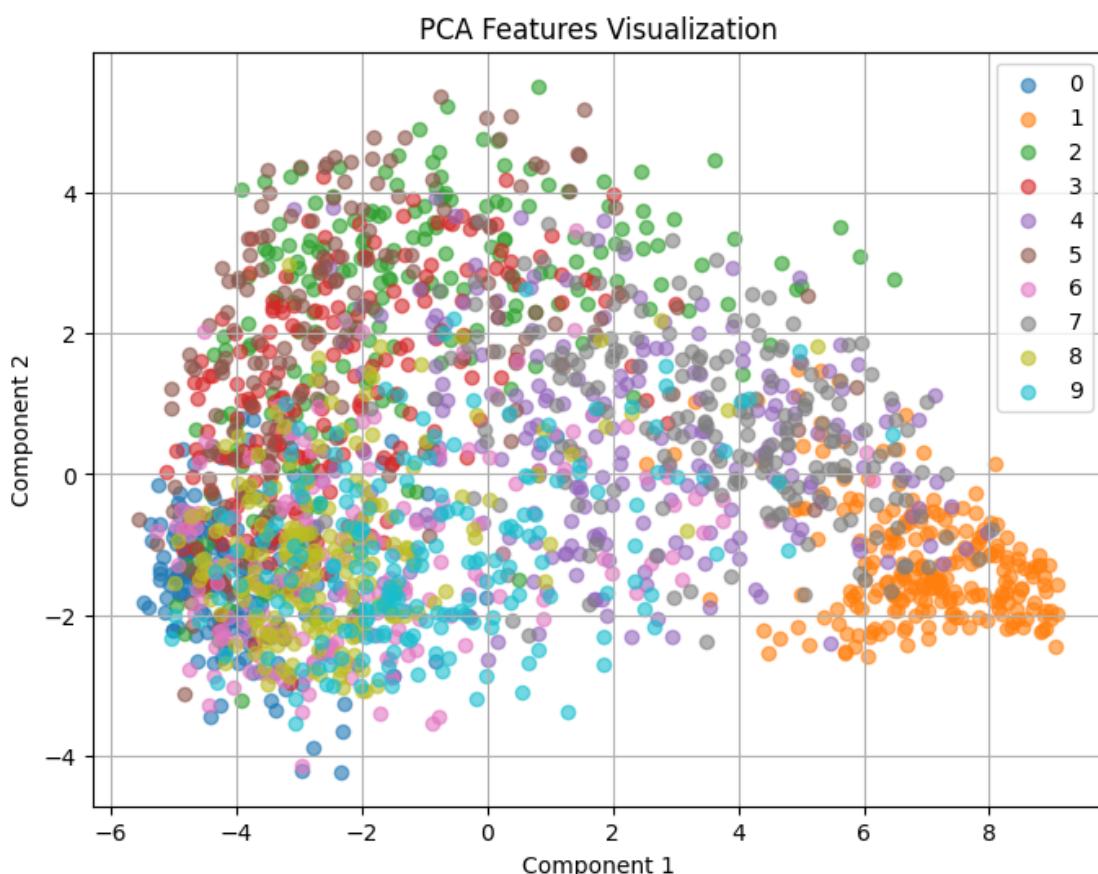
    plt.figure(figsize=(8,6))
    for i in range(10):
        plt.scatter(features_2d[labels==i,0], features_2d[labels==i,1],
                    label=str(i), alpha=0.6)
    plt.title(f"{method} Features Visualization")
    plt.xlabel("Component 1")
    plt.ylabel("Component 2")
    plt.legend()
    plt.grid(True)
    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')

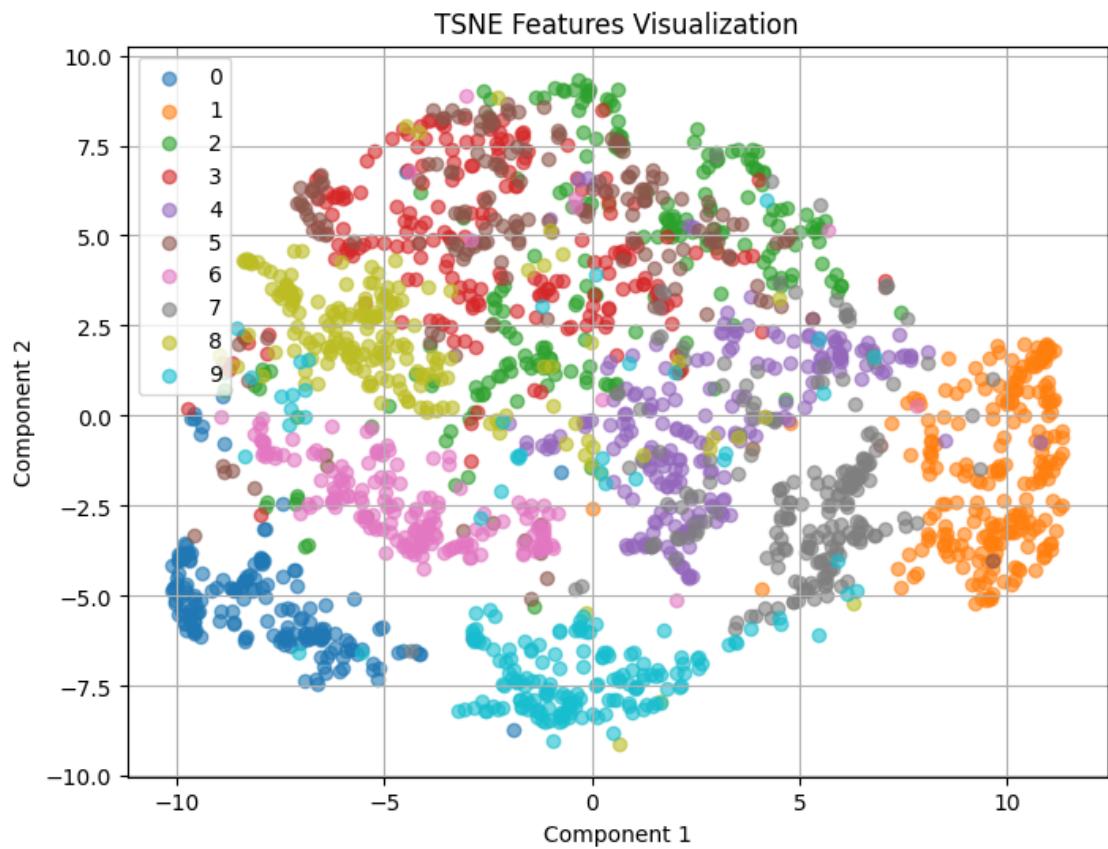
```

```
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')
```

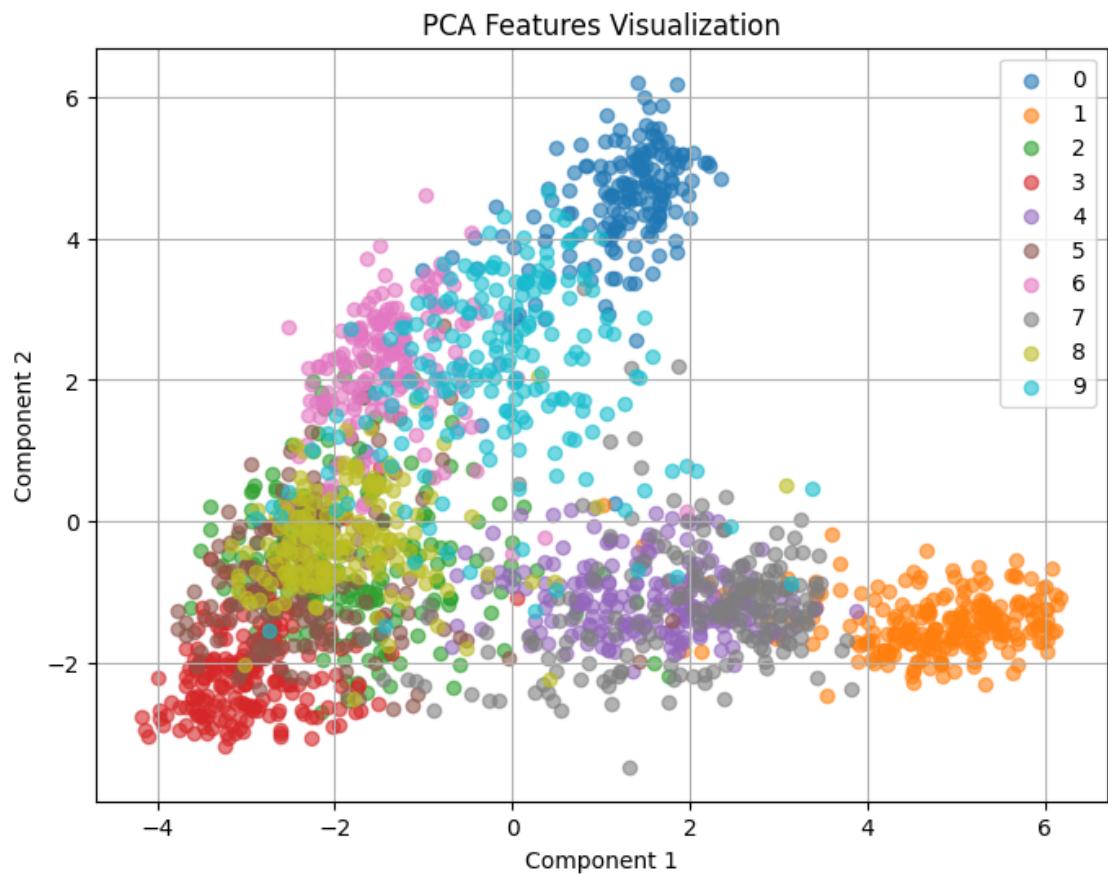
```
Epoch 1/3
422/422 - 668s - 2s/step - accuracy: 0.8057 - loss: 0.6427 - val_accuracy:
0.9367 - val_loss: 0.2254
Epoch 2/3
422/422 - 653s - 2s/step - accuracy: 0.9243 - loss: 0.2562 - val_accuracy:
0.9562 - val_loss: 0.1555
Epoch 3/3
422/422 - 686s - 2s/step - accuracy: 0.9401 - loss: 0.1965 - val_accuracy: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



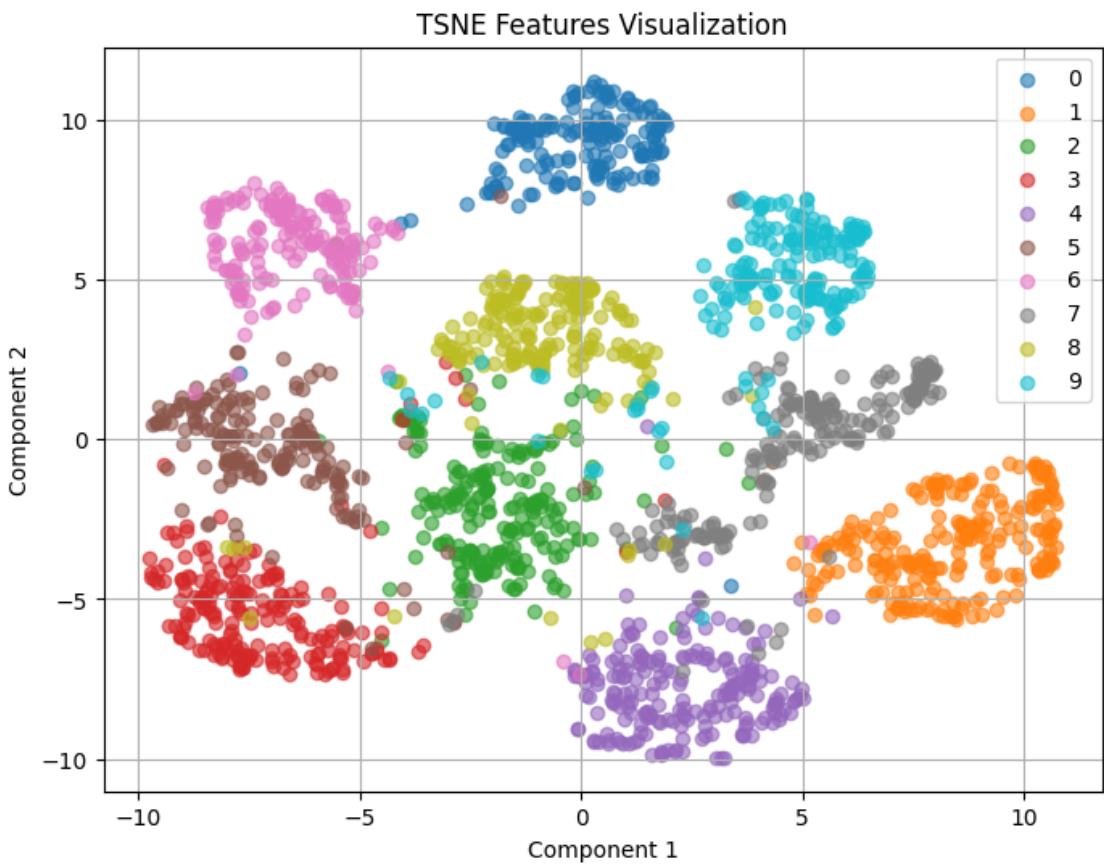
Saved: plots/features\_before\_PCA.png



Saved: plots/features\_before\_tSNE.png



Saved: plots/features\_after\_PCA.png



Saved: plots/features\_after\_tSNE.png

```
[ ]: # 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, u
    ↪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, u
    ↪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}")

```

```

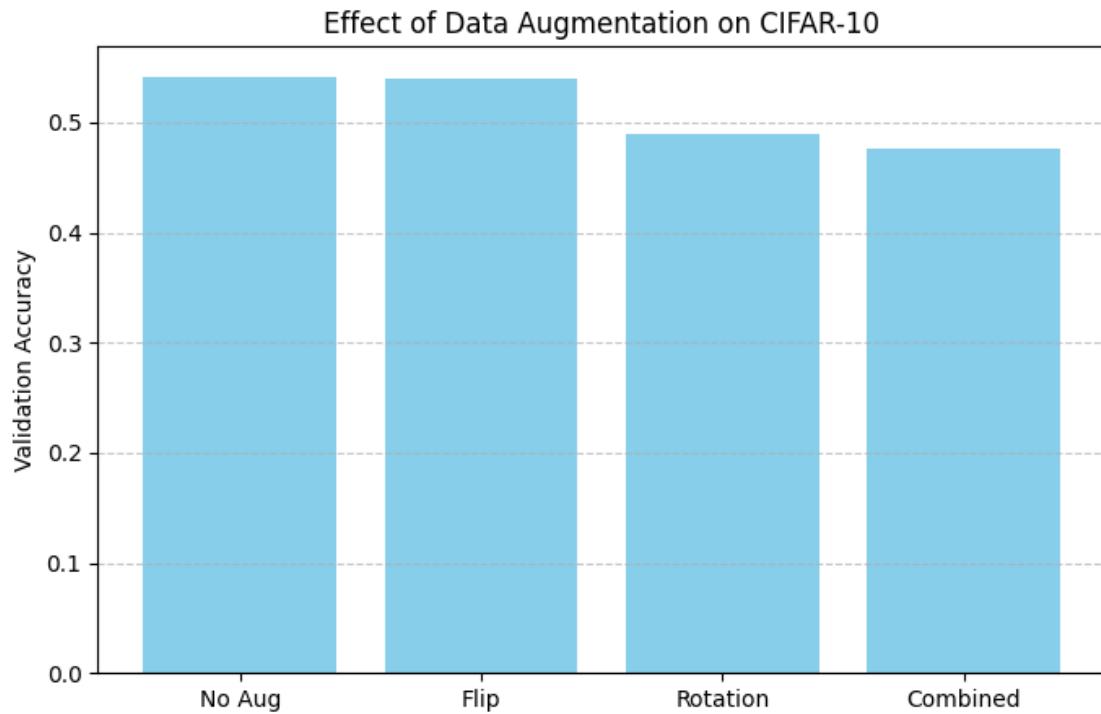
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071           2s
0us/step
Epoch 1/5
125/125          12s 67ms/step -
accuracy: 0.2310 - loss: 2.0909 - val_accuracy: 0.4128 - val_loss: 1.6531
Epoch 2/5
125/125          5s 40ms/step -
accuracy: 0.4231 - loss: 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

[ ]: # 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

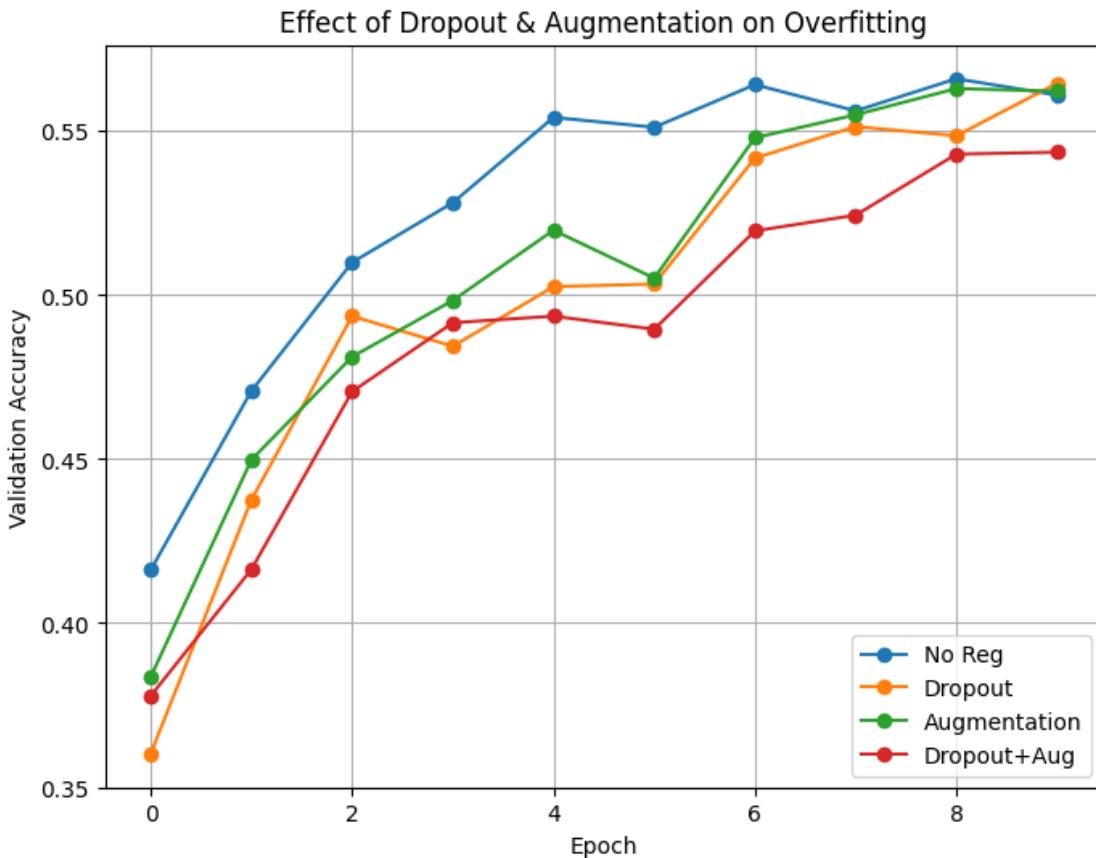
Epoch 2/10  
157/157 8s 48ms/step -  
accuracy: 0.3247 - loss: 1.8143 - val\_accuracy: 0.4376 - val\_loss: 1.5644

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()

```

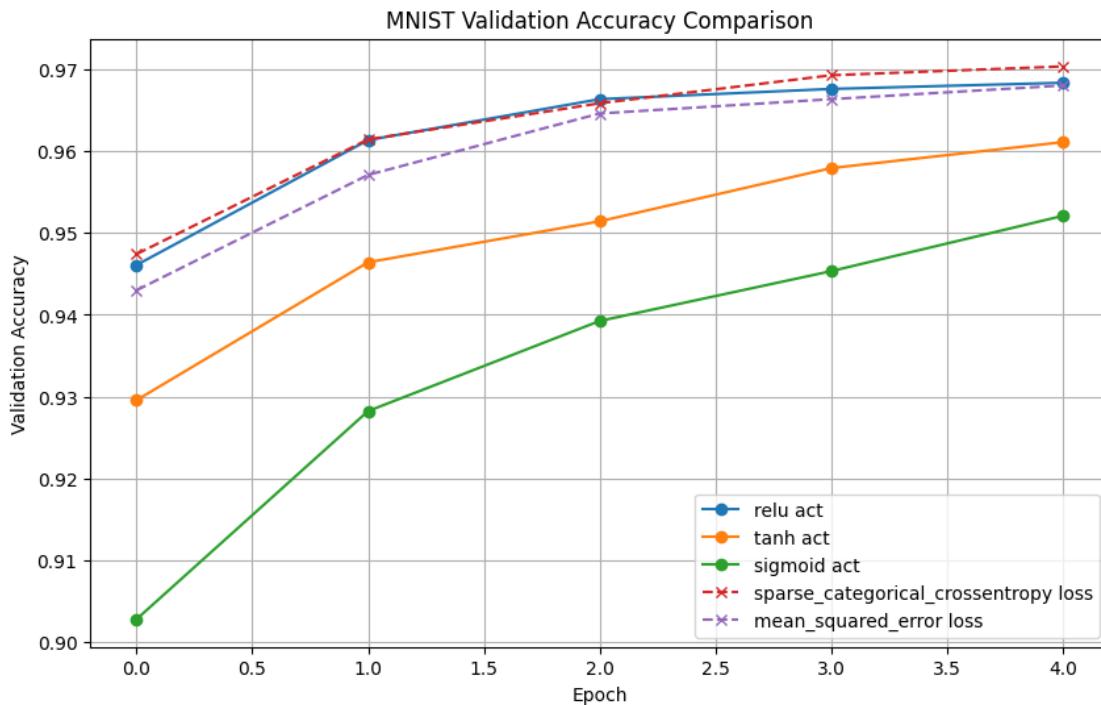
```

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
```



```
[ ]: # 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%
```

```
[ ]: # 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
)

```

```

# -----
# 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

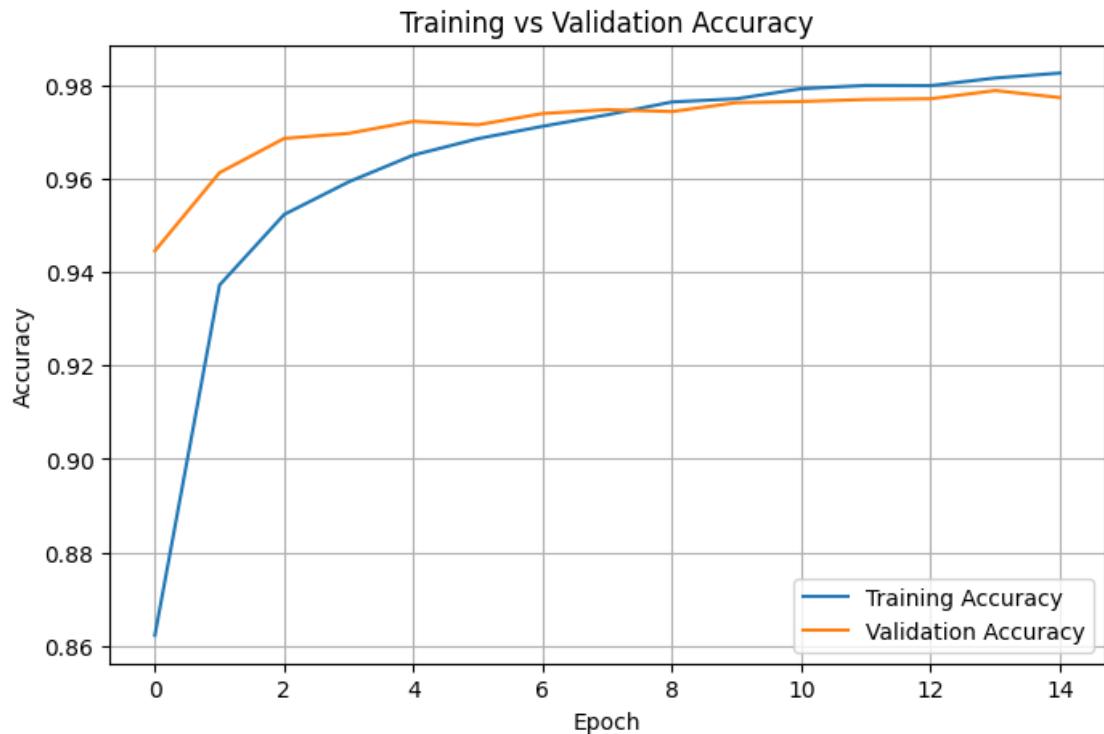
Epoch 9/15  
375/375 - 5s - 12ms/step - accuracy: 0.9765 - loss: 0.0757 - val\_accuracy:  
0.9744 - val\_loss: 0.0851

Epoch 10/15  
375/375 - 2s - 6ms/step - accuracy: 0.9772 - loss: 0.0708 - val\_accuracy: 0.9763  
- val\_loss: 0.0815

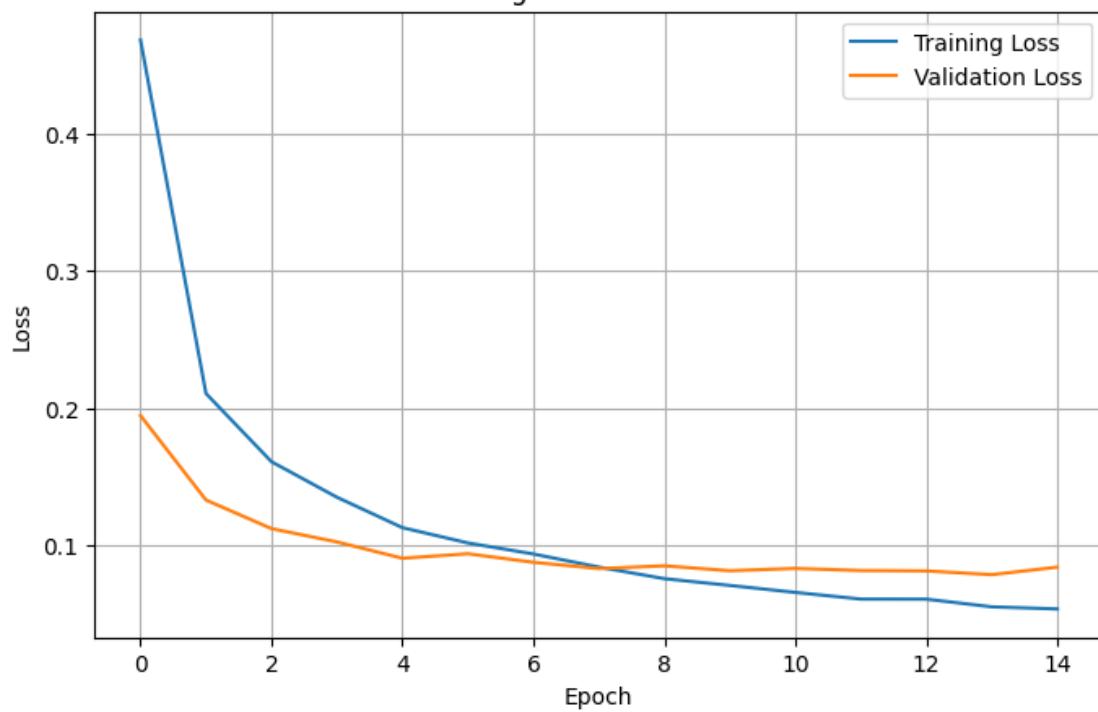
Epoch 11/15  
375/375 - 2s - 6ms/step - accuracy: 0.9793 - loss: 0.0657 - val\_accuracy: 0.9766  
- val\_loss: 0.0832

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
```



Training vs Validation Loss



Test Accuracy: 97.66%