#PROGRAM-1

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target.reshape(-1, 1)
# One-hot encoding the labels
encoder = OneHotEncoder()
y = encoder.fit_transform(y).toarray()
# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Building the neural network
model = Sequential()
model.add(Dense(10, input_shape=(X_train.shape[1],), activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(y.shape[1], activation='softmax'))
# Compiling the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
# Training the model
model.fit(X_train, y_train, epochs=50, batch_size=5, verbose=1)
# Evaluating the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Accuracy: {accuracy}')
```

#PROGRAM-2

```
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.metrics import accuracy_score
# Load the IMDB dataset
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000)
# Preprocess the data: pad sequences to ensure uniform length
X_train = pad_sequences(X_train, maxlen=200)
X_test = pad_sequences(X_test, maxlen=200)
# Convert the sequences to TF-IDF features
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train)
X_test_tfidf = tfidf_transformer.transform(X_test)
# Initialize and train the KNN classifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_tfidf, y_train)
# Make predictions
y_pred = knn.predict(X_test_tfidf)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

#PROGRAM-3

```
import numpy as np
import pandas as pd
import warnings
import tensorflow as tf  # Import TensorFlow
import numpy as np
import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import reuters
# Load the Reuters dataset
(train_data, train_labels), (test_data, test_labels) =
reuters.load_data(num_words=10000)
# Retrieve word index
word_index = reuters.get_word_index()
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
decoded_newswire = ' '.join([reverse_word_index.get(i - 3, '?') for i in train_data[0]])
# Vectorize sequences
```

```
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results
x_train = vectorize_sequences(train_data)  # 1
x_test = vectorize_sequences(test_data)    # 2
# One-hot encode labels
def to_one_hot(labels, dimension=46):
    results = np.zeros((len(labels), dimension))
    for i, label in enumerate(labels):
        results[i, label] = 1.
    return results
one_hot_train_labels = to_one_hot(train_labels)  # 1
one_hot_test_labels = to_one_hot(test_labels)    # 2
from tensorflow.keras.utils import to_categorical
one_hot_train_labels = to_categorical(train_labels)
one_hot_test_labels = to_categorical(test_labels)
# Define the model
model = keras.Sequential([
    layers.Dense(64, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(46, activation='softmax')
])
# Compile the model
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
# Validation and partial training sets
x_val = x_train[:1000]
partial_x_train = x_train[1000:]
y_val = one_hot_train_labels[:1000]
partial_y_train = one_hot_train_labels[1000:]
# Train the model
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=25,
                    batch_size=512,
                    validation_data=(x_val, y_val))
# Evaluate the model
results = model.evaluate(x_test, one_hot_test_labels)
```

#PROGRAM-4

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
# Load the MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()
# Preprocess the data
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32') / 255
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32') / 255
# Convert labels to one-hot encoding
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
# Build the CNN model
model = Sequential([
Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)),
MaxPooling2D(pool_size=(2, 2)),
Conv2D(64, kernel_size=(3, 3), activation='relu'),
MaxPooling2D(pool_size=(2, 2)),
Flatten(),
Dense(128, activation='relu'),
Dense(10, activation='softmax')
])
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Accuracy: {accuracy}')
```

#PROGRAM-5

```
import tensorflow as tf
import tensorflow_datasets as tfds
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
import matplotlib.pyplot as plt
# Load dataset
dataset_name = 'cats_vs_dogs'
(data_train, data_test), dataset_info = tfds.load(
    dataset_name,
    split=['train[:80%]', 'train[80%:]'],
    as_supervised=True,  # Include labels
    with_info=True       # Include dataset info
)
# Data preprocessing
IMG_SIZE = 150
def preprocess_image(image, label):
    image = tf.image.resize(image, (IMG_SIZE, IMG_SIZE))
    image = image / 255.0  # Normalize pixel values
    return image, label
train_dataset = data_train.map(preprocess_image).shuffle(1000).batch(32).prefetch(1)
test_dataset = data_test.map(preprocess_image).batch(32).prefetch(1)
# Visualize a few samples
def plot_samples(dataset, n_samples=5):
    plt.figure(figsize=(12, 8))
    for i, (image, label) in enumerate(dataset.take(n_samples)):
        ax = plt.subplot(1, n_samples, i + 1)
        plt.imshow(image.numpy())
        plt.title('Cat' if label.numpy() == 0 else 'Dog')
        plt.axis('off')
    plt.show()
plot_samples(data_train.map(preprocess_image))
# Build CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_SIZE, IMG_SIZE, 3)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
```

```
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')  # Binary classification
])
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
# Train the model
history = model.fit(
    train_dataset,
    validation_data=test_dataset,
    epochs=10
)
# Evaluate the model
loss, accuracy = model.evaluate(test_dataset)
print(f"Test Accuracy: {accuracy:.2f}")
# Plot training history
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(acc))
plt.figure(figsize=(12, 8))
plt.plot(epochs, acc, 'r', label='Training Accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.figure(figsize=(12, 8))
plt.plot(epochs, loss, 'r', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```

#PROGRAM-6

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

```
from tensorflow.keras.datasets import imdb
# Load the IMDb dataset
vocab_size = 10000
max_length = 100
(train_data, train_labels), (test_data, test_labels) =
imdb.load_data(num_words=vocab_size)
# Preprocess the data: Pad sequences to ensure uniform input size
train_padded = pad_sequences(train_data, maxlen=max_length, padding='post',
truncating='post')
test_padded = pad_sequences(test_data, maxlen=max_length, padding='post',
truncating='post')
# Build the RNN model
model = Sequential([
Embedding(input_dim=vocab_size, output_dim=32, input_length=max_length),  #
Embedding
layer
LSTM(64),  # LSTM layer with 64 units
Dropout(0.5),  # Dropout for regularization
Dense(1, activation='sigmoid')  # Output layer for binary classification
])
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# Train the model
model.fit(train_padded, train_labels, epochs=10, validation_data=(test_padded,
test_labels))
```