NAME : ASHIKA.C

ROLL NO : 225229105

LAB - 7 Exploration of DNN design choices using MNIST dataset

STEPS

1.NUMBER OF NODES:

In [1]:
```
import tensorflow as tf
import keras
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Flatten
```

In [2]:
```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.lo
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras
-datasets/train-labels-idx1-ubyte.gz (https://storage.googleapis.com/ten
sorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz)
32768/29515 [=================================] - 0s 1us/step
40960/29515 [==========================================] - 0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras
-datasets/train-images-idx3-ubyte.gz (https://storage.googleapis.com/ten
sorflow/tf-keras-datasets/train-images-idx3-ubyte.gz)
26427392/26421880 [==============================] - 15s 1us/step
26435584/26421880 [==============================] - 15s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras
-datasets/t10k-labels-idx1-ubyte.gz (https://storage.googleapis.com/tens
orflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz)
16384/5148 [============================================================
==================================] - 0s 0s/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras
-datasets/t10k-images-idx3-ubyte.gz (https://storage.googleapis.com/tens
orflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz)
4423680/4422102 [==============================] - 3s 1us/step
4431872/4422102 [==============================] - 3s 1us/step
```

In [41]:

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist

# Load and preprocess the dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Iterate through different numbers of nodes
num_nodes_list = [4, 32, 128, 512, 2056]
for num_nodes in num_nodes_list:
    # Build the model
    model = tf.keras.models.Sequential([
        tf.keras.layers.Flatten(input_shape=(28, 28)),
        tf.keras.layers.Dense(num_nodes, activation='relu'),
        tf.keras.layers.Dense(10, activation='softmax')
    ])

    # Compile the model with an optimizer, loss function, and metric
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    # Print the number of parameters in the model
    print(f"Number of parameters with {num_nodes} nodes: {model.count_par

    # Train the model
    model.fit(x_train, y_train, epochs=10, verbose=0)

    # Evaluate on the test set
    test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=0)
    print(f"Test accuracy with {num_nodes} nodes: {test_accuracy:.4f}")
```

```
Number of parameters with 4 nodes: 3190
Test accuracy with 4 nodes: 0.8527
Number of parameters with 32 nodes: 25450
Test accuracy with 32 nodes: 0.9649
Number of parameters with 128 nodes: 101770
Test accuracy with 128 nodes: 0.9804
Number of parameters with 512 nodes: 407050
Test accuracy with 512 nodes: 0.9798
Number of parameters with 2056 nodes: 1634530
Test accuracy with 2056 nodes: 0.9800
```

2.NUMBER OF LAYERS:

In [42]:
```
import time
num_nodes = 32

# Iterate through different numbers of hidden layers
num_hidden_layers_list = [4, 6, 8, 16]
for num_hidden_layers in num_hidden_layers_list:
    # Build the model
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.Flatten(input_shape=(28, 28)))

    for _ in range(num_hidden_layers):
        model.add(tf.keras.layers.Dense(num_nodes, activation='relu'))

    model.add(tf.keras.layers.Dense(10, activation='softmax'))

    # Compile the model with an optimizer, loss function, and metric
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    # Print the number of parameters in the model
    num_params = model.count_params()
    print(f"Number of parameters with {num_hidden_layers} hidden layers:

    # Train the model and measure training time
    start_time = time.time()
    model.fit(x_train, y_train, epochs=10, verbose=0)
    end_time = time.time()

    # Calculate training time
    training_time = end_time - start_time
    print(f"Training time with {num_hidden_layers} hidden layers: {traini

    # Evaluate on the test set
    test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=0)
    print(f"Test accuracy with {num_hidden_layers} hidden layers: {test_a
```

```
Number of parameters with 4 hidden layers: 28618
Training time with 4 hidden layers: 48.02 seconds
Test accuracy with 4 hidden layers: 0.9621
Number of parameters with 6 hidden layers: 30730
Training time with 6 hidden layers: 49.79 seconds
Test accuracy with 6 hidden layers: 0.9665
Number of parameters with 8 hidden layers: 32842
Training time with 8 hidden layers: 61.57 seconds
Test accuracy with 8 hidden layers: 0.9697
Number of parameters with 16 hidden layers: 41290
Training time with 16 hidden layers: 79.49 seconds
Test accuracy with 16 hidden layers: 0.9635
```

3.ACTIVATION FUNCTION:

In [27]:

```
# Define the number of nodes in each hidden layer
num_nodes = 32

# Define activation functions to experiment with
activation_functions = ['sigmoid', 'tanh', 'relu']

# Iterate through different activation functions
for activation_function in activation_functions:
    # Build the model
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.Flatten(input_shape=(28, 28)))

    for _ in range(3):
        model.add(tf.keras.layers.Dense(num_nodes, activation=activation_

    model.add(tf.keras.layers.Dense(10, activation='softmax'))

    # Compile the model with an optimizer, loss function, and metric
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    # Print the activation function being used
    print(f"Activation function: {activation_function}")

    # Train the model for 10 epochs and measure training time
    start_time = time.time()
    model.fit(x_train, y_train, epochs=10, verbose=0)
    end_time = time.time()

    # Calculate training time
    training_time = end_time - start_time
    print(f"Training time: {training_time:.2f} seconds")

    # Evaluate on the test set
    test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=0)
    print(f"Test accuracy: {test_accuracy:.4f}\n")
```

```
Activation function: sigmoid
Training time: 40.20 seconds
Test accuracy: 0.9625

Activation function: tanh
Training time: 38.42 seconds
Test accuracy: 0.9675

Activation function: relu
Training time: 38.67 seconds
Test accuracy: 0.9690
```

4.ACTIVATION FUNCTION COMBINATIONS:

In [28]:

```
# Load and preprocess the dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Define activation function combinations to experiment with
activation_combinations = [
    ['sigmoid', 'relu', 'tanh'],
    ['relu', 'sigmoid', 'tanh'],
    ['tanh', 'relu', 'sigmoid'],
    # Add more combinations as needed
]

# Iterate through different activation function combinations
for combination in activation_combinations:
    # Build the model
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.Flatten(input_shape=(28, 28)))

    for activation_function in combination:
        model.add(tf.keras.layers.Dense(32, activation=activation_functio

    model.add(tf.keras.layers.Dense(10, activation='softmax'))

    # Compile the model with an optimizer, loss function, and metric
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    # Print the activation function combination being used
    print(f"Activation function combination: {combination}")

    # Train the model for 10 epochs and measure training time
    start_time = time.time()
    model.fit(x_train, y_train, epochs=10, verbose=0)
    end_time = time.time()

    # Calculate training time
    training_time = end_time - start_time
    print(f"Training time: {training_time:.2f} seconds")

    # Evaluate on the test set
    test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=0)
    print(f"Test accuracy: {test_accuracy:.4f}\n")
```

```
Activation function combination: ['sigmoid', 'relu', 'tanh']
Training time: 38.89 seconds
Test accuracy: 0.9663

Activation function combination: ['relu', 'sigmoid', 'tanh']
Training time: 40.27 seconds
Test accuracy: 0.9675

Activation function combination: ['tanh', 'relu', 'sigmoid']
Training time: 38.45 seconds
Test accuracy: 0.9655
```

5. LAYER-NODE COMBINATIONS:

In [29]:

```
# Load and preprocess the dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Define activation function combinations to experiment with
activation_combinations = [
    ['sigmoid', 'relu', 'tanh'],
    ['relu', 'sigmoid', 'tanh'],
    ['tanh', 'relu', 'sigmoid'],
    # Add more combinations as needed
]

# Iterate through different activation function combinations
for combination in activation_combinations:
    # Build the model
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.Flatten(input_shape=(28, 28)))

    for activation_function in combination:
        model.add(tf.keras.layers.Dense(32, activation=activation_functio

    model.add(tf.keras.layers.Dense(10, activation='softmax'))

    # Compile the model with an optimizer, loss function, and metric
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    # Print the activation function combination being used
    print(f"Activation function combination: {combination}")

    # Train the model for 10 epochs and measure training time
    start_time = time.time()
    model.fit(x_train, y_train, epochs=10, verbose=0)
    end_time = time.time()

    # Calculate training time
    training_time = end_time - start_time
    print(f"Training time: {training_time:.2f} seconds")

    # Evaluate on the test set
    test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=0)
    print(f"Test accuracy: {test_accuracy:.4f}\n")
```

```
Activation function combination: ['sigmoid', 'relu', 'tanh']
Training time: 37.81 seconds
Test accuracy: 0.9688

Activation function combination: ['relu', 'sigmoid', 'tanh']
Training time: 39.58 seconds
Test accuracy: 0.9679

Activation function combination: ['tanh', 'relu', 'sigmoid']
Training time: 45.14 seconds
Test accuracy: 0.9686
```

6.OPTIMIZER:

In [30]:

```
import time
import tensorflow as tf
from tensorflow.keras.datasets import mnist

# Load and preprocess the dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Define the number of nodes in each hidden layer
num_nodes = 32

# Common parameters
activation_function = 'relu'
epochs = 10

# List of optimizers to experiment with
optimizers = ['sgd', 'momentum', 'rmsprop', 'adam']

for optimizer in optimizers:
    print(f"Optimizer: {optimizer}")

    # Build the model
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.Flatten(input_shape=(28, 28)))

    for _ in range(3):
        model.add(tf.keras.layers.Dense(num_nodes, activation=activation_

    model.add(tf.keras.layers.Dense(10, activation='softmax'))

    # Compile the model with different optimizers
    if optimizer == 'sgd':
        optimizer_instance = tf.keras.optimizers.SGD(learning_rate=0.01)
    elif optimizer == 'momentum':
        optimizer_instance = tf.keras.optimizers.SGD(learning_rate=0.01,
    elif optimizer == 'rmsprop':
        optimizer_instance = tf.keras.optimizers.RMSprop(learning_rate=0.
    elif optimizer == 'adam':
        optimizer_instance = tf.keras.optimizers.Adam(learning_rate=0.001
    else:
        raise ValueError(f"Unknown optimizer: {optimizer}")

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

    # Train the model and measure training time
    start_time = time.time()
    model.fit(x_train, y_train, epochs=epochs, verbose=0)
    end_time = time.time()

    # Calculate training time
    training_time = end_time - start_time
    print(f"Training time: {training_time:.2f} seconds")

    # Evaluate on the test set
    test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=0)
```

```
    print(f"Test accuracy: {test_accuracy:.4f}\n")
```

```
Optimizer: sgd
Training time: 36.16 seconds
Test accuracy: 0.9565

Optimizer: momentum
Training time: 37.77 seconds
Test accuracy: 0.9702

Optimizer: rmsprop
Training time: 38.13 seconds
Test accuracy: 0.9682

Optimizer: adam
Training time: 38.30 seconds
Test accuracy: 0.9677
```

7.L1,L2 REGULARIZATION:

In [31]:

```
# Load and preprocess the dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Define the number of nodes in each hidden layer
num_nodes = 128

# Common parameters
activation_function = 'relu'
epochs = 10

# List of lambda values for L1 regularization
l1_lambda_values = [0.0001, 0.001, 0.01]

for l1_lambda in l1_lambda_values:
    print(f"L1 Regularization Lambda: {l1_lambda}")

    # Build the model with L1 regularization
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.Flatten(input_shape=(28, 28)))

    for _ in range(3):
        model.add(tf.keras.layers.Dense(num_nodes, activation=activation_

    model.add(tf.keras.layers.Dense(10, activation='softmax'))

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

    # Train the model and measure training time
    start_time = time.time()
    model.fit(x_train, y_train, epochs=epochs, verbose=0)
    end_time = time.time()

    # Calculate training time
    training_time = end_time - start_time
    print(f"Training time: {training_time:.2f} seconds")

    # Evaluate on the test set
    test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=0)
    print(f"Test accuracy: {test_accuracy:.4f}\n")
```

```
L1 Regularization Lambda: 0.0001
Training time: 54.76 seconds
Test accuracy: 0.9758

L1 Regularization Lambda: 0.001
Training time: 57.31 seconds
Test accuracy: 0.9532

L1 Regularization Lambda: 0.01
Training time: 55.78 seconds
Test accuracy: 0.1135
```

8.DROPOUT REGULARIZATION:

In [33]:

```
# Load and preprocess the dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Define the number of nodes in each hidden layer
num_nodes = 128

# Common parameters
activation_function = 'relu'
epochs = 10

# List of dropout rates to experiment with
dropout_rates = [0.2, 0.5, 0.8]

for dropout_rate in dropout_rates:
    print(f"Dropout Rate: {dropout_rate}")

    # Build the model with dropout layers
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.Flatten(input_shape=(28, 28)))

    for _ in range(3):
        model.add(tf.keras.layers.Dense(num_nodes, activation=activation_
        model.add(tf.keras.layers.Dropout(dropout_rate))

    model.add(tf.keras.layers.Dense(10, activation='softmax'))

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

    # Train the model and measure training time
    start_time = time.time()
    model.fit(x_train, y_train, epochs=epochs, verbose=0)
    end_time = time.time()

    # Calculate training time
    training_time = end_time - start_time
    print(f"Training time: {training_time:.2f} seconds")

    # Evaluate on the test set
    test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=0)
    print(f"Test accuracy: {test_accuracy:.4f}\n")
```

```
Dropout Rate: 0.2
Training time: 50.77 seconds
Test accuracy: 0.9800

Dropout Rate: 0.5
Training time: 50.82 seconds
Test accuracy: 0.9692

Dropout Rate: 0.8
Training time: 50.25 seconds
Test accuracy: 0.6489
```

10.DATASET SPLIT

In [36]:

```
# Load and preprocess the CIFAR-10 dataset
(x_train_full, y_train_full), (x_test_full, y_test_full) = cifar10.load_d
x_train_full, x_test_full = x_train_full / 255.0, x_test_full / 255.0

# Define the number of nodes in each hidden layer
num_nodes = 128

# Common parameters
activation_function = 'relu'
epochs = 10

dataset_sizes = [5000, 10000, 15000, 20000, 25000]

for size in dataset_sizes:
    print(f"Dataset Size: {size}")

    # Use a subset of the training and testing data
    x_train = x_train_full[:size]
    y_train = y_train_full[:size]
    x_test = x_test_full[:size]
    y_test = y_test_full[:size]

    # Build the model
    model = tf.keras.models.Sequential()
    model.add(tf.keras.layers.Flatten(input_shape=(32, 32, 3)))

    for _ in range(3):
        model.add(tf.keras.layers.Dense(num_nodes, activation=activation_

    model.add(tf.keras.layers.Dense(10, activation='softmax'))

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

    # Train the model and measure training time
    start_time = time.time()
    model.fit(x_train, y_train, epochs=epochs, verbose=0)
    end_time = time.time()

    # Calculate training time
    training_time = end_time - start_time
    print(f"Training time: {training_time:.2f} seconds")

    # Evaluate on the test set
    test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=0)
    print(f"Test accuracy: {test_accuracy:.4f}\n")
```

```
Dataset Size: 5000
Training time: 10.63 seconds
Test accuracy: 0.3872

Dataset Size: 10000
Training time: 15.06 seconds
Test accuracy: 0.3971

Dataset Size: 15000
Training time: 24.16 seconds
Test accuracy: 0.4127

Dataset Size: 20000
Training time: 29.13 seconds
Test accuracy: 0.4390

Dataset Size: 25000
Training time: 32.80 seconds
Test accuracy: 0.4421
```