VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JNANA SANGAMA”, BELAGAVI - 590 018

NEURAL NETWORKS AND DEEP LEARNING LABORATORY

(21AIL75) REPORT

Submitted by

Rayson M Fernandes **4SF21AD043**

In partial fulfilment of the requirements for the VII semester

of

BACHELOR OF ENGINEERING

in

ARTIFICIAL INTELLIGENCE & DATA SCIENCE

at

SAHYADRI

COLLEGE OF ENGINEERING & MANAGEMENT

An Autonomous Institution

Adyar, Mangaluru - 575 007

Academic Year: 2024 - 25

SAHYADRI
COLLEGE OF ENGINEERING & MANAGEMENT
An Autonomous Institution
MANGALURU

CERTIFICATE

This is to certify that Mr. Rayson M Fernandes bearing USN: 4SF21AD043 has satisfactorily completed the course of experiments in Neural Networks and Deep Learning Laboratory (21AIL75) in partial fulfilment of the requirements of VII semester of Bachelor of Engineering Degree Course in Artificial Intelligence & Data Science as prescribed by the Visvesvaraya Technological University during the year 2024-25.

Date:

Internal Assessment Marks

Faculty In charge

Experiment Details

Experiment No.	Experiment Name	Date of Execution	Page No.	Marks (10)
1	Activation Functions – sigmoid, tanh, ReLU and softmax	24/09/2024	1	
2	a. Single unit Perceptron b. AND, OR, XOR for single unit perception	01/10/2024	3	
3	Deep Feed-forward Neural Network	08/10/2024	7	
4	CNN to classify multi-category image dataset	15/10/2024	9	
5	Image classification model using Deep feed-forward neural network	07/11/2024	12	
6	Bi-Directional LSTM for Sentiment analysis	21/11/2024	14	
7	Standard VGG16 and 19 CNN architecture model to classify multicategory image dataset	05/12/2024	16	

Lab Experiment - 1

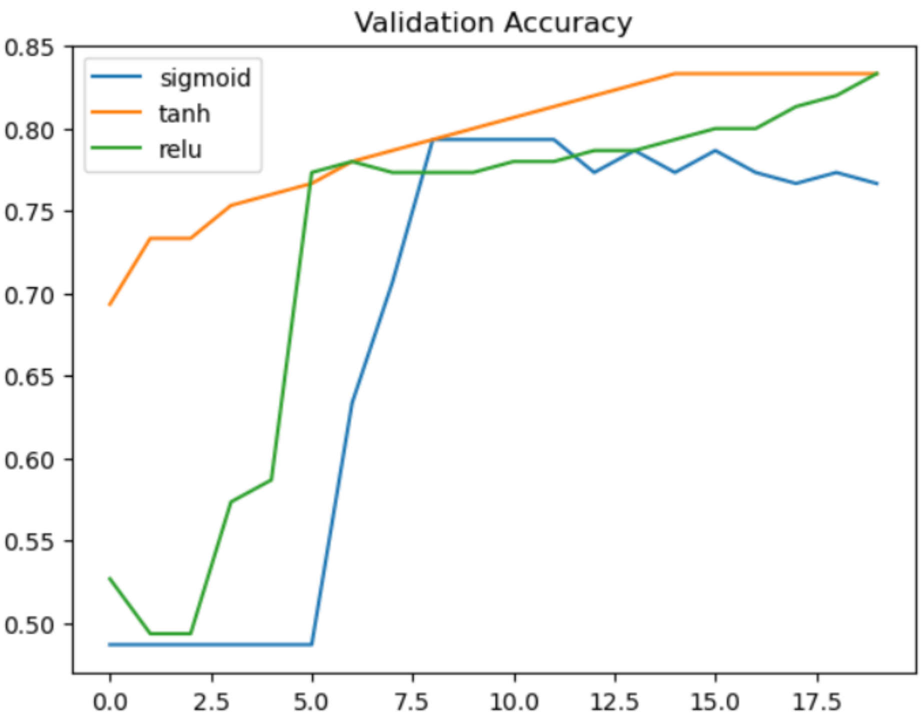
1. Write a program to demonstrate the working of different activation functions like sigmoid, tanh, ReLU & softmax to train a neural network.

Program:

```
import numpy as np
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt

# Dataset
X, y = make_moons(n_samples=500)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Compare activation functions
activations = ['sigmoid', 'tanh', 'relu']
histories = {}

for act in activations:
    model = Sequential([
        Dense(16, activation=act, input_dim=2),
        Dense(8, activation=act),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    histories[act] = model.fit(X_train,
                               y_train,
                               validation_data=(X_test, y_test),
```

```
                  epochs=20,
                  verbose=0).history

# Plot validation accuracy
for act in activations:
    plt.plot(histories[act]['val_accuracy'], label=act)
plt.legend()
plt.title('Validation Accuracy')
plt.show()
```

Output:



Lab Experiment - 2

2.

a. **Design a single unit perceptron for classification og linearly separable binary dataset without using pre-defined models. Use the perceptron from sklearn.**
b. **Identify the problems in single unit perceptron using AND, OR, XOR data and analyze the results.**

Program:

A.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.linear_model import Perceptron
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Generate a linearly separable binary dataset
X, y = make_classification(n_samples=100, n_features=2, n_classes=2,
                           n_informative=2, n_redundant=0, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the Perceptron
model = Perceptron(max_iter=1000, tol=1e-3)
model.fit(X_train, y_train)

# Evaluate and print accuracy
accuracy = model.score(X_test, y_test) * 100
print(f"Accuracy: {accuracy:.2f}%")

# Plot decision boundary
```

```
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.8, cmap=plt.cm.coolwarm)
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', cmap=plt.cm.coolwarm)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Perceptron Decision Boundary')
plt.show()
```

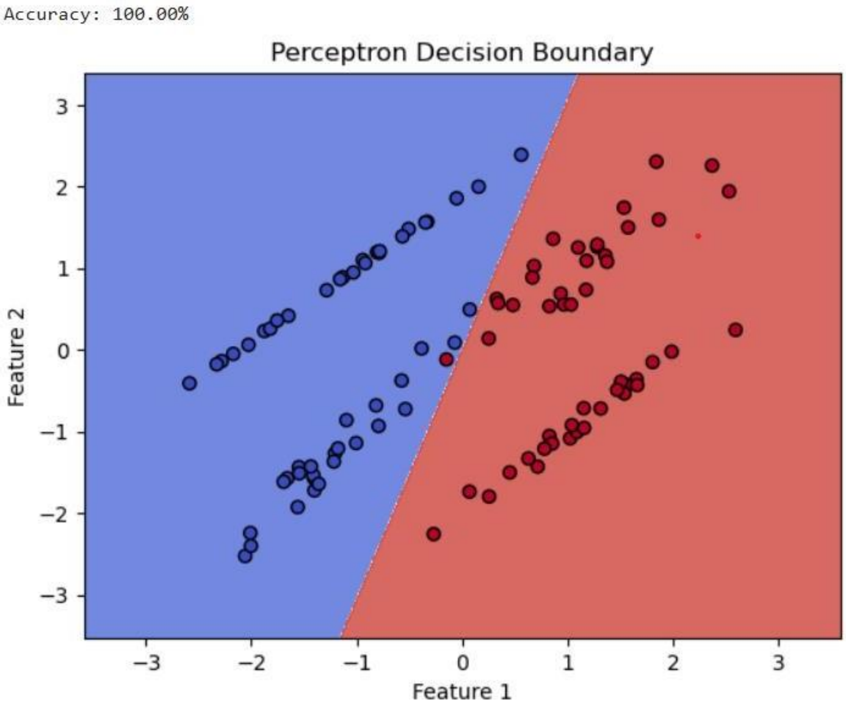
B.

```
from sklearn.linear_model import Perceptron
import numpy as np
import matplotlib.pyplot as plt

# Define the datasets
datasets = {
    "AND": (np.array([[0, 0], [0, 1], [1, 0], [1, 1]]), np.array([0, 0, 0, 1])),
    "OR": (np.array([[0, 0], [0, 1], [1, 0], [1, 1]]), np.array([0, 1, 1, 1])),
    "XOR": (np.array([[0, 0], [0, 1], [1, 0], [1, 1]]), np.array([0, 1, 1, 0]))
}

# Function to train and evaluate the perceptron
def train_and_evaluate(X, y, title):
    perceptron = Perceptron(max_iter=1000, eta0=1, random_state=0).fit(X, y)
    predictions = perceptron.predict(X)

    # Plot data and predictions
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', edgecolor='k', s=100)
```

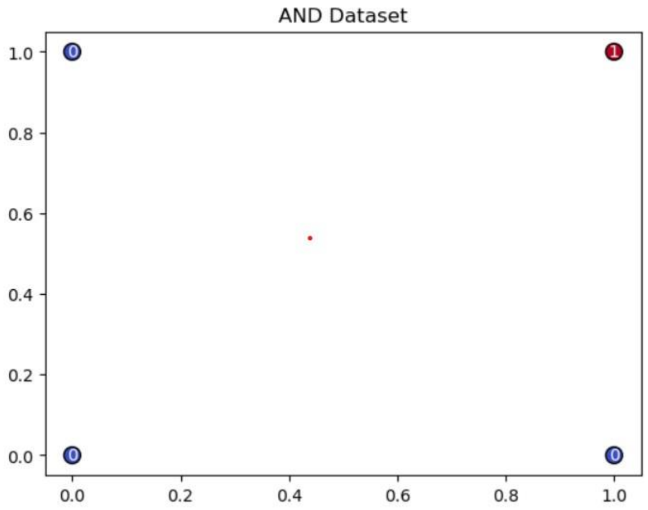
```
    for i, pred in enumerate(predictions):
        plt.text(X[i, 0], X[i, 1], str(pred), color='white', ha='center', va='center')
    plt.title(f'{title} Dataset')
    plt.show()

    print(f'{title} Accuracy: {np.mean(predictions == y) * 100:.2f}%\n')

# Evaluate datasets
for name, (X, y) in datasets.items():
    train_and_evaluate(X, y, name)
```

Output:

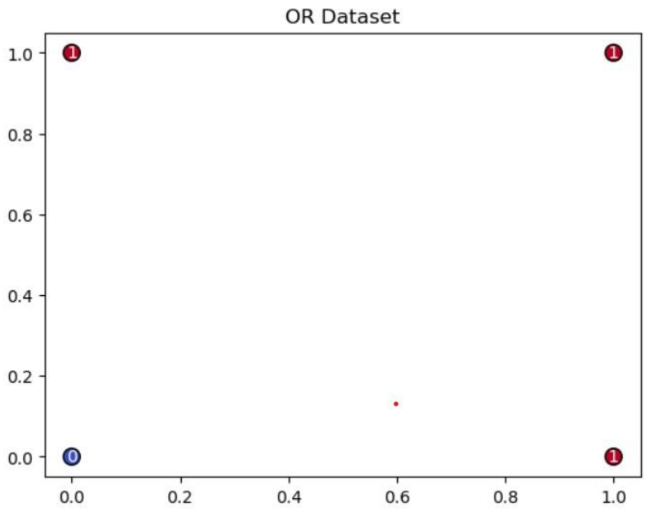
A.



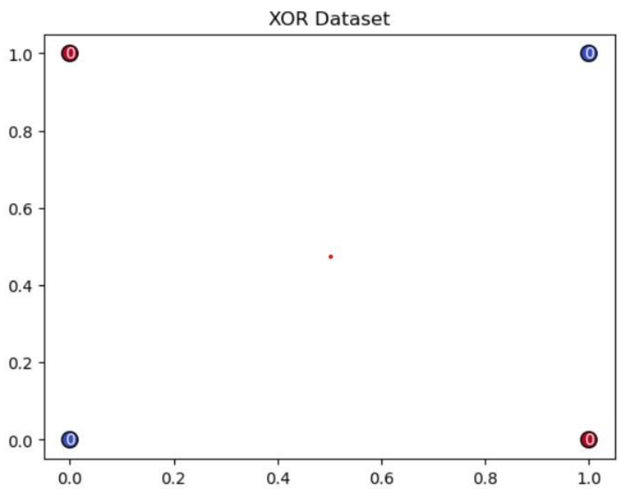
B.



AND Accuracy: 100.00%



OR Accuracy: 100.00%



XOR Accuracy: 50.00%

Lab Experiment - 3

3. Build a deep feed-forward Artificial Neural Network by implementing the back propagation algorithm and test the same using appropriate datasets. Use the number of hidden layers greater than or equal to 4.

Program:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Input
from tensorflow.keras.datasets import mnist

# Load and preprocess MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0  # Normalize inputs to [0, 1]

# Define the model
model = Sequential([
    Input(shape=(28, 28)),  # Define the input shape explicitly using Input layer
    Flatten(),  # Flatten 28x28 images into a vector
    Dense(128, activation='relu'),  # First hidden layer
    Dense(128, activation='relu'),  # Second hidden layer
    Dense(64, activation='relu'),   # Third hidden layer
    Dense(64, activation='relu'),   # Fourth hidden layer
    Dense(10, activation='softmax') # Output layer with 10 classes
])

# Compile and train the model
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
model.fit(x_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
print("\n Training complete! \n")
```

```
# Evaluate the model
loss, accuracy = model.evaluate(x_test, y_test)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")
```

Output:

```
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.8417 - loss: 0.5038 - val_accuracy: 0.9591 - val_loss: 0.1405
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9628 - loss: 0.1192 - val_accuracy: 0.9667 - val_loss: 0.1127
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9731 - loss: 0.0834 - val_accuracy: 0.9683 - val_loss: 0.1094
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9814 - loss: 0.0599 - val_accuracy: 0.9634 - val_loss: 0.1200
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9841 - loss: 0.0499 - val_accuracy: 0.9639 - val_loss: 0.1308
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9876 - loss: 0.0393 - val_accuracy: 0.9732 - val_loss: 0.1088
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9887 - loss: 0.0369 - val_accuracy: 0.9752 - val_loss: 0.1032
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9909 - loss: 0.0290 - val_accuracy: 0.9748 - val_loss: 0.1025
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9920 - loss: 0.0243 - val_accuracy: 0.9727 - val_loss: 0.1131
Epoch 10/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9930 - loss: 0.0234 - val_accuracy: 0.9740 - val_loss: 0.1150

 Training complete!

313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9678 - loss: 0.1184
Test Loss: 0.0973
Test Accuracy: 0.9746
```

Lab Experiment - 4

4. **Design and implement a CNN model with 4+ layers of convolution to classify multicategory image datasets. Use the concept of regularization and dropout while designing CNN model. Use the fashion MNIST datset. Record the training accuracy corresponding to the following architecture:**
    a. **Base model**
    b. **Model with L1 Regularization**
    c. **Model with L2 Regularization**
    d. **Model with Dropout**

Program:

```
import tensorflow as tf
from tensorflow.keras import layers, models, regularizers
from tensorflow.keras.datasets import fashion_mnist

# Load and preprocess the dataset
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
x_train = x_train[..., None]  # Add channel dimension
x_test = x_test[..., None]
y_train, y_test = tf.keras.utils.to_categorical(y_train, 10), tf.keras.utils.to_categorical(y_test, 10)

# Function to create the model with optional regularization or dropout
def create_model(regularizer=None, dropout_rate=None):
    model = models.Sequential([
        layers.Input(shape=(28, 28, 1)),
        layers.Conv2D(32, (3, 3), activation='relu'),
        layers.MaxPooling2D(),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D(),
```

```
        layers.Flatten(),
        layers.Dense(128, activation='relu', kernel_regularizer=regularizer),
        layers.Dropout(dropout_rate) if dropout_rate else layers.Dense(128, activation='relu'),
        layers.Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# List of configurations for model creation
configurations = [
    ("Base Model", None, None),
    ("Model with L1 Regularization", regularizers.l1(1e-4), None),
    ("Model with L2 Regularization", regularizers.l2(1e-4), None),
    ("Model with Dropout", None, 0.5)
]

# Train and evaluate each model configuration
for name, regularizer, dropout_rate in configurations:
    print(f"\nTraining {name}...")
    model = create_model(regularizer, dropout_rate)
    model.fit(x_train, y_train, epochs=5, batch_size=32, verbose=2)
    test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
    print(f"{name} Test Accuracy: {test_acc:.4f}")
```

Output:

```
Training Base Model...
Epoch 1/5
1875/1875 - 9s - 5ms/step - accuracy: 0.8291 - loss: 0.4624
Epoch 2/5
1875/1875 - 7s - 4ms/step - accuracy: 0.8901 - loss: 0.2992
Epoch 3/5
1875/1875 - 7s - 4ms/step - accuracy: 0.9044 - loss: 0.2553
Epoch 4/5
1875/1875 - 7s - 4ms/step - accuracy: 0.9146 - loss: 0.2247
Epoch 5/5
1875/1875 - 7s - 4ms/step - accuracy: 0.9254 - loss: 0.1987
313/313 - 1s - 4ms/step - accuracy: 0.9076 - loss: 0.2510
Base Model Test Accuracy: 0.9076
```

```
Training Model with L1 Regularization...
Epoch 1/5
1875/1875 - 10s - 5ms/step - accuracy: 0.8265 - loss: 0.6524
Epoch 2/5
1875/1875 - 8s - 4ms/step - accuracy: 0.8791 - loss: 0.4322
Epoch 3/5
1875/1875 - 8s - 4ms/step - accuracy: 0.8942 - loss: 0.3756
Epoch 4/5
1875/1875 - 8s - 4ms/step - accuracy: 0.9029 - loss: 0.3452
Epoch 5/5
1875/1875 - 8s - 4ms/step - accuracy: 0.9092 - loss: 0.3225
313/313 - 1s - 3ms/step - accuracy: 0.9034 - loss: 0.3442
Model with L1 Regularization Test Accuracy: 0.9034

Training Model with L2 Regularization...
Epoch 1/5
1875/1875 - 9s - 5ms/step - accuracy: 0.8322 - loss: 0.4804
Epoch 2/5
1875/1875 - 8s - 4ms/step - accuracy: 0.8899 - loss: 0.3252
Epoch 3/5
1875/1875 - 8s - 4ms/step - accuracy: 0.9051 - loss: 0.2883
Epoch 4/5
1875/1875 - 8s - 4ms/step - accuracy: 0.9145 - loss: 0.2627
Epoch 5/5
1875/1875 - 8s - 4ms/step - accuracy: 0.9226 - loss: 0.2441
313/313 - 1s - 4ms/step - accuracy: 0.9080 - loss: 0.2930
Model with L2 Regularization Test Accuracy: 0.9080
```

```
Training Model with Dropout...
Epoch 1/5
1875/1875 - 9s - 5ms/step - accuracy: 0.7993 - loss: 0.5546
Epoch 2/5
1875/1875 - 7s - 4ms/step - accuracy: 0.8637 - loss: 0.3780
Epoch 3/5
1875/1875 - 7s - 4ms/step - accuracy: 0.8822 - loss: 0.3244
Epoch 4/5
1875/1875 - 7s - 4ms/step - accuracy: 0.8928 - loss: 0.2936
Epoch 5/5
1875/1875 - 7s - 4ms/step - accuracy: 0.9013 - loss: 0.2739
313/313 - 1s - 3ms/step - accuracy: 0.9067 - loss: 0.2614
Model with Dropout Test Accuracy: 0.9067
```

Lab Experiment - 5

5. Design and implement an image classification model to classify a dataset of images using deep feed-forward neural network. Record the accuracy corresponding to the number of epochs. Use MNIST dataset.

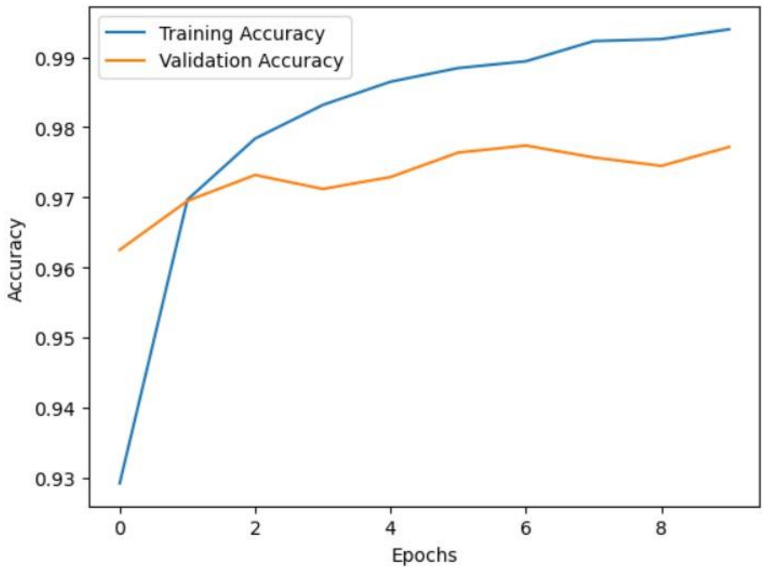
Program:

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt

# Load and preprocess the MNIST dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train, x_test = x_train.reshape(-1, 784) / 255.0, x_test.reshape(-1, 784) / 255.0

# Define and compile the model
model = Sequential([
    Dense(128, activation='relu', input_shape=(784,)),  # First hidden layer
    Dense(64, activation='relu'),  # Second hidden layer
    Dense(10, activation='softmax')  # Output layer with 10 neurons (for 10 classes)
])

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

# Train the model
history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=10,
batch_size=32)

# Plot accuracy
```

```
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Evaluate the model
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

Output:

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8726 - loss: 0.4361 - val_accuracy: 0.9625 - val_loss: 0.1233
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9674 - loss: 0.1059 - val_accuracy: 0.9695 - val_loss: 0.0961
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9782 - loss: 0.0714 - val_accuracy: 0.9732 - val_loss: 0.0851
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9836 - loss: 0.0554 - val_accuracy: 0.9712 - val_loss: 0.0983
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9873 - loss: 0.0406 - val_accuracy: 0.9729 - val_loss: 0.0929
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9895 - loss: 0.0315 - val_accuracy: 0.9764 - val_loss: 0.0821
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9907 - loss: 0.0263 - val_accuracy: 0.9774 - val_loss: 0.0832
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9932 - loss: 0.0210 - val_accuracy: 0.9757 - val_loss: 0.0945
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9934 - loss: 0.0190 - val_accuracy: 0.9745 - val_loss: 0.0982
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9942 - loss: 0.0178 - val_accuracy: 0.9772 - val_loss: 0.0849
```




```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9739 - loss: 0.0963
Test Accuracy: 97.72%
```

Lab Experiment - 6

6. Implement Bi-directional LSTM for Sentiment analysis on movie reviews.

Program:

```
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Bidirectional, LSTM, Dense

# Parameters
max_features = 10000  # Top 10000 most common words
max_len = 200  # Max length of each review

# Load and preprocess data
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
x_train, x_test = map(lambda x: pad_sequences(x, maxlen=max_len), (x_train, x_test))

# Build, compile, and train the model
model = Sequential([
    Embedding(input_dim=max_features, output_dim=64, input_length=max_len),
    Bidirectional(LSTM(32)),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5, batch_size=64, validation_split=0.2)

# Evaluate the model
print(f"Test Accuracy: {model.evaluate(x_test, y_test)[1]:.2f}")
```

```
# Test on a custom review

example_review = "The movie was absolutely amazing, I loved it!"

encoded_review = [imdb.get_word_index().get(word, 2) for word in
example_review.lower().split()]

padded_review = pad_sequences([encoded_review], maxlen=max_len)

prediction = model.predict(padded_review)[0][0]

print(f"{'Positive' if prediction < 0.5 else 'Negative'} sentiment with confidence {1 -
prediction if prediction < 0.5 else prediction:.2f}")
```

Output:

```
Epoch 1/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 20s 49ms/step - accuracy: 0.6815 - loss: 0.5702 - val_accuracy: 0.8542 - val_loss: 0.3453
Epoch 2/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 15s 48ms/step - accuracy: 0.9011 - loss: 0.2594 - val_accuracy: 0.8574 - val_loss: 0.3220
Epoch 3/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 14s 46ms/step - accuracy: 0.9356 - loss: 0.1778 - val_accuracy: 0.8550 - val_loss: 0.3749
Epoch 4/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 15s 46ms/step - accuracy: 0.9533 - loss: 0.1340 - val_accuracy: 0.8678 - val_loss: 0.3708
Epoch 5/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 15s 48ms/step - accuracy: 0.9655 - loss: 0.1010 - val_accuracy: 0.8236 - val_loss: 0.4593
782/782 ━━━━━━━━━━━━━━━━━━━━ 9s 11ms/step - accuracy: 0.8287 - loss: 0.4585
Test Accuracy: 0.83
WARNING:tensorflow:5 out of the last 5 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_d
istributed at 0x000002478E77FA60> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings
could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python
objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_re
tracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#
controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 362ms/step
Positive sentiment with confidence 0.74
```

Lab Experiment - 7

7. Implement the standard VGG16 and 19 CNN architecture model to classify multicategory image dataset and check the accuracy.

Program:

A.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.utils import to_categorical

# Load and preprocess Fashion MNIST dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()
x_train, x_test = x_train[..., None] / 255.0, x_test[..., None] / 255.0  # Normalize and add
channel
y_train, y_test = to_categorical(y_train, 10), to_categorical(y_test, 10)

# Define a simpler VGG-like model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

# Compile and train the model
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=5, batch_size=128)

# Evaluate the model
loss, accuracy = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {accuracy:.2f}")
```

B.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.utils import to_categorical

# Load and preprocess Fashion MNIST dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()
x_train, x_test = x_train[..., None] / 255.0, x_test[..., None] / 255.0  # Normalize and add channel
y_train, y_test = to_categorical(y_train, 10), to_categorical(y_test, 10)

# Define a VGG19-inspired model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

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

    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
```

```
    Dense(10, activation='softmax')
])

# Compile and train the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=5, batch_size=128)

# Evaluate the model
loss, accuracy = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {accuracy:.2f}")
```

Output:

A.

```
Epoch 1/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 10s 17ms/step - accuracy: 0.6738 - loss: 0.9151 - val_accuracy: 0.8505 - val_loss: 0.4138
Epoch 2/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 8s 17ms/step - accuracy: 0.8452 - loss: 0.4266 - val_accuracy: 0.8729 - val_loss: 0.3492
Epoch 3/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 7s 16ms/step - accuracy: 0.8716 - loss: 0.3583 - val_accuracy: 0.8845 - val_loss: 0.3161
Epoch 4/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 7s 15ms/step - accuracy: 0.8814 - loss: 0.3294 - val_accuracy: 0.8946 - val_loss: 0.2956
Epoch 5/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 7s 15ms/step - accuracy: 0.8908 - loss: 0.3004 - val_accuracy: 0.8849 - val_loss: 0.3072
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8859 - loss: 0.3101
Test Accuracy: 0.88
```

B.

```
Epoch 1/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 11s 19ms/step - accuracy: 0.6629 - loss: 0.9462 - val_accuracy: 0.8574 - val_loss: 0.3962
Epoch 2/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 8s 18ms/step - accuracy: 0.8542 - loss: 0.4047 - val_accuracy: 0.8816 - val_loss: 0.3274
Epoch 3/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 8s 17ms/step - accuracy: 0.8806 - loss: 0.3351 - val_accuracy: 0.8908 - val_loss: 0.2912
Epoch 4/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 9s 18ms/step - accuracy: 0.8949 - loss: 0.2979 - val_accuracy: 0.9027 - val_loss: 0.2668
Epoch 5/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 8s 18ms/step - accuracy: 0.9030 - loss: 0.2704 - val_accuracy: 0.9016 - val_loss: 0.2627
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9030 - loss: 0.2699
Test Accuracy: 0.90
```