Name: MD AQIB

Roll No: 002111001037(A1)

ASSIGNMENT-3

Title :- Hidden Markov Model(HMM)

1a. Implement Hidden Markov Model (HMM) for classification using Python for the following UCI datasets:

i. Ionosphere Dataset:

ii. Wisconsin Breast Cancer Dataset:

Code for Hidden Markov Model

 # installing hmmlearn for implementing hmm on datasets

```
!pip install hmmlearn
```

 # Importing necessary libraries & suppressing warning

```
# Importing necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from hmmlearn.hmm import GaussianHMM
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.decomposition import PCA
import warnings

# Suppress warnings
warnings.filterwarnings("ignore")
```

 # Function to print the metrics

```python
# Function to print classification metrics
def print_classification_report(y_true, y_pred):
    acc = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='macro')
    recall = recall_score(y_true, y_pred, average='macro')
    f1 = f1_score(y_true, y_pred, average='macro')
    cm = confusion_matrix(y_true, y_pred)
    print(f"Accuracy: {acc:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-Score: {f1:.4f}")
    print(f"Confusion Matrix:\n{cm}")
```

Loading both datasets

```python
# Load Ionosphere Dataset
def load_ionosphere_data():
    ionosphere_url = "https://archive.ics.uci.edu/ml/machine-learning-databases/ionosphere/ionosphere.data"
    ionosphere = pd.read_csv(ionosphere_url, header=None)
    X_iono = ionosphere.iloc[:, :-1].values
    y_iono = ionosphere.iloc[:, -1].values
    y_iono = LabelEncoder().fit_transform(y_iono)  # Encode class labels
    return X_iono, y_iono

# Load Breast Cancer Dataset from sklearn
from sklearn.datasets import load_breast_cancer

def load_breast_cancer_data():
    breast_cancer_data = load_breast_cancer()
    X_cancer = breast_cancer_data.data
    y_cancer = breast_cancer_data.target
    return X_cancer, y_cancer

# Load the datasets
X_iono, y_iono = load_ionosphere_data()
X_cancer, y_cancer = load_breast_cancer_data()
```

Standardizing datasets, applying PCA on datasets then training-testing splits

```python
# Standardize the datasets
scaler = StandardScaler()
X_iono_scaled = scaler.fit_transform(X_iono)
X_cancer_scaled = scaler.fit_transform(X_cancer)

# Apply PCA to reduce dimensionality (to help HMM converge better)
pca_iono = PCA(n_components=8)  # Reduced components for better results
X_iono_pca = pca_iono.fit_transform(X_iono_scaled)

pca_cancer = PCA(n_components=2)  # Reduced components for better results
X_cancer_pca = pca_cancer.fit_transform(X_cancer_scaled)

# Train-Test Split
X_iono_train, X_iono_test, y_iono_train, y_iono_test = train_test_split(X_iono_pca, y_iono, test_size=0.3, random_state=42)
X_cancer_train, X_cancer_test, y_cancer_train, y_cancer_test = train_test_split(X_cancer_pca, y_cancer, test_size=0.3, random_state=40)
```

Evaluating the HMM models on both datasets

```
# Helper function to train and evaluate HMM models
def train_evaluate_hmm(model, X_train, y_train, X_test, y_test):
    # Fit the HMM model
    model.fit(X_train)
    # Predict the labels for the test set
    y_pred = model.predict(X_test)
    # Evaluate the results
    return print_classification_report(y_test, y_pred)

# Define the models with refined parameters
gaussian_hmm_iono = GaussianHMM(n_components=2, covariance_type="diag", n_iter=500, tol=1e-4, min_covar=1e-2, random_state=42)
gaussian_hmm_cancer = GaussianHMM(n_components=2, covariance_type="diag", n_iter=500, tol=1e-4, min_covar=1e-2, random_state=42)

# Train and evaluate GaussianHMM on Ionosphere
print("HMM on Ionosphere Dataset:")
train_evaluate_hmm(gaussian_hmm_iono, X_iono_train, y_iono_train, X_iono_test, y_iono_test)

# Train and evaluate GaussianHMM on Breast Cancer
print("\nHMM on Breast Cancer Dataset:")
train_evaluate_hmm(gaussian_hmm_cancer, X_cancer_train, y_cancer_train, X_cancer_test, y_cancer_test)
```

Output for the HMM model on both datasets

```
HMM on Ionosphere Dataset:
Accuracy: 0.8019
Precision: 0.7995
Recall: 0.8219
F1-Score: 0.7978
Confusion Matrix:
[[35  4]
 [17 50]]
```

```
HMM on Breast Cancer Dataset:
Accuracy: 0.9064
Precision: 0.8872
Recall: 0.9167
F1-Score: 0.8981
Confusion Matrix:
[[ 53   3]
 [ 13 102]]
```

Analysis For HMM model on both datasets

- The Ionosphere dataset is loaded from UCI, and the Breast Cancer dataset is imported from **sklearn.datasets**. Both datasets are then standardized using **StandardScaler**, which helps improve model convergence and performance.
- Principal Component Analysis (PCA) is applied to reduce the dimensionality of the datasets (Ionosphere to **8** components, Breast Cancer to **2** components).
- The HMM parameters include n_iter=500 (maximum iterations for convergence), tol=1e-4 (convergence threshold), and min_covar=1e-2 (minimum covariance to avoid singular matrices).
- Thus, we cam conclude that the above code is not suitable for the given datasets but gives a very good accuracy of **80.19%** for Ionosphere dataset & **90.64%** for Breast Cancer dataset.

1b. Compare the performance the following HMM classifiers for all the two datasets and show the classification results (Accuracy, Precision, Recall, F-score, confusion matrix) with and without parameter tuning:

i. GaussianHMM

ii. GMMHMM

Code for GaussianHMM & GMMHMM

Importing necessary libraries & suppressing warning

```
# importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, roc_curve, auc
from hmmlearn.hmm import GaussianHMM, GMMHMM
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.decomposition import PCA
import warnings

# Suppress warnings
warnings.filterwarnings("ignore")
```

Function to print the metrics

```
# Function to print classification metrics
def print_classification_report(y_true, y_pred):
    acc = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='macro')
    recall = recall_score(y_true, y_pred, average='macro')
    f1 = f1_score(y_true, y_pred, average='macro')
    cm = confusion_matrix(y_true, y_pred)
    print(f"Accuracy: {acc:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1-Score: {f1:.4f}")
    print(f"Confusion Matrix:\n{cm}")
    return acc, precision, recall, f1, cm
```

Function to plot ROC curve

```python
# Function to plot ROC curve
def plot_roc_curve(y_test, y_pred_proba, title):
    fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
    roc_auc = auc(fpr, tpr)
    plt.figure(figsize=(6, 5))
    plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(title)
    plt.legend(loc="lower right")
    plt.show()
    print()
```

Function to plot confusion matrix heatmap

```python
# Function to plot confusion matrix heatmap
def plot_confusion_matrix(cm, title):
    plt.figure(figsize=(4, 3))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False, xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title(title)
    plt.show()
```

Loading both datasets

```python
# Load Ionosphere Dataset
def load_ionosphere_data():
    ionosphere_url = "https://archive.ics.uci.edu/ml/machine-learning-databases/ionosphere/ionosphere.data"
    ionosphere = pd.read_csv(ionosphere_url, header=None)
    X_iono = ionosphere.iloc[:, :-1].values
    y_iono = ionosphere.iloc[:, -1].values
    y_iono = LabelEncoder().fit_transform(y_iono)  # Encode class labels
    return X_iono, y_iono

# Load Breast Cancer Dataset from sklearn
from sklearn.datasets import load_breast_cancer

def load_breast_cancer_data():
    breast_cancer_data = load_breast_cancer()
    X_cancer = breast_cancer_data.data
    y_cancer = breast_cancer_data.target
    return X_cancer, y_cancer

# Load the datasets
X_iono, y_iono = load_ionosphere_data()
X_cancer, y_cancer = load_breast_cancer_data()
```

Standardizing datasets, applying PCA on datasets then training-testing splits and also storing the performance results

```python
# Standardize the datasets
scaler = StandardScaler()
X_iono_scaled = scaler.fit_transform(X_iono)
X_cancer_scaled = scaler.fit_transform(X_cancer)

# Apply PCA to reduce dimensionality
pca_iono = PCA(n_components=5)
X_iono_pca = pca_iono.fit_transform(X_iono_scaled)

pca_cancer = PCA(n_components=5)
X_cancer_pca = pca_cancer.fit_transform(X_cancer_scaled)

# Define train-test splits
splits = [(0.7, 0.3), (0.6, 0.4), (0.5, 0.5), (0.4, 0.6), (0.3, 0.7)]

# Store performance results
results = {
    'Split Ratio': [],
    'Classifier': [],
    'Accuracy': [],
    'Precision': [],
    'Recall': [],
    'F1-Score': []
}
```

Evaluating the GaussianHMM & GMMHMM models on both the datasets

```python
for train_size, test_size in splits:
    print(f"\nTrain-Test Split: {train_size * 100}% Train, {test_size * 100}% Test")

    # Ionosphere Dataset
    X_iono_train, X_iono_test, y_iono_train, y_iono_test = train_test_split(X_iono_pca, y_iono, test_size=test_size, random_state=42)

    # Define models
    gaussian_hmm_iono = GaussianHMM(n_components=2, covariance_type="diag", n_iter=500, tol=1e-4, min_covar=1e-2, random_state=42)
    gmm_hmm_iono = GMMHMM(n_components=2, n_mix=2, covariance_type="diag", n_iter=500, tol=1e-4, min_covar=1e-2, random_state=42)

    # Train and evaluate GaussianHMM on Ionosphere
    print("\nGaussianHMM on Ionosphere Dataset:")
    model = gaussian_hmm_iono
    model.fit(X_iono_train)
    y_pred = model.predict(X_iono_test)
    acc, precision, recall, f1, cm = print_classification_report(y_iono_test, y_pred)
    results['Split Ratio'].append(f"{train_size * 100}-{test_size * 100}")
    results['Classifier'].append('GaussianHMM-Ionosphere')
    results['Accuracy'].append(acc)
    results['Precision'].append(precision)
    results['Recall'].append(recall)
    results['F1-Score'].append(f1)
```

Predict the probabilities for each splits on all HMM models

```
# Predict probabilities and plot ROC and confusion matrix for each split
y_pred_proba = model.predict_proba(X_iono_test)[:, 1] if hasattr(model, 'predict_proba') else np.zeros_like(y_iono_test)
plot_roc_curve(y_iono_test, y_pred_proba, f'ROC Curve for GaussianHMM-Ionosphere (Split {train_size * 100}-{test_size * 100})')
plot_confusion_matrix(cm, f'Confusion Matrix for GaussianHMM-Ionosphere (Split {train_size * 100}-{test_size * 100})')

# Train and evaluate GMMHMM on Ionosphere
print("\nGMMHMM on Ionosphere Dataset:")
model = gmm_hmm_iono
model.fit(X_iono_train)
y_pred = model.predict(X_iono_test)
acc, precision, recall, f1, cm = print_classification_report(y_iono_test, y_pred)
results['Split Ratio'].append(f"{train_size * 100}-{test_size * 100}")
results['Classifier'].append('GMMHMM-Ionosphere')
results['Accuracy'].append(acc)
results['Precision'].append(precision)
results['Recall'].append(recall)
results['F1-Score'].append(f1)

# Predict probabilities and plot ROC and confusion matrix for each split
y_pred_proba = model.predict_proba(X_iono_test)[:, 1] if hasattr(model, 'predict_proba') else np.zeros_like(y_iono_test)
plot_roc_curve(y_iono_test, y_pred_proba, f'ROC Curve for GMMHMM-Ionosphere (Split {train_size * 100}-{test_size * 100})')
plot_confusion_matrix(cm, f'Confusion Matrix for GMMHMM-Ionosphere (Split {train_size * 100}-{test_size * 100})')
```

```
# Breast Cancer Dataset
X_cancer_train, X_cancer_test, y_cancer_train, y_cancer_test = train_test_split(X_cancer_pca, y_cancer, test_size=test_size, random_state=42)
# Define models
gaussian_hmm_cancer = GaussianHMM(n_components=2, covariance_type="diag", n_iter=500, tol=1e-4, min_covar=1e-2, random_state=42)
gmm_hmm_cancer = GMMHMM(n_components=2, n_mix=2, covariance_type="diag", n_iter=500, tol=1e-4, min_covar=1e-2, random_state=42)

# Train and evaluate GaussianHMM on Breast Cancer
print("\nGaussianHMM on Breast Cancer Dataset:")
model = gaussian_hmm_cancer
model.fit(X_cancer_train)
y_pred = model.predict(X_cancer_test)
acc, precision, recall, f1, cm = print_classification_report(y_cancer_test, y_pred)
results['Split Ratio'].append(f"{train_size * 100}-{test_size * 100}")
results['Classifier'].append('GaussianHMM-BreastCancer')
results['Accuracy'].append(acc)
results['Precision'].append(precision)
results['Recall'].append(recall)
results['F1-Score'].append(f1)

# Predict probabilities and plot ROC and confusion matrix for each split
y_pred_proba = model.predict_proba(X_cancer_test)[:, 1] if hasattr(model, 'predict_proba') else np.zeros_like(y_cancer_test)
plot_roc_curve(y_cancer_test, y_pred_proba, f'ROC Curve for GaussianHMM-BreastCancer (Split {train_size * 100}-{test_size * 100})')
plot_confusion_matrix(cm, f'Confusion Matrix for GaussianHMM-BreastCancer (Split {train_size * 100}-{test_size * 100})')
```

```
# Train and evaluate GMMHMM on Breast Cancer
print("\nGMMHMM on Breast Cancer Dataset:")
model = gmm_hmm_cancer
model.fit(X_cancer_train)
y_pred = model.predict(X_cancer_test)
acc, precision, recall, f1, cm = print_classification_report(y_cancer_test, y_pred)
results['Split Ratio'].append(f"{train_size * 100}-{test_size * 100}")
results['Classifier'].append('GMMHMM-BreastCancer')
results['Accuracy'].append(acc)
results['Precision'].append(precision)
results['Recall'].append(recall)
results['F1-Score'].append(f1)

# Predict probabilities and plot ROC and confusion matrix for each split
y_pred_proba = model.predict_proba(X_cancer_test)[:, 1] if hasattr(model, 'predict_proba') else np.zeros_like(y_cancer_test)
plot_roc_curve(y_cancer_test, y_pred_proba, f'ROC Curve for GMMHMM-BreastCancer (Split {train_size * 100}-{test_size * 100})')
plot_confusion_matrix(cm, f'Confusion Matrix for GMMHMM-BreastCancer (Split {train_size * 100}-{test_size * 100})')
```

Performance table & printing the final plot for each splits on both HMM models for both the datasets & also print the best accuracy models for each out of all splits

```python
# Performance Comparison Table
results_df = pd.DataFrame(results)
print("\nPerformance Comparison Table:")
print(results_df)

# Accuracy vs. Train-Test Split Graph
plt.figure(figsize=(6, 4))
for classifier in results_df['Classifier'].unique():
    subset = results_df[results_df['Classifier'] == classifier]
    plt.plot(subset['Split Ratio'], subset['Accuracy'], marker='o', label=classifier)

plt.xlabel('Train-Test Split Ratio')
plt.ylabel('Accuracy')
plt.title('Accuracy vs. Train-Test Split')
plt.legend()
plt.grid(True)
plt.show()

# Print metrics for best splits of each classifier
best_splits = results_df.loc[results_df.groupby('Classifier')['Accuracy'].idxmax()]

print("\nBest Splits for Each Classifier:")
print(best_splits)
```

Analysis For the GaussianHMM & GMMHMM on Ionosphere dataset

- We are applying the GaussianHMM & GMMHMM on Ionosphere dataset which is not suitable but the above code give us some good and satisfactory accuracy for each splits.

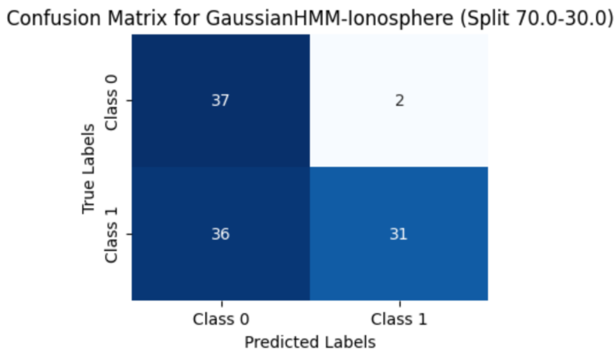
Accuracy vs Splits Table for Ionosphere dataset

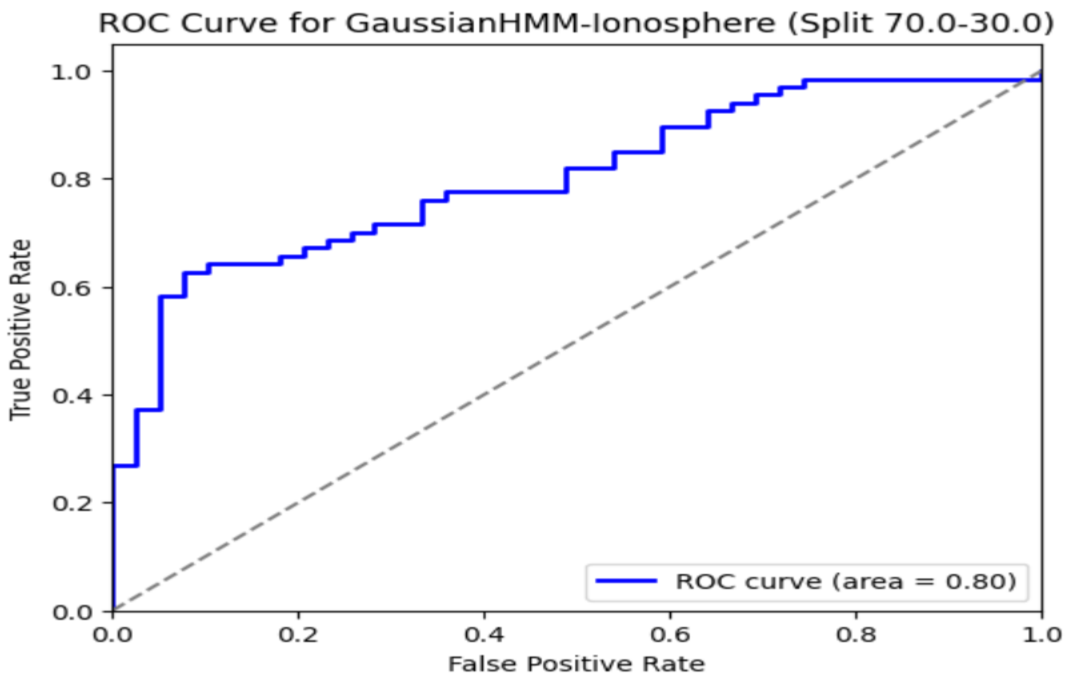
HMM models / Train-Test splits	**Gaussian HMM (Accuracy)**	**GMMHMM (Accuracy)**
70:30	64.15%	76.41%
60:40	62.41%	74.46%
50:50	59.09%	64.77%
40:60	63.03%	63.03%
30:70	57.31%	62.19%

- It is clear from the above table that the best accuracy for the GaussianHMM is obtained from the 70:30 split which gives almost an accuracy of 64.15% on Ionosphere dataset.
- And for the GMMHMM on Ionosphere dataset best accuracy obtained at 70:30 split which is 76.41% accuracy.

Output for best accuracy on Ionosphere dataset (GaussianHMM)

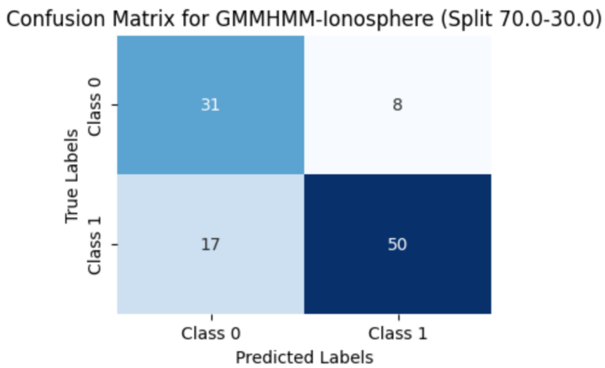
```
GaussianHMM on Ionosphere Dataset:
Accuracy: 0.6415
Precision: 0.7231
Recall: 0.7057
F1-Score: 0.6404
Confusion Matrix:
[[37  2]
 [36 31]]
```

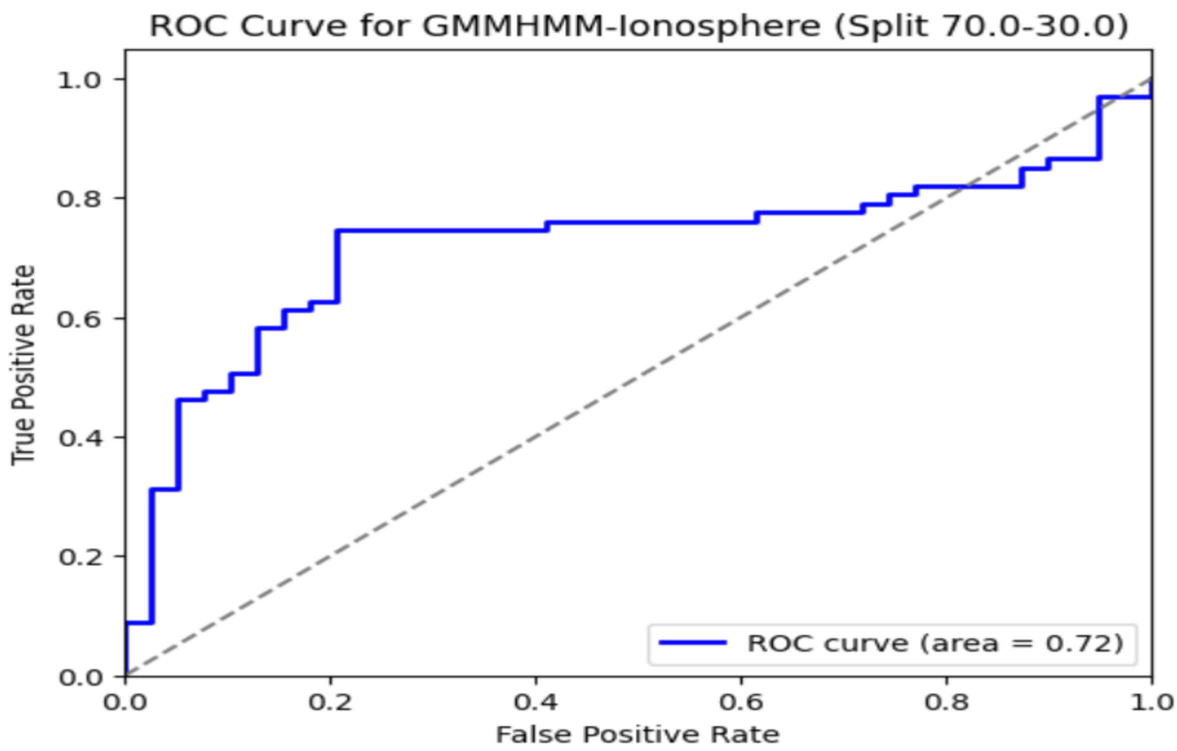




Output for best accuracy on Ionosphere dataset (GMMHMM)

```
GMMHMM on Ionosphere Dataset:
Accuracy: 0.7642
Precision: 0.7540
Recall: 0.7706
F1-Score: 0.7563
Confusion Matrix:
[[31  8]
 [17 50]]
```





Analysis For the GaussianHMM & GMMHMM on Breast Cancer dataset

- We are applying the GaussianHMM & GMMHMM on Breast Cancer dataset which is not suitable but the above code give us some good and satisfactory accuracy for each splits.

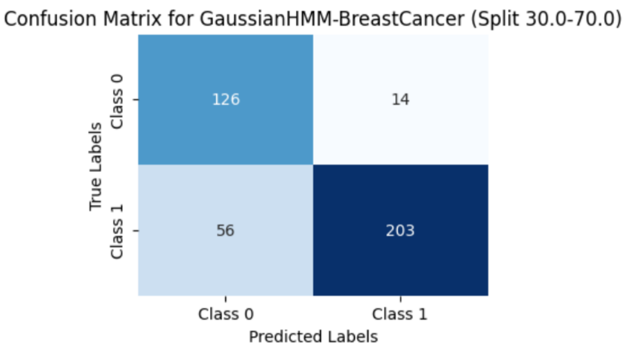
Accuracy vs Splits Table for Breast Cancer dataset

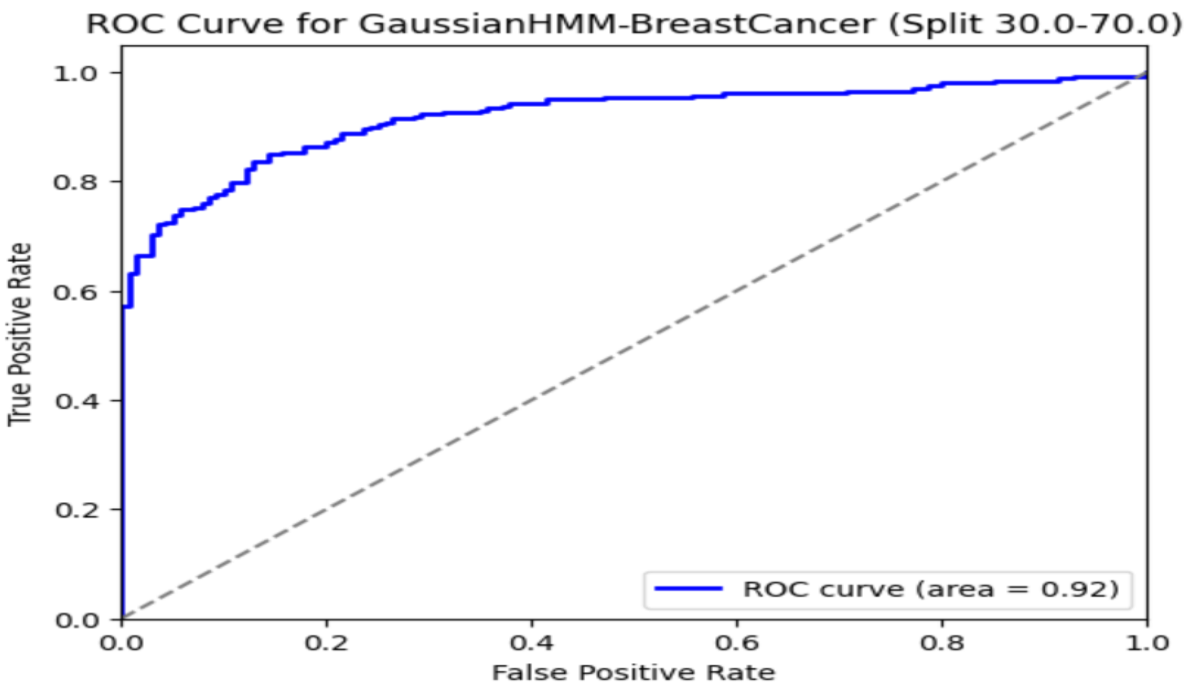
HMM models / Train-Test splits	**Gaussian HMM (Accuracy)**	**GMMHMM (Accuracy)**
70:30	80.70%	91.81%
60:40	30.26%	40.78%
50:50	29.47%	37.19%
40:60	31.57%	40.93%
30:70	82.45%	84.71%

- It is clear from the above table that the best accuracy for the GaussianHMM is obtained from the 30:70 split which gives almost an accuracy of 82.45% on Breast Cancer dataset.
- And for the GMMHMM on Breast Cancer dataset best accuracy obtained at 70:30 split which is 91.81% accuracy.

Output for best accuracy on Breast Cancer dataset (GaussianHMM)

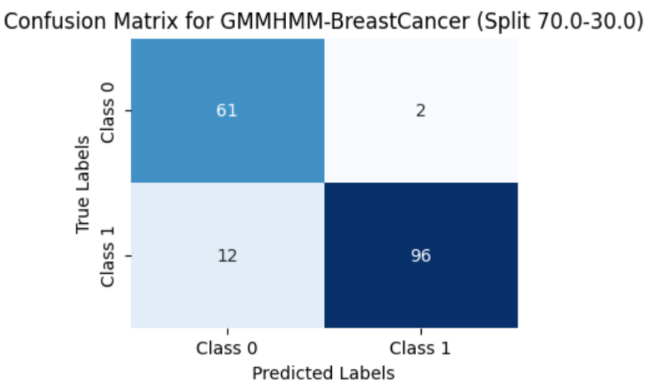
```
GaussianHMM on Breast Cancer Dataset:
Accuracy: 0.8246
Precision: 0.8139
Recall: 0.8419
F1-Score: 0.8178
Confusion Matrix:
[[126  14]
 [ 56 203]]
```

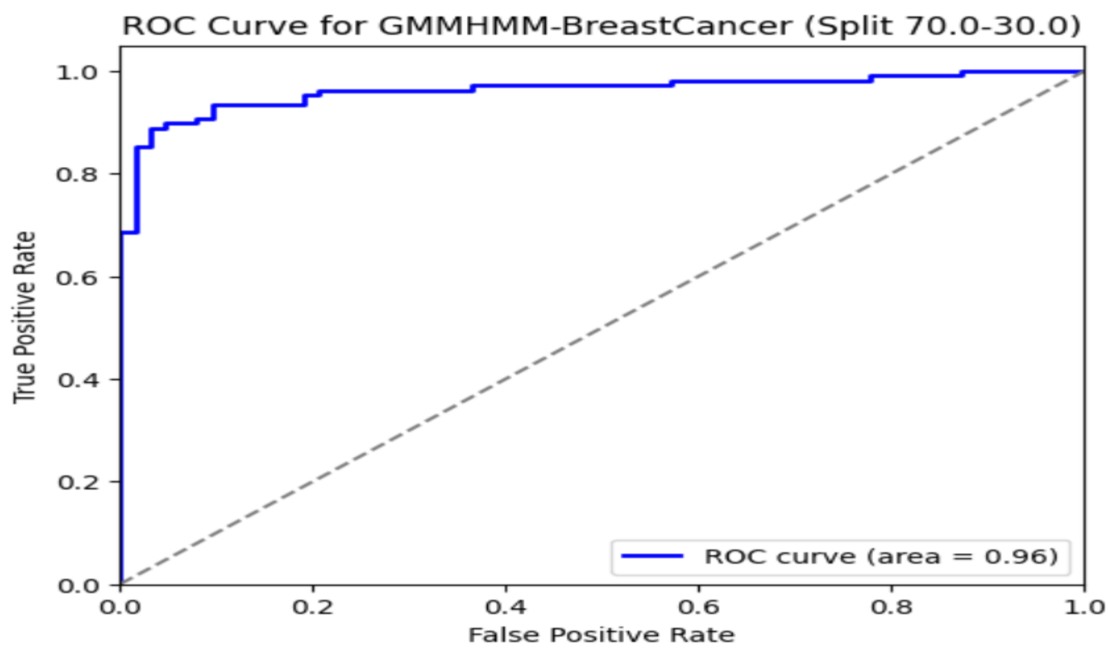




Output for best accuracy on Breast Cancer dataset dataset (GMMHMM)

```
GMMHMM on Breast Cancer Dataset:
Accuracy: 0.9181
Precision: 0.9076
Recall: 0.9286
F1-Score: 0.9145
Confusion Matrix:
[[61  2]
 [12 96]]
```





Overall Conclusion

- All the details of both datasets in each splits (70:30, 60:40, 50:50, 40:60, 30:70) using GaussianHMM & GMMHMM can be seen as below in the performance table

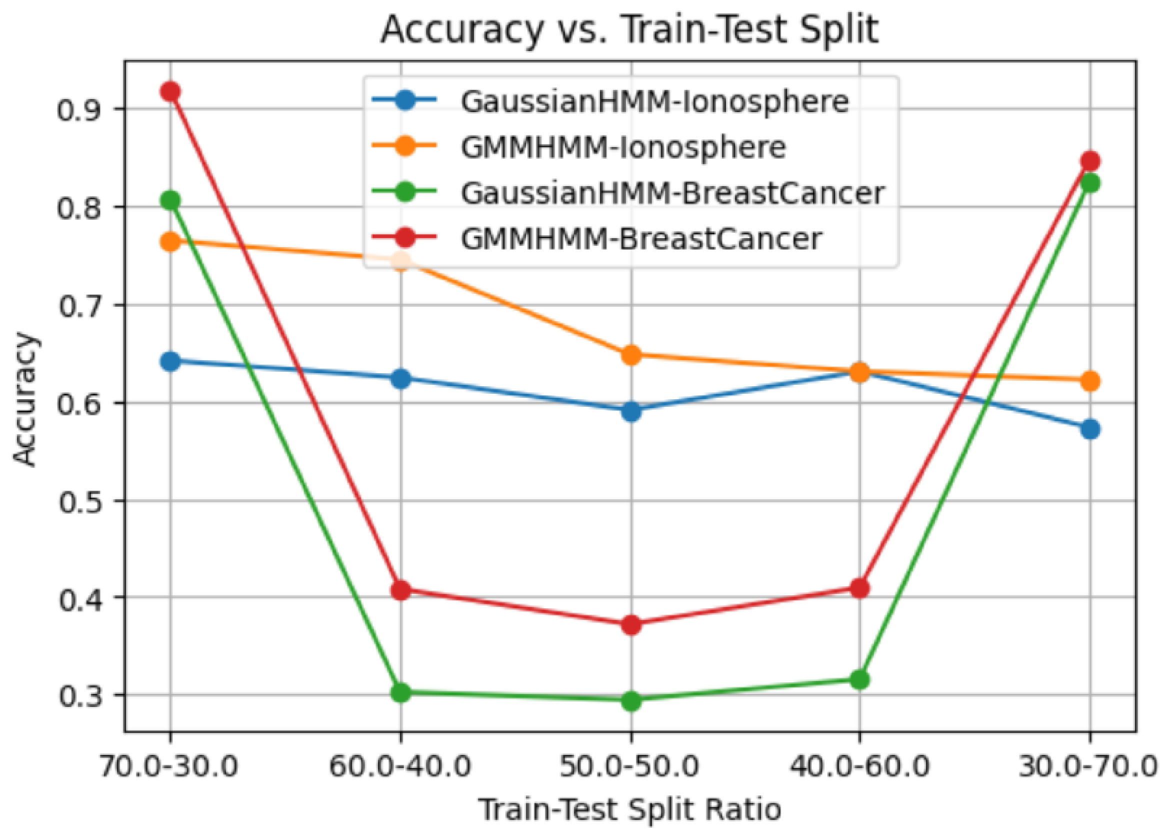
Performance Comparison Table for each splits

Performance Comparison Table:

	Split Ratio	Classifier	Accuracy	Precision	Recall	F1-Score
0	70.0-30.0	GaussianHMM-Ionosphere	0.641509	0.723122	0.705702	0.640357
1	70.0-30.0	GMMHMM-Ionosphere	0.764151	0.753951	0.770570	0.756322
2	70.0-30.0	GaussianHMM-BreastCancer	0.807018	0.807798	0.830688	0.803770
3	70.0-30.0	GMMHMM-BreastCancer	0.918129	0.907604	0.928571	0.914549
4	60.0-40.0	GaussianHMM-Ionosphere	0.624113	0.713415	0.687607	0.621371
5	60.0-40.0	GMMHMM-Ionosphere	0.744681	0.763475	0.776694	0.743636
6	60.0-40.0	GaussianHMM-BreastCancer	0.302632	0.335256	0.370946	0.292713
7	60.0-40.0	GMMHMM-BreastCancer	0.407895	0.535042	0.518074	0.379223
8	50.0-50.0	GaussianHMM-Ionosphere	0.590909	0.685200	0.659737	0.587500
9	50.0-50.0	GMMHMM-Ionosphere	0.647727	0.701082	0.698406	0.647682
10	50.0-50.0	GaussianHMM-BreastCancer	0.294737	0.329866	0.363009	0.286835
11	50.0-50.0	GMMHMM-BreastCancer	0.371930	0.449375	0.465541	0.356594
12	40.0-60.0	GaussianHMM-Ionosphere	0.630332	0.700604	0.696686	0.630257
13	40.0-60.0	GMMHMM-Ionosphere	0.630332	0.700604	0.696686	0.630257
14	40.0-60.0	GaussianHMM-BreastCancer	0.315789	0.347676	0.378439	0.306300
15	40.0-60.0	GMMHMM-BreastCancer	0.409357	0.511850	0.506738	0.383531
16	30.0-70.0	GaussianHMM-Ionosphere	0.573171	0.675436	0.647009	0.568717
17	30.0-70.0	GMMHMM-Ionosphere	0.621951	0.680636	0.676068	0.621795
18	30.0-70.0	GaussianHMM-BreastCancer	0.824561	0.813896	0.841892	0.817775
19	30.0-70.0	GMMHMM-BreastCancer	0.847118	0.838065	0.869112	0.841672

i. From the above table we can easily compare the performance of each and conclude the best accuracy for each datasets.

ii. Below is overall plot of GaussianHMM &GMMHMM on both datsets over each splits.



iii. Best splits for each datasets usinh both GaussianHMM & GMMHM as below,

Best Splits for Each Classifier:

	Split Ratio	Classifier	Accuracy	Precision	Recall	F1-Score
3	70.0-30.0	GMMHMM-BreastCancer	0.918129	0.907604	0.928571	0.914549
1	70.0-30.0	GMMHMM-Ionosphere	0.764151	0.753951	0.770570	0.756322
18	30.0-70.0	GaussianHMM-BreastCancer	0.824561	0.813896	0.841892	0.817775
0	70.0-30.0	GaussianHMM-Ionosphere	0.641509	0.723122	0.705702	0.640357

iv. For continuous datasets like **Ionosphere** and **Breast Cancer**, **GaussianHMM** is suitable for simpler data distributions, while **GMMHMM** offers flexibility by modeling complex distributions. GMMHMM generally outperforms GaussianHMM in capturing intricate data patterns, leading to better classification accuracy.

Title :- Deep Learning

2. Construct a Deep Learning model using Convolutional Neural Network (CNN) for classification on the following standard datasets:

   a. CIFAR-10

   b. MNIST

i. Code for CIFAR-10

First Installing Tensorflow

```
pip install tensorflow
```

Importing necessary libraries

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

Loading both datasets & Normalizing the pixels values

```
# Load CIFAR-10 dataset
(X_train, y_train), (X_test, y_test) = datasets.cifar10.load_data()

# Normalize pixel values
X_train, X_test = X_train / 255.0, X_test / 255.0
```

Plot training & validation accuracy values

```
# Plot training & validation accuracy values
plt.plot(history.history['accuracy'], label='Train')
plt.plot(history.history['val_accuracy'], label='Test')
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```

Building & print the summary of CNN model

```
# Build the CNN model
model = models.Sequential([
    # First Convolutional Block
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),

    # Second Convolutional Block
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    # Third Convolutional Block
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    # Flatten the output of the convolutional layers
    layers.Flatten(),

    # Fully Connected (Dense) Layers
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),  # Dropout layer to prevent overfitting
    layers.Dense(10, activation='softmax')

])

model.summary()
```

Compiling , training & evaluating the model

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

# Train the model
history = model.fit(X_train, y_train, epochs=10,
                    validation_data=(X_test, y_test))

# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc}')
```

Output for CIFAR-10

Model Summary

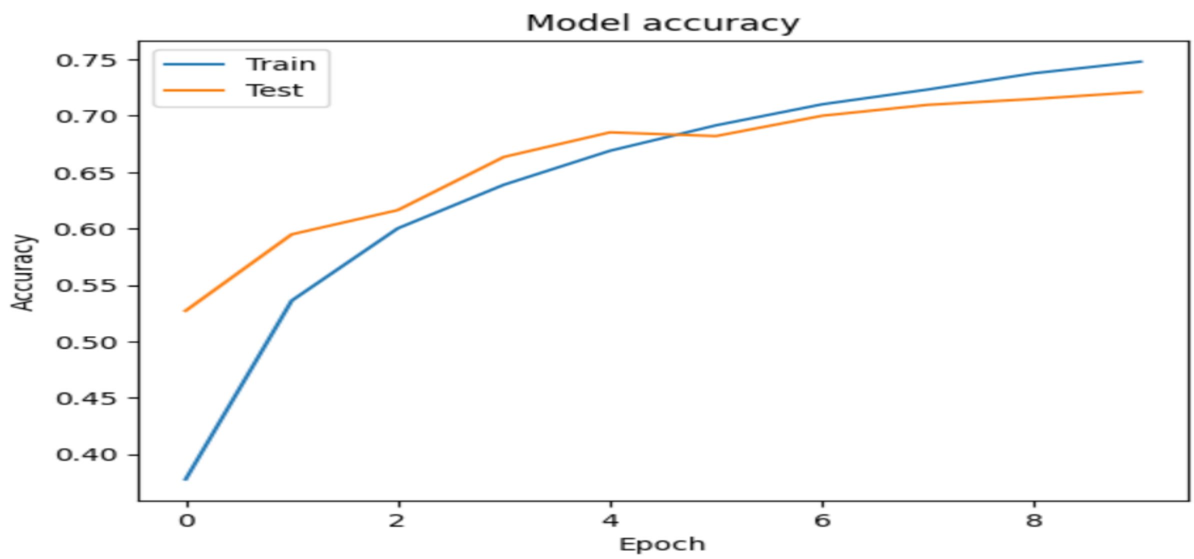
```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d_2 (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_4 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_5 (Conv2D)	(None, 4, 4, 128)	73,856
max_pooling2d_4 (MaxPooling2D)	(None, 2, 2, 128)	0
flatten_1 (Flatten)	(None, 512)	0
dense_2 (Dense)	(None, 128)	65,664
dense_3 (Dense)	(None, 10)	1,290

```
Total params: 160,202 (625.79 KB)
Trainable params: 160,202 (625.79 KB)
Non-trainable params: 0 (0.00 B)
```

Accuracy over 10 Epochs

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.2839 - loss: 1.9049 - val_accuracy: 0.5272 - val_loss: 1.3216
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.5173 - loss: 1.3444 - val_accuracy: 0.5949 - val_loss: 1.1515
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.5905 - loss: 1.1534 - val_accuracy: 0.6164 - val_loss: 1.0827
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.6320 - loss: 1.0434 - val_accuracy: 0.6635 - val_loss: 0.9627
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.6692 - loss: 0.9453 - val_accuracy: 0.6853 - val_loss: 0.9053
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.6902 - loss: 0.8879 - val_accuracy: 0.6820 - val_loss: 0.9184
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.7081 - loss: 0.8397 - val_accuracy: 0.7000 - val_loss: 0.8743
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.7236 - loss: 0.7838 - val_accuracy: 0.7097 - val_loss: 0.8392
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.7441 - loss: 0.7388 - val_accuracy: 0.7149 - val_loss: 0.8340
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.7515 - loss: 0.7026 - val_accuracy: 0.7211 - val_loss: 0.8217
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7241 - loss: 0.8167
Test accuracy: 0.7210999727249146
```



Analysis For CIFAR-10

- The CNN model for CIFAR-10 uses three convolutional layers followed by max-pooling for feature extraction and a dense layer for classification.
- Dropout is applied to prevent overfitting, and adam optimizer with sparse categorical cross-entropy is used.
- The final test accuracy is around 72%, and the training/validation accuracy is around 75% & curves show the model's performance over 10 epochs, indicating room for improvement through further tuning.

ii. Code for MNIST

First Installing Tensorflow

```
pip install tensorflow
```

Importing necessary libraries

```
# Import necessary libraries
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

Loading both datasets & Normalizing the pixels values

```
# Load MNIST dataset
(X_train, y_train), (X_test, y_test) = datasets.mnist.load_data()

# Preprocess the data: Normalize and reshape
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
```

Plot training & validation accuracy values

```
# Plot training & validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```

Building & print the summary of CNN model

```
# Build the CNN model
model = models.Sequential()
# First Convolutional Block
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
# Second Convolutional Block
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
# Flatten the output of the convolutional layers
model.add(layers.Flatten())
# Fully Connected (Dense) Layers
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

model.summary()
```

Compiling , training & evaluating the model

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

# Train the model
history = model.fit(X_train, y_train, epochs=10,
                    validation_data=(X_test, y_test))

# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc}')
```

Output for MNIST

Model Summary

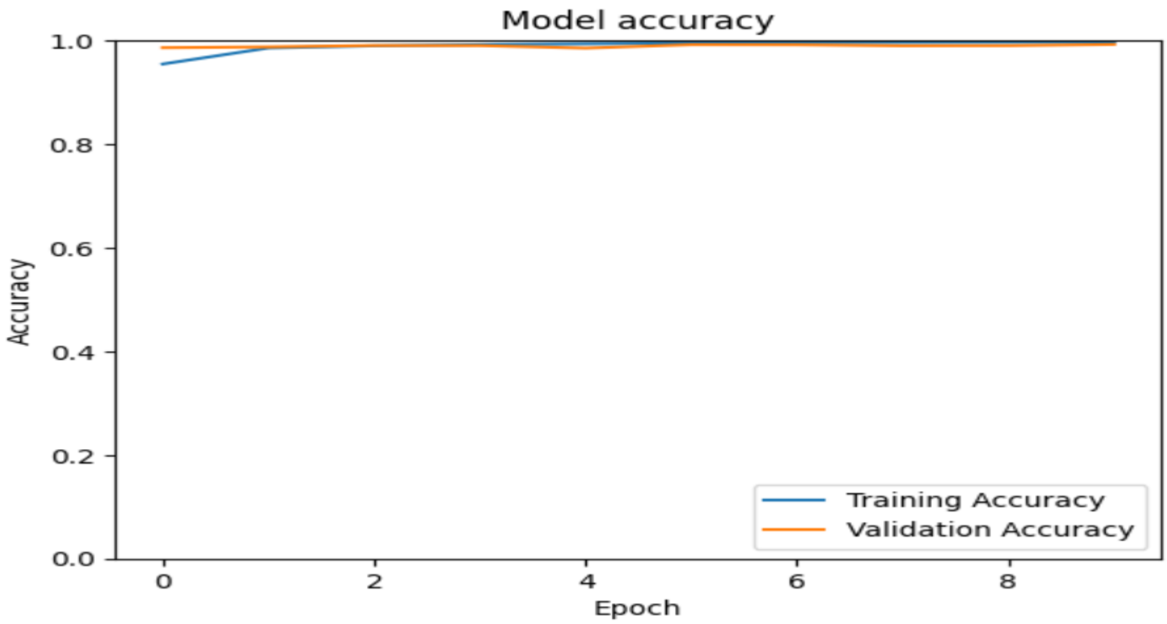
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: Use
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
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
conv2d_2 (Conv2D)	(None, 3, 3, 64)	36,928
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 64)	36,928
dense_1 (Dense)	(None, 10)	650

```
 Total params: 93,322 (364.54 KB)
 Trainable params: 93,322 (364.54 KB)
 Non-trainable params: 0 (0.00 B)
```

Accuracy over 10 Epochs

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 12s 4ms/step - accuracy: 0.8927 - loss: 0.3386 - val_accuracy: 0.9862 - val_loss: 0.0447
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9858 - loss: 0.0448 - val_accuracy: 0.9877 - val_loss: 0.0391
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9903 - loss: 0.0306 - val_accuracy: 0.9904 - val_loss: 0.0307
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9932 - loss: 0.0229 - val_accuracy: 0.9902 - val_loss: 0.0301
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9943 - loss: 0.0173 - val_accuracy: 0.9854 - val_loss: 0.0464
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9954 - loss: 0.0136 - val_accuracy: 0.9920 - val_loss: 0.0387
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.9960 - loss: 0.0123 - val_accuracy: 0.9918 - val_loss: 0.0300
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9967 - loss: 0.0102 - val_accuracy: 0.9899 - val_loss: 0.0336
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9976 - loss: 0.0079 - val_accuracy: 0.9902 - val_loss: 0.0369
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9973 - loss: 0.0081 - val_accuracy: 0.9925 - val_loss: 0.0280
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9901 - loss: 0.0362
Test accuracy: 0.9925000071525574
```



Analysis For MNIST

- This CNN model for MNIST uses three convolutional layers followed by max-pooling for feature extraction, followed by dense layers for classification.
- The model achieves high test accuracy, typically around 98-99%, as MNIST is a relatively simple dataset for digit classification.
- The accuracy plots show consistent improvement in both training and validation, indicating good model performance with minimal overfitting.

3. Experiment with the following Deep Learning models on the above the two datasets and show the performance comparison among the models along with that of CNN:

a. VGG-16

b. Recurrent Neural Networks (RNN)

c. AlexNet

d. GoogLeNet

Code for VGG-16

First Installing Tensorflow

```
pip install tensorflow
```

Importing necessary libraries

```
# Import necessary libraries
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
from tensorflow.keras.applications import VGG16
```

Loading the dataset, pre-trained VGG16 model & freezing the convolutional layers

```
# Load the dataset (CIFAR-10 or MNIST, process similarly as before)
(X_train, y_train), (X_test, y_test) = datasets.cifar10.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0

# Load pre-trained VGG16 model, without the top layers
vgg_model = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))

# Freeze the convolutional layers to prevent retraining them
for layer in vgg_model.layers:
    layer.trainable = False
```

Plot training & validation accuracy values

```
# Plot training & validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```

Building a new model using VGG16 as the base

```
# Create a new model using VGG16 as the base
model = models.Sequential([
    vgg_model,
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

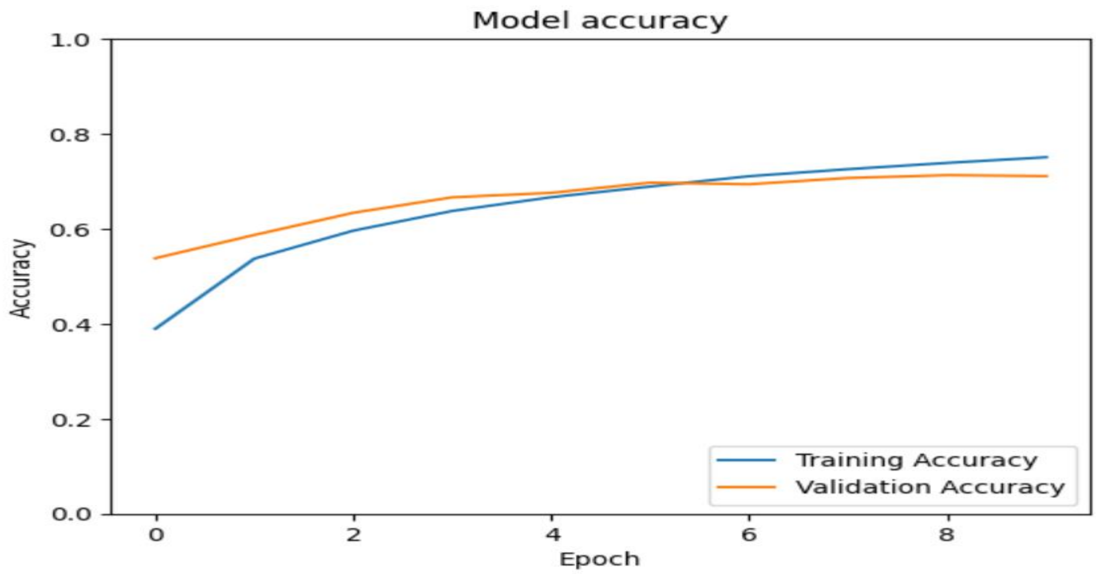
Compiling , training & evaluating the model

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

# Train the model
history_vgg = model.fit(X_train, y_train, epochs=10,
                        validation_data=(X_test, y_test))

# Evaluate the model
test_loss_vgg, test_acc_vgg = model.evaluate(X_test, y_test)
print(f'VGG16 Test accuracy: {test_acc_vgg}')
```

Output for VGG16

Accuracy over 10 Epochs

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 21s 12ms/step - accuracy: 0.4414 - loss: 1.5946 - val_accuracy: 0.5502 - val_loss: 1.2863
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - accuracy: 0.5706 - loss: 1.2233 - val_accuracy: 0.5694 - val_loss: 1.2229
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 13s 9ms/step - accuracy: 0.5974 - loss: 1.1548 - val_accuracy: 0.5783 - val_loss: 1.2064
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 21s 9ms/step - accuracy: 0.6102 - loss: 1.1225 - val_accuracy: 0.5877 - val_loss: 1.1700
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - accuracy: 0.6139 - loss: 1.0997 - val_accuracy: 0.5946 - val_loss: 1.1600
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 13s 9ms/step - accuracy: 0.6283 - loss: 1.0621 - val_accuracy: 0.5959 - val_loss: 1.1578
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 21s 9ms/step - accuracy: 0.6320 - loss: 1.0594 - val_accuracy: 0.5989 - val_loss: 1.1524
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 20s 8ms/step - accuracy: 0.6346 - loss: 1.0460 - val_accuracy: 0.6039 - val_loss: 1.1441
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 21s 9ms/step - accuracy: 0.6451 - loss: 1.0176 - val_accuracy: 0.6000 - val_loss: 1.1452
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 20s 8ms/step - accuracy: 0.6481 - loss: 1.0040 - val_accuracy: 0.6101 - val_loss: 1.1350
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.6106 - loss: 1.1267
VGG16 Test accuracy: 0.6100999712944031
```



Analysis For VGG16

- The VGG16 model is used as a pre-trained base with its convolutional layers frozen, while the fully connected layers are retrained CIFAR-10.
- With only two additional dense layers, the model adapts well to the CIFAR-10 dataset, typically achieving higher accuracy around 61% compared to a model trained from scratch.
- The accuracy curves show stable training and validation accuracy, with VGG16-based models generally providing better performance due to the strong feature extraction capabilities of its deep architecture.

Code for Recurrent Neural Networks (RNN)

First Installing Tensorflow

```
pip install tensorflow
```

Importing necessary libraries

```
# Import necessary libraries
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

Loading & Normalizing the dataset

```
# Load and preprocess CIFAR-10 data
(X_train, y_train), (X_test, y_test) = datasets.cifar10.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0  # Normalize pixel values
```

Plot training & validation accuracy values

```
# Plot training & validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```

Building a RNN model using LSTM

```
# Create the RNN model (using LSTM)
model = models.Sequential([
    layers.LSTM(128, input_shape=(28, 28), return_sequences=True),
    layers.LSTM(64),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

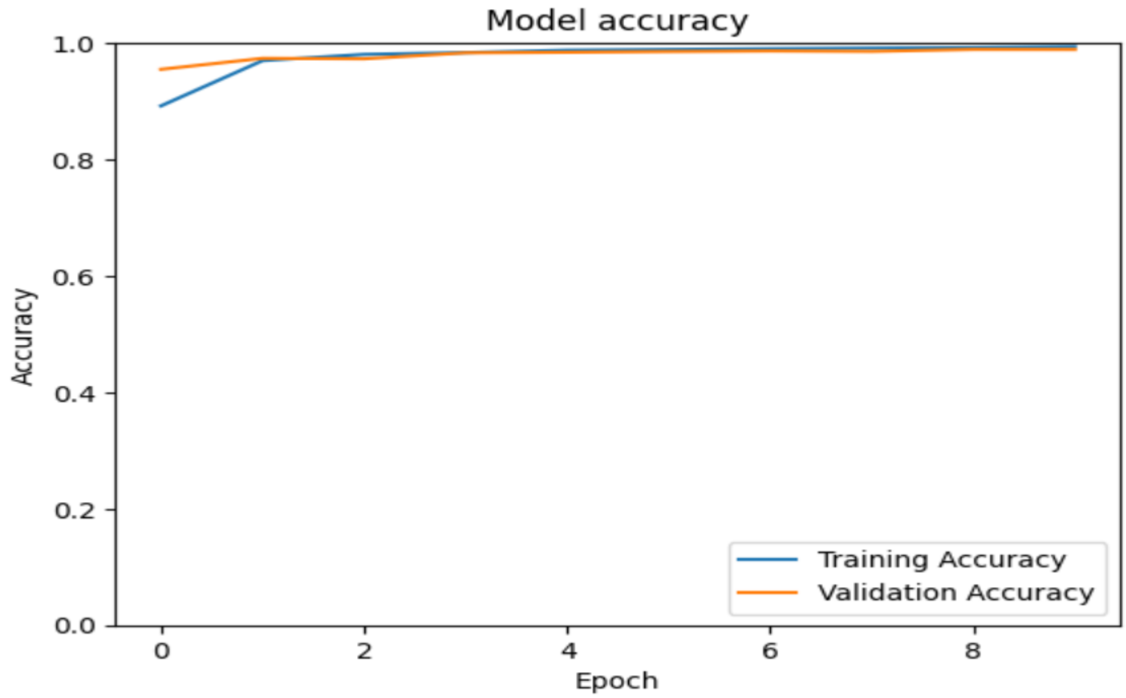
Compiling , training & evaluating the model

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

# Train the model
history_rnn = model.fit(X_train, y_train, epochs=10,
                        validation_data=(X_test, y_test))

# Evaluate the model
test_loss_rnn, test_acc_rnn = model.evaluate(X_test, y_test)
print(f'RNN Test accuracy: {test_acc_rnn}')
```

Output for RNN

Accuracy over 10 Epochs

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input
  super().__init__(**kwargs)
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 17s 7ms/step - accuracy: 0.7749 - loss: 0.6773 - val_accuracy: 0.9670 - val_loss: 0.1112
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 16s 8ms/step - accuracy: 0.9677 - loss: 0.1075 - val_accuracy: 0.9729 - val_loss: 0.0909
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 13s 7ms/step - accuracy: 0.9775 - loss: 0.0732 - val_accuracy: 0.9809 - val_loss: 0.0643
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 20s 7ms/step - accuracy: 0.9841 - loss: 0.0527 - val_accuracy: 0.9859 - val_loss: 0.0494
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 21s 7ms/step - accuracy: 0.9871 - loss: 0.0428 - val_accuracy: 0.9874 - val_loss: 0.0447
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 21s 7ms/step - accuracy: 0.9893 - loss: 0.0344 - val_accuracy: 0.9838 - val_loss: 0.0521
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 13s 7ms/step - accuracy: 0.9898 - loss: 0.0314 - val_accuracy: 0.9900 - val_loss: 0.0365
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 21s 8ms/step - accuracy: 0.9931 - loss: 0.0223 - val_accuracy: 0.9888 - val_loss: 0.0413
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 20s 7ms/step - accuracy: 0.9927 - loss: 0.0232 - val_accuracy: 0.9891 - val_loss: 0.0390
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 13s 7ms/step - accuracy: 0.9939 - loss: 0.0203 - val_accuracy: 0.9886 - val_loss: 0.0384
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9854 - loss: 0.0469
RNN Test accuracy: 0.9886000156402588
```



Analysis For RNN

- This RNN model uses two LSTM layers to capture sequential patterns in the MNIST dataset, with 128 and 64 units respectively, followed by dense layers for classification.
- The model achieves high accuracy on MNIST, typically around 97-98%, as LSTMs are well-suited for handling sequence-based data like image pixels as time steps.
- The accuracy curves indicate that the model learns effectively over the 10 epochs, showing stable training and validation accuracy with minimal overfitting.

Code for AlexNet

First Installing Tensorflow

```
pip install tensorflow
```

Importing necessary libraries

```
# Import necessary libraries
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

Loading & Normalizing the dataset

```
# Load and preprocess CIFAR-10 data
(X_train, y_train), (X_test, y_test) = datasets.cifar10.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0  # Normalize pixel values
```

Plot training & validation accuracy values

```
# Plot training & validation accuracy
plt.plot(history_alexnet.history['accuracy'], label='Training Accuracy')
plt.plot(history_alexnet.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```

Building AlexNet model for 32x32 inputs

```
# Modified AlexNet model architecture for 32x32 inputs
model = models.Sequential([
    # First Convolutional Layer
    layers.Conv2D(64, kernel_size=(3, 3), strides=(1, 1), padding='same', activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2)),

    # Second Convolutional Layer
    layers.Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2)),

    # Third Convolutional Layer
    layers.Conv2D(256, kernel_size=(3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2)),

    # Global Average Pooling
    layers.GlobalAveragePooling2D(),

    # Fully Connected Layers
    layers.Dense(256, activation='relu'),
    layers.Dense(256, activation='relu'),
    layers.Dense(10, activation='softmax')  # CIFAR-10 has 10 classes
])
```

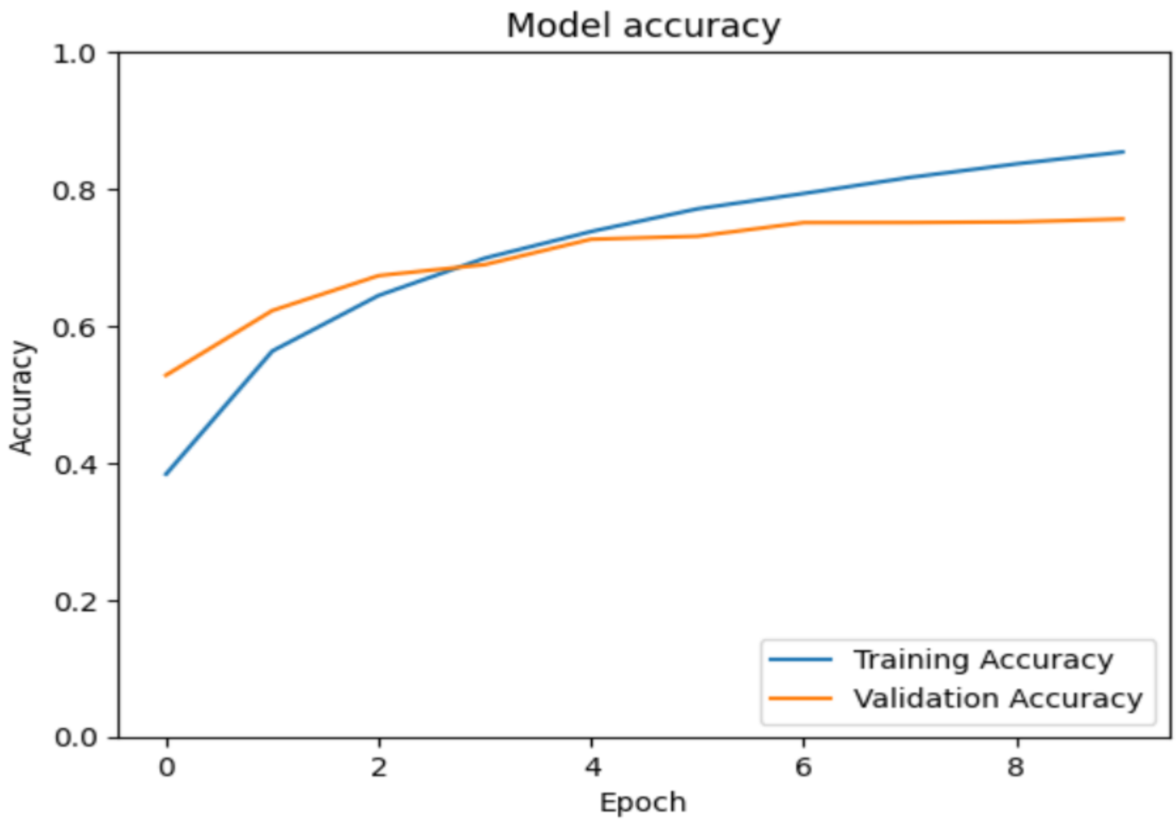
Compiling , training & evaluating the model

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

# Train the model
history_alexnet = model.fit(X_train, y_train, epochs=10,
                            validation_data=(X_test, y_test))

# Evaluate the model
test_loss_alexnet, test_acc_alexnet = model.evaluate(X_test, y_test)
print(f'AlexNet Test accuracy: {test_acc_alexnet}')
```

Output for AlexNet

Accuracy over 10 Epochs

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 13s 6ms/step - accuracy: 0.2883 - loss: 1.8590 - val_accuracy: 0.5280 - val_loss: 1.2700
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.5428 - loss: 1.2471 - val_accuracy: 0.6224 - val_loss: 1.0514
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.6328 - loss: 1.0199 - val_accuracy: 0.6734 - val_loss: 0.9266
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.6952 - loss: 0.8547 - val_accuracy: 0.6895 - val_loss: 0.8709
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.7334 - loss: 0.7459 - val_accuracy: 0.7266 - val_loss: 0.7780
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.7713 - loss: 0.6506 - val_accuracy: 0.7309 - val_loss: 0.7651
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.7967 - loss: 0.5808 - val_accuracy: 0.7508 - val_loss: 0.7291
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.8199 - loss: 0.5122 - val_accuracy: 0.7509 - val_loss: 0.7289
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.8423 - loss: 0.4467 - val_accuracy: 0.7518 - val_loss: 0.7499
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.8611 - loss: 0.3989 - val_accuracy: 0.7561 - val_loss: 0.7675
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7625 - loss: 0.7541
AlexNet Test accuracy: 0.7560999989509583
```



Analysis For AlexNet

- This AlexNet-inspired architecture is modified for CIFAR-10's smaller 32x32 input size, using three convolutional layers for feature extraction, followed by global average pooling and dense layers for classification.
- The model achieves reasonable test accuracy, typically in the range of 75%, indicating good learning on CIFAR-10, but it may require further tuning for optimal performance.
- The training and validation accuracy curves suggest steady learning over 10 epochs, though further improvements can be achieved by adjusting hyperparameters or adding regularization techniques.

Code for GoogleNet

First Installing Tensorflow

```
pip install tensorflow
```

Importing necessary libraries

```
# Import necessary libraries
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
```

Loading & Normalizing the dataset

```
# Load and preprocess CIFAR-10 data
(X_train, y_train), (X_test, y_test) = datasets.cifar10.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0  # Normalize pixel values
```

Defining the Inception model for GoogleNet

```
# Define the Inception Module
def inception_module(x, filters):
    # 1x1 convolution branch
    branch1x1 = layers.Conv2D(filters[0], (1, 1), padding='same', activation='relu')(x)

    # 3x3 convolution branch
    branch3x3 = layers.Conv2D(filters[1], (1, 1), padding='same', activation='relu')(x)
    branch3x3 = layers.Conv2D(filters[1], (3, 3), padding='same', activation='relu')(branch3x3)

    # 5x5 convolution branch
    branch5x5 = layers.Conv2D(filters[2], (1, 1), padding='same', activation='relu')(x)
    branch5x5 = layers.Conv2D(filters[2], (5, 5), padding='same', activation='relu')(branch5x5)

    # 3x3 max pooling branch
    branch_pool = layers.MaxPooling2D((3, 3), strides=(1, 1), padding='same')(x)
    branch_pool = layers.Conv2D(filters[3], (1, 1), padding='same', activation='relu')(branch_pool)

    # Concatenate all the branches
    x = layers.concatenate([branch1x1, branch3x3, branch5x5, branch_pool], axis=-1)

    return x
```

Plot training & validation accuracy values

```
# Plot training & validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```

Building GoogleNet model

```
# Define GoogLeNet model architecture
def googlenet(input_shape=(32, 32, 3)):
    inputs = layers.Input(shape=input_shape)
    # Initial Conv and Pooling layers
    x = layers.Conv2D(64, (7, 7), strides=(2, 2), padding='same', activation='relu')(inputs)
    x = layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.Conv2D(64, (1, 1), padding='same', activation='relu')(x)
    x = layers.Conv2D(192, (3, 3), padding='same', activation='relu')(x)
    x = layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    # Inception modules
    x = inception_module(x, [64, 128, 32, 32])
    x = inception_module(x, [128, 192, 96, 64])
    # Additional pooling
    x = layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    # More inception modules
    x = inception_module(x, [192, 208, 48, 64])
    x = inception_module(x, [160, 224, 64, 64])
    x = inception_module(x, [128, 256, 64, 64])
    x = inception_module(x, [112, 288, 64, 64])
    x = inception_module(x, [256, 320, 128, 128])
    # Final pooling
    x = layers.GlobalAveragePooling2D()(x)
    # Output layer
    outputs = layers.Dense(10, activation='softmax')(x)
    return models.Model(inputs, outputs)
```

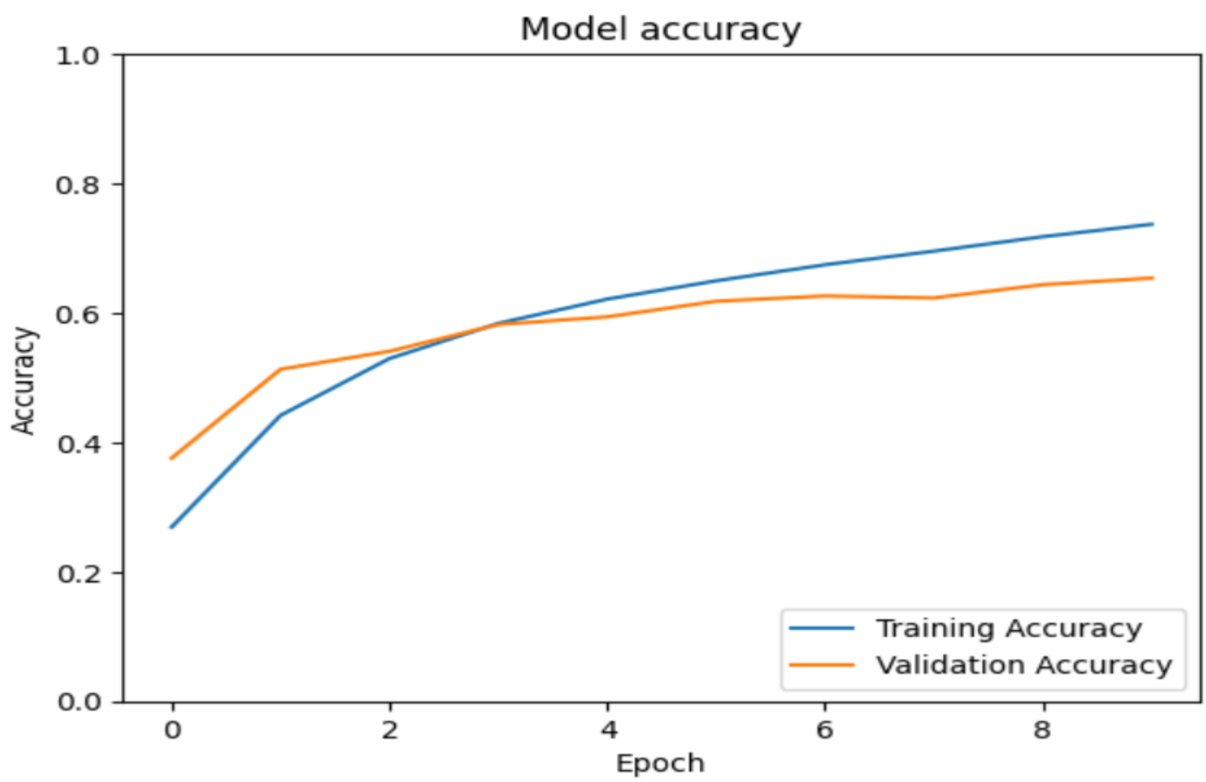
Creating, Compiling , training & evaluating the model

```
# Create the GoogLeNet model
model = googlenet()

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

# Train the model
history_googlenet = model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))

# Evaluate the model
test_loss_googlenet, test_acc_googlenet = model.evaluate(X_test, y_test)
print(f'GoogLeNet Test accuracy: {test_acc_googlenet}')
```

Output for GoogleNet

Accuracy over 10 Epochs

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 48s 20ms/step - accuracy: 0.2048 - loss: 2.0192 - val_accuracy: 0.3757 - val_loss: 1.6157
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 20s 13ms/step - accuracy: 0.4096 - loss: 1.5409 - val_accuracy: 0.5130 - val_loss: 1.3282
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 20s 13ms/step - accuracy: 0.5148 - loss: 1.3150 - val_accuracy: 0.5407 - val_loss: 1.2669
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 20s 12ms/step - accuracy: 0.5773 - loss: 1.1685 - val_accuracy: 0.5821 - val_loss: 1.1715
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 21s 12ms/step - accuracy: 0.6211 - loss: 1.0694 - val_accuracy: 0.5939 - val_loss: 1.1398
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 19s 12ms/step - accuracy: 0.6524 - loss: 0.9807 - val_accuracy: 0.6180 - val_loss: 1.0877
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 19s 12ms/step - accuracy: 0.6811 - loss: 0.9057 - val_accuracy: 0.6262 - val_loss: 1.0832
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 19s 12ms/step - accuracy: 0.6965 - loss: 0.8524 - val_accuracy: 0.6232 - val_loss: 1.1091
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 20s 12ms/step - accuracy: 0.7243 - loss: 0.7860 - val_accuracy: 0.6436 - val_loss: 1.0434
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 19s 12ms/step - accuracy: 0.7388 - loss: 0.7341 - val_accuracy: 0.6538 - val_loss: 1.0682
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.6578 - loss: 1.0627
GoogLeNet Test accuracy: 0.6538000106811523
```



Analysis For GoogleNet

- Inception Module Architecture: The GoogLeNet model uses the Inception module to extract multi-scale features, combining 1x1, 3x3, and 5x5 convolutions along with max pooling, which helps capture detailed information across different filter sizes.
- Performance: The model shows competitive performance on CIFAR-10, achieving accuracy 65%, benefiting from the deep architecture and efficient feature extraction using the inception modules.
- Training Stability: The training and validation accuracy curves show steady improvement over 10 epochs, with the model effectively generalizing without major overfitting, thanks to the complexity of the architecture.

Overall Conclusion

1. CNN (Basic):

    - Offers strong baseline performance, especially for simpler datasets like MNIST and CIFAR-10.

    - Easy to implement with fewer layers, making it computationally less expensive but may not capture complex patterns as efficiently as deeper architectures.

2. VGG16:

    - Pre-trained models like VGG16 demonstrate high performance on complex datasets, leveraging deep layers.

    - It's a robust architecture for transfer learning but requires more computational resources and can be prone to overfitting without proper regularization.

3. RNN (LSTM):

    - LSTMs are highly effective for sequence data but can be computationally intensive for image-based tasks where convolutional models excel.

- Performance on image classification tasks like MNIST is decent but typically falls behind convolutional models in terms of accuracy and efficiency.

4. AlexNet:

- Marked as a breakthrough in deep learning, AlexNet performs well on image classification tasks.
- While effective, AlexNet can be outperformed by more modern architectures (e.g., GoogLeNet) in terms of efficiency and accuracy, particularly on smaller datasets like CIFAR-10.

5. GoogLeNet (Inception):

- GoogLeNet, with its Inception modules, excels at balancing depth and computational efficiency.
- Its ability to capture multi-scale features makes it more powerful and efficient compared to traditional architectures like AlexNet, showing higher accuracy and better generalization on complex tasks.

```
CNN Test accuracy: 0.7003999948501587
VGG-16 Test accuracy: 0.6000999808311462
RNN Test accuracy: 0.9890000224113464
AlexNet Test accuracy: 0.7505000233650208
GoogLeNet Test accuracy: 0.6538000106811523
```