GURU TEGH BAHADUR INSTITUTE OF TECHNOLOGY

Supervised & Deep Learning

Practical File

SUBMITTED TO: - Ms. Shipra Thareja

NAME: - Akkshar Chauhan

BRANCH: - IT-1(ML & DA)

ENROLLMENT NO: - 03513203121

Index

Sr. No.	List Of Experiment	Date	Signature
1	Linear regression: Implement linear regression on a dataset and evaluate the model's performance.		
2	Logistic regression: Implement logistic regression on a binary classification dataset and evaluate the model's performance.		
3	k-Nearest Neighbors (k-NN): Implement k-NN algorithm on a dataset and evaluate the model's performance.		
4	Decision Trees: Implement decision trees on a dataset and evaluate the model's performance.		
5	Random Forest: Implement random forest algorithm on a dataset and evaluate the model's performance.		
6	Support Vector Machines (SVM): Implement SVM on a dataset and evaluate the model's performance.		
7	Naive Bayes: Implement Naive Bayes algorithm on a dataset and evaluate the model's performance.		
8	Gradient Boosting: Implement gradient boosting algorithm on a dataset and evaluate the model's performance.		
9	Convolutional Neural Networks (CNN): Implement CNN on an image classification dataset and evaluate the model's performance.		
10	Recurrent Neural Networks (RNN): Implement RNN on a text classification dataset and evaluate the model's performance.		
11	Long Short-Term Memory Networks (LSTM): Implement LSTM on a time-series dataset and evaluate the model's performance.		
12	Autoencoders: Implement autoencoders on an image dataset and evaluate the model's performance.		
13	Transfer Learning: Implement transfer learning on an image dataset and evaluate the model's performance.		
14	Reinforcement Learning: Implement reinforcement learning on a game environment and evaluate the model's performance.		

Experiment 1

Aim: Linear regression: Implement linear regression on a dataset and evaluate the model's performance.

Dataset Used:

```
Index(['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup',
       'Latitude', 'Longitude'],
      dtype='object')
```

Code:

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import pandas as pd
import matplotlib.pyplot as plt

# Load the California Housing dataset
california = fetch_california_housing(as_frame=True)
X = california.data
y = california.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Initialize and train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"R² Score: {r2}")
```

```python
# Display actual vs predicted values
results = pd.DataFrame({"Actual": y_test, "Predicted": y_pred})
print(results.head())

# Plot actual vs predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.7)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Linear Regression - Actual vs Predicted")
plt.show()

# Plot residuals
residuals = y_test - y_pred
plt.figure(figsize=(10, 6))
plt.scatter(y_pred, residuals, alpha=0.7)
plt.axhline(y=0, color="red", linestyle="--")
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Residuals vs Predicted Values")
plt.show()
```

Output:

```
Mean Squared Error (MSE): 0.5305677824766755
Mean Absolute Error (MAE): 0.5272474538305955
R² Score: 0.5957702326061662
        Actual   Predicted
20046  0.47700   0.726049
3024   0.45800   1.767434
15663  5.00001   2.710922
20484  2.18600   2.835147
9814   2.78000   2.606958
```

Experiment 2

Aim: Logistic regression: Implement logistic regression on a binary classification dataset and evaluate the model's performance.

Dataset Used:

```
...   Index(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
             'mean smoothness', 'mean compactness', 'mean concavity',
             'mean concave points', 'mean symmetry', 'mean fractal dimension',
             'radius error', 'texture error', 'perimeter error', 'area error',
             'smoothness error', 'compactness error', 'concavity error',
             'concave points error', 'symmetry error', 'fractal dimension error',
             'worst radius', 'worst texture', 'worst perimeter', 'worst area',
             'worst smoothness', 'worst compactness', 'worst concavity',
             'worst concave points', 'worst symmetry', 'worst fractal dimension'],
            dtype='object')
```

Code:

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load the Breast Cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Initialize and train the Logistic Regression model
model = LogisticRegression(max_iter=10000, solver='lbfgs')
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
```

```python
print(f"Accuracy: {accuracy:.2f}")

# Display classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=data.target_names))

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
conf_df = pd.DataFrame(conf_matrix, index=["Benign", "Malignant"],
columns=["Predicted Benign", "Predicted Malignant"])

# Visualize the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_df, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

Output:

```
Accuracy: 0.98

Classification Report:
              precision    recall  f1-score   support

   malignant       0.97      0.97      0.97        63
      benign       0.98      0.98      0.98       108

    accuracy                           0.98       171
   macro avg       0.97      0.97      0.97       171
weighted avg       0.98      0.98      0.98       171
```

Experiment 3

Aim: k-Nearest Neighbors (k-NN): Implement k-NN algorithm on a dataset and evaluate the model's performance.

Dataset Used:

```
Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)'],
      dtype='object')
```

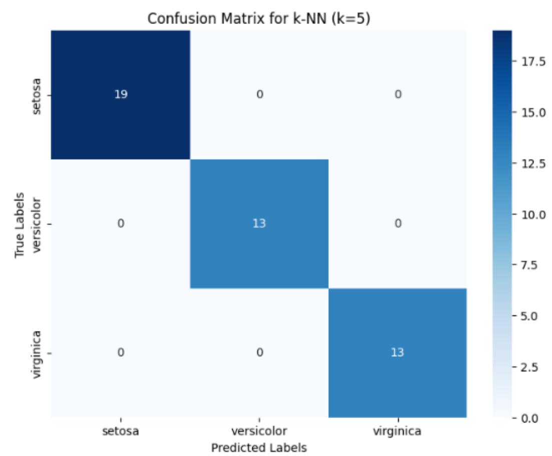
Code:

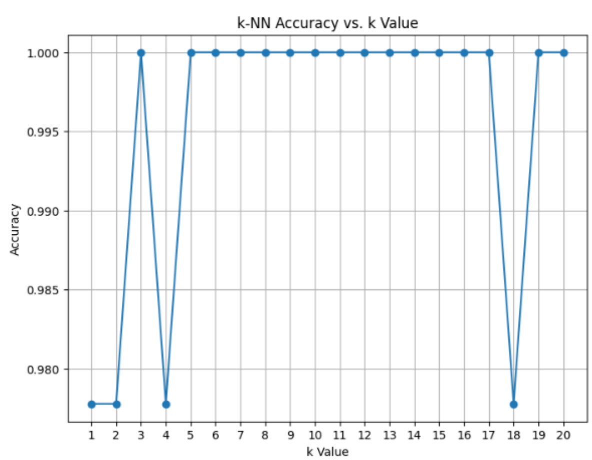
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize and train the k-NN model
k = 5  # Number of neighbors
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)

# Make predictions
y_pred = knn.predict(X_test)

# Evaluate the model
```

```python
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
conf_df = pd.DataFrame(conf_matrix, index=iris.target_names,
columns=iris.target_names)

# Visualize the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_df, annot=True, fmt="d", cmap="Blues")
plt.title(f"Confusion Matrix for k-NN (k={k})")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()

# Visualize accuracy vs. different k values
k_values = range(1, 21)
accuracies = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred_k = knn.predict(X_test)
    accuracies.append(accuracy_score(y_test, y_pred_k))

plt.figure(figsize=(8, 6))
plt.plot(k_values, accuracies, marker='o')
plt.title("k-NN Accuracy vs. k Value")
plt.xlabel("k Value")
plt.ylabel("Accuracy")
plt.xticks(k_values)
plt.grid()
plt.show()
```

Output:

```
Accuracy: 1.00

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```





Experiment 4

Aim: Decision Trees: Implement decision trees on a dataset and evaluate the model's performance.

Dataset Used:

```
Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)'],
      dtype='object')
```

Code:

```python
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Initialize and train the Decision Tree model
decision_tree = DecisionTreeClassifier(random_state=42)
decision_tree.fit(X_train, y_train)

# Make predictions
y_pred = decision_tree.predict(X_test)

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Print detailed classification report
report = classification_report(y_test, y_pred, target_names=iris.target_names)
print("Classification Report:\n", report)
```

Output:

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

Experiment 5

Aim: Random Forest: Implement random forest algorithm on a dataset and evaluate the model's performance.

Dataset Used:

```
Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)'],
      dtype='object')
```

Code:

```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Initialize and train the Random Forest model
random_forest = RandomForestClassifier(random_state=42, n_estimators=100)
random_forest.fit(X_train, y_train)

# Make predictions
y_pred = random_forest.predict(X_test)

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Print detailed classification report
report = classification_report(y_test, y_pred, target_names=iris.target_names)
print("Classification Report:\n", report)
```

Output:

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

Experiment 6

Aim: Support Vector Machines (SVM): Implement SVM on a dataset and evaluate the model's performance.

Dataset Used:

```
Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)'],
      dtype='object')
```

Code:

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42, stratify=y)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize and train the SVM model
model = SVC(kernel='linear', C=1.0, random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
```

```python
print(f"Accuracy: {accuracy:.2f}")

# Classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
conf_df = pd.DataFrame(conf_matrix, index=iris.target_names,
columns=iris.target_names)

# Visualize the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_df, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix for SVM")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

Output:

```
Accuracy: 0.91

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       0.82      0.93      0.88        15
   virginica       0.92      0.80      0.86        15

    accuracy                           0.91        45
   macro avg       0.92      0.91      0.91        45
weighted avg       0.92      0.91      0.91        45
```

Experiment 7

Aim: Naive Bayes: Implement Naive Bayes algorithm on a dataset and evaluate the model's performance.

Dataset Used:

```
Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)'],
      dtype='object')
```

Code:

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix

# Load dataset
dataset = load_iris()
X = dataset.data
y = dataset.target

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=0)

# Create and fit the SVM model
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(X_train, y_train)

# Make predictions
y_pred = classifier.predict(X_test)

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)


# Cross-validation
accuracies = cross_val_score(estimator=classifier, X=X_train, y=y_train, cv=10)
print("Accuracy: {:.2f}%".format(accuracies.mean() * 100))
print("Standard Deviation: {:.2f}%".format(accuracies.std() * 100))
```

Output:

```
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
Accuracy: 98.18 %
Standard Deviation: 3.64 %
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

Experiment 8

Aim: Gradient Boosting: Implement gradient boosting algorithm on a dataset and evaluate the model's performance.

Dataset Used:

```
Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)'],
      dtype='object')
```

Code:

```
from sklearn.datasets import load_iris
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Initialize and train the Gradient Boosting model
gradient_boosting = GradientBoostingClassifier(random_state=42, n_estimators=100,
learning_rate=0.1, max_depth=3)
gradient_boosting.fit(X_train, y_train)

# Make predictions
y_pred = gradient_boosting.predict(X_test)

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Print detailed classification report
report = classification_report(y_test, y_pred, target_names=iris.target_names)
print("Classification Report:\n", report)
```

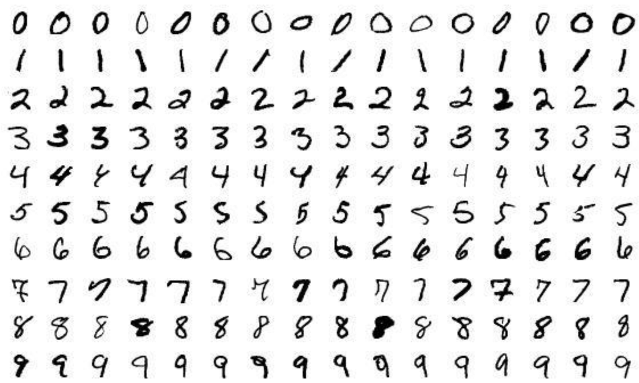
Output:

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

Experiment 9

Aim: Convolutional Neural Networks (CNN): Implement CNN on an image classification dataset and evaluate the model's performance.

Dataset Used:



Code:

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import classification_report

# 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 = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Build the CNN model
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
```

```python
model.add(layers.Dense(10, activation='softmax'))

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

# Train the model
history = model.fit(X_train, y_train, epochs=5, batch_size=64,
validation_split=0.2)

# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_accuracy}")

# Generate predictions and classification report
y_pred = model.predict(X_test)
y_pred_classes = y_pred.argmax(axis=1)
y_test_classes = y_test.argmax(axis=1)

print("Classification Report:\n", classification_report(y_test_classes,
y_pred_classes))
```

Output:

```
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8567 - loss: 0.4793 - val_accuracy: 0.9761 - val_loss: 0.0802
Epoch 2/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9814 - loss: 0.0596 - val_accuracy: 0.9834 - val_loss: 0.0569
Epoch 3/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9883 - loss: 0.0384 - val_accuracy: 0.9872 - val_loss: 0.0447
Epoch 4/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9912 - loss: 0.0277 - val_accuracy: 0.9876 - val_loss: 0.0425
Epoch 5/5
750/750 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9925 - loss: 0.0230 - val_accuracy: 0.9864 - val_loss: 0.0441
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 932us/step - accuracy: 0.9835 - loss: 0.0422
Test Accuracy: 0.9872999787330627
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Classification Report:
               precision    recall  f1-score   support

           0       0.99      0.99      0.99       980
           1       1.00      0.99      0.99      1135
           2       0.97      1.00      0.98      1032
           3       0.99      0.99      0.99      1010
           4       0.98      0.99      0.99       982
           5       0.99      0.98      0.99       892
           6       0.98      0.99      0.99       958
           7       0.99      0.97      0.98      1028
           8       0.99      0.99      0.99       974
...
    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

Experiment 10

Aim: Recurrent Neural Networks (RNN): Implement RNN on a text classification dataset and evaluate the model's performance.

Dataset Used: IMDB Movie Reviews

```
so i loved the fact there was a real connection with this film the witty remarks throughout the film were grea
ovie up at target for 5 because i figured hey it's sandler i can get some cheap laughs i was wrong completely
lms of the 1990s when my friends i were watching this film being the target audience it was aimed at we just s
```

Code:

```python
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense

# Load the IMDB dataset
max_features = 10000  # Number of unique words to consider
maxlen = 200  # Cut texts after this number of words
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=max_features)

# Preprocess the data (padding sequences)
X_train = pad_sequences(X_train, maxlen=maxlen)
X_test = pad_sequences(X_test, maxlen=maxlen)

# Build the RNN model
model = Sequential()
model.add(Embedding(max_features, 32, input_length=maxlen))  # Embedding layer
model.add(SimpleRNN(32, return_sequences=False))  # RNN layer
model.add(Dense(1, activation='sigmoid'))  # Output layer

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

# Train the model
history = model.fit(X_train, y_train,
                    epochs=5,
                    batch_size=64,
                    validation_split=0.2)

# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test)
```

```python
print(f"Test Accuracy: {test_accuracy}")

# Generate predictions
y_pred = (model.predict(X_test) > 0.5).astype("int32")

# Display classification report
from sklearn.metrics import classification_report
print("Classification Report:\n", classification_report(y_test, y_pred))
```

Output:

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - accuracy: 0.5581 - loss: 0.6732 - val_accuracy: 0.8236 - val_loss: 0.4065
Epoch 2/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 0.8484 - loss: 0.3645 - val_accuracy: 0.7540 - val_loss: 0.5109
Epoch 3/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.8821 - loss: 0.3020 - val_accuracy: 0.8286 - val_loss: 0.4128
Epoch 4/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.9567 - loss: 0.1325 - val_accuracy: 0.8306 - val_loss: 0.4639
Epoch 5/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.9843 - loss: 0.0552 - val_accuracy: 0.8066 - val_loss: 0.5827
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8110 - loss: 0.5539
Test Accuracy: 0.8121200203895569
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
Classification Report:
               precision    recall  f1-score   support

           0       0.81      0.81      0.81     12500
           1       0.81      0.81      0.81     12500

    accuracy                           0.81     25000
   macro avg       0.81      0.81      0.81     25000
weighted avg       0.81      0.81      0.81     25000
```

Experiment 11

Aim: Long Short-Term Memory Networks (LSTM): Implement LSTM on a time-series dataset and evaluate the model's performance.

Dataset Used

```
       Month  Passengers
0    1949-01         112
1    1949-02         118
2    1949-03         132
3    1949-04         129
4    1949-05         121
..       ...         ...
139  1960-08         606
140  1960-09         508
141  1960-10         461
142  1960-11         390
143  1960-12         432

[144 rows x 2 columns]
```

Code:

```python
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.metrics import mean_squared_error
from tensorflow.keras.layers import Dropout
# Load the dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/airline-passengers.csv"
data = pd.read_csv(url, usecols=[1], header=0)
values = data.values.astype('float32')

# Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
values_scaled = scaler.fit_transform(values)

# Prepare the data for supervised learning
def create_dataset(dataset, look_back=1):
```

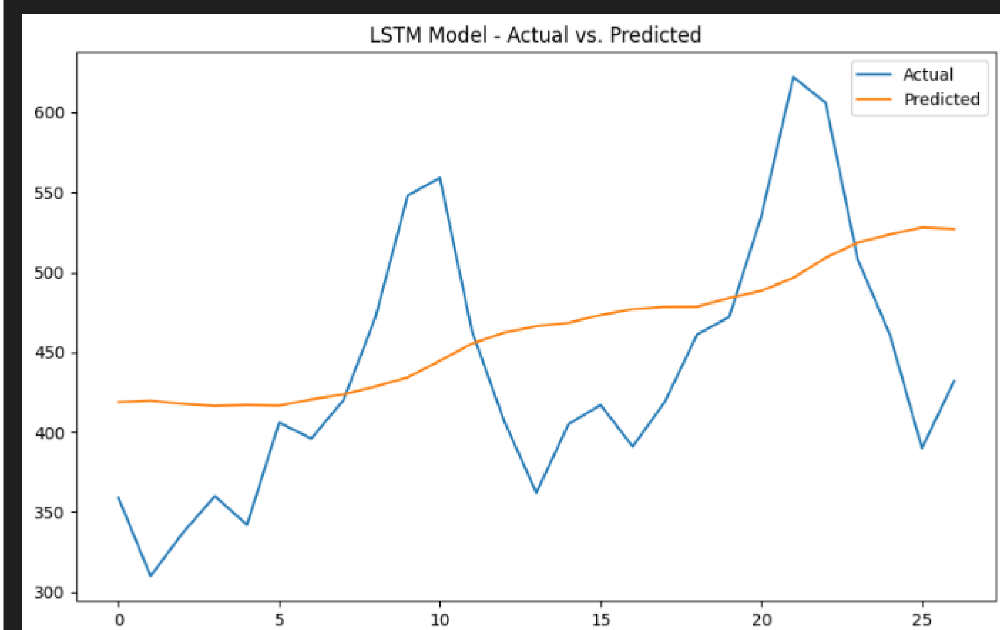
```
    X, y = [], []
    for i in range(len(dataset) - look_back):
        X.append(dataset[i:i + look_back, 0])
        y.append(dataset[i + look_back, 0])
    return np.array(X), np.array(y)

# Hyperparameters
look_back = 12  # Use the last 12 months to predict the next one
X, y = create_dataset(values_scaled, look_back)

# Reshape input to [samples, time steps, features] for LSTM
X = X.reshape((X.shape[0], X.shape[1], 1))

# Split into training and test sets
train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# Build the LSTM model
model = Sequential()
model.add(LSTM(500, input_shape=(look_back, 1)))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32,
validation_split=0.2, verbose=1)

# Make predictions
y_pred = model.predict(X_test)

# Invert scaling for predictions and actual values
y_pred_inverted = scaler.inverse_transform(y_pred)
y_test_inverted = scaler.inverse_transform(y_test.reshape(-1, 1))

# Evaluate the model
rmse = np.sqrt(mean_squared_error(y_test_inverted, y_pred_inverted))
print(f"Root Mean Squared Error: {rmse}")

# Plot the results
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.plot(y_test_inverted, label="Actual")
plt.plot(y_pred_inverted, label="Predicted")
```

```python
plt.legend()
plt.title("LSTM Model - Actual vs. Predicted")
plt.show()
```

Output:

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 65ms/step - loss: 0.0599 - val_loss: 0.1873
Epoch 2/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0354 - val_loss: 0.1142
Epoch 3/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0186 - val_loss: 0.0578
Epoch 4/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0099 - val_loss: 0.0232
Epoch 5/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0067 - val_loss: 0.0139
Epoch 6/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0097 - val_loss: 0.0137
Epoch 7/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 0.0100 - val_loss: 0.0147
Epoch 8/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0074 - val_loss: 0.0191
Epoch 9/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0057 - val_loss: 0.0251
Epoch 10/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0054 - val_loss: 0.0296
Epoch 11/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0075 - val_loss: 0.0298
Epoch 12/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0070 - val_loss: 0.0276
Epoch 13/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0052 - val_loss: 0.0241
...
Epoch 50/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.0038 - val_loss: 0.0133
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step
Root Mean Squared Error: 74.90377807617188
```

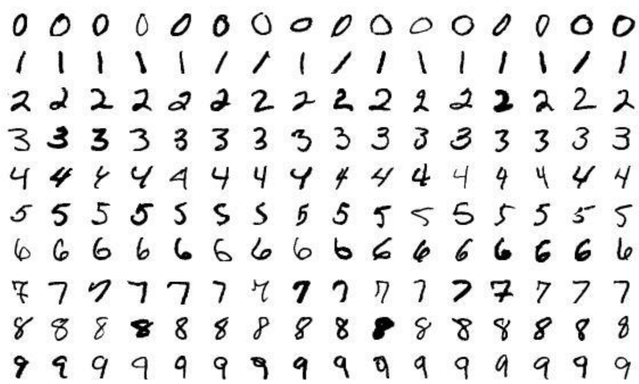
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...



Experiment 12

Aim: Autoencoders: Implement autoencoders on an image dataset and evaluate the model's performance.

Dataset Used:



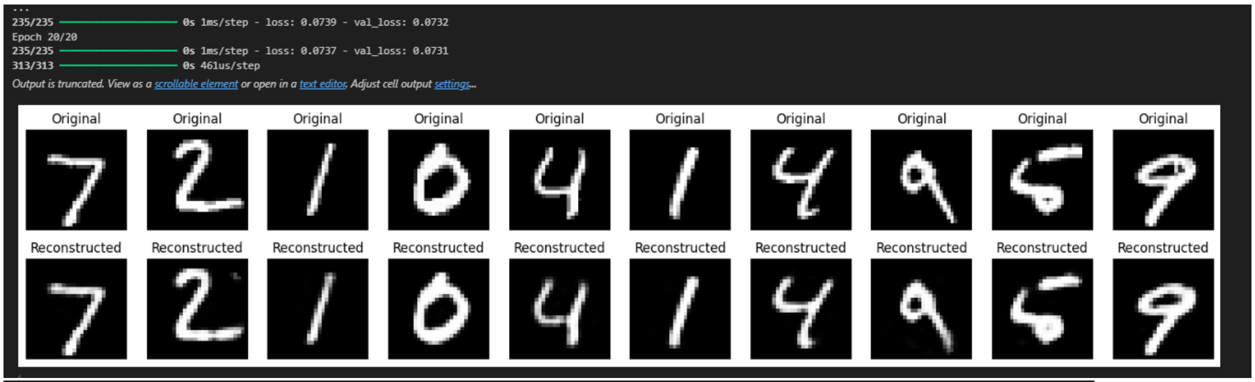
Code:

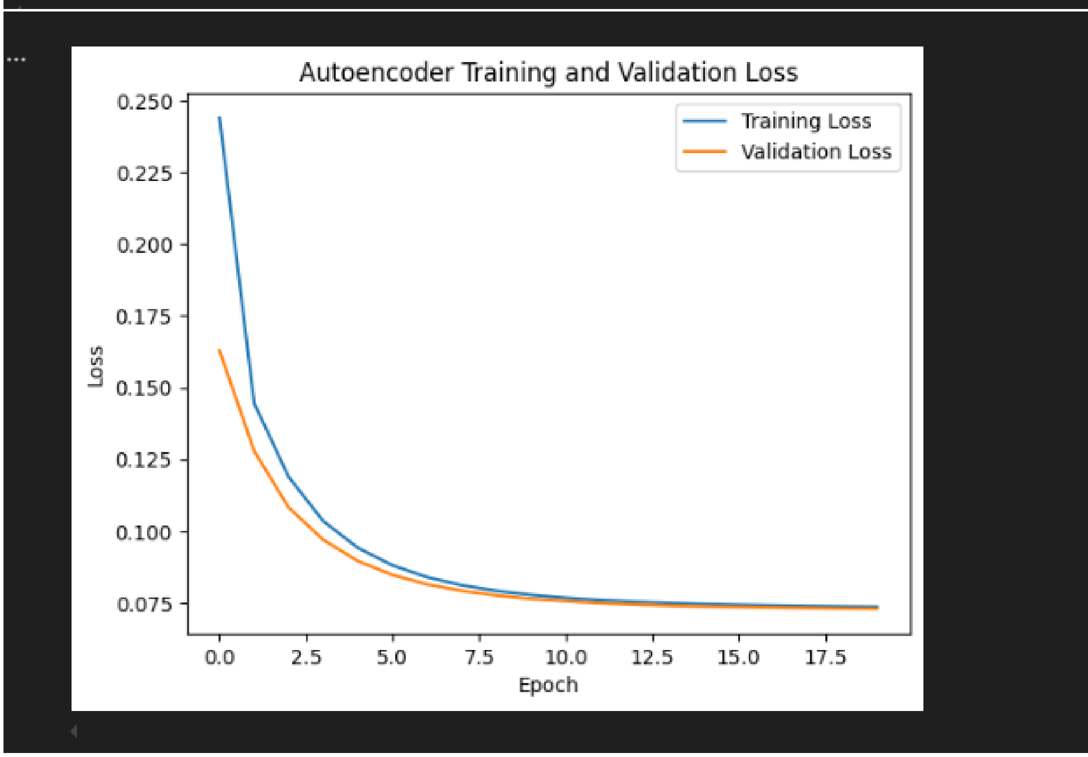
```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Input, Dense, Flatten, Reshape
from tensorflow.keras.optimizers import Adam

# Load the MNIST dataset
(X_train, _), (X_test, _) = mnist.load_data()

# Normalize the data
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# Flatten the data for fully connected autoencoder
X_train = X_train.reshape((X_train.shape[0], -1))
X_test = X_test.reshape((X_test.shape[0], -1))

# Define the dimensions of the autoencoder
input_dim = X_train.shape[1]  # 28x28 = 784
encoding_dim = 64  # Dimensionality of the encoded space

# Build the autoencoder model
# Encoder
```

```python
input_img = Input(shape=(input_dim,))
encoded = Dense(encoding_dim, activation='relu')(input_img)

# Decoder
decoded = Dense(input_dim, activation='sigmoid')(encoded)

# Autoencoder model
autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer=Adam(), loss='binary_crossentropy')

# Train the autoencoder
history = autoencoder.fit(X_train, X_train,
                          epochs=20,
                          batch_size=256,
                          shuffle=True,
                          validation_data=(X_test, X_test))

# Encode and decode some images
decoded_imgs = autoencoder.predict(X_test)

# Reshape the images back to 28x28 for visualization
X_test_reshaped = X_test.reshape((X_test.shape[0], 28, 28))
decoded_imgs_reshaped = decoded_imgs.reshape((decoded_imgs.shape[0], 28, 28))

# Plot original and reconstructed images
n = 10  # Number of images to display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Original images
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(X_test_reshaped[i], cmap="gray")
    plt.title("Original")
    plt.axis("off")

    # Reconstructed images
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs_reshaped[i], cmap="gray")
    plt.title("Reconstructed")
    plt.axis("off")
plt.show()

# Plot training loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
```

```python
plt.ylabel('Loss')
plt.legend()
plt.title('Autoencoder Training and Validation Loss')
plt.show()
```

Output:





Experiment 13

Aim: Transfer Learning: Implement transfer learning on an image dataset and evaluate the model's performance.

Dataset Used:

Code:

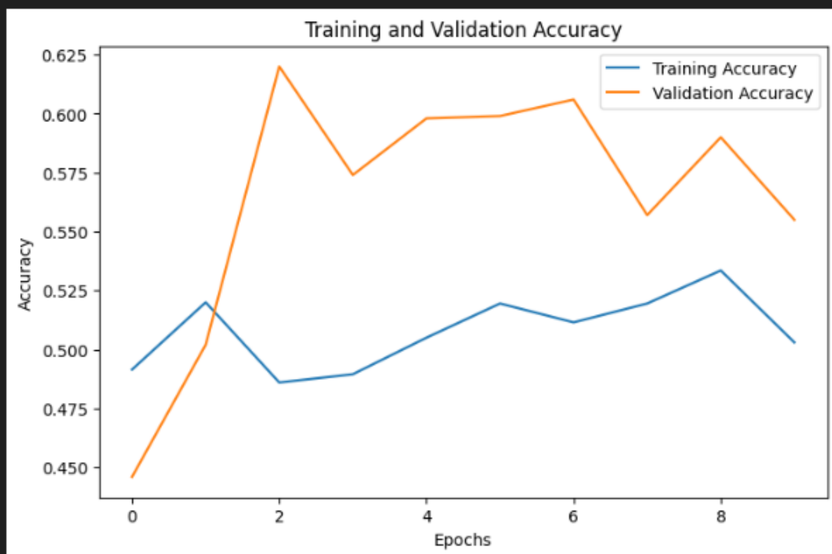
```
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
import os

# Set paths for the Cats vs. Dogs dataset
base_dir = r"./cats_and_dogs_filtered"
train_dir = os.path.join(base_dir, "train")
validation_dir = os.path.join(base_dir, "validation")
print(f"Validation directory: {validation_dir}")

# Image data generators for preprocessing
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode="nearest"
)
```

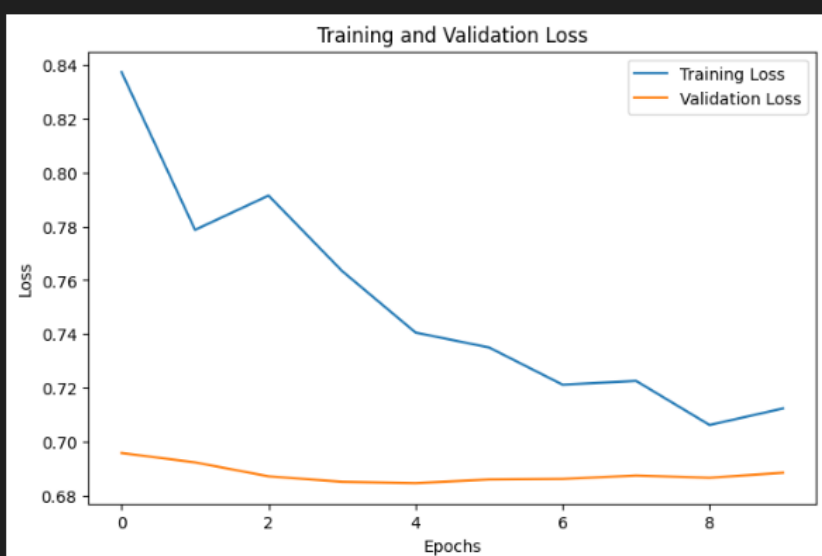
```python
validation_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary'
)

validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary'
)

# Load the ResNet50 model pre-trained on ImageNet
base_model = ResNet50(weights="imagenet", include_top=False, input_shape=(224,
224, 3))

# Freeze all layers of the base model
base_model.trainable = False

# Add custom layers on top
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dropout(0.5),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(
    optimizer=Adam(learning_rate=0.0001),
    loss='binary_crossentropy',
    metrics=['accuracy']
)

# Early stopping for better performance
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
```

```python
# Train the model
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator,
    callbacks=[early_stopping]
)

# Evaluate the model
loss, accuracy = model.evaluate(validation_generator)
print(f"Validation Loss: {loss:.4f}")
print(f"Validation Accuracy: {accuracy:.4f}")

# Plot training and validation accuracy
plt.figure(figsize=(8, 5))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
plt.figure(figsize=(8, 5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Output:

```
Epoch 9/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 46s 726ms/step - accuracy: 0.5256 - loss: 0.7063 - val_accuracy: 0.5900 - val_loss: 0.6865
Epoch 10/10
63/63 ━━━━━━━━━━━━━━━━━━━━ 43s 682ms/step - accuracy: 0.5077 - loss: 0.7077 - val_accuracy: 0.5550 - val_loss: 0.6884
32/32 ━━━━━━━━━━━━━━━━━━━━ 13s 413ms/step - accuracy: 0.6039 - loss: 0.6847
Validation Loss: 0.6845
Validation Accuracy: 0.5980
```





Experiment 14

Aim: Reinforcement Learning: Implement reinforcement learning on a game environment and evaluate the model's performance.

Code:

```
import gym
import numpy as np
import warnings

# Suppress specific deprecation warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)

# Load the environment with render mode specified
env = gym.make('CartPole-v1', render_mode="human")

# Initialize the environment to get the initial state
state = env.reset()

# Print the state space and action space
print("State space:", env.observation_space)
print("Action space:", env.action_space)


for _ in range(10):
    env.render()
    action = env.action_space.sample()

    step_result = env.step(action)

    if len(step_result) == 4:
        next_state, reward, done, info = step_result
        terminated = False
    else:
        next_state, reward, done, truncated, info = step_result
        terminated = done or truncated

    print(f"Action: {action}, Reward: {reward}, Next State: {next_state}, Done: {done}, Info: {info}")

    if terminated:
        state = env.reset()

env.close()
```

Output:

```
1]   ✓ 2.4s                                                                                                    Pythor

State space: Box([-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38], [4.8000002e+00 3.4028235e+38 4.1
Action space: Discrete(2)
Action: 0, Reward: 1.0, Next State: [-0.00504123 -0.24225019  0.04615125  0.26604646], Done: False, Info: {}
Action: 1, Reward: 1.0, Next State: [-0.00988624 -0.04781627  0.05147218 -0.01173022], Done: False, Info: {}
Action: 1, Reward: 1.0, Next State: [-0.01084256  0.14653116  0.05123757 -0.28773913], Done: False, Info: {}
Action: 1, Reward: 1.0, Next State: [-0.00791194  0.3408864   0.04548279 -0.56383216], Done: False, Info: {}
Action: 1, Reward: 1.0, Next State: [-0.00109421  0.5353417   0.03420615 -0.84184605], Done: False, Info: {}
Action: 0, Reward: 1.0, Next State: [ 0.00961262  0.3397699   0.01736923 -0.5386054 ], Done: False, Info: {}
Action: 1, Reward: 1.0, Next State: [ 0.01640802  0.5346434   0.00659712 -0.82576525], Done: False, Info: {}
Action: 1, Reward: 1.0, Next State: [ 0.02710089  0.7296745  -0.00991819 -1.116366  ], Done: False, Info: {}
Action: 0, Reward: 1.0, Next State: [ 0.04169438  0.5346842  -0.03224551 -0.8268108 ], Done: False, Info: {}
Action: 0, Reward: 1.0, Next State: [ 0.05238806  0.34001765 -0.04878172 -0.5444413 ], Done: False, Info: {}
```