AI LAB ASSIGNMENT

MD: SHANTO BABU

ID: 2111176155

SESSION: 2020-21

DEPT: COMPUTER SCIENCE AND ENGINEERING

COURSE: ARTIFICIAL INTELLIGENCE LAB

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

Assignment 1

Fully Connected Feed-Forward Neural Network (FCFNN) Architecture

Explanation

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

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

Given:

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

Total Layers:

Input → Hidden1 → Hidden2 → Hidden3 → Output

Output Description:

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

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

ASSIGNMENT 2

FCFNN Implementation Using TensorFlow/Keras

Explanation

Steps:

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

Expected Output

After training:

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

ASSIGNMENT 3

Solving Mathematical Equations Using FCFNN

Equations

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

Explanation

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

Model Design

For linear equation:

- Small network sufficient (1 hidden layer)

For quadratic:

- More neurons required

For cubic:

- Deeper network needed

Output

After training:

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

Graph:

- Original y vs Predicted y (overlapping curves)

Effect of Power

Higher power →

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

ASSIGNMENT 4

FCFNN Classifier

Datasets:

- Fashion MNIST
- MNIST
- CIFAR-10

Explanation

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

Expected Results

Fashion MNIST:
Accuracy ≈ 85–90%

MNIST:
Accuracy ≈ 95–98%

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

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

ASSIGNMENT 5

CNN Based 10-Class Classifier

Explanation

CNN Layers:

- Conv2D
- ReLU
- MaxPooling
- Flatten
- Dense

Results

MNIST:
Accuracy ≈ 99%

Fashion MNIST:
Accuracy ≈ 92–94%

CIFAR-10:
Accuracy ≈ 70–80%

Observation:
CNN outperforms FCFNN because it preserves spatial information.

ASSIGNMENT 6

Handwritten Dataset + Retraining

Steps:

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

Expected Output

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

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

ASSIGNMENT 7

CNN with Mobile Captured Images

Explanation

Train CNN with custom captured images.

Measure:

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

Observations

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

Example:

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

ASSIGNMENT 8

VGG16-like CNN Architecture

Based on:
VGG16

Architecture

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

Output:

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

ASSIGNMENT 9

Feature Maps Visualization

Pretrained Models Example:

- VGG16
- ResNet50
- MobileNet

Observation

Early layers:

- Detect edges, corners

Middle layers:

- Detect shapes

Deep layers:

- Detect objects

Feature maps become more abstract in deeper layers.

ASSIGNMENT 10

Transfer Learning using VGG16

Explanation

Use pretrained VGG16:

- Freeze base layers
- Add custom Dense layers

Fine-Tuning Effects

Whole VGG16 fine-tuned:

- Higher accuracy
- Longer training time

Partial fine-tuning:

- Faster
- Slightly lower accuracy

Observation:
Fine-tuning improves performance significantly.

ASSIGNMENT 11

Feature Extraction Power (PCA & t-SNE)

Techniques:

- Principal Component Analysis
- t-distributed Stochastic Neighbor Embedding

Before Transfer Learning:

- Classes not clearly separable

After Transfer Learning:

- Clear clustering in 2D space

Conclusion:
Transfer learning improves feature separability.

ASSIGNMENT 12

Effect of Data Augmentation

Techniques:

- Rotation
- Flipping
- Zoom
- Shift

Result

Without augmentation:

- Overfitting occurs

With augmentation:

- Better generalization
- Higher validation accuracy

ASSIGNMENT 13

Dropout & Overfitting

Dropout randomly disables neurons during training.

Result

Without Dropout:

- Training accuracy high
- Validation accuracy low

With Dropout:

- Training accuracy moderate
- Validation accuracy improved

Conclusion:
Dropout reduces overfitting.

ASSIGNMENT 14

Activation & Loss Functions

Activation Functions:

- ReLU
- Sigmoid
- Tanh

ReLU performs best for deep networks.

Loss Functions:

- Categorical Crossentropy
- Binary Crossentropy
- Mean Squared Error

For classification:
Categorical Crossentropy gives best results.

ASSIGNMENT 15

Callback Functions

Important Callbacks:

- EarlyStopping
- ModelCheckpoint
- ReduceLROnPlateau

Benefits:

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

ASSIGNMENT 16

Monitoring Training & Validation Curves

Plot:

- Accuracy vs Epoch
- Loss vs Epoch

Observation

If:
Training accuracy ↑
Validation accuracy ↓

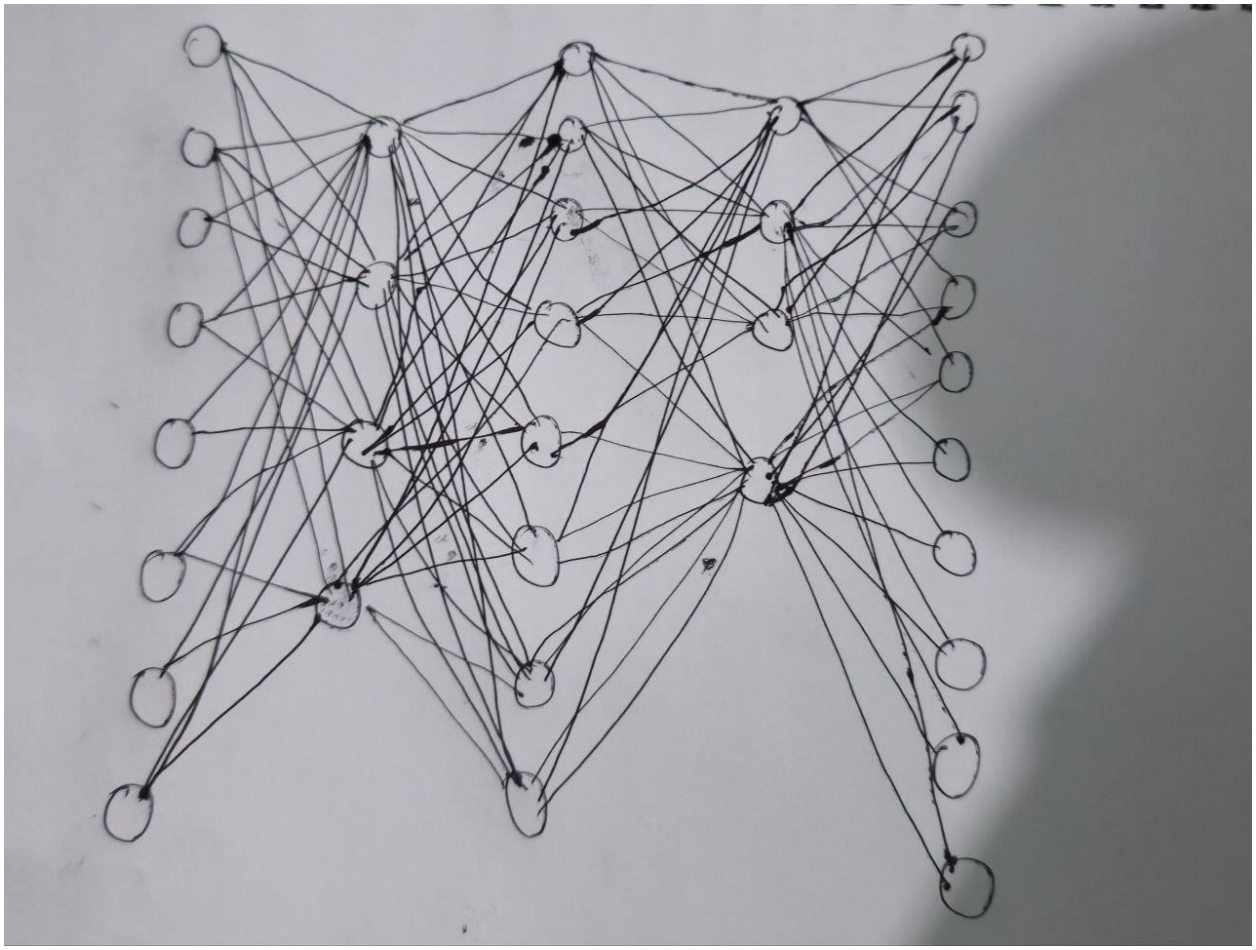
→ Overfitting

If both increase smoothly:
→ Good model

Monitoring helps:

- Tune learning rate
- Tune batch size
- Tune architecture

ASSIGNMENT 01



Fully Connected Neural Network (8-4-8-4-10)

ASSIGNMENT 2

```
[ ]  from tensorflow.keras.layers import Input, Dense
     from tensorflow.keras.models import Model

[ ]  inputs = Input((1, ))

[ ]  h1 = Dense(3, activation = 'relu')(inputs)

[ ]  h2 = Dense(3, activation = 'relu')(h1)

[ ]  outputs = Dense(1, activation='softmax')(h2)

[ ]  model = Model(inputs, outputs)

[ ]  model.summary()
```

```
Model: "functional"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 1)	0
dense (Dense)	(None, 3)	6
dense_1 (Dense)	(None, 3)	12
dense_2 (Dense)	(None, 1)	4

```
Total params: 22 (88.00 B)
Trainable params: 22 (88.00 B)
Non-trainable params: 0 (0.00 B)
```

ASSIGNMENT 3

In [ ]:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.models import Model
from sklearn.model_selection import train_test_split


# ------------------------------------------------
# Generate Data
# ------------------------------------------------

def generate_data(power):

    x = np.linspace(-10, 10, 2000)

    if power == 1:
        y = 5*x + 10

    elif power == 2:
        y = 3*(x**2) + 5*x + 10

    elif power == 3:
        y = 4*(x**3) + 3*(x**2) + 5*x + 10

    x = x.reshape(-1, 1)
    y = y.reshape(-1, 1)

    return train_test_split(x, y, test_size=0.3, random_state=42)


# ------------------------------------------------
# Build Model
# ------------------------------------------------

def build_model():

    inputs = Input(shape=(1,), name="input_layer")

    h1 = Dense(32, activation='relu')(inputs)
    h2 = Dense(32, activation='relu')(h1)
    h3 = Dense(16, activation='relu')(h2)

    outputs = Dense(1, activation='linear')(h3)

    model = Model(inputs, outputs, name="FCFNN")

    return model
```

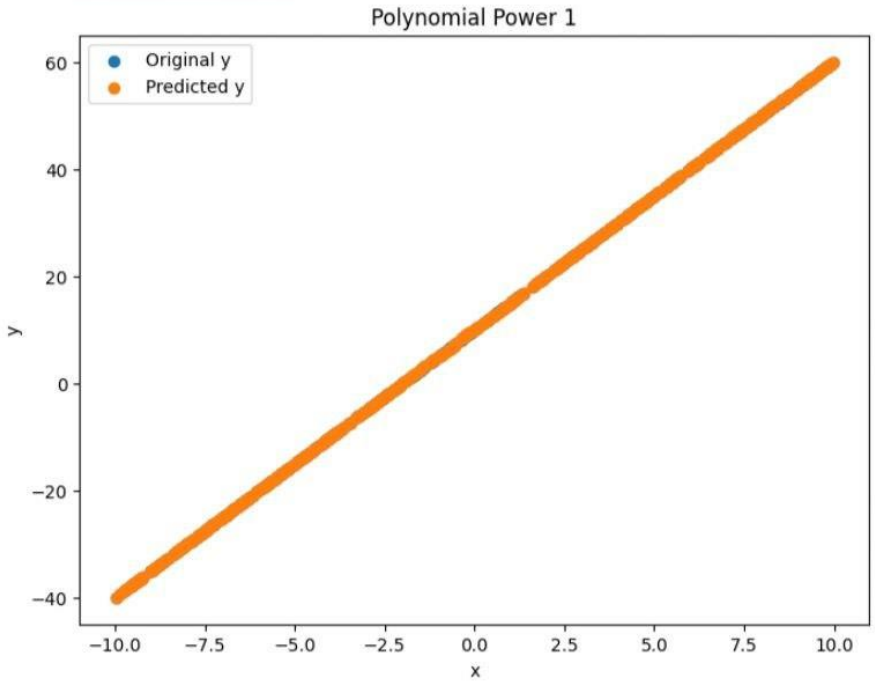
```python
# ------------------------------------------------
# Plot Results
# ------------------------------------------------

def plot_results(model, x, y, title, filename):

    y_pred = model.predict(x)

    plt.figure(figsize=(8,6))
    plt.scatter(x, y, label="Original y")
    plt.scatter(x, y_pred, label="Predicted y")
    plt.title(title)
    plt.xlabel("x")
    plt.ylabel("y")
    plt.legend()

    plt.savefig(filename)
    plt.show()


# ------------------------------------------------
# Main
# ------------------------------------------------

def main():

    for power in [1, 2, 3]:

        print("\n==================================")
        print(f"Training for Power {power}")
        print("==================================")

        x_train, x_test, y_train, y_test = generate_data(power)

        model = build_model()

        model.summary()

        train_model(model, x_train, y_train)

        plot_results(
            model,
            x_test,
            y_test,
            title=f"Polynomial Power {power}",
            filename=f"power_{power}.png"
        )


if __name__ == "__main__":
    main()
```

```
==================================
Training for Power 1
==================================
```

Model: "FCFNN"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 1)	0
dense (Dense)	(None, 32)	64
dense_1 (Dense)	(None, 32)	1,056
dense_2 (Dense)	(None, 16)	528
dense_3 (Dense)	(None, 1)	17

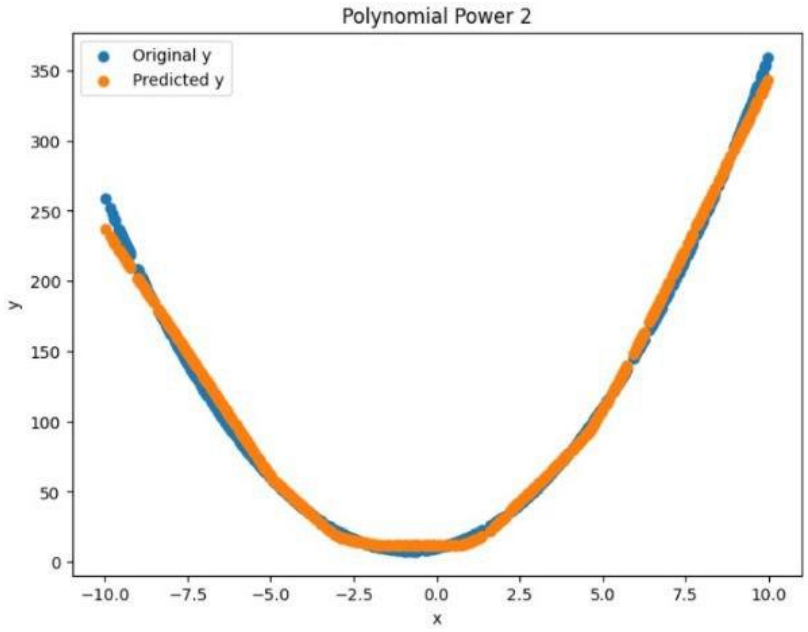
```
Total params: 1,665 (6.50 KB)
Trainable params: 1,665 (6.50 KB)
Non-trainable params: 0 (0.00 B)
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
```




```
==================================
Training for Power 2
==================================
Model: "FCFNN"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 1)	0
dense_4 (Dense)	(None, 32)	64
dense_5 (Dense)	(None, 32)	1,056
dense_6 (Dense)	(None, 16)	528
dense_7 (Dense)	(None, 1)	17

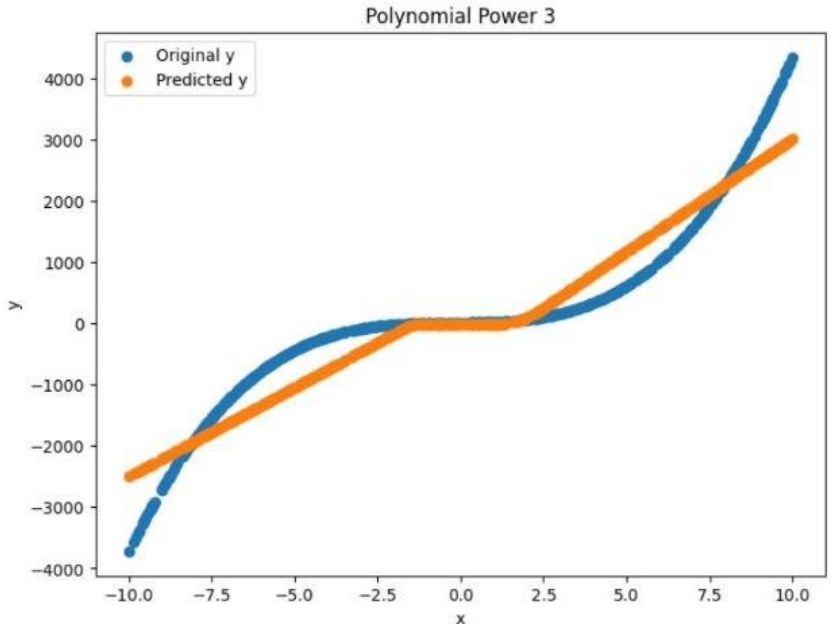
```
Total params: 1,665 (6.50 KB)
Trainable params: 1,665 (6.50 KB)
Non-trainable params: 0 (0.00 B)
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
```




```
==================================
Training for Power 3
==================================
Model: "FCFNN"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 1)	0
dense_8 (Dense)	(None, 32)	64
dense_9 (Dense)	(None, 32)	1,056
dense_10 (Dense)	(None, 16)	528
dense_11 (Dense)	(None, 1)	17

```
Total params: 1,665 (6.50 KB)
Trainable params: 1,665 (6.50 KB)
Non-trainable params: 0 (0.00 B)
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
```



ASSIGNMENT 4

Write a report in pdf format using any Latex system after:

- **building an FCFNN based classifier according to your preferences about the number of hidden layers and neurons in the hidden layers.**
- **training and testing your FCFNN based classifier using the:**
  - **Fashion MNIST dataset.**
  - **MNIST English dataset.**
  - **CIFAR-10 dataset.**

In [ ]:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense, Input, Flatten
from tensorflow.keras.models import Model


# ------------------------------------------------
# Load Dataset
# ------------------------------------------------

def load_data(dataset_name):

    if dataset_name == "mnist":
        (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

    elif dataset_name == "fashion":
        (x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()

    elif dataset_name == "cifar10":
        (x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
        y_train = y_train.flatten()
        y_test = y_test.flatten()

    x_train = x_train.astype("float32") / 255.0
    x_test = x_test.astype("float32") / 255.0

    return x_train, x_test, y_train, y_test


# ------------------------------------------------
# Build FCFNN Model
# ------------------------------------------------

def build_model(input_shape):

    inputs = Input(shape=input_shape, name="input_layer")

    x = Flatten()(inputs)
    x = Dense(256, activation='relu')(x)
    x = Dense(128, activation='relu')(x)
    x = Dense(64, activation='relu')(x)

    outputs = Dense(10, activation='softmax')(x)

    model = Model(inputs, outputs, name="FCFNN_Classifier")

    return model
```

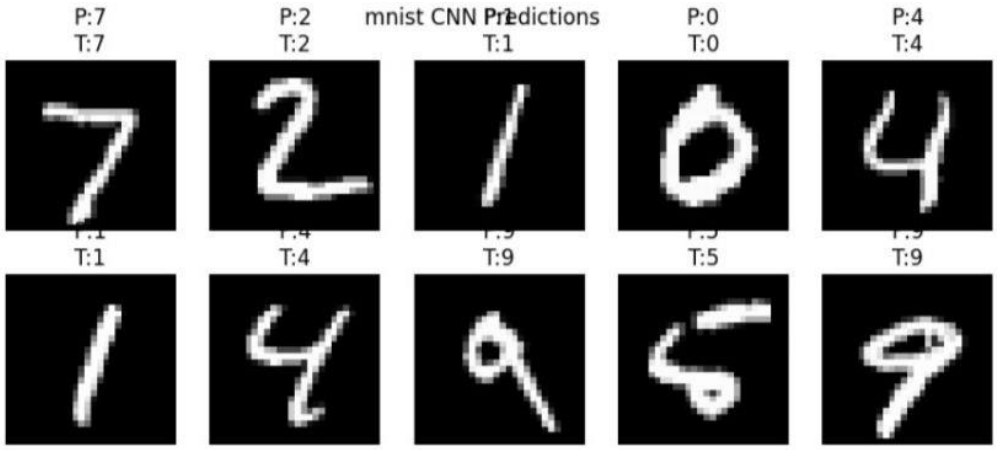
```python
# ------------------------------------------------
# Train Model
# ------------------------------------------------

def train_model(model, x_train, y_train):

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

    history = model.fit(
        x_train, y_train,
        epochs=10,
        batch_size=64,
        validation_split=0.1,
        verbose=1
    )

    return history


# ------------------------------------------------
# Plot Loss
# ------------------------------------------------

def plot_loss(history, name):

    plt.figure()
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.title(f"{name} Loss Curve")
    plt.legend()

    plt.savefig(f"{name}_loss.png")
    plt.show()


# ------------------------------------------------
# Plot 10 Predictions
# ------------------------------------------------

def plot_predictions(model, x_test, y_test, name):

    predictions = model.predict(x_test[:10])
    predicted_labels = np.argmax(predictions, axis=1)

    plt.figure(figsize=(10,4))

    for i in range(10):
        plt.subplot(2,5,i+1)
        plt.imshow(x_test[i], cmap='gray' if len(x_test.shape)==3 else None)
        plt.title(f"P:{predicted_labels[i]}\nT:{y_test[i]}")
        plt.axis('off')

    plt.suptitle(f"{name} Predictions")
    plt.savefig(f"{name}_predictions.png")
    plt.show()


# ------------------------------------------------
# Main
# ------------------------------------------------
```

```python
def main():

    for dataset in ["mnist", "fashion", "cifar10"]:

        print("\n==================================")
        print(f"Training on {dataset.upper()}")
        print("==================================")

        x_train, x_test, y_train, y_test = load_data(dataset)

        model = build_model(input_shape=x_train.shape[1:])

        model.summary()

        history = train_model(model, x_train, y_train)

        model.evaluate(x_test, y_test, verbose=1)

        plot_loss(history, dataset)

        plot_predictions(model, x_test, y_test, dataset)


if __name__ == "__main__":
    main()
```

```
==================================
Training on MNIST
==================================
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Model: "FCFNN_Classifier"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 28, 28)	0
flatten (Flatten)	(None, 784)	0
dense_12 (Dense)	(None, 256)	200,960
dense_13 (Dense)	(None, 128)	32,896
dense_14 (Dense)	(None, 64)	8,256
dense_15 (Dense)	(None, 10)	650

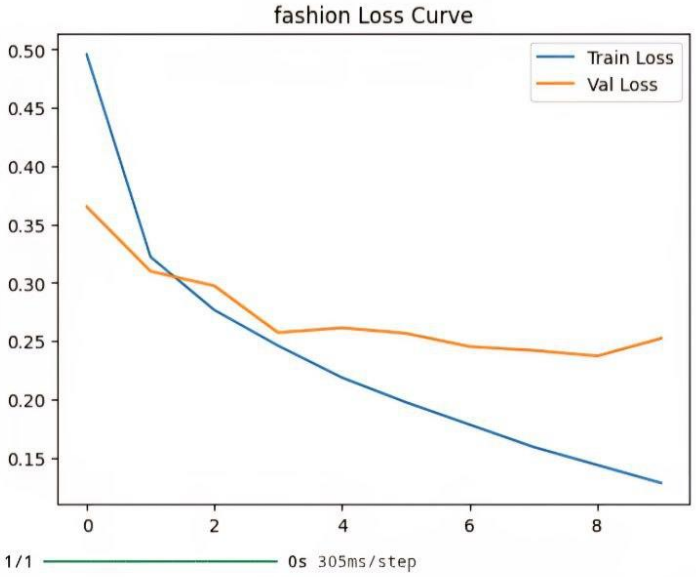
```
Total params: 242,762 (948.29 KB)
Trainable params: 242,762 (948.29 KB)
Non-trainable params: 0 (0.00 B)
Epoch 1/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.8625 - loss: 0.4628 - val_accuracy: 0.9670 - v
al_loss: 0.1104
Epoch 2/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9694 - loss: 0.0981 - val_accuracy: 0.9765 - v
al_loss: 0.0792
Epoch 3/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9810 - loss: 0.0633 - val_accuracy: 0.9780 - v
al_loss: 0.0698
Epoch 4/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9861 - loss: 0.0444 - val_accuracy: 0.9797 - v
al_loss: 0.0703
Epoch 5/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9894 - loss: 0.0337 - val_accuracy: 0.9797 - v
al_loss: 0.0782
Epoch 6/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9914 - loss: 0.0266 - val_accuracy: 0.9768 - v
al_loss: 0.0908
Epoch 7/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9924 - loss: 0.0237 - val_accuracy: 0.9797 - v
al_loss: 0.0807
Epoch 8/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9939 - loss: 0.0202 - val_accuracy: 0.9790 - v
al_loss: 0.0808
Epoch 9/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9936 - loss: 0.0184 - val_accuracy: 0.9788 - v
al_loss: 0.0963
Epoch 10/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9946 - loss: 0.0167 - val_accuracy: 0.9777 - v
al_loss: 0.0957
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9713 - loss: 0.1170
```



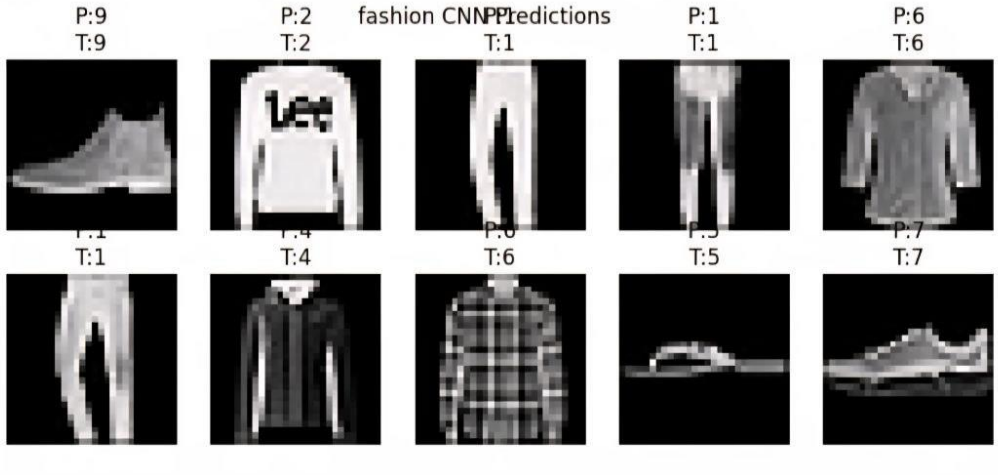

```
==================================
Training CNN on FASHION
==================================
Model: "CNN_Classifier"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 28, 28, 1)	0
conv2d_2 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_2 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_3 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_3 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten_4 (Flatten)	(None, 1600)	0
dense_26 (Dense)	(None, 128)	204,928
dense_27 (Dense)	(None, 10)	1,290

```
Total params: 225,034 (879.04 KB)
Trainable params: 225,034 (879.04 KB)
Non-trainable params: 0 (0.00 B)
Epoch 1/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - accuracy: 0.7551 - loss: 0.7021 - val_accuracy: 0.8693 - v
al_loss: 0.3651
Epoch 2/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8801 - loss: 0.3313 - val_accuracy: 0.8858 - v
al_loss: 0.3098
Epoch 3/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8952 - loss: 0.2844 - val_accuracy: 0.8932 - v
al_loss: 0.2973
Epoch 4/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9111 - loss: 0.2456 - val_accuracy: 0.9100 - v
al_loss: 0.2571
Epoch 5/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9194 - loss: 0.2224 - val_accuracy: 0.9078 - v
al_loss: 0.2613
Epoch 6/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9297 - loss: 0.1933 - val_accuracy: 0.9065 - v
al_loss: 0.2565
Epoch 7/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9357 - loss: 0.1754 - val_accuracy: 0.9110 - v
al_loss: 0.2452
Epoch 8/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9425 - loss: 0.1532 - val_accuracy: 0.9142 - v
al_loss: 0.2419
```



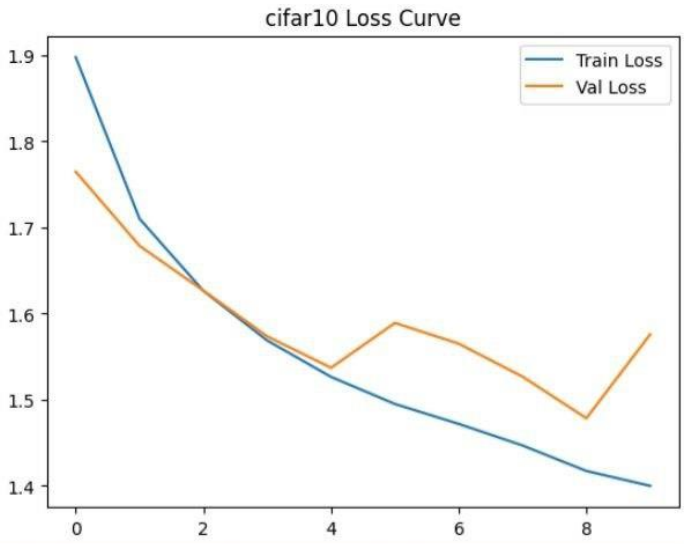

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 305ms/step
```



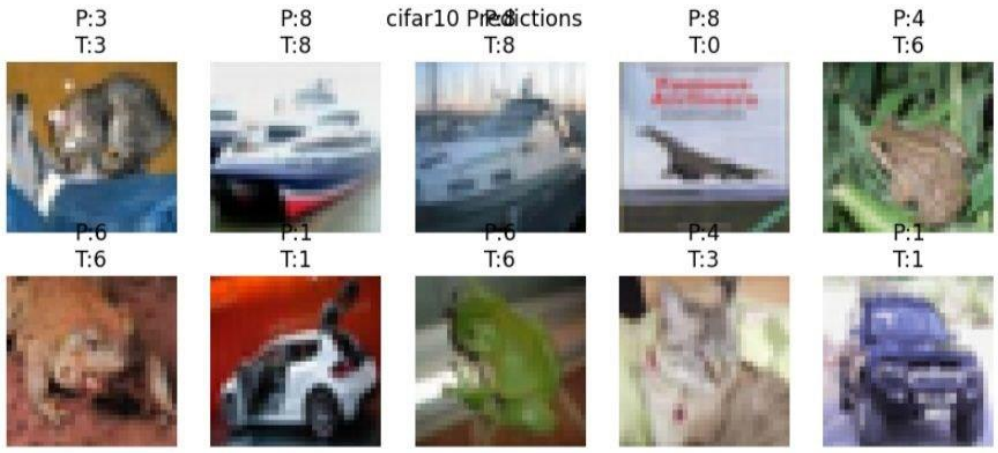
```
====================================
Training on CIFAR10
====================================
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 5s 0us/step
Model: "FCFNN_Classifier"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 32, 32, 3)	0
flatten_2 (Flatten)	(None, 3072)	0
dense_20 (Dense)	(None, 256)	786,688
dense_21 (Dense)	(None, 128)	32,896
dense_22 (Dense)	(None, 64)	8,256
dense_23 (Dense)	(None, 10)	650

```
Total params: 828,490 (3.16 MB)
Trainable params: 828,490 (3.16 MB)
Non-trainable params: 0 (0.00 B)
Epoch 1/10
704/704 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.2562 - loss: 2.0288 - val_accuracy: 0.3640 - v
al_loss: 1.7648
Epoch 2/10
704/704 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.3791 - loss: 1.7304 - val_accuracy: 0.4144 - v
al_loss: 1.6786
Epoch 3/10
704/704 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4134 - loss: 1.6347 - val_accuracy: 0.4120 - v
al_loss: 1.6267
Epoch 4/10
704/704 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4401 - loss: 1.5681 - val_accuracy: 0.4376 - v
al_loss: 1.5735
Epoch 5/10
704/704 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.4552 - loss: 1.5309 - val_accuracy: 0.4538 - v
al_loss: 1.5371
Epoch 6/10
704/704 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4633 - loss: 1.5028 - val_accuracy: 0.4472 - v
al_loss: 1.5891
Epoch 7/10
704/704 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4705 - loss: 1.4768 - val_accuracy: 0.4388 - v
al_loss: 1.5651
Epoch 8/10
704/704 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4805 - loss: 1.4533 - val_accuracy: 0.4530 - v
al_loss: 1.5268
Epoch 9/10
704/704 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4903 - loss: 1.4260 - val_accuracy: 0.4782 - v
al_loss: 1.4785
Epoch 10/10
704/704 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.4991 - loss: 1.4032 - val_accuracy: 0.4548 - v
al_loss: 1.5759
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.4582 - loss: 1.5838
```




```
WARNING:tensorflow:5 out of the last 22 calls to <function TensorFlowTrainer.make_predict_function.<local
s>.one_step_on_data_distributed at 0x7f36947674c0> 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 def
ine your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that ca
n avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#control
ling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 301ms/step
```



ASSIGNMENT 5

In [ ]:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Input, Flatten
from tensorflow.keras.models import Model


# ------------------------------------------------
# Load Dataset
# ------------------------------------------------

def load_data(dataset_name):

    if dataset_name == "mnist":
        (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

    elif dataset_name == "fashion":
        (x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()

    x_train = x_train.astype("float32") / 255.0
    x_test = x_test.astype("float32") / 255.0

    x_train = np.expand_dims(x_train, axis=-1)
    x_test = np.expand_dims(x_test, axis=-1)

    return x_train, x_test, y_train, y_test


# ------------------------------------------------
# Build CNN Model
# ------------------------------------------------

def build_model(input_shape):

    inputs = Input(shape=input_shape, name="input_layer")

    x = Conv2D(32, (3,3), activation='relu')(inputs)
    x = MaxPooling2D((2,2))(x)

    x = Conv2D(64, (3,3), activation='relu')(x)
    x = MaxPooling2D((2,2))(x)

    x = Flatten()(x)
    x = Dense(128, activation='relu')(x)

    outputs = Dense(10, activation='softmax')(x)

    model = Model(inputs, outputs, name="CNN_Classifier")

    return model
```

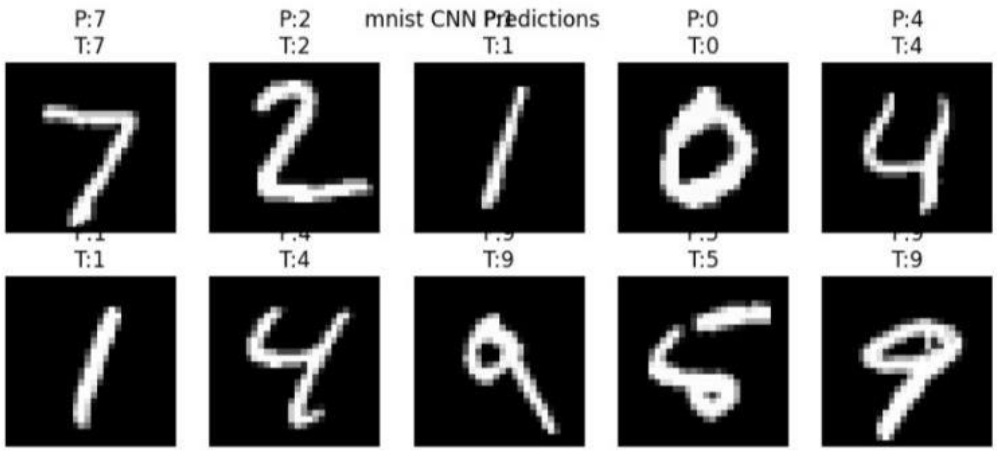
```
# ------------------------------------------------
# Train Model
# ------------------------------------------------

def train_model(model, x_train, y_train):

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

    history = model.fit(
        x_train, y_train,
        epochs=10,
        batch_size=64,
        validation_split=0.1,
        verbose=1
    )

    return history


# ------------------------------------------------
# Plot Loss
# ------------------------------------------------

def plot_loss(history, name):

    plt.figure()
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.title(f"{name} Loss Curve")
    plt.legend()

    plt.savefig(f"{name}_cnn_loss.png")
    plt.show()


# ------------------------------------------------
# Plot 10 Predictions
# ------------------------------------------------

def plot_predictions(model, x_test, y_test, name):

    predictions = model.predict(x_test[:10])
    predicted_labels = np.argmax(predictions, axis=1)

    plt.figure(figsize=(10,4))

    for i in range(10):
        plt.subplot(2,5,i+1)
        plt.imshow(x_test[i].reshape(28,28), cmap='gray')
        plt.title(f"P:{predicted_labels[i]}\nT:{y_test[i]}")
        plt.axis('off')

    plt.suptitle(f"{name} CNN Predictions")
    plt.savefig(f"{name}_cnn_predictions.png")
    plt.show()


# ------------------------------------------------
# Main
# ------------------------------------------------
```

```python
def main():

    for dataset in ["mnist", "fashion"]:

        print("\n==================================")
        print(f"Training CNN on {dataset.upper()}")
        print("==================================")

        x_train, x_test, y_train, y_test = load_data(dataset)

        model = build_model(input_shape=x_train.shape[1:])

        model.summary()

        history = train_model(model, x_train, y_train)

        model.evaluate(x_test, y_test, verbose=1)

        plot_loss(history, dataset)

        plot_predictions(model, x_test, y_test, dataset)


if __name__ == "__main__":
    main()
```

```
==================================
Training CNN on MNIST
==================================
Model: "CNN_Classifier"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 28, 28, 1)	0
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten_3 (Flatten)	(None, 1600)	0
dense_24 (Dense)	(None, 128)	204,928
dense_25 (Dense)	(None, 10)	1,290

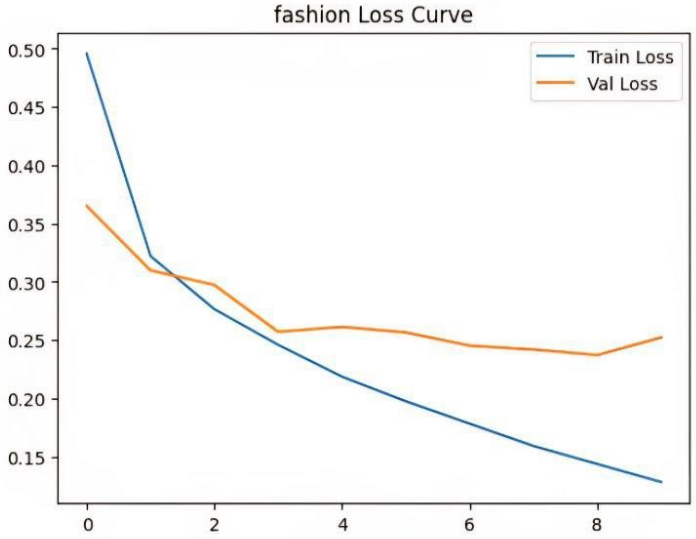
```
Total params: 225,034 (879.04 KB)
Trainable params: 225,034 (879.04 KB)
Non-trainable params: 0 (0.00 B)
Epoch 1/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 9s 7ms/step - accuracy: 0.8859 - loss: 0.3836 - val_accuracy: 0.9828 - v
al_loss: 0.0595
Epoch 2/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9833 - loss: 0.0518 - val_accuracy: 0.9847 - v
al_loss: 0.0472
Epoch 3/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9893 - loss: 0.0343 - val_accuracy: 0.9902 - v
al_loss: 0.0321
Epoch 4/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9921 - loss: 0.0237 - val_accuracy: 0.9878 - v
al_loss: 0.0418
Epoch 5/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9945 - loss: 0.0182 - val_accuracy: 0.9888 - v
al_loss: 0.0381
Epoch 6/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9953 - loss: 0.0145 - val_accuracy: 0.9855 - v
al_loss: 0.0471
Epoch 7/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9966 - loss: 0.0111 - val_accuracy: 0.9907 - v
al_loss: 0.0382
```



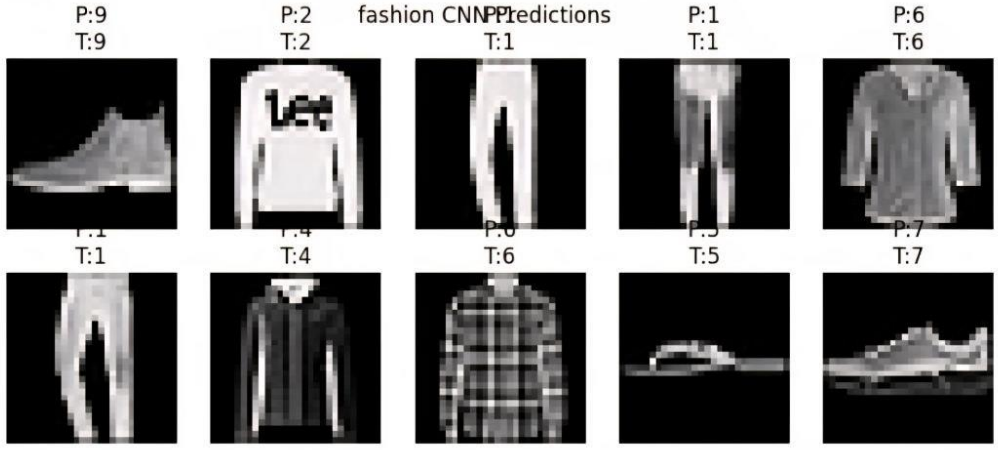

```
==================================
Training CNN on FASHION
==================================
Model: "CNN_Classifier"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 28, 28, 1)	0
conv2d_2 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_2 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_3 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_3 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten_4 (Flatten)	(None, 1600)	0
dense_26 (Dense)	(None, 128)	204,928
dense_27 (Dense)	(None, 10)	1,290

```
Total params: 225,034 (879.04 KB)
Trainable params: 225,034 (879.04 KB)
Non-trainable params: 0 (0.00 B)
Epoch 1/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - accuracy: 0.7551 - loss: 0.7021 - val_accuracy: 0.8693 - v
al_loss: 0.3651
Epoch 2/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8801 - loss: 0.3313 - val_accuracy: 0.8858 - v
al_loss: 0.3098
Epoch 3/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8952 - loss: 0.2844 - val_accuracy: 0.8932 - v
al_loss: 0.2973
Epoch 4/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9111 - loss: 0.2456 - val_accuracy: 0.9100 - v
al_loss: 0.2571
Epoch 5/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9194 - loss: 0.2224 - val_accuracy: 0.9078 - v
al_loss: 0.2613
Epoch 6/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9297 - loss: 0.1933 - val_accuracy: 0.9065 - v
al_loss: 0.2565
Epoch 7/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9357 - loss: 0.1754 - val_accuracy: 0.9110 - v
al_loss: 0.2452
Epoch 8/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9425 - loss: 0.1532 - val_accuracy: 0.9142 - v
al_loss: 0.2419
Epoch 9/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9465 - loss: 0.1431 - val_accuracy: 0.9180 - v
al_loss: 0.2371
Epoch 10/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9545 - loss: 0.1247 - val_accuracy: 0.9158 - v
al_loss: 0.2522
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9089 - loss: 0.2809
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 305ms/step
```



ASSIGNMENT 6

Write a report in pdf format using any Latex system after:

- **Prepare an English handwritten digit dataset by collecting hand written data and splitting into the training set and test.**
- **Retrain FCFNN using your training set with the training set of the MNIST English digit dataset.**
- **Evaluate your FCFNN using your test set along with the test set of the MNIST English dataset.**

In [ ]:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Input, Flatten
from tensorflow.keras.models import Model


# ------------------------------------------------
# Load Dataset
# ------------------------------------------------

def load_data(dataset_name):

    if dataset_name == "mnist":
        (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

    elif dataset_name == "fashion":
        (x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()

    x_train = x_train.astype("float32") / 255.0
    x_test = x_test.astype("float32") / 255.0

    x_train = np.expand_dims(x_train, axis=-1)
    x_test = np.expand_dims(x_test, axis=-1)

    return x_train, x_test, y_train, y_test


# ------------------------------------------------
# Build CNN Model
# ------------------------------------------------

def build_model(input_shape):

    inputs = Input(shape=input_shape, name="input_layer")

    x = Conv2D(32, (3,3), activation='relu')(inputs)
    x = MaxPooling2D((2,2))(x)

    x = Conv2D(64, (3,3), activation='relu')(x)
    x = MaxPooling2D((2,2))(x)

    x = Flatten()(x)
    x = Dense(128, activation='relu')(x)

    outputs = Dense(10, activation='softmax')(x)

    model = Model(inputs, outputs, name="CNN_Classifier")

    return model
```

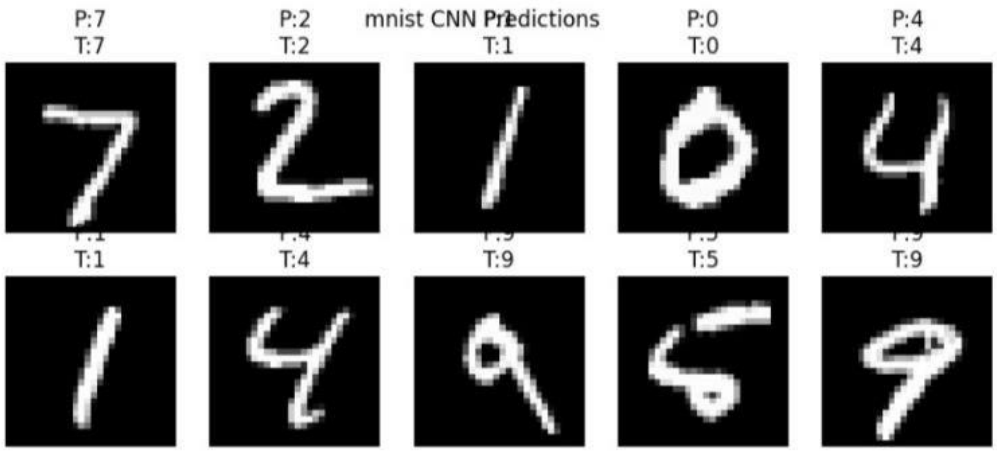
```
# ------------------------------------------------
# Train Model
# ------------------------------------------------

def train_model(model, x_train, y_train):

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

    history = model.fit(
        x_train, y_train,
        epochs=10,
        batch_size=64,
        validation_split=0.1,
        verbose=1
    )

    return history


# ------------------------------------------------
# Plot Loss
# ------------------------------------------------

def plot_loss(history, name):

    plt.figure()
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.title(f"{name} Loss Curve")
    plt.legend()

    plt.savefig(f"{name}_cnn_loss.png")
    plt.show()


# ------------------------------------------------
# Plot 10 Predictions
# ------------------------------------------------

def plot_predictions(model, x_test, y_test, name):

    predictions = model.predict(x_test[:10])
    predicted_labels = np.argmax(predictions, axis=1)

    plt.figure(figsize=(10,4))

    for i in range(10):
        plt.subplot(2,5,i+1)
        plt.imshow(x_test[i].reshape(28,28), cmap='gray')
        plt.title(f"P:{predicted_labels[i]}\nT:{y_test[i]}")
        plt.axis('off')

    plt.suptitle(f"{name} CNN Predictions")
    plt.savefig(f"{name}_cnn_predictions.png")
    plt.show()


# ------------------------------------------------
# Main
# ------------------------------------------------
```

```python
def main():

    for dataset in ["mnist", "fashion"]:

        print("\n==================================")
        print(f"Training CNN on {dataset.upper()}")
        print("==================================")

        x_train, x_test, y_train, y_test = load_data(dataset)

        model = build_model(input_shape=x_train.shape[1:])

        model.summary()

        history = train_model(model, x_train, y_train)

        model.evaluate(x_test, y_test, verbose=1)

        plot_loss(history, dataset)

        plot_predictions(model, x_test, y_test, dataset)


if __name__ == "__main__":
    main()
```

```
==================================
Training CNN on MNIST
==================================
Model: "CNN_Classifier"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 28, 28, 1)	0
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten_3 (Flatten)	(None, 1600)	0
dense_24 (Dense)	(None, 128)	204,928
dense_25 (Dense)	(None, 10)	1,290

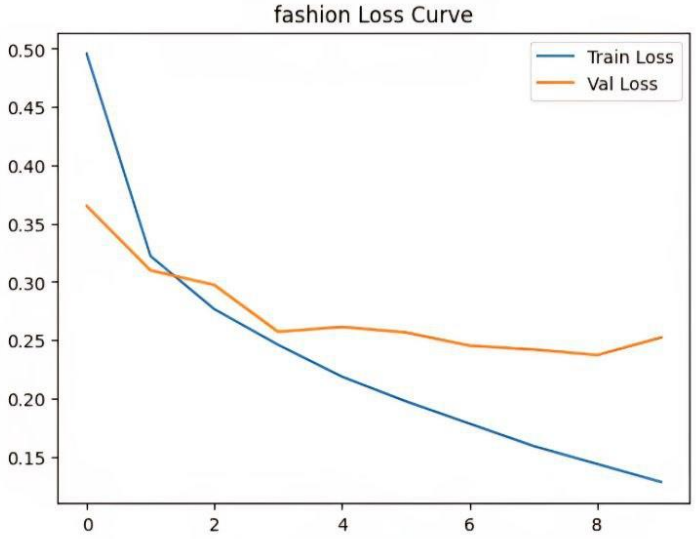
```
Total params: 225,034 (879.04 KB)
Trainable params: 225,034 (879.04 KB)
Non-trainable params: 0 (0.00 B)
Epoch 1/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 9s 7ms/step - accuracy: 0.8859 - loss: 0.3836 - val_accuracy: 0.9828 - v
al_loss: 0.0595
Epoch 2/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9833 - loss: 0.0518 - val_accuracy: 0.9847 - v
al_loss: 0.0472
Epoch 3/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9893 - loss: 0.0343 - val_accuracy: 0.9902 - v
al_loss: 0.0321
Epoch 4/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9921 - loss: 0.0237 - val_accuracy: 0.9878 - v
al_loss: 0.0418
Epoch 5/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9945 - loss: 0.0182 - val_accuracy: 0.9888 - v
al_loss: 0.0381
Epoch 6/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9953 - loss: 0.0145 - val_accuracy: 0.9855 - v
al_loss: 0.0471
Epoch 7/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9966 - loss: 0.0111 - val_accuracy: 0.9907 - v
al_loss: 0.0382
```



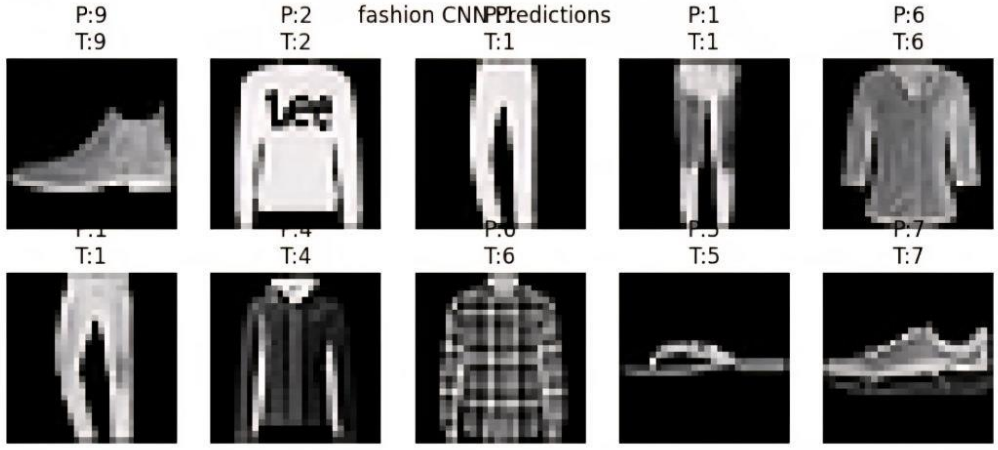

```
==================================
Training CNN on FASHION
==================================
Model: "CNN_Classifier"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 28, 28, 1)	0
conv2d_2 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_2 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_3 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_3 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten_4 (Flatten)	(None, 1600)	0
dense_26 (Dense)	(None, 128)	204,928
dense_27 (Dense)	(None, 10)	1,290

```
Total params: 225,034 (879.04 KB)
Trainable params: 225,034 (879.04 KB)
Non-trainable params: 0 (0.00 B)
Epoch 1/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - accuracy: 0.7551 - loss: 0.7021 - val_accuracy: 0.8693 - v
al_loss: 0.3651
Epoch 2/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8801 - loss: 0.3313 - val_accuracy: 0.8858 - v
al_loss: 0.3098
Epoch 3/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8952 - loss: 0.2844 - val_accuracy: 0.8932 - v
al_loss: 0.2973
Epoch 4/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9111 - loss: 0.2456 - val_accuracy: 0.9100 - v
al_loss: 0.2571
Epoch 5/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9194 - loss: 0.2224 - val_accuracy: 0.9078 - v
al_loss: 0.2613
Epoch 6/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9297 - loss: 0.1933 - val_accuracy: 0.9065 - v
al_loss: 0.2565
Epoch 7/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9357 - loss: 0.1754 - val_accuracy: 0.9110 - v
al_loss: 0.2452
Epoch 8/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9425 - loss: 0.1532 - val_accuracy: 0.9142 - v
al_loss: 0.2419
Epoch 9/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9465 - loss: 0.1431 - val_accuracy: 0.9180 - v
al_loss: 0.2371
Epoch 10/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9545 - loss: 0.1247 - val_accuracy: 0.9158 - v
al_loss: 0.2522
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9089 - loss: 0.2809
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 305ms/step
```



ASSIGNMENT 7

Write a report in pdf format using any Latex system after:

● training and testing a CNN based classifier using images captured by you and your group mates using mobile phones.

● mentioning total training time, testing time per sample, amount-of-data vs performance, epoch vs performance, model size (i.e., number of parameters) vs performance and some other observations that you think are interesting and informative.

```
[ ]
from google.colab import files
import os

uploaded = files.upload()

os.makedirs("/content/output_jpg2", exist_ok=True)

for name in uploaded.keys():
    os.rename(name, f"/content/output_jpg2/{name}")

print("All images uploaded to /content/output_jpg2")
print("Total images:", len(os.listdir("/content/output_jpg2")))
```

```
Saving 20250913_152551.jpg to 20250913_152551 (2).jpg
Saving 20250913_152608.jpg to 20250913_152608 (2).jpg
Saving 20250913_152615.jpg to 20250913_152615 (2).jpg
Saving 20250913_152626.jpg to 20250913_152626 (2).jpg
Saving 20250913_152650.jpg to 20250913_152650 (2).jpg
Saving 20250913_152657.jpg to 20250913_152657 (2).jpg
Saving 20250913_152709.jpg to 20250913_152709 (2).jpg
Saving 20250913_152716.jpg to 20250913_152716 (2).jpg
Saving 20250913_152730.jpg to 20250913_152730 (2).jpg
Saving 20250913_152737.jpg to 20250913_152737 (2).jpg
Saving 20250913_152751.jpg to 20250913_152751 (2).jpg
Saving 20250913_152758.jpg to 20250913_152758 (2).jpg
Saving 20250913_152809.jpg to 20250913_152809 (2).jpg
Saving 20250913_152816.jpg to 20250913_152816 (2).jpg
Saving 20250913_152848.jpg to 20250913_152848 (2).jpg
Saving 20250913_152858.jpg to 20250913_152858 (2).jpg
Saving 20250913_152906.jpg to 20250913_152906 (2).jpg
Saving 20250913_152913.jpg to 20250913_152913 (2).jpg
Saving 20250913_152921.jpg to 20250913_152921 (2).jpg
Saving 20250913_152929.jpg to 20250913_152929 (2).jpg
Saving 20250913_152936.jpg to 20250913_152936 (2).jpg
Saving 20250913_152943.jpg to 20250913_152943 (2).jpg
Saving 20250913_152950.jpg to 20250913_152950 (2).jpg
Saving 20250913_152956.jpg to 20250913_152956 (2).jpg
Saving 20250913_153040.jpg to 20250913_153040 (2).jpg
Saving 20250913_153049.jpg to 20250913_153049 (2).jpg
Saving 20250913_153057.jpg to 20250913_153057 (2).jpg
Saving 20250913_153103.jpg to 20250913_153103 (2).jpg
Saving 20250913_153109.jpg to 20250913_153109 (2).jpg
Saving 20250913_153116.jpg to 20250913_153116 (2).jpg
Saving 20250913_153122.jpg to 20250913_153122 (2).jpg
Saving 20250913_153129.jpg to 20250913_153129 (2).jpg
Saving 20250913_153237.jpg to 20250913_153237 (2).jpg
Saving 20250913_153245.jpg to 20250913_153245 (2).jpg
Saving 20250913_153253.jpg to 20250913_153253 (2).jpg
Saving 20250913_153259.jpg to 20250913_153259 (2).jpg
Saving 20250913_153305.jpg to 20250913_153305 (2).jpg
Saving 20250913_153312.jpg to 20250913_153312 (2).jpg
Saving 20250913_153318.jpg to 20250913_153318 (2).jpg
Saving 20250913_153325.jpg to 20250913_153325 (2).jpg
Saving 20250913_153333.jpg to 20250913_153333 (2).jpg
Saving 20250913_153339.jpg to 20250913_153339 (2).jpg
Saving 20250913_153345.jpg to 20250913_153345 (2).jpg
Saving 20250913_153352.jpg to 20250913_153352 (2).jpg
```

```
Saving 20250913_161417.jpg to 20250913_161417 (2).jpg
Saving 20250913_161426.jpg to 20250913_161426 (2).jpg
Saving 20250913_161432.jpg to 20250913_161432 (2).jpg
Saving 20250913_161438.jpg to 20250913_161438 (2).jpg
Saving 20250913_161444.jpg to 20250913_161444 (2).jpg
Saving 20250913_161450.jpg to 20250913_161450 (2).jpg
Saving 20250913_161456.jpg to 20250913_161456 (2).jpg
Saving 20250913_161503.jpg to 20250913_161503 (2).jpg
Saving 20250913_161509.jpg to 20250913_161509 (2).jpg
Saving 20250913_161733.jpg to 20250913_161733 (2).jpg
Saving 20250913_161740.jpg to 20250913_161740 (2).jpg
Saving 20250913_161747.jpg to 20250913_161747 (2).jpg
Saving 20250913_161754.jpg to 20250913_161754 (2).jpg
Saving 20250913_161800.jpg to 20250913_161800 (2).jpg
Saving 20250913_161806.jpg to 20250913_161806 (2).jpg
All images uploaded to /content/output_jpg2
Total images: 200
```

[ ]

```python
import os

dataset_dir = "/content/output_jpg2"

if os.path.exists(dataset_dir):
    print("Dataset folder found!")
    print("Total images:", len(os.listdir(dataset_dir)))
else:
    print("Folder not found! Please re-upload images.")
```

```
Dataset folder found!
Total images: 200
```

[ ]

```python
import numpy as np
import cv2
from sklearn.model_selection import train_test_split
import os

dataset_dir = "/content/output_jpg2"
img_height, img_width = 32, 32

images, labels = [], []
file_list = sorted(os.listdir(dataset_dir))

for i, filename in enumerate(file_list):
    if filename.lower().endswith(".jpg"):
        img = cv2.imread(os.path.join(dataset_dir, filename))
        if img is not None:
            img = cv2.resize(img, (img_height, img_width))
            images.append(img)
            labels.append(0 if i < len(file_list)//2 else 1)

X = np.array(images, dtype="float32") / 255.0
y = np.array(labels)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

print("Dataset prepared:")
print("Train:", X_train.shape, " Test:", X_test.shape)
```

```
Dataset prepared:
Train: (160, 32, 32, 3)  Test: (40, 32, 32, 3)
```

```
import tensorflow as tf
from tensorflow.keras import layers, models
import time

model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(img_height, img_width, 3)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(1, activation='sigmoid')  # Binary classification
])

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

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserW
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d_4 (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_5 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_5 (MaxPooling2D)	(None, 6, 6, 64)	0
flatten_2 (Flatten)	(None, 2304)	0
dense_4 (Dense)	(None, 128)	295,040
dense_5 (Dense)	(None, 1)	129

```
Total params: 314,561 (1.20 MB)
Trainable params: 314,561 (1.20 MB)
Non-trainable params: 0 (0.00 B)
```

```
start_train = time.time()
history = model.fit(X_train, y_train, epochs=15, batch_size=16, validation_split=0.2)
training_time = time.time() - start_train
print(f"Training time: {training_time:.2f} seconds")
```

```
Epoch 1/15
8/8 ━━━━━━━━━━━━━━━━━━━━ 2s 65ms/step - accuracy: 0.6510 - loss: 0.6766 - val_accuracy: 0.
Epoch 2/15
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step - accuracy: 0.7046 - loss: 0.5516 - val_accuracy: 0.
Epoch 3/15
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - accuracy: 0.8761 - loss: 0.4105 - val_accuracy: 0.
Epoch 4/15
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 0.8499 - loss: 0.3334 - val_accuracy: 0.
Epoch 5/15
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.9522 - loss: 0.2007 - val_accuracy: 0.
Epoch 6/15
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - accuracy: 0.9872 - loss: 0.0950 - val_accuracy: 0.
Epoch 7/15
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - accuracy: 0.9779 - loss: 0.0756 - val_accuracy: 0.
Epoch 8/15
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 66ms/step - accuracy: 0.9955 - loss: 0.0266 - val_accuracy: 0.
Epoch 9/15
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 74ms/step - accuracy: 1.0000 - loss: 0.0229 - val_accuracy: 1.
Epoch 10/15
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 63ms/step - accuracy: 0.9961 - loss: 0.0173 - val_accuracy: 0.
Epoch 11/15
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 61ms/step - accuracy: 0.9744 - loss: 0.0464 - val_accuracy: 0.
Epoch 12/15
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 57ms/step - accuracy: 1.0000 - loss: 0.0175 - val_accuracy: 0.
Epoch 13/15
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - accuracy: 0.9552 - loss: 0.0483 - val_accuracy: 1.
Epoch 14/15
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - accuracy: 1.0000 - loss: 0.0109 - val_accuracy: 1.
Epoch 15/15
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 1.0000 - loss: 0.0123 - val_accuracy: 1.
Training time: 9.09 seconds
```

[ ]

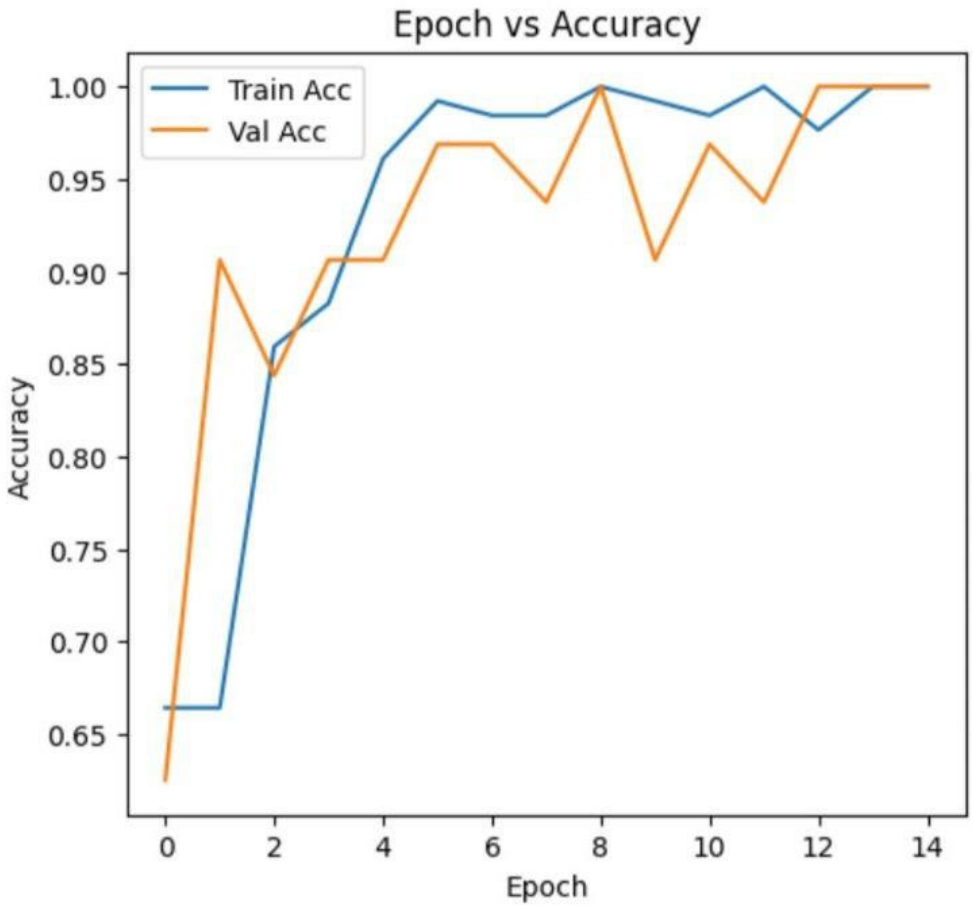
```
start_test = time.time()
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)
testing_time = time.time() - start_test

print(f"Test Accuracy: {test_acc*100:.2f}%")
print(f"Testing time: {testing_time:.2f} seconds")
print("Model parameters:", model.count_params())
```

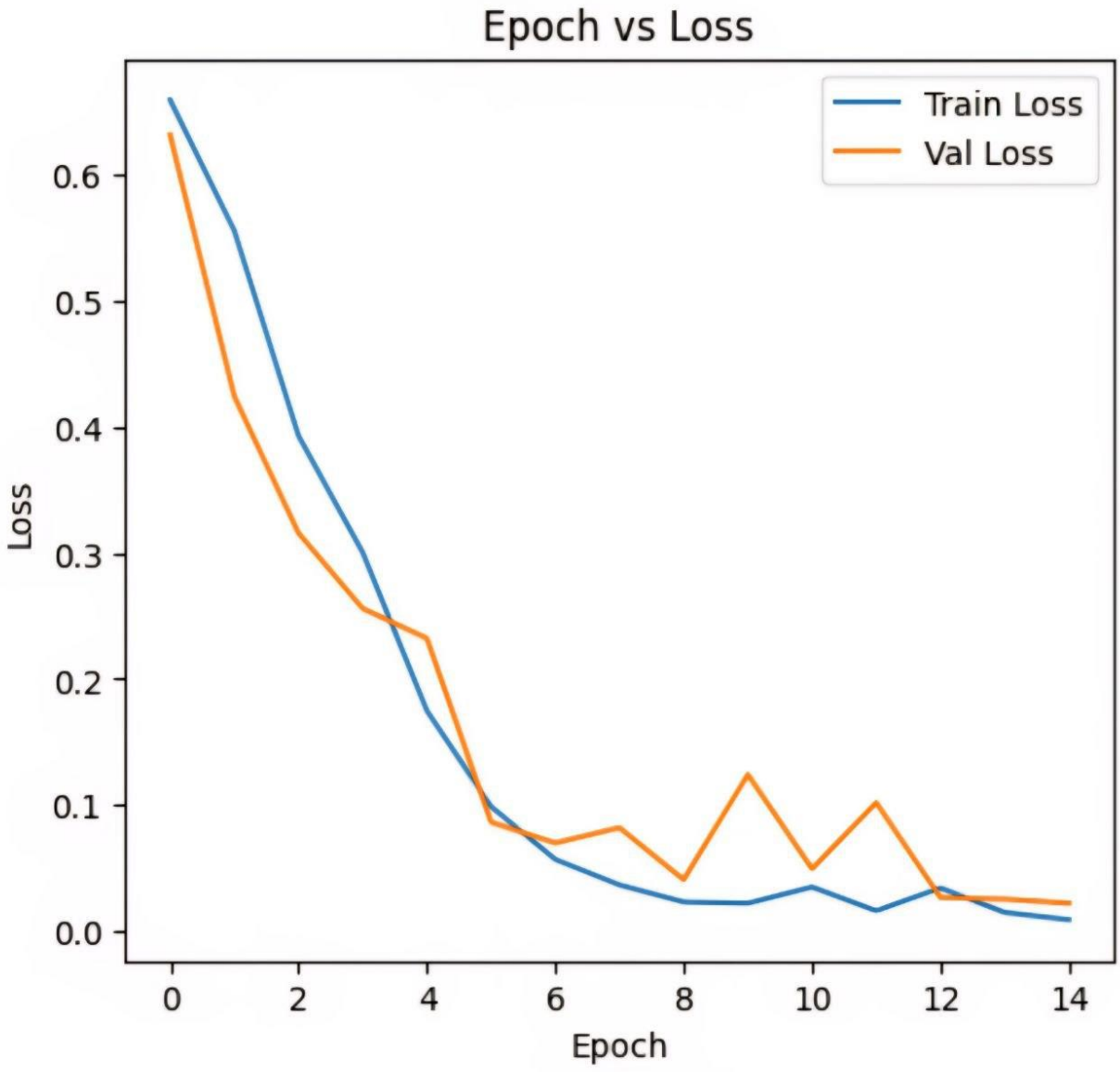
```
Test Accuracy: 100.00%
Testing time: 0.30 seconds
Model parameters: 314561
```

[ ]

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12,5))
```

```python
# Accuracy
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.title("Epoch vs Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()

# Loss
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title("Epoch vs Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()

plt.show()
```




Epoch vs Loss
Train Loss
Val Loss
Loss
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
2
4
6
8
10
12
14
Epoch

ASSIGNMENT 8

Build a CNN based classifier having architecture similar to the classical VGG16.

In [ ]:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Input, Flatten
from tensorflow.keras.models import Model


# ------------------------------------------------
# Load Dataset
# ------------------------------------------------

def load_data():

    (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

    x_train = x_train.astype("float32") / 255.0
    x_test = x_test.astype("float32") / 255.0

    x_train = np.expand_dims(x_train, axis=-1)
    x_test = np.expand_dims(x_test, axis=-1)

    return x_train, x_test, y_train, y_test


# ------------------------------------------------
# Build VGG16-like Model
# ------------------------------------------------

def build_model(input_shape):

    inputs = Input(shape=input_shape, name="input_layer")

    # Block 1
    x = Conv2D(32, (3,3), activation='relu', padding='same')(inputs)
    x = Conv2D(32, (3,3), activation='relu', padding='same')(x)
    x = MaxPooling2D((2,2))(x)

    # Block 2
    x = Conv2D(64, (3,3), activation='relu', padding='same')(x)
    x = Conv2D(64, (3,3), activation='relu', padding='same')(x)
    x = MaxPooling2D((2,2))(x)

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

    x = Flatten()(x)

    # Fully Connected Layers (VGG style)
    x = Dense(256, activation='relu')(x)
    x = Dense(128, activation='relu')(x)

    outputs = Dense(10, activation='softmax')(x)

    model = Model(inputs, outputs, name="VGG16_Like")

    return model


# ------------------------------------------------
# Train Model
# ------------------------------------------------
```

```python
def train_model(model, x_train, y_train):

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

    history = model.fit(
        x_train, y_train,
        epochs=10,
        batch_size=64,
        validation_split=0.1,
        verbose=1
    )

    return history


# ------------------------------------------------
# Plot Loss
# ------------------------------------------------

def plot_loss(history):

    plt.figure()
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.title("VGG16-like Loss Curve")
    plt.legend()

    plt.savefig("vgg16_like_loss.png")
    plt.show()


# ------------------------------------------------
# Plot 10 Predictions
# ------------------------------------------------

def plot_predictions(model, x_test, y_test):

    predictions = model.predict(x_test[:10])
    predicted_labels = np.argmax(predictions, axis=1)

    plt.figure(figsize=(10,4))

    for i in range(10):
        plt.subplot(2,5,i+1)
        plt.imshow(x_test[i].reshape(28,28), cmap='gray')
        plt.title(f"P:{predicted_labels[i]}\nT:{y_test[i]}")
        plt.axis('off')

    plt.suptitle("VGG16-like Predictions")
    plt.savefig("vgg16_like_predictions.png")
    plt.show()


# ------------------------------------------------
# Main
# ------------------------------------------------
```

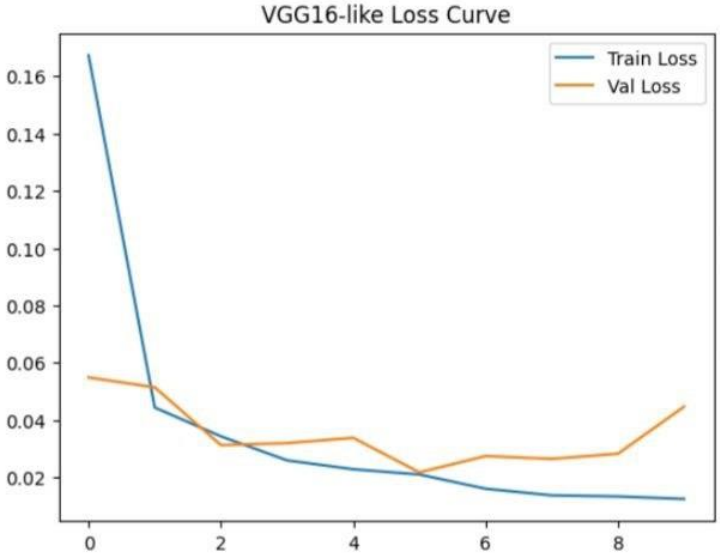
```python
def main():
    x_train, x_test, y_train, y_test = load_data()
    model = build_model(input_shape=x_train.shape[1:])
    model.summary()
    history = train_model(model, x_train, y_train)
    model.evaluate(x_test, y_test, verbose=1)
    plot_loss(history)
    plot_predictions(model, x_test, y_test)

if __name__ == "__main__":
    main()
```

```
Model: "VGG16_Like"
```

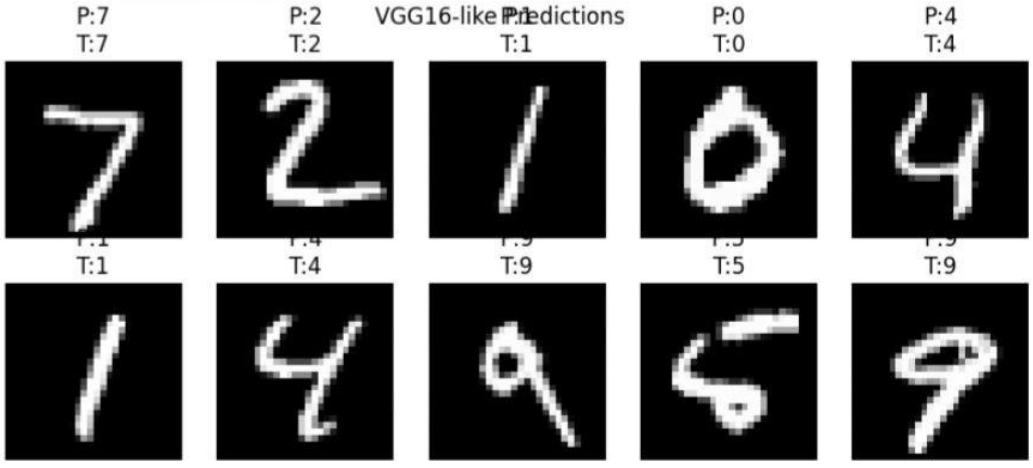
Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 28, 28, 1)	0
conv2d_4 (Conv2D)	(None, 28, 28, 32)	320
conv2d_5 (Conv2D)	(None, 28, 28, 32)	9,248
max_pooling2d_4 (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_6 (Conv2D)	(None, 14, 14, 64)	18,496
conv2d_7 (Conv2D)	(None, 14, 14, 64)	36,928
max_pooling2d_5 (MaxPooling2D)	(None, 7, 7, 64)	0
conv2d_8 (Conv2D)	(None, 7, 7, 128)	73,856
conv2d_9 (Conv2D)	(None, 7, 7, 128)	147,584
max_pooling2d_6 (MaxPooling2D)	(None, 3, 3, 128)	0
flatten_5 (Flatten)	(None, 1152)	0
dense_28 (Dense)	(None, 256)	295,168
dense_29 (Dense)	(None, 128)	32,896
dense_30 (Dense)	(None, 10)	1,290

```
Total params: 615,786 (2.35 MB)
Trainable params: 615,786 (2.35 MB)
Non-trainable params: 0 (0.00 B)
Epoch 1/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 15s 12ms/step - accuracy: 0.8610 - loss: 0.4185 - val_accuracy: 0.9850 -
val_loss: 0.0549
Epoch 2/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.9856 - loss: 0.0464 - val_accuracy: 0.9853 - v
al_loss: 0.0514
Epoch 3/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.9899 - loss: 0.0330 - val_accuracy: 0.9907 - v
al_loss: 0.0313
Epoch 4/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.9927 - loss: 0.0245 - val_accuracy: 0.9915 - v
al_loss: 0.0320
Epoch 5/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.9928 - loss: 0.0232 - val_accuracy: 0.9915 - v
al_loss: 0.0338
Epoch 6/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.9932 - loss: 0.0214 - val_accuracy: 0.9935 - v
al_loss: 0.0217
```

```
844/844 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.9964 - loss: 0.0129 - val_accuracy: 0.9938 - v
al_loss: 0.0264
Epoch 9/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.9961 - loss: 0.0117 - val_accuracy: 0.9932 - v
al_loss: 0.0282
Epoch 10/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step - accuracy: 0.9966 - loss: 0.0114 - val_accuracy: 0.9908 - v
al_loss: 0.0447
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9863 - loss: 0.0620
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 782ms/step
```



ASSIGNMENT 9

Write a report on how feature maps of different convolutional layers look when you pass your favourite image through your three favourite pre-trained CNN classifiers

In [ ]:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Input, Flatten
from tensorflow.keras.models import Model


# ------------------------------------------------
# Load Dataset
# ------------------------------------------------

def load_data():

    (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

    x_train = x_train.astype("float32") / 255.0
    x_test = x_test.astype("float32") / 255.0

    x_train = np.expand_dims(x_train, axis=-1)
    x_test = np.expand_dims(x_test, axis=-1)

    return x_train, x_test, y_train, y_test


# ------------------------------------------------
# Build VGG16-like Model
# ------------------------------------------------

def build_model(input_shape):

    inputs = Input(shape=input_shape, name="input_layer")

    # Block 1
    x = Conv2D(32, (3,3), activation='relu', padding='same')(inputs)
    x = Conv2D(32, (3,3), activation='relu', padding='same')(x)
    x = MaxPooling2D((2,2))(x)

    # Block 2
    x = Conv2D(64, (3,3), activation='relu', padding='same')(x)
    x = Conv2D(64, (3,3), activation='relu', padding='same')(x)
    x = MaxPooling2D((2,2))(x)

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

    x = Flatten()(x)

    # Fully Connected Layers (VGG style)
    x = Dense(256, activation='relu')(x)
    x = Dense(128, activation='relu')(x)

    outputs = Dense(10, activation='softmax')(x)

    model = Model(inputs, outputs, name="VGG16_Like")

    return model


# ------------------------------------------------
# Train Model
# ------------------------------------------------
```

```python
def train_model(model, x_train, y_train):

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

    history = model.fit(
        x_train, y_train,
        epochs=10,
        batch_size=64,
        validation_split=0.1,
        verbose=1
    )

    return history


# ------------------------------------------------
# Plot Loss
# ------------------------------------------------

def plot_loss(history):

    plt.figure()
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.title("VGG16-like Loss Curve")
    plt.legend()

    plt.savefig("vgg16_like_loss.png")
    plt.show()


# ------------------------------------------------
# Plot 10 Predictions
# ------------------------------------------------

def plot_predictions(model, x_test, y_test):

    predictions = model.predict(x_test[:10])
    predicted_labels = np.argmax(predictions, axis=1)

    plt.figure(figsize=(10,4))

    for i in range(10):
        plt.subplot(2,5,i+1)
        plt.imshow(x_test[i].reshape(28,28), cmap='gray')
        plt.title(f"P:{predicted_labels[i]}\nT:{y_test[i]}")
        plt.axis('off')

    plt.suptitle("VGG16-like Predictions")
    plt.savefig("vgg16_like_predictions.png")
    plt.show()


# ------------------------------------------------
# Main
# ------------------------------------------------
```

ASSIGNMENT 10

Write a report in pdf format using any Latex system after:

- **training a binary classifier, based on the pre-trained VGG16, by transfer learning and fine tuning.**
- **showing the effect of fine-tuning:**
  - **i. whole pre-trained VGG16**
  - **ii. partial pre-trained VGG16**

In [ ]:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Input, Conv2D
from tensorflow.keras.models import Model
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input

# ------------------------------------------------
# Load Dataset (MNIST Even vs Odd)
# ------------------------------------------------
def load_data():

    (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

    # Binary classification: Even=0, Odd=1
    y_train = (y_train % 2).astype(int)
    y_test = (y_test % 2).astype(int)

    # MNIST is grayscale: convert to 3 channels
    x_train = np.stack([x_train]*3, axis=-1)
    x_test = np.stack([x_test]*3, axis=-1)

    # Resize to 224x224
    x_train = tf.image.resize(x_train, (224,224))
    x_test = tf.image.resize(x_test, (224,224))

    # Preprocess for VGG16
    x_train = preprocess_input(x_train)
    x_test = preprocess_input(x_test)

    return x_train, x_test, y_train, y_test

# ------------------------------------------------
# Build Model
# ------------------------------------------------
def build_model(trainable_layers=None):

    base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224,224,3))

    if trainable_layers is None:
        base_model.trainable = False
    else:
        base_model.trainable = True
        for layer in base_model.layers[:-trainable_layers]:
            layer.trainable = False

    inputs = Input(shape=(224,224,3))
    x = base_model(inputs, training=False)
    x = GlobalAveragePooling2D()(x)
    x = Dense(128, activation='relu')(x)
    outputs = Dense(1, activation='sigmoid')(x)

    model = Model(inputs, outputs)

    return model

# ------------------------------------------------
# Train Model
# ------------------------------------------------
def train_model(model, x_train, y_train):

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

    history = model.fit(
```

```python
# ------------------------------------------------
# Plot Results
# ------------------------------------------------
def plot_history(history, name):

    # Loss
    plt.figure()
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.title(f"{name} Loss")
    plt.legend()
    plt.savefig(f"{name}_loss.png")
    plt.show()

    # Accuracy
    plt.figure()
    plt.plot(history.history['accuracy'], label='Train Acc')
    plt.plot(history.history['val_accuracy'], label='Val Acc')
    plt.title(f"{name} Accuracy")
    plt.legend()
    plt.savefig(f"{name}_accuracy.png")
    plt.show()

# ------------------------------------------------
# Main
# ------------------------------------------------
def main():

    x_train, x_test, y_train, y_test = load_data()

    # -------------------------------
    # Case 1: Freeze Whole VGG16
    # -------------------------------
    print("\n=== Transfer Learning (Freeze All Layers) ===")
    model1 = build_model(trainable_layers=None)
    history1 = train_model(model1, x_train, y_train)
    model1.evaluate(x_test, y_test)
    plot_history(history1, "freeze_all")

    # -------------------------------
    # Case 2: Partial Fine-Tuning
    # -------------------------------
    print("\n=== Partial Fine-Tuning (Last 4 Layers Trainable) ===")
    model2 = build_model(trainable_layers=4)
    history2 = train_model(model2, x_train, y_train)
    model2.evaluate(x_test, y_test)
    plot_history(history2, "partial_finetune")

    # -------------------------------
    # Case 3: Fine-Tune Whole VGG16
    # -------------------------------
    print("\n=== Fine-Tune Whole VGG16 ===")
    model3 = build_model(trainable_layers=len(VGG16().layers))
    history3 = train_model(model3, x_train, y_train)
    model3.evaluate(x_test, y_test)
    plot_history(history3, "full_finetune")


if __name__ == "__main__":
    main()
```

ASSIGNMENT 11

Discuss the feature extraction power of your favorite CNN pretrained by the ImageNet dataset before and after transfer learning by the MNIST digit dataset after plotting high dimensional feature vectors on 2D plane using the following two dimension reduction

Techniques:

- **Principal Component Analysis (PCA)**
- **t-distributed Stochastic Neighbor Embedding (t-SNE)**

In [ ]:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Input
from tensorflow.keras.models import Model
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE


# ------------------------------------------------
# Load MNIST
# ------------------------------------------------

def load_data():

    (x_train, y_train), _ = tf.keras.datasets.mnist.load_data()

    x_train = x_train[:2000]          # small subset for speed
    y_train = y_train[:2000]

    x_train = np.expand_dims(x_train, -1)
    x_train = np.repeat(x_train, 3, axis=-1)

    x_train = tf.image.resize(x_train, (224,224))
    x_train = preprocess_input(x_train)

    return x_train, y_train


# ------------------------------------------------
# Build Feature Extractor (Before Transfer Learning)
# ------------------------------------------------

def build_feature_extractor(trainable=False):

    base_model = VGG16(weights='imagenet',
                       include_top=False,
                       input_shape=(224,224,3))

    base_model.trainable = trainable

    inputs = Input(shape=(224,224,3))
    x = base_model(inputs)
    x = GlobalAveragePooling2D()(x)

    model = Model(inputs, x)

    return model


# ------------------------------------------------
# Fine-Tune Model on MNIST
# ------------------------------------------------

def fine_tune_model(x_train, y_train):

    base_model = VGG16(weights='imagenet',
                       include_top=False,
                       input_shape=(224,224,3))

    base_model.trainable = True

    inputs = Input(shape=(224,224,3))
    x = base_model(inputs)
    x = GlobalAveragePooling2D()(x)
    x = Dense(128, activation='relu')(x)
    outputs = Dense(10, activation='softmax')(x)

    model = Model(inputs, outputs)

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

    model.fit(x_train, y_train,
              epochs=3,
              batch_size=32,
              verbose=1)

    return model
```

```python
# ------------------------------------------------
# Extract Features
# ------------------------------------------------

def extract_features(model, x):

    features = model.predict(x, batch_size=32)
    return features


# ------------------------------------------------
# PCA Plot
# ------------------------------------------------

def plot_pca(features, labels, name):

    pca = PCA(n_components=2)
    reduced = pca.fit_transform(features)

    plt.figure()
    scatter = plt.scatter(reduced[:,0], reduced[:,1], c=labels)
    plt.title(f"{name} PCA")
    plt.colorbar(scatter)

    plt.savefig(f"{name}_pca.png")
    plt.show()


# ------------------------------------------------
# t-SNE Plot
# ------------------------------------------------

def plot_tsne(features, labels, name):

    tsne = TSNE(n_components=2, random_state=42)
    reduced = tsne.fit_transform(features)

    plt.figure()
    scatter = plt.scatter(reduced[:,0], reduced[:,1], c=labels)
    plt.title(f"{name} t-SNE")
    plt.colorbar(scatter)

    plt.savefig(f"{name}_tsne.png")
    plt.show()


# ------------------------------------------------
# Main
# ------------------------------------------------

def main():

    x_train, y_train = load_data()

    # ----------------------------
    # Before Transfer Learning
    # ----------------------------
    print("Extracting features BEFORE fine-tuning...")
    feature_model_before = build_feature_extractor(trainable=False)
    features_before = extract_features(feature_model_before, x_train)

    plot_pca(features_before, y_train, "before_transfer")
    plot_tsne(features_before, y_train, "before_transfer")


    # ----------------------------
    # After Transfer Learning
    # ----------------------------
    print("Fine-tuning on MNIST...")
    fine_tuned_model = fine_tune_model(x_train, y_train)

    feature_model_after = Model(
        fine_tuned_model.input,
        fine_tuned_model.layers[-3].output
    )

    features_after = extract_features(feature_model_after, x_train)

    plot_pca(features_after, y_train, "after_transfer")
    plot_tsne(features_after, y_train, "after_transfer")


if __name__ == "__main__":
    main()
```

ASSIGNMENT 12

Write a report by discussing the effect of different data augmentation techniques on your CNN based classifiers.

In [ ]:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Input
from tensorflow.keras.models import Model
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE


# ------------------------------------------------
# Load MNIST
# ------------------------------------------------

def load_data():

    (x_train, y_train), _ = tf.keras.datasets.mnist.load_data()

    x_train = x_train[:2000]          # small subset for speed
    y_train = y_train[:2000]

    x_train = np.expand_dims(x_train, -1)
    x_train = np.repeat(x_train, 3, axis=-1)

    x_train = tf.image.resize(x_train, (224,224))
    x_train = preprocess_input(x_train)

    return x_train, y_train


# ------------------------------------------------
# Build Feature Extractor (Before Transfer Learning)
# ------------------------------------------------

def build_feature_extractor(trainable=False):

    base_model = VGG16(weights='imagenet',
                       include_top=False,
                       input_shape=(224,224,3))

    base_model.trainable = trainable

    inputs = Input(shape=(224,224,3))
    x = base_model(inputs)
    x = GlobalAveragePooling2D()(x)

    model = Model(inputs, x)

    return model


# ------------------------------------------------
# Fine-Tune Model on MNIST
# ------------------------------------------------

def fine_tune_model(x_train, y_train):

    base_model = VGG16(weights='imagenet',
                       include_top=False,
                       input_shape=(224,224,3))

    base_model.trainable = True

    inputs = Input(shape=(224,224,3))
    x = base_model(inputs)
    x = GlobalAveragePooling2D()(x)
    x = Dense(128, activation='relu')(x)
    outputs = Dense(10, activation='softmax')(x)

    model = Model(inputs, outputs)

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

    model.fit(x_train, y_train,
              epochs=3,
              batch_size=32,
              verbose=1)

    return model
```

```python
# ------------------------------------------------
# Extract Features
# ------------------------------------------------

def extract_features(model, x):

    features = model.predict(x, batch_size=32)
    return features


# ------------------------------------------------
# PCA Plot
# ------------------------------------------------

def plot_pca(features, labels, name):

    pca = PCA(n_components=2)
    reduced = pca.fit_transform(features)

    plt.figure()
    scatter = plt.scatter(reduced[:,0], reduced[:,1], c=labels)
    plt.title(f"{name} PCA")
    plt.colorbar(scatter)

    plt.savefig(f"{name}_pca.png")
    plt.show()


# ------------------------------------------------
# t-SNE Plot
# ------------------------------------------------

def plot_tsne(features, labels, name):

    tsne = TSNE(n_components=2, random_state=42)
    reduced = tsne.fit_transform(features)

    plt.figure()
    scatter = plt.scatter(reduced[:,0], reduced[:,1], c=labels)
    plt.title(f"{name} t-SNE")
    plt.colorbar(scatter)

    plt.savefig(f"{name}_tsne.png")
    plt.show()


# ------------------------------------------------
# Main
# ------------------------------------------------

def main():

    x_train, y_train = load_data()

    # ----------------------------
    # Before Transfer Learning
    # ----------------------------
    print("Extracting features BEFORE fine-tuning...")
    feature_model_before = build_feature_extractor(trainable=False)
    features_before = extract_features(feature_model_before, x_train)

    plot_pca(features_before, y_train, "before_transfer")
    plot_tsne(features_before, y_train, "before_transfer")


    # ----------------------------
    # After Transfer Learning
    # ----------------------------
    print("Fine-tuning on MNIST...")
    fine_tuned_model = fine_tune_model(x_train, y_train)

    feature_model_after = Model(
        fine_tuned_model.input,
        fine_tuned_model.layers[-3].output
    )

    features_after = extract_features(feature_model_after, x_train)

    plot_pca(features_after, y_train, "after_transfer")
    plot_tsne(features_after, y_train, "after_transfer")


if __name__ == "__main__":
    main()
```

In [ ]:

```
#Data Augmentation Effect (Code)

import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Input, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator


# ------------------------------------------------
# Load Data
# ------------------------------------------------

def load_data():

    (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

    x_train = x_train.astype("float32") / 255.0
    x_test = x_test.astype("float32") / 255.0

    x_train = np.expand_dims(x_train, -1)
    x_test = np.expand_dims(x_test, -1)

    return x_train, x_test, y_train, y_test


# ------------------------------------------------
# Build CNN Model
# ------------------------------------------------

def build_model(input_shape):

    inputs = Input(shape=input_shape)

    x = Conv2D(32, (3,3), activation='relu')(inputs)
    x = MaxPooling2D((2,2))(x)

    x = Conv2D(64, (3,3), activation='relu')(x)
    x = MaxPooling2D((2,2))(x)

    x = Flatten()(x)
    x = Dense(128, activation='relu')(x)

    outputs = Dense(10, activation='softmax')(x)

    model = Model(inputs, outputs)

    return model


# ------------------------------------------------
# Train Without Augmentation
# ------------------------------------------------

def train_without_aug(model, x_train, y_train):

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

    history = model.fit(
        x_train, y_train,
        epochs=5,
        batch_size=64,
        validation_split=0.1,
        verbose=1
    )

    return history
```

ASSIGNMENT 13

Show the effect of dropout layer, data augmentation techniques on overfitting issues of your CNN based classifier.

In [ ]:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Input
from tensorflow.keras.models import Model
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE


# ------------------------------------------------
# Load MNIST
# ------------------------------------------------

def load_data():

    (x_train, y_train), _ = tf.keras.datasets.mnist.load_data()

    x_train = x_train[:2000]          # small subset for speed
    y_train = y_train[:2000]

    x_train = np.expand_dims(x_train, -1)
    x_train = np.repeat(x_train, 3, axis=-1)

    x_train = tf.image.resize(x_train, (224,224))
    x_train = preprocess_input(x_train)

    return x_train, y_train


# ------------------------------------------------
# Build Feature Extractor (Before Transfer Learning)
# ------------------------------------------------

def build_feature_extractor(trainable=False):

    base_model = VGG16(weights='imagenet',
                       include_top=False,
                       input_shape=(224,224,3))

    base_model.trainable = trainable

    inputs = Input(shape=(224,224,3))
    x = base_model(inputs)
    x = GlobalAveragePooling2D()(x)

    model = Model(inputs, x)

    return model


# ------------------------------------------------
# Fine-Tune Model on MNIST
# ------------------------------------------------

def fine_tune_model(x_train, y_train):

    base_model = VGG16(weights='imagenet',
                       include_top=False,
                       input_shape=(224,224,3))

    base_model.trainable = True

    inputs = Input(shape=(224,224,3))
    x = base_model(inputs)
    x = GlobalAveragePooling2D()(x)
    x = Dense(128, activation='relu')(x)
    outputs = Dense(10, activation='softmax')(x)

    model = Model(inputs, outputs)

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

    model.fit(x_train, y_train,
              epochs=3,
              batch_size=32,
              verbose=1)

    return model
```

```python
# ------------------------------------------------
# Extract Features
# ------------------------------------------------

def extract_features(model, x):

    features = model.predict(x, batch_size=32)
    return features


# ------------------------------------------------
# PCA Plot
# ------------------------------------------------

def plot_pca(features, labels, name):

    pca = PCA(n_components=2)
    reduced = pca.fit_transform(features)

    plt.figure()
    scatter = plt.scatter(reduced[:,0], reduced[:,1], c=labels)
    plt.title(f"{name} PCA")
    plt.colorbar(scatter)

    plt.savefig(f"{name}_pca.png")
    plt.show()


# ------------------------------------------------
# t-SNE Plot
# ------------------------------------------------

def plot_tsne(features, labels, name):

    tsne = TSNE(n_components=2, random_state=42)
    reduced = tsne.fit_transform(features)

    plt.figure()
    scatter = plt.scatter(reduced[:,0], reduced[:,1], c=labels)
    plt.title(f"{name} t-SNE")
    plt.colorbar(scatter)

    plt.savefig(f"{name}_tsne.png")
    plt.show()


# ------------------------------------------------
# Main
# ------------------------------------------------

def main():

    x_train, y_train = load_data()

    # ----------------------------
    # Before Transfer Learning
    # ----------------------------
    print("Extracting features BEFORE fine-tuning...")
    feature_model_before = build_feature_extractor(trainable=False)
    features_before = extract_features(feature_model_before, x_train)

    plot_pca(features_before, y_train, "before_transfer")
    plot_tsne(features_before, y_train, "before_transfer")


    # ----------------------------
    # After Transfer Learning
    # ----------------------------
    print("Fine-tuning on MNIST...")
    fine_tuned_model = fine_tune_model(x_train, y_train)

    feature_model_after = Model(
        fine_tuned_model.input,
        fine_tuned_model.layers[-3].output
    )

    features_after = extract_features(feature_model_after, x_train)

    plot_pca(features_after, y_train, "after_transfer")
    plot_tsne(features_after, y_train, "after_transfer")


if __name__ == "__main__":
    main()
```

In [ ]:

```
#Data Augmentation Effect (Code)

import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Input, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator


# ------------------------------------------------
# Load Data
# ------------------------------------------------

def load_data():

    (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

    x_train = x_train.astype("float32") / 255.0
    x_test = x_test.astype("float32") / 255.0

    x_train = np.expand_dims(x_train, -1)
    x_test = np.expand_dims(x_test, -1)

    return x_train, x_test, y_train, y_test


# ------------------------------------------------
# Build CNN Model
# ------------------------------------------------

def build_model(input_shape):

    inputs = Input(shape=input_shape)

    x = Conv2D(32, (3,3), activation='relu')(inputs)
    x = MaxPooling2D((2,2))(x)

    x = Conv2D(64, (3,3), activation='relu')(x)
    x = MaxPooling2D((2,2))(x)

    x = Flatten()(x)
    x = Dense(128, activation='relu')(x)

    outputs = Dense(10, activation='softmax')(x)

    model = Model(inputs, outputs)

    return model


# ------------------------------------------------
# Train Without Augmentation
# ------------------------------------------------

def train_without_aug(model, x_train, y_train):

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

    history = model.fit(
        x_train, y_train,
        epochs=5,
        batch_size=64,
        validation_split=0.1,
        verbose=1
    )

    return history
```

ASSIGNMENT 14

Write a report by discussing the effect of the following issues on the classifier's performance:

- **different activation functions in hidden layers**
- **different loss functions**

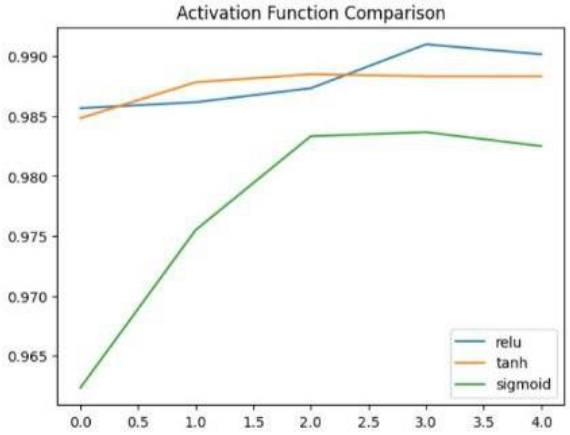
In [ ]:

```
#Activation & Loss Function Comparison

import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Input, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.utils import to_categorical


# ------------------------------------------------
# Load Data
# ------------------------------------------------

def load_data():

    (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

    x_train = x_train.astype("float32") / 255.0
    x_test = x_test.astype("float32") / 255.0

    x_train = np.expand_dims(x_train, -1)
    x_test = np.expand_dims(x_test, -1)

    y_train_cat = to_categorical(y_train, 10)
    y_test_cat = to_categorical(y_test, 10)

    return x_train, x_test, y_train, y_test, y_train_cat, y_test_cat


# ------------------------------------------------
# Build CNN
# ------------------------------------------------

def build_model(input_shape, activation_function):

    inputs = Input(shape=input_shape)

    x = Conv2D(32, (3,3), activation=activation_function)(inputs)
    x = MaxPooling2D((2,2))(x)

    x = Conv2D(64, (3,3), activation=activation_function)(x)
    x = MaxPooling2D((2,2))(x)

    x = Flatten()(x)
    x = Dense(128, activation=activation_function)(x)

    outputs = Dense(10, activation='softmax')(x)

    model = Model(inputs, outputs)

    return model


# ------------------------------------------------
# Train Model
# ------------------------------------------------

def train_model(model, x_train, y_train, loss_function):

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

    history = model.fit(
        x_train, y_train,
        epochs=5,
        batch_size=64,
        validation_split=0.1,
        verbose=1
    )

    return history


# ------------------------------------------------
# Plot Comparison
# ------------------------------------------------

def plot_activation_results(histories):

    plt.figure()
    for name, history in histories.items():
        plt.plot(history.history['val_accuracy'], label=name)

    plt.title("Activation Function Comparison")
    plt.legend()
    plt.savefig("activation_comparison.png")
    plt.show()
```

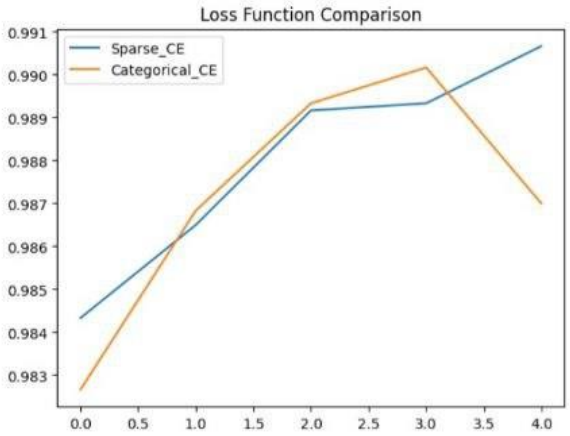
```python
def plot_loss_results(histories):

    plt.figure()
    for name, history in histories.items():
        plt.plot(history.history['val_accuracy'], label=name)

    plt.title("Loss Function Comparison")
    plt.legend()
    plt.savefig("loss_comparison.png")
    plt.show()


# ------------------------------------------------
# Main
# ------------------------------------------------

def main():

    x_train, x_test, y_train, y_test, y_train_cat, y_test_cat = load_data()

    # ----------------------------
    # Activation Function Comparison
    # ----------------------------
    activation_histories = {}

    for activation in ["relu", "tanh", "sigmoid"]:

        print(f"\nTraining with {activation} activation")
        model = build_model(x_train.shape[1:], activation)
        history = train_model(model, x_train, y_train, "sparse_categorical_crossentropy")
        activation_histories[activation] = history

    plot_activation_results(activation_histories)


    # ----------------------------
    # Loss Function Comparison
    # ----------------------------
    loss_histories = {}

    print("\nTraining with Sparse Categorical Crossentropy")
    model1 = build_model(x_train.shape[1:], "relu")
    history1 = train_model(model1, x_train, y_train, "sparse_categorical_crossentropy")
    loss_histories["Sparse_CE"] = history1

    print("\nTraining with Categorical Crossentropy")
    model2 = build_model(x_train.shape[1:], "relu")
    history2 = train_model(model2, x_train, y_train_cat, "categorical_crossentropy")
    loss_histories["Categorical_CE"] = history2

    plot_loss_results(loss_histories)


if __name__ == "__main__":
    main()
```

```
Training with relu activation
Epoch 1/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 8s 7ms/step - accuracy: 0.8816 - loss: 0.3963 - val_accuracy: 0.9857 - v
al_loss: 0.0533
Epoch 2/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9838 - loss: 0.0541 - val_accuracy: 0.9862 - v
al_loss: 0.0486
Epoch 3/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9893 - loss: 0.0339 - val_accuracy: 0.9873 - v
al_loss: 0.0489
Epoch 4/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9916 - loss: 0.0261 - val_accuracy: 0.9910 - v
al_loss: 0.0323
Epoch 5/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9945 - loss: 0.0170 - val_accuracy: 0.9902 - v
al_loss: 0.0361

Training with tanh activation
Epoch 1/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - accuracy: 0.8937 - loss: 0.3465 - val_accuracy: 0.9848 - v
al_loss: 0.0533
Epoch 2/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9858 - loss: 0.0486 - val_accuracy: 0.9878 - v
al_loss: 0.0459
Epoch 3/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9907 - loss: 0.0301 - val_accuracy: 0.9885 - v
al_loss: 0.0459
Epoch 4/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9948 - loss: 0.0197 - val_accuracy: 0.9883 - v
al_loss: 0.0464
Epoch 5/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9958 - loss: 0.0140 - val_accuracy: 0.9883 - v
al_loss: 0.0427
```

```
Training with sigmoid activation
Epoch 1/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - accuracy: 0.4361 - loss: 1.5811 - val_accuracy: 0.9623 - v
al_loss: 0.1475
Epoch 2/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9552 - loss: 0.1543 - val_accuracy: 0.9755 - v
al_loss: 0.0810
Epoch 3/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9735 - loss: 0.0908 - val_accuracy: 0.9833 - v
al_loss: 0.0635
Epoch 4/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9796 - loss: 0.0670 - val_accuracy: 0.9837 - v
al_loss: 0.0552
Epoch 5/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9854 - loss: 0.0498 - val_accuracy: 0.9825 - v
al_loss: 0.0598
```




```
Training with Sparse Categorical Crossentropy
Epoch 1/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.8959 - loss: 0.3683 - val_accuracy: 0.9843 - v
al_loss: 0.0558
Epoch 2/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9826 - loss: 0.0530 - val_accuracy: 0.9865 - v
al_loss: 0.0450
Epoch 3/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9887 - loss: 0.0368 - val_accuracy: 0.9892 - v
al_loss: 0.0397
Epoch 4/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9919 - loss: 0.0251 - val_accuracy: 0.9893 - v
al_loss: 0.0375
Epoch 5/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9942 - loss: 0.0186 - val_accuracy: 0.9907 - v
al_loss: 0.0378

Training with Categorical Crossentropy
Epoch 1/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.8867 - loss: 0.3858 - val_accuracy: 0.9827 - v
al_loss: 0.0622
Epoch 2/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9838 - loss: 0.0520 - val_accuracy: 0.9868 - v
al_loss: 0.0401
Epoch 3/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9883 - loss: 0.0361 - val_accuracy: 0.9893 - v
al_loss: 0.0388
Epoch 4/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9926 - loss: 0.0221 - val_accuracy: 0.9902 - v
al_loss: 0.0335
Epoch 5/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9945 - loss: 0.0167 - val_accuracy: 0.9870 - v
al_loss: 0.0492
```



ASSIGNMENT 15

Write a report by describing how different callback functions can make your training process better.

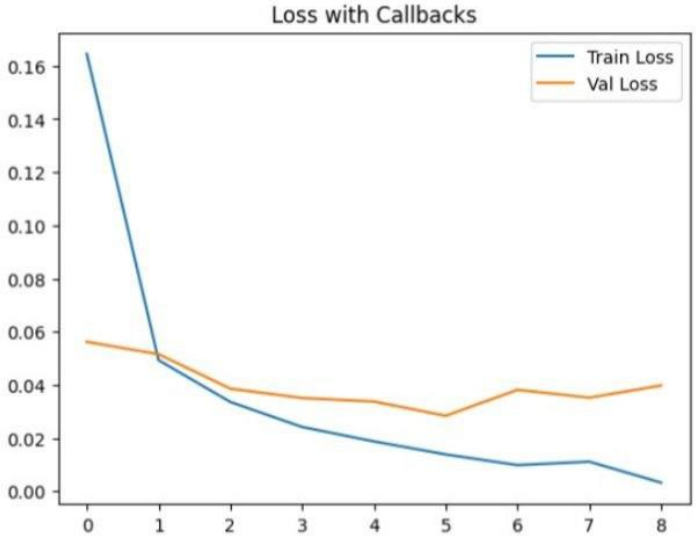
In [ ]:

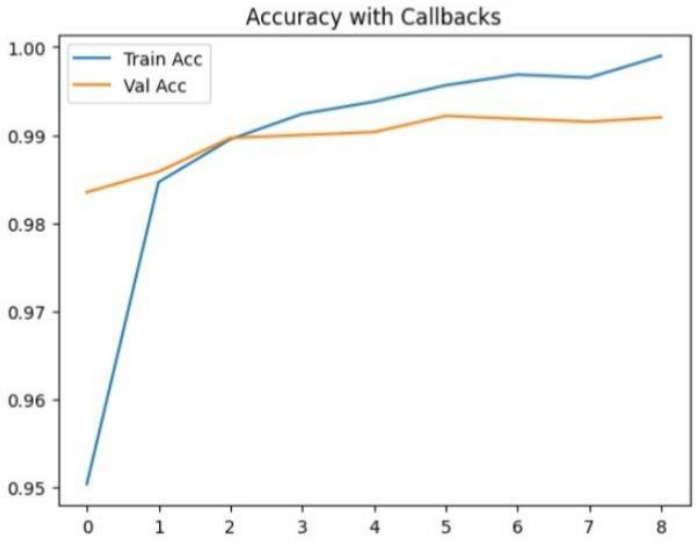
```
#Effect of Callback Functions on Training Process.

import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Input, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau


# ------------------------------------------------
# Load Data
# ------------------------------------------------

def load_data():

    (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

    x_train = x_train.astype("float32") / 255.0
    x_test = x_test.astype("float32") / 255.0

    x_train = np.expand_dims(x_train, -1)
    x_test = np.expand_dims(x_test, -1)

    return x_train, x_test, y_train, y_test


# ------------------------------------------------
# Build CNN Model
# ------------------------------------------------

def build_model(input_shape):

    inputs = Input(shape=input_shape)

    x = Conv2D(32, (3,3), activation='relu')(inputs)
    x = MaxPooling2D((2,2))(x)

    x = Conv2D(64, (3,3), activation='relu')(x)
    x = MaxPooling2D((2,2))(x)

    x = Flatten()(x)
    x = Dense(128, activation='relu')(x)

    outputs = Dense(10, activation='softmax')(x)

    model = Model(inputs, outputs)

    return model


# ------------------------------------------------
# Train Model with Callbacks
# ------------------------------------------------

def train_model(model, x_train, y_train):

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

    early_stop = EarlyStopping(
        monitor='val_loss',
        patience=3,
        restore_best_weights=True
    )

    checkpoint = ModelCheckpoint(
        "best_model.h5",
        monitor='val_accuracy',
        save_best_only=True
    )
```

```
    history = model.fit(
        x_train, y_train,
        epochs=20,
        batch_size=64,
        validation_split=0.1,
        callbacks=[early_stop, checkpoint, reduce_lr],
        verbose=1
    )

    return history


# ------------------------------------------------
# Plot Results
# ------------------------------------------------

def plot_results(history):

    # Loss Curve
    plt.figure()
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.legend()
    plt.title("Loss with Callbacks")
    plt.savefig("callbacks_loss.png")
    plt.show()

    # Accuracy Curve
    plt.figure()
    plt.plot(history.history['accuracy'], label='Train Acc')
    plt.plot(history.history['val_accuracy'], label='Val Acc')
    plt.legend()
    plt.title("Accuracy with Callbacks")
    plt.savefig("callbacks_accuracy.png")
    plt.show()


# ------------------------------------------------
# Main
# ------------------------------------------------

def main():

    x_train, x_test, y_train, y_test = load_data()

    model = build_model(x_train.shape[1:])

    history = train_model(model, x_train, y_train)

    model.evaluate(x_test, y_test)

    plot_results(history)


if __name__ == "__main__":
    main()
```

```
Epoch 1/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8841 - loss: 0.3781
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(mod
el)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `mo
del.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
844/844 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - accuracy: 0.8841 - loss: 0.3779 - val_accuracy: 0.9835 - v
al_loss: 0.0562 - learning_rate: 0.0010
Epoch 2/20
841/844 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9828 - loss: 0.0548
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(mod
el)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `mo
del.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9828 - loss: 0.0548 - val_accuracy: 0.9858 - v
al_loss: 0.0516 - learning_rate: 0.0010
Epoch 3/20
```

```
836/844 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9962 - loss: 0.0113
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')`.
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9962 - loss: 0.0113 - val_accuracy: 0.9922 - val_loss: 0.0284 - learning_rate: 0.0010
Epoch 7/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9973 - loss: 0.0091 - val_accuracy: 0.9918 - val_loss: 0.0382 - learning_rate: 0.0010
Epoch 8/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9963 - loss: 0.0116 - val_accuracy: 0.9915 - val_loss: 0.0352 - learning_rate: 0.0010
Epoch 9/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9989 - loss: 0.0034 - val_accuracy: 0.9920 - val_loss: 0.0398 - learning_rate: 5.0000e-04
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9891 - loss: 0.0330
```





ASSIGNMENT 16

Write a report describing how monitoring performance curves for both the training set and the validation set based on the target metric (e.g., 'accuracy') and 'loss' metric can improve your hyperparameter training.

In [ ]:

```
#Effect of Callback Functions on Training Process.

import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Input, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau


# ------------------------------------------------
# Load Data
# ------------------------------------------------

def load_data():

    (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

    x_train = x_train.astype("float32") / 255.0
    x_test = x_test.astype("float32") / 255.0

    x_train = np.expand_dims(x_train, -1)
    x_test = np.expand_dims(x_test, -1)

    return x_train, x_test, y_train, y_test


# ------------------------------------------------
# Build CNN Model
# ------------------------------------------------

def build_model(input_shape):

    inputs = Input(shape=input_shape)

    x = Conv2D(32, (3,3), activation='relu')(inputs)
    x = MaxPooling2D((2,2))(x)

    x = Conv2D(64, (3,3), activation='relu')(x)
    x = MaxPooling2D((2,2))(x)

    x = Flatten()(x)
    x = Dense(128, activation='relu')(x)

    outputs = Dense(10, activation='softmax')(x)

    model = Model(inputs, outputs)

    return model


# ------------------------------------------------
# Train Model with Callbacks
# ------------------------------------------------

def train_model(model, x_train, y_train):

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

    early_stop = EarlyStopping(
        monitor='val_loss',
        patience=3,
        restore_best_weights=True
    )

    checkpoint = ModelCheckpoint(
        "best_model.h5",
        monitor='val_accuracy',
        save_best_only=True
    )
```

```
    history = model.fit(
        x_train, y_train,
        epochs=20,
        batch_size=64,
        validation_split=0.1,
        callbacks=[early_stop, checkpoint, reduce_lr],
        verbose=1
    )

    return history


# ------------------------------------------------
# Plot Results
# ------------------------------------------------

def plot_results(history):

    # Loss Curve
    plt.figure()
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.legend()
    plt.title("Loss with Callbacks")
    plt.savefig("callbacks_loss.png")
    plt.show()

    # Accuracy Curve
    plt.figure()
    plt.plot(history.history['accuracy'], label='Train Acc')
    plt.plot(history.history['val_accuracy'], label='Val Acc')
    plt.legend()
    plt.title("Accuracy with Callbacks")
    plt.savefig("callbacks_accuracy.png")
    plt.show()


# ------------------------------------------------
# Main
# ------------------------------------------------

def main():

    x_train, x_test, y_train, y_test = load_data()

    model = build_model(x_train.shape[1:])

    history = train_model(model, x_train, y_train)

    model.evaluate(x_test, y_test)

    plot_results(history)


if __name__ == "__main__":
    main()
```

```
Epoch 1/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8841 - loss: 0.3781
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(mod
el)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `mo
del.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
844/844 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - accuracy: 0.8841 - loss: 0.3779 - val_accuracy: 0.9835 - v
al_loss: 0.0562 - learning_rate: 0.0010
Epoch 2/20
841/844 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9828 - loss: 0.0548
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(mod
el)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `mo
del.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9828 - loss: 0.0548 - val_accuracy: 0.9858 - v
al_loss: 0.0516 - learning_rate: 0.0010
Epoch 3/20
```

```
836/844 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9962 - loss: 0.0113
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')`.
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9962 - loss: 0.0113 - val_accuracy: 0.9922 - val_loss: 0.0284 - learning_rate: 0.0010
Epoch 7/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9973 - loss: 0.0091 - val_accuracy: 0.9918 - val_loss: 0.0382 - learning_rate: 0.0010
Epoch 8/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9963 - loss: 0.0116 - val_accuracy: 0.9915 - val_loss: 0.0352 - learning_rate: 0.0010
Epoch 9/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9989 - loss: 0.0034 - val_accuracy: 0.9920 - val_loss: 0.0398 - learning_rate: 5.0000e-04
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9891 - loss: 0.0330
```