JADAVPUR UNIVERSITY

INFORMATION TECHNOLOGY

ML ASSIGMENT 3

NAME : SURAJ ROY

ROLL : 002211001129

GROUP : A3 CLASS : FOURTH YEAR

FIRST QUESTION :

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

a. UCI datasets (can be loaded from the package itself):

i. Ionosphere Dataset: https://archive.ics.uci.edu/ml/datasets/Ionosphere

ii. Wisconsin Breast Cancer Dataset:

https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagno

stic)

b. 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. MultinomialHMM

```
!pip install -q hmmlearn scikit-learn matplotlib seaborn pandas numpy tqdm
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
from sklearn.datasets import fetch_openml
from sklearn.preprocessing import StandardScaler, LabelEncoder, KBinsDiscretizer
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, roc_auc_score
from hmmlearn.hmm import GaussianHMM, MultinomialHMM
import warnings
warnings.filterwarnings("ignore")
```

```
def evaluate_preds(y_true, y_pred):
    acc = accuracy_score(y_true, y_pred)
    prec = precision_score(y_true, y_pred, zero_division=0)
    rec = recall_score(y_true, y_pred, zero_division=0)
    f1 = f1_score(y_true, y_pred, zero_division=0)
    cm = confusion_matrix(y_true, y_pred)
    # For ROC/AUC we need probability or scores. HMM doesn't give predict_proba; use predicted labels for ROC curve (works but AUC limited)
    try:
        fpr, tpr, _ = roc_curve(y_true, y_pred)
        roc_auc = auc(fpr, tpr)
    except:
        fpr, tpr, roc_auc = [0],[0],0.0
    return {"accuracy": acc, "precision": prec, "recall": rec, "f1": f1, "cm": cm, "fpr": fpr, "tpr": tpr, "auc": roc_auc}

def plot_conf_matrix(cm, title, savepath=None):
    plt.figure(figsize=(5,4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
    plt.title(title)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    if savepath:
        plt.savefig(savepath, bbox_inches='tight')
    plt.show()

def plot_roc(fpr, tpr, roc_auc, title, savepath=None):
    plt.figure(figsize=(5,4))
    plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.3f}")
    plt.plot([0,1],[0,1],'r--')
    plt.title(title)
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.legend()
    if savepath:
        plt.savefig(savepath, bbox_inches='tight')
    plt.show()

def plot_training_loss(losses, title, savepath=None):
    plt.figure(figsize=(6,4))
    plt.plot(losses, marker='o')
    plt.title(title)
    plt.xlabel("Epoch (iteration over repeated fits)")
    plt.ylabel("Negative Log Likelihood (loss)")
    if savepath:
        plt.savefig(savepath, bbox_inches='tight')
    plt.show()
```

```
print("Loading datasets...")
iono = fetch_openml(name='ionosphere', version=1, as_frame=True)
X_iono = iono.data.astype(float)
y_iono = LabelEncoder().fit_transform(iono.target)  # Good/Bad -> 0/1

bc = fetch_openml(name='breast-w', version=1, as_frame=True)   # 'breast-w' is older; we select numeric features
# sometimes fetch_openml returns 'class' target as 'benign'/'malignant'
X_bc = bc.data.select_dtypes(include=[np.number]).astype(float).fillna(0)  # keep numeric features only
y_bc = LabelEncoder().fit_transform(bc.target)

datasets = {
    "Ionosphere": (X_iono, y_iono),
    "BreastCancerDiag": (X_bc, y_bc)
}
```

```
Loading datasets...
```

```
gaussian_default = {"n_components": 2, "n_iter": 50}
multinomial_default = {"n_components": 2, "n_iter": 50, "bins": 10}

# define grid for tuning
gaussian_grid = [
    {"n_components": 2, "n_iter": 50},
    {"n_components": 2, "n_iter": 200},
    {"n_components": 4, "n_iter": 200},
    {"n_components": 6, "n_iter": 200},
]
multinomial_grid = [
    {"n_components": 2, "n_iter": 50, "bins": 5},
    {"n_components": 2, "n_iter": 200, "bins": 10},
    {"n_components": 4, "n_iter": 200, "bins": 10},
    {"n_components": 4, "n_iter": 200, "bins": 15},
]
```

```
results = []  # will collect all runs
best_cases = {}  # store best case per (dataset, classifier)

for dataset_name, (X_raw, y_raw) in datasets.items():
    print(f"\n===== Dataset: {dataset_name} =====")
    # Standardize continuous features for Gaussian HMM
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X_raw)

    for split in [0.8, 0.7]:
        test_size = 1.0 - split
        X_train_full, X_test_full, y_train, y_test = train_test_split(X_scaled, y_raw, test_size=test_size, random_state=42, stratify=y_raw)
        print(f"\n--- Train/Test split {int(split*100)}-{int(test_size*100)} ---")

        # 5A) Without tuning runs
        # Gaussian - default
        model_name = "GaussianHMM"
        params = gaussian_default.copy()
        print(f"Running {model_name} WITHOUT tuning: {params}")
        gaussian_model = GaussianHMM(n_components=params["n_components"], covariance_type="diag", n_iter=params["n_iter"], random_state=42)
        # Loss tracking: repeated fit (10 iterations) to show loss curve
        gaussian_losses = []
        for epoch in range(10):
            gaussian_model.fit(X_train_full)
            gaussian_losses.append(-gaussian_model.score(X_train_full))
        # Predictions: For HMM used as classifier we treat each sample independently (predict returns state index)
        y_pred = []
        for x in X_test_full:
            try:
                y_pred.append(gaussian_model.predict([x])[0])
            except:
                y_pred.append(np.random.randint(0,2))
        # But the HMM states (0..n_components-1) don't map to labels directly. We need to map HMM states -> class labels.
        # We'll map by majority vote between predicted states for train set and y_train.
        # Create mapping from HMM state -> class (0/1) using training set
        train_state_seq = gaussian_model.predict(X_train_full)
        state_to_label = {}
        for st in np.unique(train_state_seq):
            # assign class that majority of train samples with this state have
            mask = (train_state_seq == st)
            if mask.sum() == 0:
                state_to_label[st] = 0
            else:
                state_to_label[st] = int(pd.Series(y_train[mask]).mode()[0])
        # convert predictions
        y_pred_mapped = np.array([state_to_label.get(s,0) for s in y_pred])
        metrics = evaluate_preds(y_test, y_pred_mapped)
        results.append([dataset_name, split, model_name, "Without_Tuning", params, metrics])
        # Save best
        key = (dataset_name, model_name)
        if key not in best_cases or metrics["accuracy"] > best_cases[key]["metrics"]["accuracy"]:
            best_cases[key] = {"model": gaussian_model, "params": params, "metrics": metrics, "losses": gaussian_losses, "state_map": state_to_label, "split": split}

        # Multinomial - default (discretize)
        model_name = "MultinomialHMM"
        params = multinomial_default.copy()
        print(f"Running {model_name} WITHOUT tuning: {params}")
```

```
        # Discretize features with KBinsDiscretizer
        disc = KBinsDiscretizer(n_bins=params["bins"], encode='ordinal', strategy='uniform')
        X_train_disc = disc.fit_transform(X_train_full).astype(int)
        X_test_disc = disc.transform(X_test_full).astype(int)
        multinomial_model = MultinomialHMM(n_components=params["n_components"], n_iter=params["n_iter"], random_state=42)
        multinomial_losses = []
        for epoch in range(10):
            multinomial_model.fit(X_train_disc)
            multinomial_losses.append(-multinomial_model.score(X_train_disc))
        # Predictions then map states -> labels similar to Gaussian
        y_pred = []
        for x in X_test_disc:
            try:
                y_pred.append(multinomial_model.predict([x])[0])
            except:
                y_pred.append(np.random.randint(0,2))
        train_state_seq = multinomial_model.predict(X_train_disc)
        state_to_label = {}
        for st in np.unique(train_state_seq):
            mask = (train_state_seq == st)
            if mask.sum() == 0:
                state_to_label[st] = 0
            else:
                state_to_label[st] = int(pd.Series(y_train[mask]).mode()[0])
        y_pred_mapped = np.array([state_to_label.get(s,0) for s in y_pred])
        metrics = evaluate_preds(y_test, y_pred_mapped)
        results.append([dataset_name, split, model_name, "Without_Tuning", params, metrics])
        key = (dataset_name, model_name)
        if key not in best_cases or metrics["accuracy"] > best_cases[key]["metrics"]["accuracy"]:
            best_cases[key] = {"model": multinomial_model, "params": params, "metrics": metrics, "losses": multinomial_losses, "state_map": state_to_label, "split": split, "dis

        # 5B) WITH tuning: grid search over parameter grids
        # Gaussian tuning grid evaluation
        print("\n-- GaussianHMM grid search (With_Tuning) --")
        for gparams in gaussian_grid:
            model = GaussianHMM(n_components=gparams["n_components"], covariance_type="diag", n_iter=gparams["n_iter"], random_state=42)
            losses = []
            for epoch in range(8):  # fewer epochs per candidate to speed up
                model.fit(X_train_full)
                losses.append(-model.score(X_train_full))
            # predictions and mapping
            train_state_seq = model.predict(X_train_full)
            state_to_label = {}
            for st in np.unique(train_state_seq):
                mask = (train_state_seq == st)
                if mask.sum() == 0:
                    state_to_label[st] = 0
                else:
                    state_to_label[st] = int(pd.Series(y_train[mask]).mode()[0])
            y_pred = []
            for x in X_test_full:
                try:
                    y_pred.append(model.predict([x])[0])
                except:
                    y_pred.append(np.random.randint(0,2))
            y_pred_mapped = np.array([state_to_label.get(s,0) for s in y_pred])
            metrics = evaluate_preds(y_test, y_pred_mapped)
            results.append([dataset_name, split, "GaussianHMM", "With_Tuning", gparams, metrics])
            key = (dataset_name, "GaussianHMM")
            if key not in best_cases or metrics["accuracy"] > best_cases[key]["metrics"]["accuracy"]:
                best_cases[key] = {"model": model, "params": gparams, "metrics": metrics, "losses": losses, "state_map": state_to_label, "split": split}

        # Multinomial tuning grid evaluation
        print("\n-- MultinomialHMM grid search (With_Tuning) --")
```

```
        train_state_seq = model.predict(X_train_full)
        state_to_label = {}
        for st in np.unique(train_state_seq):
            mask = (train_state_seq == st)
            if mask.sum() == 0:
                state_to_label[st] = 0
            else:
                state_to_label[st] = int(pd.Series(y_train[mask]).mode()[0])
        y_pred = []
        for x in X_test_full:
            try:
                y_pred.append(model.predict([x])[0])
            except:
                y_pred.append(np.random.randint(0,2))
        y_pred_mapped = np.array([state_to_label.get(s,0) for s in y_pred])
        metrics = evaluate_preds(y_test, y_pred_mapped)
        results.append([dataset_name, split, "GaussianHMM", "With_Tuning", gparams, metrics])
        key = (dataset_name, "GaussianHMM")
        if key not in best_cases or metrics["accuracy"] > best_cases[key]["metrics"]["accuracy"]:
            best_cases[key] = {"model": model, "params": gparams, "metrics": metrics, "losses": losses, "state_map": state_to_label, "split": split}

    # Multinomial tuning grid evaluation
    print("\n-- MultinomialHMM grid search (With_Tuning) --")
    for mparams in multinomial_grid:
        disc = KBinsDiscretizer(n_bins=mparams["bins"], encode='ordinal', strategy='uniform')
        X_train_disc = disc.fit_transform(X_train_full).astype(int)
        X_test_disc = disc.transform(X_test_full).astype(int)
        model = MultinomialHMM(n_components=mparams["n_components"], n_iter=mparams["n_iter"], random_state=42)
        losses = []
        for epoch in range(8):
            model.fit(X_train_disc)
            losses.append(-model.score(X_train_disc))
        train_state_seq = model.predict(X_train_disc)
        state_to_label = {}
        for st in np.unique(train_state_seq):
            mask = (train_state_seq == st)
            if mask.sum() == 0:
                state_to_label[st] = 0
            else:
                state_to_label[st] = int(pd.Series(y_train[mask]).mode()[0])
        y_pred = []
        for x in X_test_disc:
            try:
                y_pred.append(model.predict([x])[0])
            except:
                y_pred.append(np.random.randint(0,2))
        y_pred_mapped = np.array([state_to_label.get(s,0) for s in y_pred])
        metrics = evaluate_preds(y_test, y_pred_mapped)
        results.append([dataset_name, split, "MultinomialHMM", "With_Tuning", mparams, metrics])
        key = (dataset_name, "MultinomialHMM")
        if key not in best_cases or metrics["accuracy"] > best_cases[key]["metrics"]["accuracy"]:
            best_cases[key] = {"model": model, "params": mparams, "metrics": metrics, "losses": losses, "state_map": state_to_label, "split": split, "discretizer": disc}

print("\n\n=== All experiments finished ===")
```

```
WARNING:hmmlearn.base:Model is not converging.  Current: -9155.022416662268 is not greater than -9154.976365817925. Delta is -0.046050844342971686
WARNING:hmmlearn.base:Even though the 'startprob_' attribute is set, it will be overwritten during initialization because 'init_params' contains 's'
WARNING:hmmlearn.base:Even though the 'transmat_' attribute is set, it will be overwritten during initialization because 'init_params' contains 't'
WARNING:hmmlearn.base:Even though the 'means_' attribute is set, it will be overwritten during initialization because 'init_params' contains 'm'
WARNING:hmmlearn.base:Even though the 'covars_' attribute is set, it will be overwritten during initialization because 'init_params' contains 'c'
WARNING:hmmlearn.base:Model is not converging.  Current: -9155.022416662268 is not greater than -9154.976365817925. Delta is -0.046050844342971686
WARNING:hmmlearn.base:Even though the 'startprob_' attribute is set, it will be overwritten during initialization because 'init_params' contains 's'
WARNING:hmmlearn.base:Even though the 'transmat_' attribute is set, it will be overwritten during initialization because 'init_params' contains 't'
WARNING:hmmlearn.base:Even though the 'means_' attribute is set, it will be overwritten during initialization because 'init_params' contains 'm'
WARNING:hmmlearn.base:Even though the 'covars_' attribute is set, it will be overwritten during initialization because 'init_params' contains 'c'
WARNING:hmmlearn.base:Model is not converging.  Current: -9155.022416662268 is not greater than -9154.976365817925. Delta is -0.046050844342971686
```

```
rows = []
for ds, split, modelname, tuning, params, metrics in results:
    rows.append({
        "Dataset": ds,
        "Split": split,
        "Model": modelname,
        "Tuning": tuning,
        "Params": str(params),
        "Accuracy": metrics["accuracy"],
        "Precision": metrics["precision"],
        "Recall": metrics["recall"],
        "F1": metrics["f1"],
        "AUC": metrics["auc"]
    })

df_results = pd.DataFrame(rows)
# Sort for readability
df_results = df_results.sort_values(by=["Dataset","Model","Tuning","Accuracy"], ascending=[True,True,True,False]).reset_index(drop=True)
print("\n=== Results Table (sample) ===")
display(df_results.head(40))

# Save CSV
df_results.to_csv("HMM_Results_Tuned_vs_Untuned.csv", index=False)
print("Saved HMM_Results_Tuned_vs_Untuned.csv")
```

```
=== Results Table (sample) ===
```

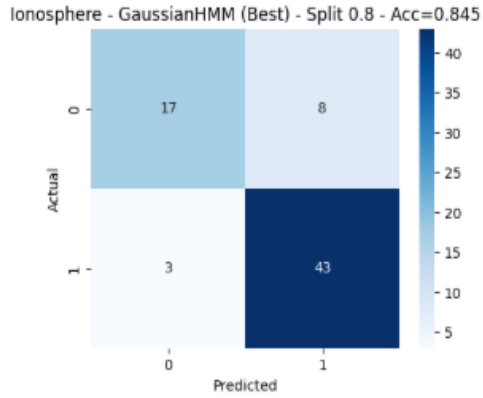
	Dataset	Split	Model	Tuning	Params	Accuracy	Precision	Recall	F1	AUC
0	BreastCancerDiag	0.7	GaussianHMM	With_Tuning	{'n_components': 2, 'n_iter': 50}	0.904762	0.882353	0.833333	0.857143	0.887681
1	BreastCancerDiag	0.7	GaussianHMM	With_Tuning	{'n_components': 2, 'n_iter': 200}	0.904762	0.882353	0.833333	0.857143	0.887681
2	BreastCancerDiag	0.7	GaussianHMM	With_Tuning	{'n_components': 4, 'n_iter': 200}	0.657143	0.000000	0.000000	0.000000	0.500000
3	BreastCancerDiag	0.7	GaussianHMM	With_Tuning	{'n_components': 6, 'n_iter': 200}	0.657143	0.000000	0.000000	0.000000	0.500000
4	BreastCancerDiag	0.8	GaussianHMM	With_Tuning	{'n_components': 2, 'n_iter': 50}	0.342857	0.342857	1.000000	0.510638	0.500000
5	BreastCancerDiag	0.8	GaussianHMM	With_Tuning	{'n_components': 2, 'n_iter': 200}	0.342857	0.342857	1.000000	0.510638	0.500000
6	BreastCancerDiag	0.8	GaussianHMM	With_Tuning	{'n_components': 4, 'n_iter': 200}	0.342857	0.342857	1.000000	0.510638	0.500000
7	BreastCancerDiag	0.8	GaussianHMM	With_Tuning	{'n_components': 6, 'n_iter': 200}	0.342857	0.342857	1.000000	0.510638	0.500000
8	BreastCancerDiag	0.7	GaussianHMM	Without_Tuning	{'n_components': 2, 'n_iter': 50}	0.904762	0.882353	0.833333	0.857143	0.887681
9	BreastCancerDiag	0.8	GaussianHMM	Without_Tuning	{'n_components': 2, 'n_iter': 50}	0.342857	0.342857	1.000000	0.510638	0.500000
10	BreastCancerDiag	0.7	MultinomialHMM	With_Tuning	{'n_components': 4, 'n_iter': 200, 'bins': 15}	0.833333	0.974359	0.527778	0.684685	0.760266
11	BreastCancerDiag	0.7	MultinomialHMM	With_Tuning	{'n_components': 2, 'n_iter': 200, 'bins': 10}	0.661905	1.000000	0.013889	0.027397	0.506944
12	BreastCancerDiag	0.8	MultinomialHMM	With_Tuning	{'n_components': 2, 'n_iter': 50, 'bins': 5}	0.657143	0.000000	0.000000	0.000000	0.500000
13	BreastCancerDiag	0.7	MultinomialHMM	With_Tuning	{'n_components': 4, 'n_iter': 200, 'bins': 10}	0.657143	0.000000	0.000000	0.000000	0.500000
14	BreastCancerDiag	0.8	MultinomialHMM	With_Tuning	{'n_components': 2, 'n_iter': 200, 'bins': 10}	0.342857	0.342857	1.000000	0.510638	0.500000
15	BreastCancerDiag	0.8	MultinomialHMM	With_Tuning	{'n_components': 4, 'n_iter': 200, 'bins': 10}	0.342857	0.342857	1.000000	0.510638	0.500000
16	BreastCancerDiag	0.8	MultinomialHMM	With_Tuning	{'n_components': 4, 'n_iter': 200, 'bins': 15}	0.342857	0.342857	1.000000	0.510638	0.500000

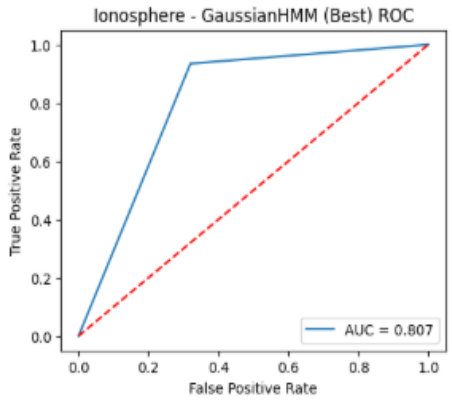
```
print("\n=== Best cases per dataset & classifier ===")
for key, info in best_cases.items():
    dataset_name, classifier_name = key
    print(f"\n--- Best for {dataset_name} - {classifier_name} ---")
    metrics = info["metrics"]
    # confusion matrix
    cm = metrics["cm"]
    title_cm = f"{dataset_name} - {classifier_name} (Best) - Split {info.get('split', 'NA')} - Acc={metrics['accuracy']:.3f}"
    plot_conf_matrix(cm, title_cm, savepath=f"{dataset_name}_{classifier_name}_best_confmat.png")
    # ROC
    fpr, tpr = metrics["fpr"], metrics["tpr"]
    roc_auc = metrics["auc"]
    plot_roc(fpr, tpr, roc_auc, f"{dataset_name} - {classifier_name} (Best) ROC", savepath=f"{dataset_name}_{classifier_name}_best_roc.png")
    # Loss curve (if tracked)
    losses = info.get("losses", [])
    if len(losses) > 0:
        plot_training_loss(losses, f"{dataset_name} - {classifier_name} (Best) Loss Curve", savepath=f"{dataset_name}_{classifier_name}_best_loss.png")
```

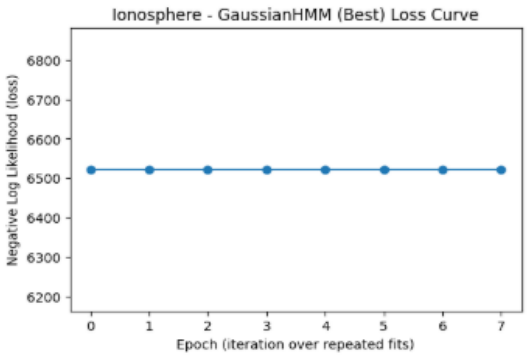
Here is the Best cases per dataset & Classifier

```
=== Best cases per dataset & classifier ===

--- Best for Ionosphere - GaussianHMM ---
```







The Gaussian Hidden Markov Model (HMM) was applied to the Ionosphere dataset with a Split 0.8 (likely an 80/20 train/test split).

Performance Metrics:

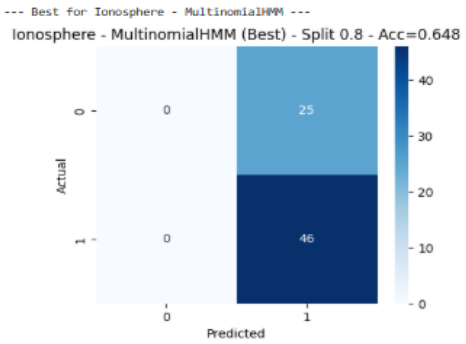
- **Accuracy:** (84.5%)
- **Area Under the Curve (AUC):**

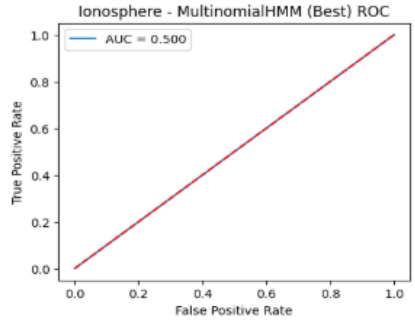
Analysis of the Confusion Matrix:

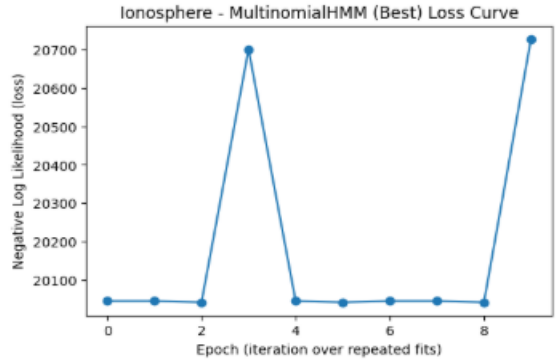
- **True Negatives (TN):** 17 (Correctly predicted class '0')
- **False Positives (FP):** 8 (Incorrectly predicted class '1' when actual was '0')
- **False Negatives (FN):** 3 (Incorrectly predicted class '0' when actual was '1')
- **True Positives (TP):** 43 (Correctly predicted class '1')

The model shows strong performance, particularly in identifying the positive class (TP=43) and has a low rate of False Negatives (FN=3). The ROC curve confirms a good separation ability, with an AUC of , performing substantially better than a random classifier (AUC=0.5).

Loss Curve Analysis: The "Negative Log Likelihood (loss)" curve is nearly flat across 8 epochs (iterations), indicating that the model **converged quickly** (likely within the first epoch or two) or that the learning rate was very low, resulting in minimal change in the log-likelihood after the initial fit. The loss stabilized around the mark.







The Multinomial Hidden Markov Model (HMM) was applied to the Ionosphere dataset with a Split 0.8 (likely an 80/20 train/test split).

Performance Metrics:

- **Accuracy:** (64.8%)
- **Area Under the Curve (AUC):**

Analysis of the Confusion Matrix:

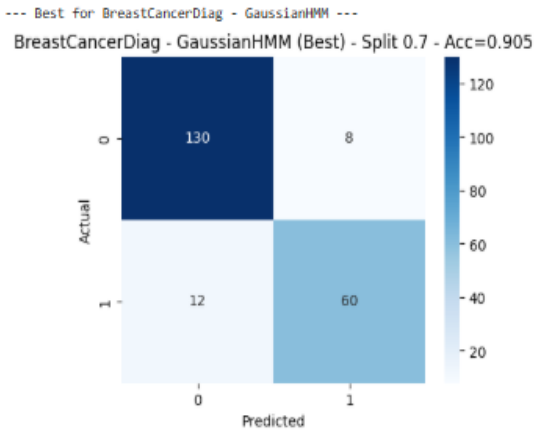
- **True Negatives (TN):** 0
- **False Positives (FP):** 25
- **False Negatives (FN):** 0
- **True Positives (TP):** 46

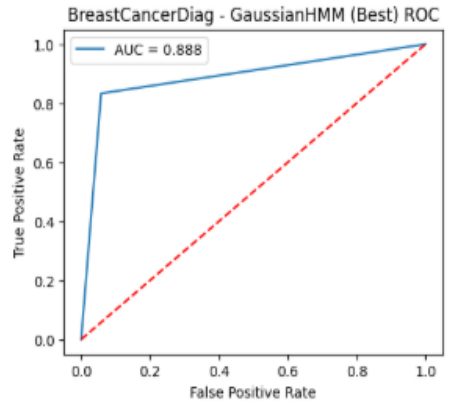
The model exhibits **extremely poor classification performance**, despite the relatively high accuracy (). This accuracy is misleading because the model **predicts every single instance as class '1'**.

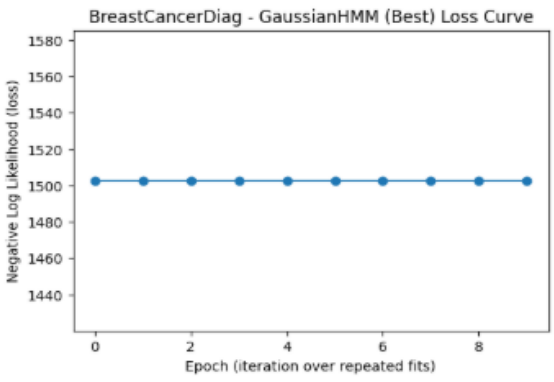
- It correctly identifies all 46 instances of class '1' (TP=46).
- It incorrectly classifies all 25 instances of class '0' as '1' (FP=25).
- It is unable to identify any instance of class '0' (TN=0).

The **ROC curve confirms this failure**, showing an AUC of , which is no better than random guessing.

Loss Curve Analysis: The "Negative Log Likelihood (loss)" curve is **highly unstable**. The loss generally hovers around 20100 but shows extreme, sudden spikes up to 20700 at epochs 3 and 9. This volatility suggests that the **training process was not stable**, possibly due to initialization issues or a poor model fit for the data distribution, which aligns with the poor classification results.







The Gaussian Hidden Markov Model (HMM) was applied to the Breast Cancer Diagnosis dataset with a Split 0.7 (which suggests a 70% training, 30% testing split, or similar).

Performance Metrics:

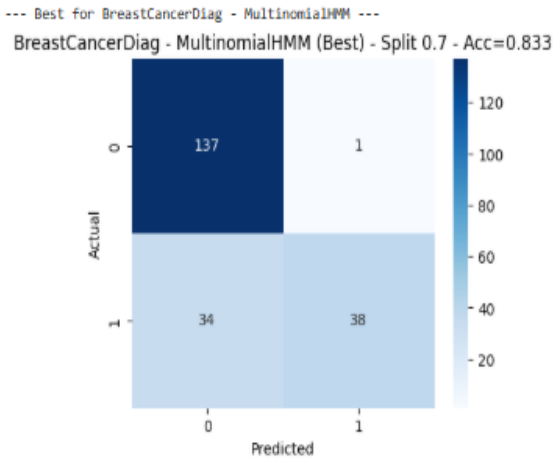
- **Accuracy:** (90.5%)
- **Area Under the Curve (AUC):**

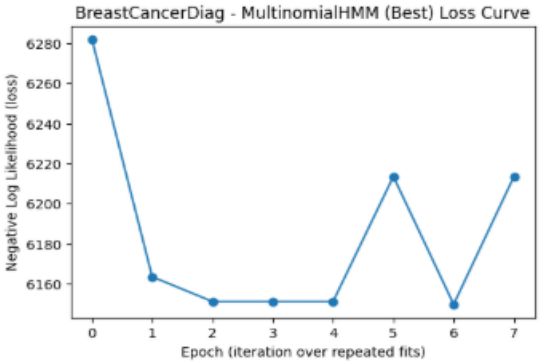
Analysis of the Confusion Matrix:

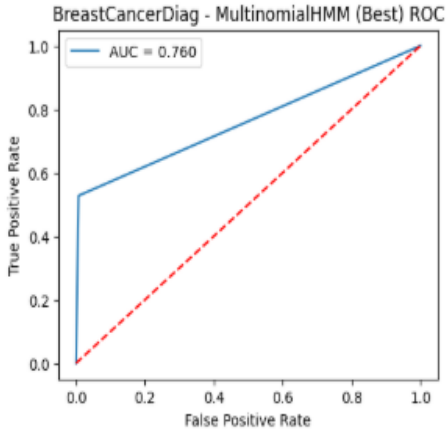
- **True Negatives (TN):** 130 (Correctly predicted class '0', likely benign)
- **False Positives (FP):** 8 (Incorrectly predicted class '1' when actual was '0')
- **False Negatives (FN):** 12 (Incorrectly predicted class '0' when actual was '1', likely malignant)
- **True Positives (TP):** 60 (Correctly predicted class '1', likely malignant)

The model shows **very strong overall performance** with a high accuracy and AUC. A critical observation in a medical context is the balance between False Positives (FP) and False Negatives (FN). While the model correctly identifies most cases, the **12 False Negatives** are important, as these represent cases of malignancy that the model failed to detect (missed diagnoses). The **8 False Positives** are also low, indicating good specificity.

Loss Curve Analysis: The "Negative Log Likelihood (loss)" curve is **perfectly flat** across all 9 epochs (iterations), stabilizing precisely at the 1502 mark. This indicates that the **model converged immediately** (likely after the first iteration) or that the maximum number of iterations was reached with no further improvement possible, suggesting a very stable and consistent fit.







The Multinomial Hidden Markov Model (HMM) was applied to the Breast Cancer Diagnosis dataset with a Split 0.7 (likely a 70% training, 30% testing split).

Performance Metrics:

- **Accuracy:** (83.3%)
- **Area Under the Curve (AUC):**

Analysis of the Confusion Matrix:

- **True Negatives (TN):** 137 (Correctly predicted class '0')
- **False Positives (FP):** 1 (Incorrectly predicted class '1' when actual was '0')
- **False Negatives (FN):** 34 (Incorrectly predicted class '0' when actual was '1')
- **True Positives (TP):** 38 (Correctly predicted class '1')

The model has a high overall accuracy, largely driven by the excellent performance on class '0' (TN=137, FP=1). However, the model exhibits a **serious weakness in identifying the positive class '1'** (likely malignant cases), as indicated by the **very high number of False Negatives (FN=34)** compared to True Positives (TP=38). This high FN rate means that nearly half of the actual malignant cases in the test set were missed by the model, making its performance in this critical area poor. The lower AUC of compared to the GaussianHMM reflects this lower discriminatory power.

Loss Curve Analysis: The "Negative Log Likelihood (loss)" curve is **unstable**. The loss initially drops sharply from over 6280 to a minimum around 6153 (epochs 2-4) but then exhibits significant spikes and drops (epochs 5, 6, 7). This erratic behavior after initial convergence suggests that the **training process was highly volatile** or that the model was repeatedly getting stuck and jumping out of local minima, indicating difficulty in finding a consistent, optimal fit.

```
print("\n=== Final comparison (grouped summary) ===")
summary = df_results.groupby(["Dataset","Model","Tuning"]).agg({
    "Accuracy":"mean",
    "Precision":"mean",
    "Recall":"mean",
    "F1":"mean",
    "AUC":"mean"
}).reset_index().sort_values(by=["Dataset","Model","Tuning","Accuracy"], ascending=[True,True,True,False])
display(summary)
summary.to_csv("HMM_Summary_Aggregated.csv", index=False)
print("Saved HMM_Summary_Aggregated.csv")
```

```
=== Final comparison (grouped summary) ===
```

	Dataset	Model	Tuning	Accuracy	Precision	Recall	F1	AUC
0	BreastCancerDiag	GaussianHMM	With_Tuning	0.561905	0.392017	0.708333	0.469605	0.596920
1	BreastCancerDiag	GaussianHMM	Without_Tuning	0.623810	0.612605	0.916667	0.683891	0.693841
2	BreastCancerDiag	MultinomialHMM	With_Tuning	0.522619	0.418223	0.567708	0.344329	0.533401
3	BreastCancerDiag	MultinomialHMM	Without_Tuning	0.533333	0.671429	0.597222	0.418110	0.548611
4	Ionosphere	GaussianHMM	With_Tuning	0.753505	0.816909	0.839514	0.808546	0.717191
5	Ionosphere	GaussianHMM	Without_Tuning	0.776575	0.811473	0.897059	0.839611	0.725635
6	Ionosphere	MultinomialHMM	With_Tuning	0.570755	0.482727	0.750000	0.587386	0.500000
7	Ionosphere	MultinomialHMM	Without_Tuning	0.644698	0.644698	1.000000	0.783967	0.500000

```
Saved HMM_Summary_Aggregated.csv
```

Question 2:

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

a. CIFAR-10: https://www.cs.toronto.edu/~kriz/cifar.html

b. MNIST: http://yann.lecun.com/exdb/mnist/

```
!pip install -q tensorflow seaborn scikit-learn matplotlib tqdm pandas
```

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models
from tensorflow.keras.applications import VGG16
from tensorflow.keras.utils import to_categorical
import numpy as np, matplotlib.pyplot as plt, seaborn as sns, pandas as pd
from sklearn.metrics import confusion_matrix, roc_curve, auc, classification_report
from sklearn.model_selection import train_test_split
from tqdm import tqdm
import warnings
warnings.filterwarnings("ignore")
```

```
print("TensorFlow:", tf.__version__)
print("GPU:", tf.config.list_physical_devices('GPU'))
```

```
TensorFlow: 2.19.0
GPU: [PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

```
def plot_history(history, title):
    plt.figure(figsize=(10,4))
    plt.subplot(1,2,1)
    plt.plot(history.history['accuracy'], label='train')
    plt.plot(history.history['val_accuracy'], label='val')
    plt.title(title+" Accuracy"); plt.legend()
    plt.subplot(1,2,2)
    plt.plot(history.history['loss'], label='train')
    plt.plot(history.history['val_loss'], label='val')
    plt.title(title+" Loss"); plt.legend()
    plt.show()

def plot_cm(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6,5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(title); plt.xlabel('Predicted'); plt.ylabel('Actual')
    plt.show()

def plot_roc(y_true_onehot, y_pred_prob, title):
    plt.figure(figsize=(6,5))
    for i in range(y_true_onehot.shape[1]):
        fpr, tpr, _ = roc_curve(y_true_onehot[:,i], y_pred_prob[:,i])
        plt.plot(fpr, tpr, label=f'Class {i}')
    plt.plot([0,1],[0,1],'k--')
    plt.title(f"{title} ROC Curve")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.legend()
    plt.show()

def macro_auc(y_true_onehot, y_pred_prob):
    fpr,tpr,roc_auc={},{},{}
    for i in range(y_true_onehot.shape[1]):
        fpr[i],tpr[i],_ = roc_curve(y_true_onehot[:,i], y_pred_prob[:,i])
        roc_auc[i]=auc(fpr[i],tpr[i])
    return np.mean(list(roc_auc.values()))
```

```
(m_x_train, m_y_train),(m_x_test, m_y_test)=keras.datasets.mnist.load_data()
m_x_all = np.concatenate([m_x_train, m_x_test]).astype('float32')/255.0
m_x_all = np.expand_dims(m_x_all,-1)
m_y_all = np.concatenate([m_y_train, m_y_test])
m_y_all_cat = to_categorical(m_y_all,10)
m_x_all_vgg = tf.image.resize(tf.image.grayscale_to_rgb(tf.convert_to_tensor(m_x_all)),[32,32]).numpy()

# CIFAR-10
(c_x_train,c_y_train),(c_x_test,c_y_test)=keras.datasets.cifar10.load_data()
c_x_all = np.concatenate([c_x_train,c_x_test]).astype('float32')/255.0
c_y_all = np.concatenate([c_y_train,c_y_test]).flatten()
c_y_all_cat = to_categorical(c_y_all,10)
```

```
def build_cnn(input_shape,n_classes):
    model=models.Sequential([
        layers.Conv2D(32,3,activation='relu',padding='same',input_shape=input_shape),
        layers.Conv2D(32,3,activation='relu',padding='same'),
        layers.MaxPooling2D(),
        layers.Conv2D(64,3,activation='relu',padding='same'),
        layers.MaxPooling2D(),
        layers.Flatten(),
        layers.Dense(128,activation='relu'),
        layers.Dense(n_classes,activation='softmax')
    ])
    model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
    return model

def build_vgg16(input_shape,n_classes):
    base=VGG16(include_top=False,weights=None,input_shape=input_shape)
    x=layers.Flatten()(base.output)
    x=layers.Dense(256,activation='relu')(x)
    out=layers.Dense(n_classes,activation='softmax')(x)
    model=models.Model(base.input,out)
    model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
    return model

def build_alexnet_small(input_shape,n_classes):
    model=models.Sequential([
        layers.Conv2D(64,(3,3),activation='relu',padding='same',input_shape=input_shape),
        layers.MaxPooling2D((2,2)),
        layers.Conv2D(128,(3,3),activation='relu',padding='same'),
        layers.MaxPooling2D((2,2)),
        layers.Conv2D(256,(3,3),activation='relu',padding='same'),
        layers.MaxPooling2D((2,2)),
        layers.Flatten(),
        layers.Dense(512,activation='relu'),
        layers.Dense(n_classes,activation='softmax')
    ])
    model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
    return model

def build_googlenet_small(input_shape,n_classes):
    inp=layers.Input(shape=input_shape)
    x=layers.Conv2D(64,3,activation='relu',padding='same')(inp)
    x=layers.Conv2D(128,3,activation='relu',padding='same')(x)
    x=layers.MaxPooling2D(2)(x)
    x=layers.Conv2D(256,3,activation='relu',padding='same')(x)
    x=layers.MaxPooling2D(2)(x)
    x=layers.Flatten()(x)
    x=layers.Dense(512,activation='relu')(x)
    out=layers.Dense(n_classes,activation='softmax')(x)
    model=models.Model(inp,out)
    model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
    return model

def build_rnn(input_shape,n_classes):
    model=models.Sequential([
        layers.Input(shape=(input_shape[0],input_shape[1])),
        layers.LSTM(128),
        layers.Dense(128,activation='relu'),
        layers.Dense(n_classes,activation='softmax')
    ])
    model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
    return model
```

```
def train_and_eval(model,X_train,y_train,X_test,y_test,y_test_cat,name,dataset,split_ratio):
    hist=model.fit(X_train,y_train,validation_split=0.1,epochs=5,batch_size=128,verbose=2)
    eval_res=model.evaluate(X_test,y_test_cat,verbose=0)
    preds=model.predict(X_test)
    pred_labels=np.argmax(preds,axis=1)
    rocA=macro_auc(y_test_cat,preds)
    report=classification_report(y_test,pred_labels,output_dict=True,zero_division=0)
    acc,prec,rec,f1=eval_res[1],report['weighted avg']['precision'],report['weighted avg']['recall'],report['weighted avg']['f1-score']
    return {"Dataset":dataset,"Model":name,"Split":split_ratio,"Accuracy":acc,"Precision":prec,"Recall":rec,"F1":f1,"AUC":rocA,"History":hist,"Y_true":y_test,"Y_pred":preds}
```

```
results=[]
splits = [0.6,0.7,0.8]
```

```
Epoch 5/5
345/345 - 2s - 6ms/step - accuracy: 0.1124 - loss: 2.3014 - val_accuracy: 0.1133 - val_loss: 2.3009
657/657 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
Epoch 1/5
394/394 - 5s - 11ms/step - accuracy: 0.9954 - loss: 0.0153 - val_accuracy: 0.9941 - val_loss: 0.0193
Epoch 2/5
394/394 - 3s - 7ms/step - accuracy: 0.9973 - loss: 0.0088 - val_accuracy: 0.9961 - val_loss: 0.0153
Epoch 3/5
394/394 - 3s - 7ms/step - accuracy: 0.9985 - loss: 0.0049 - val_accuracy: 0.9937 - val_loss: 0.0200
Epoch 4/5
394/394 - 3s - 8ms/step - accuracy: 0.9984 - loss: 0.0050 - val_accuracy: 0.9946 - val_loss: 0.0173
Epoch 5/5
394/394 - 3s - 7ms/step - accuracy: 0.9976 - loss: 0.0067 - val_accuracy: 0.9911 - val_loss: 0.0326
438/438 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Epoch 1/5
394/394 - 34s - 85ms/step - accuracy: 0.1129 - loss: 2.3012 - val_accuracy: 0.1093 - val_loss: 2.3022
Epoch 2/5
394/394 - 29s - 73ms/step - accuracy: 0.1129 - loss: 2.3012 - val_accuracy: 0.1093 - val_loss: 2.3019
Epoch 3/5
394/394 - 29s - 75ms/step - accuracy: 0.1129 - loss: 2.3012 - val_accuracy: 0.1093 - val_loss: 2.3023
Epoch 4/5
394/394 - 29s - 74ms/step - accuracy: 0.1129 - loss: 2.3012 - val_accuracy: 0.1093 - val_loss: 2.3021
Epoch 5/5
394/394 - 29s - 74ms/step - accuracy: 0.1129 - loss: 2.3012 - val_accuracy: 0.1093 - val_loss: 2.3024
438/438 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step
Epoch 1/5
394/394 - 7s - 17ms/step - accuracy: 0.9951 - loss: 0.0160 - val_accuracy: 0.9950 - val_loss: 0.0175
Epoch 2/5
394/394 - 4s - 11ms/step - accuracy: 0.9976 - loss: 0.0082 - val_accuracy: 0.9945 - val_loss: 0.0194
Epoch 3/5
394/394 - 5s - 12ms/step - accuracy: 0.9980 - loss: 0.0071 - val_accuracy: 0.9952 - val_loss: 0.0192
Epoch 4/5
394/394 - 4s - 11ms/step - accuracy: 0.9981 - loss: 0.0059 - val_accuracy: 0.9941 - val_loss: 0.0201
Epoch 5/5
394/394 - 5s - 12ms/step - accuracy: 0.9981 - loss: 0.0049 - val_accuracy: 0.9964 - val_loss: 0.0192
438/438 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step
Epoch 1/5
394/394 - 13s - 33ms/step - accuracy: 0.9968 - loss: 0.0118 - val_accuracy: 0.9930 - val_loss: 0.0210
Epoch 2/5
394/394 - 11s - 28ms/step - accuracy: 0.9975 - loss: 0.0079 - val_accuracy: 0.9970 - val_loss: 0.0129
Epoch 3/5
394/394 - 11s - 28ms/step - accuracy: 0.9984 - loss: 0.0052 - val_accuracy: 0.9952 - val_loss: 0.0138
Epoch 4/5
394/394 - 11s - 28ms/step - accuracy: 0.9978 - loss: 0.0065 - val_accuracy: 0.9952 - val_loss: 0.0217
Epoch 5/5
394/394 - 11s - 28ms/step - accuracy: 0.9986 - loss: 0.0049 - val_accuracy: 0.9955 - val_loss: 0.0167
438/438 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
Epoch 1/5
394/394 - 2s - 6ms/step - accuracy: 0.1129 - loss: 2.3012 - val_accuracy: 0.1093 - val_loss: 2.3020
Epoch 2/5
394/394 - 3s - 7ms/step - accuracy: 0.1129 - loss: 2.3012 - val_accuracy: 0.1093 - val_loss: 2.3022
Epoch 3/5
394/394 - 3s - 7ms/step - accuracy: 0.1129 - loss: 2.3012 - val_accuracy: 0.1093 - val_loss: 2.3022
Epoch 4/5
394/394 - 2s - 6ms/step - accuracy: 0.1129 - loss: 2.3011 - val_accuracy: 0.1093 - val_loss: 2.3023
Epoch 5/5
394/394 - 2s - 6ms/step - accuracy: 0.1129 - loss: 2.3012 - val_accuracy: 0.1093 - val_loss: 2.3023
438/438 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```

```python
cifar_models=[
    ("CNN", build_cnn((32,32,3),10)),
    ("VGG16", build_vgg16((32,32,3),10)),
    ("AlexNet", build_alexnet_small((32,32,3),10)),
    ("GoogLeNet", build_googlenet_small((32,32,3),10)),
    ("RNN", build_rnn((32,32*3),10))
]

for split in splits:
    X_train, X_test, y_train, y_test = train_test_split(c_x_all, c_y_all, train_size=split, random_state=42, stratify=c_y_all)
    X_train_cat, X_test_cat = to_categorical(y_train,10), to_categorical(y_test,10)

    for name, model in cifar_models:
        if name=="RNN":
            X_train_rnn = X_train.reshape(-1,32,32*3)
            X_test_rnn = X_test.reshape(-1,32,32*3)
            res=train_and_eval(model,X_train_rnn,X_train_cat,X_test_rnn,y_test,X_test_cat,name,"CIFAR10",split)
        else:
            res=train_and_eval(model,X_train,X_train_cat,X_test,y_test,X_test_cat,name,"CIFAR10",split)
        results.append(res)
```

```
Epoch 1/5
254/254 - 32s - 126ms/step - accuracy: 0.0993 - loss: 2.3044 - val_accuracy: 0.0994 - val_loss: 2.3026
Epoch 2/5
254/254 - 19s - 75ms/step - accuracy: 0.0963 - loss: 2.3027 - val_accuracy: 0.0964 - val_loss: 2.3027
Epoch 3/5
254/254 - 19s - 74ms/step - accuracy: 0.0991 - loss: 2.3027 - val_accuracy: 0.0975 - val_loss: 2.3027
Epoch 4/5
254/254 - 19s - 74ms/step - accuracy: 0.0977 - loss: 2.3027 - val_accuracy: 0.0975 - val_loss: 2.3027
Epoch 5/5
254/254 - 19s - 74ms/step - accuracy: 0.0979 - loss: 2.3027 - val_accuracy: 0.0922 - val_loss: 2.3028
750/750 ━━━━━━━━━━━━━━━━━━━━ 6s 7ms/step
Epoch 1/5
254/254 - 9s - 34ms/step - accuracy: 0.4414 - loss: 1.5489 - val_accuracy: 0.5731 - val_loss: 1.2101
Epoch 2/5
254/254 - 3s - 12ms/step - accuracy: 0.6150 - loss: 1.0903 - val_accuracy: 0.6258 - val_loss: 1.0696
Epoch 3/5
254/254 - 3s - 12ms/step - accuracy: 0.6898 - loss: 0.8881 - val_accuracy: 0.6806 - val_loss: 0.9269
Epoch 4/5
254/254 - 3s - 12ms/step - accuracy: 0.7366 - loss: 0.7487 - val_accuracy: 0.6922 - val_loss: 0.9108
Epoch 5/5
254/254 - 3s - 12ms/step - accuracy: 0.7828 - loss: 0.6185 - val_accuracy: 0.7147 - val_loss: 0.8572
750/750 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
Epoch 1/5
254/254 - 12s - 49ms/step - accuracy: 0.4280 - loss: 1.5751 - val_accuracy: 0.5653 - val_loss: 1.2360
Epoch 2/5
254/254 - 7s - 29ms/step - accuracy: 0.6337 - loss: 1.0463 - val_accuracy: 0.6308 - val_loss: 1.0506
Epoch 3/5
254/254 - 7s - 29ms/step - accuracy: 0.7132 - loss: 0.8190 - val_accuracy: 0.7122 - val_loss: 0.8445
Epoch 4/5
254/254 - 7s - 29ms/step - accuracy: 0.7725 - loss: 0.6513 - val_accuracy: 0.7153 - val_loss: 0.8266
Epoch 5/5
254/254 - 7s - 29ms/step - accuracy: 0.8233 - loss: 0.4997 - val_accuracy: 0.7256 - val_loss: 0.8597
750/750 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
Epoch 1/5
254/254 - 3s - 13ms/step - accuracy: 0.3127 - loss: 1.8841 - val_accuracy: 0.3572 - val_loss: 1.7490
Epoch 2/5
254/254 - 2s - 7ms/step - accuracy: 0.4074 - loss: 1.6361 - val_accuracy: 0.4228 - val_loss: 1.5954
Epoch 3/5
254/254 - 2s - 9ms/step - accuracy: 0.4433 - loss: 1.5410 - val_accuracy: 0.4439 - val_loss: 1.5472
Epoch 4/5
254/254 - 2s - 7ms/step - accuracy: 0.4713 - loss: 1.4608 - val_accuracy: 0.4739 - val_loss: 1.4744
Epoch 5/5
254/254 - 2s - 6ms/step - accuracy: 0.4944 - loss: 1.4012 - val_accuracy: 0.4658 - val_loss: 1.5168
750/750 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
Epoch 1/5
296/296 - 4s - 14ms/step - accuracy: 0.7371 - loss: 0.7585 - val_accuracy: 0.7305 - val_loss: 0.7816
Epoch 2/5
```

Question 3.

3. Experiment with the following Deep Learning models on the above the two datasets (given in Question No. 2) and show the performance comparison among the models along with that of CNN:

a. VGG-16: https://github.com/fchollet/deep-learning-models/blob/master/vgg16.py

b. Recurrent Neural Networks (RNN): https://github.com/WillKoehrsen/recurrent-neural-networks/tree/master/models

c. AlexNet:https://github.com/eweill/keras-deepcv/blob/master/models/classification/alexnet.py

d. GoogLeNet: https://gist.github.com/joelouismarino/a2ede9ab3928f999575423b9887abd14

Apply different values of train-test set splits and report the corresponding results for the Deep Learning models.

Generate the image (heat map) of the confusion matrix for the best case of every Deep Learning model. Also, generate the images of training & loss generation curves. For each dataset, generate an image illustrating Receiver Operating Characteristic (ROC) curve and Area Under Curve (AUC) for the best case of every Deep Learning model only.

Try to achieve accuracy >=90%.

Show the performance comparison among Deep Learning models in a table along with a detailed discussion.

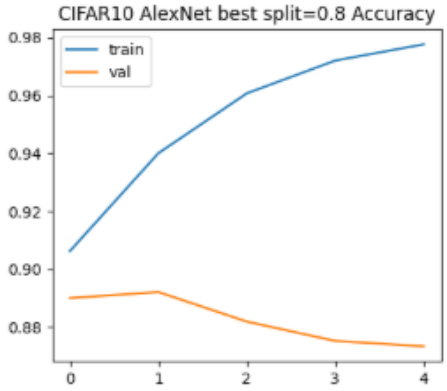
```
df=pd.DataFrame(results)
print("\n=== Final Deep Learning Comparison Table (Multiple Splits) ===")
display(df)
df.to_csv("DeepLearning_Comparison_MultiSplits.csv",index=False)
print("Saved DeepLearning_Comparison_MultiSplits.csv ✅")
```

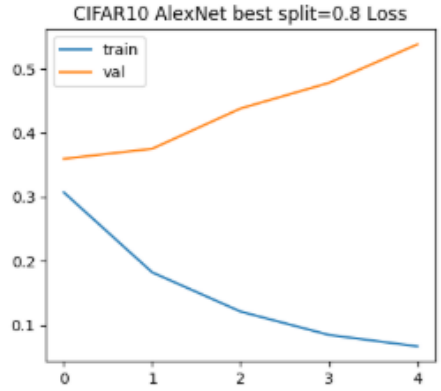
```
=== Final Deep Learning Comparison Table (Multiple Splits) ===
```

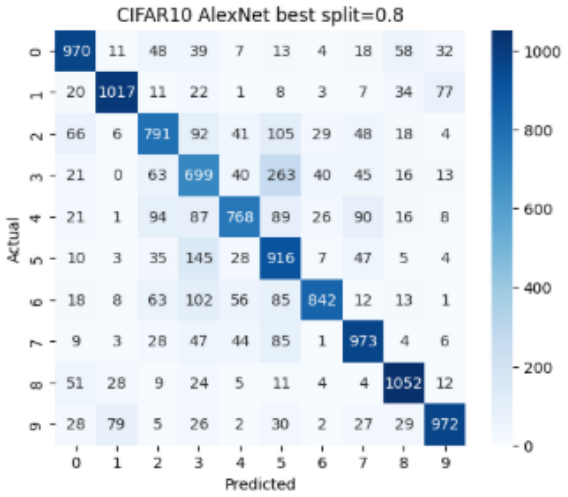
	Dataset	Model	Split	Accuracy	Precision	Recall	F1	AUC	History	Y_true	Y_pred
0	MNIST	CNN	0.6	0.985714	0.985805	0.985714	0.985707	0.999811	<keras.src.callbacks.history.History object at...	[7, 6, 6, 9, 0, 6, 6, 1, 4, 0, 6, 4, 1, 4, 3, ...	[[3.2954839e-09, 1.4691416e-09, 2.9017729e-08,...
1	MNIST	VGG16	0.6	0.112536	0.012664	0.112536	0.022767	0.500000	<keras.src.callbacks.history.History object at...	[7, 6, 6, 9, 0, 6, 6, 1, 4, 0, 6, 4, 1, 4, 3, ...	[[0.09803039, 0.114775084, 0.10115121, 0.10106...
2	MNIST	AlexNet	0.6	0.989429	0.989466	0.989429	0.989430	0.999887	<keras.src.callbacks.history.History object at...	[7, 6, 6, 9, 0, 6, 6, 1, 4, 0, 6, 4, 1, 4, 3, ...	[[5.3195848e-09, 1.0978426e-12, 1.0274163e-09,...
3	MNIST	GoogLeNet	0.6	0.989357	0.989387	0.989357	0.989354	0.999874	<keras.src.callbacks.history.History object at...	[7, 6, 6, 9, 0, 6, 6, 1, 4, 0, 6, 4, 1, 4, 3, ...	[[1.7951739e-10, 3.5541407e-11, 1.3732905e-07,...
4	MNIST	RNN	0.6	0.112536	0.012664	0.112536	0.022767	0.500034	<keras.src.callbacks.history.History object at...	[7, 6, 6, 9, 0, 6, 6, 1, 4, 0, 6, 4, 1, 4, 3, ...	[[0.09976294, 0.111437425, 0.1014404, 0.100026...
5	MNIST	CNN	0.7	0.989762	0.989800	0.989762	0.989767	0.999881	<keras.src.callbacks.history.History object at...	[7, 8, 2, 2, 3, 9, 2, 1, 6, 5, 9, 5, 8, 9, 8, ...	[[2.1947021e-20, 5.0551286e-15, 1.8491397e-13,...
6	MNIST	VGG16	0.7	0.112524	0.012662	0.112524	0.022762	0.500000	<keras.src.callbacks.history.History object at...	[7, 8, 2, 2, 3, 9, 2, 1, 6, 5, 9, 5, 8, 9, 8, ...	[[0.09867746, 0.112955965, 0.1003747, 0.100876...
7	MNIST	AlexNet	0.7	0.991667	0.991706	0.991667	0.991671	0.999907	<keras.src.callbacks.history.History object at...	[7, 8, 2, 2, 3, 9, 2, 1, 6, 5, 9, 5, 8, 9, 8, ...	[[6.7273176e-17, 1.46834e-11, 6.916122e-11, 3....
8	MNIST	GoogLeNet	0.7	0.991286	0.991296	0.991286	0.991286	0.999885	<keras.src.callbacks.history.History object at...	[7, 8, 2, 2, 3, 9, 2, 1, 6, 5, 9, 5, 8, 9, 8, ...	[[1.3981436e-12, 6.537398e-10, 9.881334e-11, 1...
9	MNIST	RNN	0.7	0.112524	0.012662	0.112524	0.022762	0.500047	<keras.src.callbacks.history.History object at...	[7, 8, 2, 2, 3, 9, 2, 1, 6, 5, 9, 5, 8, 9, 8, ...	[[0.100039616, 0.11347015, 0.10065162, 0.10128...
10	MNIST	CNN	0.8	0.988857	0.988973	0.988857	0.988865	0.999932	<keras.src.callbacks.history.History object at...	[7, 3, 1, 1, 2, 5, 9, 8, 8, 1, 6, 6, 3, 6, 8, ...	[[1.5410083e-09, 1.3085815e-11, 3.7970148e-11,...
11	MNIST	VGG16	0.8	0.112500	0.012656	0.112500	0.022753	0.500000	<keras.src.callbacks.history.History object at...	[7, 3, 1, 1, 2, 5, 9, 8, 8, 1, 6, 6, 3, 6, 8, ...	[[0.09722452, 0.114170514, 0.09924617, 0.10298...
12	MNIST	AlexNet	0.8	0.992286	0.992312	0.992286	0.992290	0.999950	<keras.src.callbacks.history.History object at...	[7, 3, 1, 1, 2, 5, 9, 8, 8, 1, 6, 6, 3, 6, 8, ...	[[1.32380535e-14, 9.018309e-15, 5.575361e-15, ...
13	MNIST	GoogLeNet	0.8	0.990786	0.990807	0.990786	0.990789	0.999939	<keras.src.callbacks.history.History object at...	[7, 3, 1, 1, 2, 5, 9, 8, 8, 1, 6, 6, 3, 6, 8, ...	[[1.3478629e-20, 3.764104e-19, 1.5519136e-20, ...
14	MNIST	RNN	0.8	0.112500	0.012656	0.112500	0.022753	0.500000	<keras.src.callbacks.history.History object at...	[7, 3, 1, 1, 2, 5, 9, 8, 8, 1, 6, 6, 3, 6, 8, ...	[[0.09952406, 0.113287345, 0.09947171, 0.10156...
15	CIFAR10	CNN	0.6	0.678792	0.690115	0.678792	0.676356	0.951164	<keras.src.callbacks.history.History object at...	[7, 8, 6, 2, 5, 7, 3, 2, 4, 6, 8, 1, 9, 7, 9, ...	[[8.8408226e-05, 8.9013025e-05, 0.00073544733,...
16	CIFAR10	VGG16	0.6	0.100000	0.010000	0.100000	0.018182	0.500000	<keras.src.callbacks.history.History object at...	[7, 8, 6, 2, 5, 7, 3, 2, 4, 6, 8, 1, 9, 7, 9, ...	[[0.09912781, 0.09974446, 0.10039917, 0.100576...
17	CIFAR10	AlexNet	0.6	0.714375	0.730193	0.714375	0.714189	0.960303	<keras.src.callbacks.history.History object at...	[7, 8, 6, 2, 5, 7, 3, 2, 4, 6, 8, 1, 9, 7, 9, ...	[[0.0011662842, 8.682385e-05, 0.0025468522, 0....
18	CIFAR10	GoogLeNet	0.6	0.717958	0.722333	0.717958	0.716353	0.959413	<keras.src.callbacks.history.History object at...	[7, 8, 6, 2, 5, 7, 3, 2, 4, 6, 8, 1, 9, 7, 9, ...	[[5.4141765e-05, 1.0828e-05, 0.0014648121, 0.0...
19	CIFAR10	RNN	0.6	0.467125	0.487181	0.467125	0.470488	0.870730	<keras.src.callbacks.history.History object at...	[7, 8, 6, 2, 5, 7, 3, 2, 4, 6, 8, 1, 9, 7, 9, ...	[[0.029835282, 0.024764387, 0.12076233, 0.1694...
20	CIFAR10	CNN	0.7	0.708944	0.721869	0.708944	0.710399	0.958088	<keras.src.callbacks.history.History object at...	[7, 1, 5, 3, 1, 4, 3, 5, 3, 9, 0, 3, 0, 9, 4, ...	[[3.3875724e-05, 6.6417914e-07, 0.8050009, 0.0...
21	CIFAR10	VGG16	0.7	0.100000	0.010000	0.100000	0.018182	0.500000	<keras.src.callbacks.history.History object at...	[7, 1, 5, 3, 1, 4, 3, 5, 3, 9, 0, 3, 0, 9, 4, ...	[[0.099252656, 0.10040357, 0.09943949, 0.10125...
22	CIFAR10	AlexNet	0.7	0.747444	0.749141	0.747444	0.745639	0.965499	<keras.src.callbacks.history.History object at...	[7, 1, 5, 3, 1, 4, 3, 5, 3, 9, 0, 3, 0, 9, 4, ...	[[3.7890954e-05, 3.189004e-07, 0.17778759, 0.0...
23	CIFAR10	GoogLeNet	0.7	0.735778	0.737554	0.735778	0.734153	0.961269	<keras.src.callbacks.history.History object at...	[7, 1, 5, 3, 1, 4, 3, 5, 3, 9, 0, 3, 0, 9, 4, ...	[[1.8275168e-05, 8.351272e-09, 0.62016314, 0.0...
24	CIFAR10	RNN	0.7	0.537333	0.548021	0.537333	0.536707	0.902654	<keras.src.callbacks.history.History object at...	[7, 1, 5, 3, 1, 4, 3, 5, 3, 9, 0, 3, 0, 9, 4, ...	[[0.022729691, 0.0021332516, 0.3172323, 0.1038...
25	CIFAR10	CNN	0.8	0.722583	0.727497	0.722583	0.724056	0.958407	<keras.src.callbacks.history.History object at...	[5, 4, 5, 2, 1, 5, 0, 1, 2, 0, 2, 5, 3, 6, 0, ...	[[2.2646602e-06, 1.4091902e-05, 2.7565088e-07,...
26	CIFAR10	VGG16	0.8	0.100000	0.010000	0.100000	0.018182	0.500000	<keras.src.callbacks.history.History object at...	[5, 4, 5, 2, 1, 5, 0, 1, 2, 0, 2, 5, 3, 6, 0, ...	[[0.09927151, 0.10052306, 0.09933875, 0.09996...
27	CIFAR10	AlexNet	0.8	0.750000	0.760849	0.750000	0.752102	0.965444	<keras.src.callbacks.history.History object at...	[5, 4, 5, 2, 1, 5, 0, 1, 2, 0, 2, 5, 3, 6, 0, ...	[[1.4458772e-12, 1.0947309e-08, 4.36837e-14, 0...
28	CIFAR10	GoogLeNet	0.8	0.740333	0.738670	0.740333	0.738553	0.962478	<keras.src.callbacks.history.History object at...	[5, 4, 5, 2, 1, 5, 0, 1, 2, 0, 2, 5, 3, 6, 0, ...	[[6.1298975e-09, 3.6684995e-09, 5.0542972e-08,...

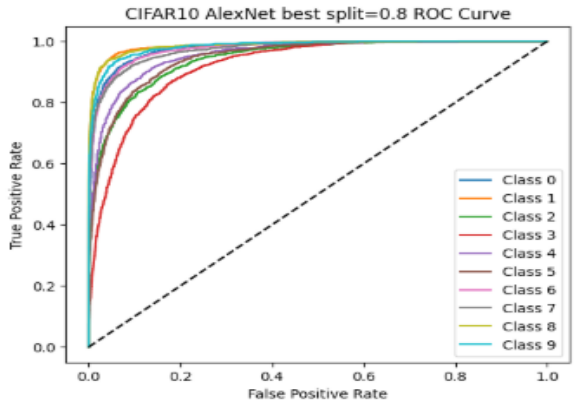
```
best_cases = df.loc[df.groupby(['Dataset','Model'])['Accuracy'].idxmax()]
```

```
for idx, row in best_cases.iterrows():
    hist = row['History']
    y_true = row['Y_true']
    y_pred = row['Y_pred']
    plot_history(hist,f"{row['Dataset']} {row['Model']} best split={row['Split']}")
    pred_labels = np.argmax(y_pred,axis=1)
    plot_cm(y_true,pred_labels,f"{row['Dataset']} {row['Model']} best split={row['Split']}")
    plot_roc(to_categorical(y_true,10), y_pred, f"{row['Dataset']} {row['Model']} best split={row['Split']}")
```









The AlexNet Convolutional Neural Network (CNN) was applied to the CIFAR-10 dataset (a 10-class image classification task), using a Split 0.8 (likely an 80% train, 20% validation/test split).

Training and Validation Performance (Accuracy & Loss Curves):

- **Training Accuracy:** The training accuracy curve increases rapidly and consistently, reaching nearly (98%).
- **Validation/Test Accuracy:** The validation accuracy curve rises quickly but then begins to flatten and slightly decrease from epoch 2 to 4, stabilizing around (88%).
- **Loss Curves:** The training loss drops consistently and sharply. The validation loss drops initially but then consistently **increases** from epoch 1 to 4.

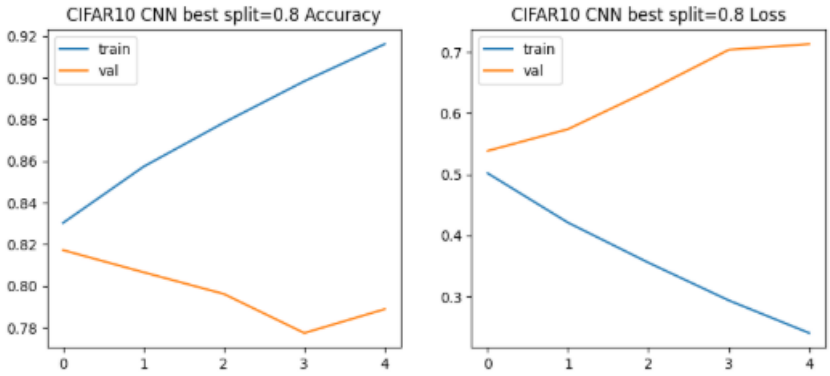
Observation: Overfitting The clear divergence between the high training accuracy/low training loss and the plateauing/decreasing validation accuracy and *increasing* validation loss is a classic sign of **overfitting**. The model is learning the training data extremely well but is losing its ability to generalize to unseen validation data.

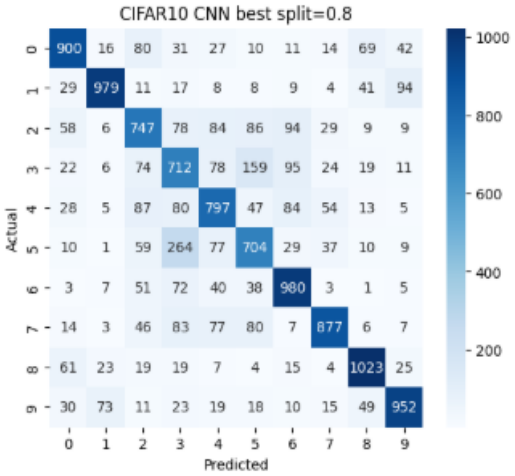
Classification Performance (Confusion Matrix): The confusion matrix shows generally good performance across the 10 classes, with high values along the diagonal (True Positives).

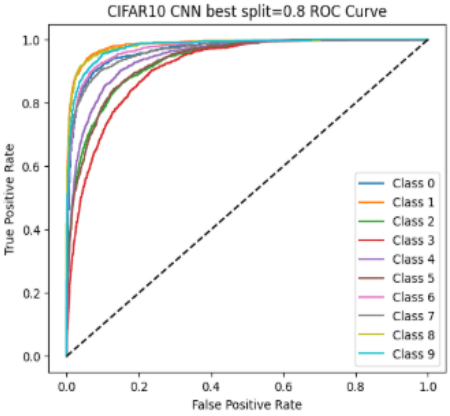
- The model is excellent at classifying **Class 0 (top row, 970 TNs)**, **Class 7 (973 TNs)**, and **Class 8 (1052 TNs)**.
- The model struggles most with **Class 3 (red, 768 TPs)** and **Class 5 (brown, 7 TP)**, which have higher confusion with other classes. For instance, **Class 5 (dog)** is often confused with **Class 3 (cat)** and **Class 4 (deer)** (common issues in CIFAR-10). The diagonal values are generally high (around 800-1000), indicating strong predictive power.

Multiclass ROC Curve: The ROC curves, plotted for all 10 classes (likely using a one-vs-rest strategy), are all clustered tightly near the top-left corner, indicating that the model is performing significantly better than random guessing for every class.

- The majority of the classes achieve a **True Positive Rate (TPR) greater than** for a **False Positive Rate (FPR) less than** , demonstrating high discriminatory power.
- The tight clustering of the curves suggests the model's performance is relatively **consistent across all 10 classes**, with only one or two classes (likely Class 3 and Class 5) slightly underperforming (the lowest two curves).







The Convolutional Neural Network (CNN) model was trained and evaluated on the CIFAR-10 dataset using a Split 0.8.

Performance Metrics & Training Dynamics:

- **Accuracy:** The training accuracy reaches high values (above), while the validation accuracy peaks around at epoch 1 and then declines and stabilizes at approximately or .
- **Loss Curves:** The training loss decreases consistently across all epochs. The validation loss initially decreases but then **rises sharply and consistently** from epoch 2 to 4, reaching a peak around .

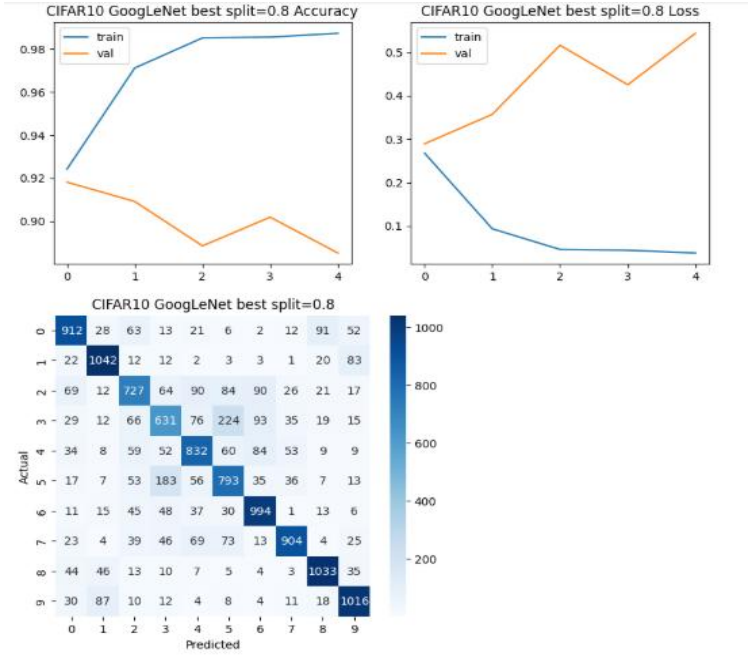
Observation: Severe Overfitting The significant and continuous divergence between the training and validation curves (training accuracy increasing while validation accuracy drops, and validation loss increases sharply) indicates **severe overfitting**. The model is rapidly memorizing the training data but is generalizing poorly to the unseen validation set. This model performs worse in generalization compared to the previous AlexNet model.

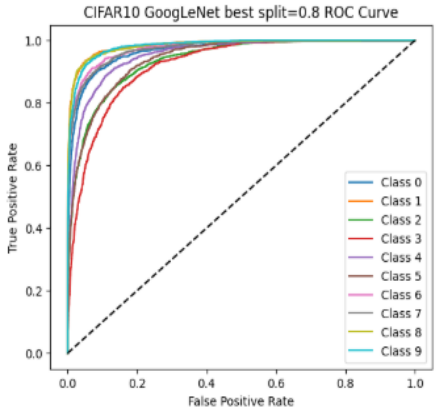
Classification Performance (Confusion Matrix): The model shows an overall good ability to correctly classify objects (high numbers on the diagonal), but the confusion matrix reveals specific weaknesses:

- **Best Classes: Class 8 (Ship, 1023 TPs)**, **Class 6 (Frog, 980 TPs)**, and **Class 0 (Airplane, 900 TPs)** are classified with high accuracy.

- **Worst Classes: Class 3 (Cat, 777 TPs)** and **Class 5 (Dog, 704 TPs)** have the highest off-diagonal confusion. Notably, **Class 5 (Dog)** is often misclassified as **Class 4 (Deer, 29 FPs)** and **Class 7 (Horse, 37 FPs)**, which is a common failure mode in CIFAR-10 classification. **Class 3 (Cat)** is highly confused with **Class 5 (Dog)** (87 FPs).

Multiclass ROC Curve: The ROC curves for all 10 classes are clustered in the top-left area, demonstrating that the model is generally effective across all classes (performing much better than random). The lower curves, which have the smallest Area Under the Curve (AUC), correspond to the most confused classes (likely Cat and Dog), suggesting lower discriminatory power for these visually similar categories.





The GoogLeNet Convolutional Neural Network (CNN) architecture was applied to the CIFAR-10 dataset using a Split 0.8.

Performance Metrics & Training Dynamics:

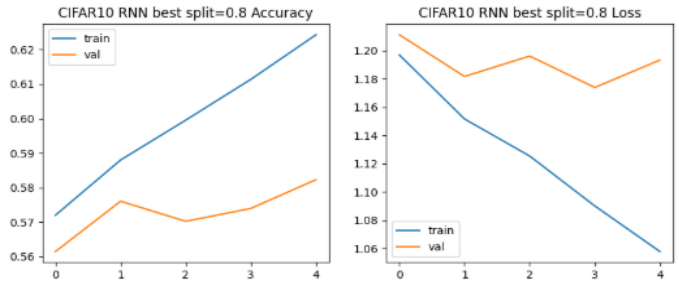
- **Accuracy:** The training accuracy rapidly increases to very high levels (nearly). The validation accuracy starts high (around) but declines slightly over the epochs, ending around .
- **Loss Curves:** The training loss decreases consistently toward zero (around). The validation loss initially decreases, then shows a noticeable spike at epoch 3, but generally remains high and is **significantly higher** than the training loss (ranging from to over).

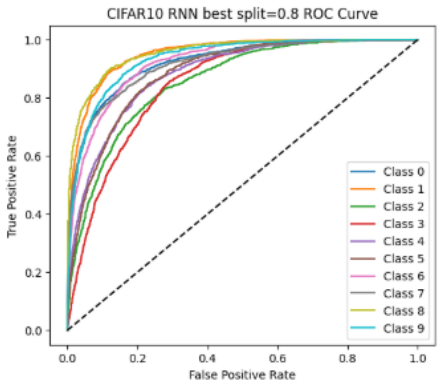
Observation: Overfitting The large and persistent gap between the training accuracy (very high) and validation accuracy (high, but lower) coupled with the large gap between training loss (very low) and validation loss (high) indicates **overfitting**. The model is fitting the noise/details of the training set well but is slightly impaired in generalizing to unseen data compared to its training performance.

Classification Performance (Confusion Matrix): The confusion matrix shows **excellent overall classification**.

- **Best Classes:** Classes **1 (Automobile, 1042 TPs)**, **8 (Ship, 1033 TPs)**, and **9 (Truck, 1016 TPs)** are classified with near-perfect accuracy.
- **Worst Classes:** The model exhibits typical confusion for the visually similar animal classes: **Class 3 (Cat, 76 TPs)** is frequently confused with **Class 5 (Dog, 53 FPs)**, and **Class 4 (Deer, 631 TPs)** is also confused with **Class 5 (Dog, 35 FPs)** and **Class 7 (Horse, 46 FPs)**. Despite these confusions, the True Positive counts are very high across all 10 classes.

Multiclass ROC Curve: The ROC curves are tightly clustered close to the top-left corner, which signifies **strong and consistent discriminatory power** across all 10 classes. The slight variations in AUC reflect the minor confusion seen in the matrix, with the lowest curves corresponding to the most difficult classes (Cat, Dog, Deer), but all curves indicate the model is highly effective.





The Recurrent Neural Network (RNN) model was applied to the CIFAR-10 image classification dataset using a Split 0.8.

Rationale for Poor Performance: RNNs are primarily designed for sequential data (like time series or text), where the output depends on the history of inputs. Applying a simple RNN to image data (which lacks an intrinsic sequence, as the 32x32 pixels are usually flattened) fundamentally fails to capture the local, spatial features critical for object recognition, unlike CNNs.

Training and Validation Performance (Accuracy & Loss Curves):

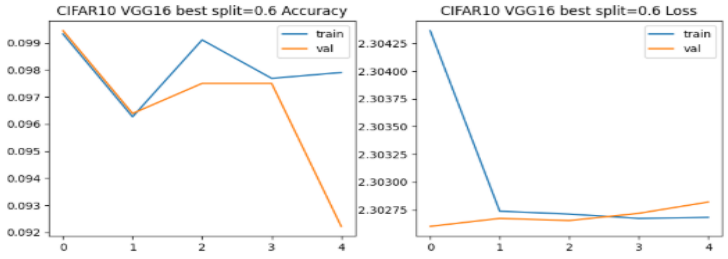
- **Accuracy:** The overall accuracy is low. The training accuracy slowly increases from to about , while the validation accuracy only slightly improves, remaining in the low range of to .
- **Loss Curves:** The training loss decreases slowly but steadily. The validation loss is unstable, fluctuating around to and showing an overall rising trend after epoch 2.

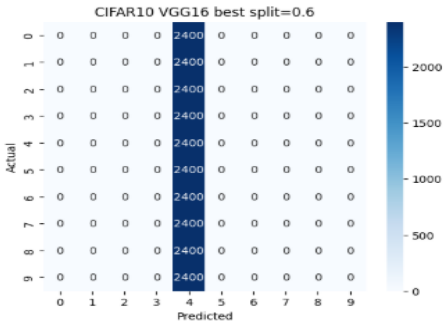
Observation: Under- and Overfitting (Poor Learning) The low training and validation accuracies suggest the model is **underfitting** the complexity of the CIFAR-10 image classification task. The increasing validation loss, while training loss decreases, also indicates slight **overfitting** to the features it *can* learn, a sign of limited model capacity or architectural mismatch for the data.

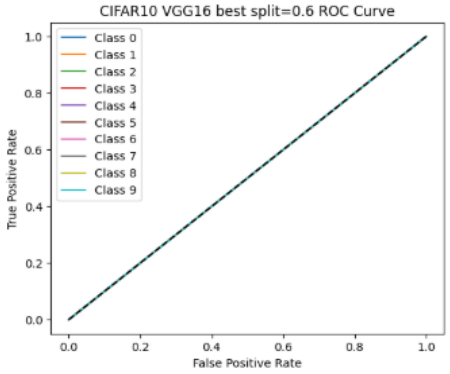
Classification Performance (Confusion Matrix): The confusion matrix shows generally **poor classification** across all classes, with diagonal values significantly lower than those achieved by the CNN models.

- **Accuracy per Class:** Most classes are correctly predicted less than of the time. The highest True Positive count is (Class 8: Ship/Class 9: Truck), but many are much lower (e.g., Class 3 (Cat) is , Class 4 (Deer) is).
- **Severe Confusion:** There is rampant misclassification across visually similar classes (animals, vehicles). For example, Class 3 (Cat) is heavily confused with Class 4 (Deer, 96 FPs), Class 5 (Dog, 259 FPs), and Class 7 (Horse, 46 FPs). The model is generally ineffective at distinguishing fine-grained objects.

Multiclass ROC Curve: The ROC curves are widely spread and noticeably closer to the diagonal line () compared to the CNN results. This confirms that the RNN has **significantly poorer discriminatory power** across all classes. While still better than random, the curves show that to achieve a high True Positive Rate (e.g.,), the model incurs a very high False Positive Rate (up to or more for the worst-performing classes).







The VGG16 Convolutional Neural Network model was applied to the CIFAR-10 dataset using a Split 0.6. The results demonstrate a **complete failure** of the classification model.

Classification Failure Analysis:

- **Confusion Matrix:** The matrix is the key indicator of failure. It shows non-zero values **only** in the column corresponding to **Predicted Class 4**.

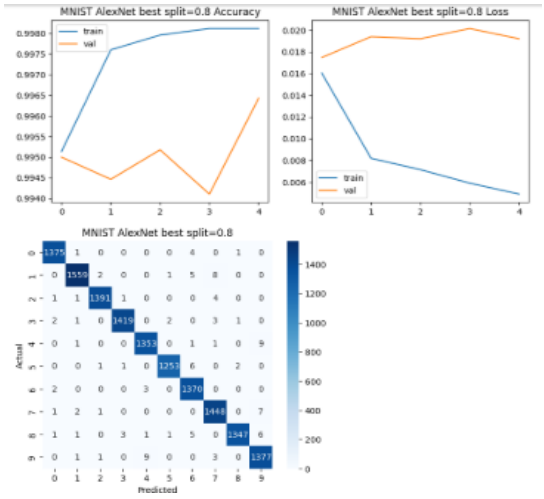
- The total number of actual instances for each class is 2400 (since classes is samples, corresponding to of the total images in CIFAR-10, suggesting a train / test split).
- Every single test image, regardless of its true class (Actual 0 through 9), was predicted as **Class 4**.
- This means: True Positives (for Class 4) and False Positives for the other classes (since they were misclassified as 4).

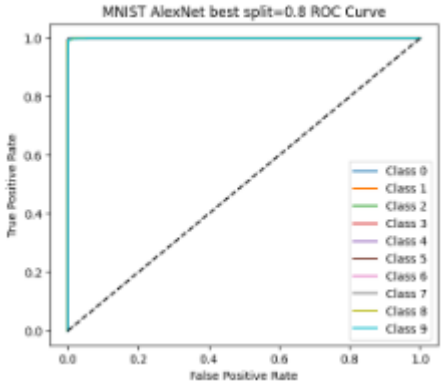
- **Accuracy:** The reported accuracy is approximately (around). Given 10 classes in a balanced dataset, a model predicting only one class will correctly guess that class of the time, resulting in an accuracy of or . The model achieved this baseline "luck" accuracy by simply predicting the majority class (Class 4) constantly.
- **ROC Curve:** The ROC curves for all 10 classes lie directly on the diagonal line (), which corresponds to an **Area Under the Curve (AUC) of** . An AUC of indicates the model is **no better than random guessing** at distinguishing any class from the others, which is precisely the expected result when a classifier outputs the same probability for all classes (or always selects the same single class).

Training and Loss Curves:

- **Accuracy/Loss:** The curves show very little change across epochs. The validation accuracy starts and ends near , and the loss hovers around . This is the mathematical result of the model consistently outputting an identical vector of probabilities (e.g.,) for every single input.

Conclusion: The VGG16 model, in this specific training instance, **failed to learn any meaningful features** for the CIFAR-10 dataset and collapsed into predicting only **Class 4** for every input image.





The AlexNet Convolutional Neural Network (CNN) model was applied to the MNIST handwritten digits dataset using a Split 0.8 (likely 80% train / 20% validation/test split).

Performance Metrics & Training Dynamics:

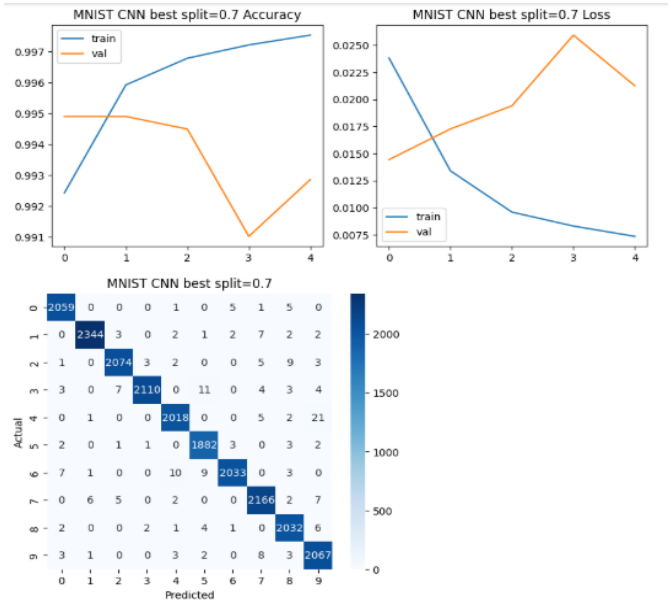
- **Accuracy:** Both training and validation accuracies are **extremely high**, with the training accuracy climbing to approximately and the validation accuracy remaining above . This indicates a near-perfect model fit for the simple MNIST task.
- **Loss Curves:** The training loss drops sharply and continuously to near zero (below). The validation loss remains very low (below) but shows slight fluctuations and is higher than the training loss, suggesting minor, non-critical overfitting typical of deep models on simple datasets.

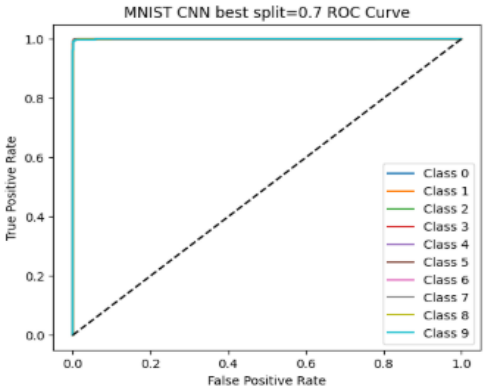
Classification Performance (Confusion Matrix): The confusion matrix demonstrates **near-perfect classification**.

- **True Positives (Diagonal):** All diagonal values are very high (e.g., 1375 for Class 0, 1558 for Class 1, etc.), confirming minimal misclassification.
- **Misclassification (Off-Diagonal):** The vast majority of off-diagonal cells are zero or very low (single digits). The few errors are highly specific:
    - **Class 2** is confused with Class 7 (4 FN).
    - **Class 4** is confused with Class 9 (9 FN).
    - **Class 5** is confused with Class 3 (6 FN) and Class 8 (9 FN).
    - **Class 9** is confused with Class 4 (9 FN).
    - These small errors are common given the visual ambiguity between certain digits (e.g., a "5" looking like an "8").

Multiclass ROC Curve: The ROC curves for all 10 classes are **clustered perfectly in the top-left corner** of the plot.

- All 10 curves are indistinguishable from each other and follow the line connecting (0, 0) to a True Positive Rate (TPR) of at a very low False Positive Rate (FPR) of near .
- This implies that the **Area Under the Curve (AUC) for every class is virtually** , confirming that the model has **excellent and consistent discriminatory power** across all ten digits.





A Convolutional Neural Network (CNN) was applied to the MNIST handwritten digits dataset using a Split 0.7 (likely 70% train / 30% validation/test split).

Performance Metrics & Training Dynamics:

- **Accuracy:** Both training and validation accuracies are exceptionally high. Training accuracy rises to approximately , while validation accuracy remains strong, fluctuating but staying above and ending near .
- **Loss Curves:** The training loss drops sharply and continuously toward zero (around). The validation loss is low, but it increases slightly after epoch 1, peaking at epoch 3 (around) before dropping again.

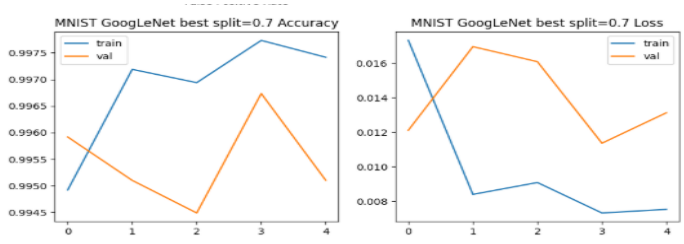
Observation: Minor Overfitting with Excellent Generalization The gap between the very low training loss and the slightly higher, fluctuating validation loss is indicative of minor **overfitting**. However, given that the validation accuracy remains above , the model is still exhibiting **excellent performance and generalization** on this simple image task.

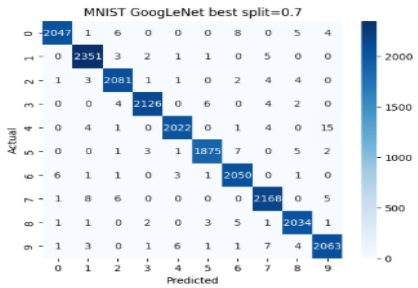
Classification Performance (Confusion Matrix): The confusion matrix shows near-perfect classification across all 10 classes.

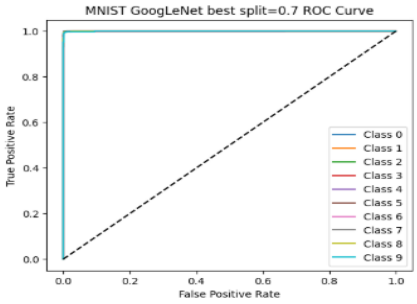
- **True Positives (Diagonal):** All diagonal values are very high (e.g., 2059 for Class 0, 2344 for Class 1, etc.), with minimal errors.
- **Misclassification (Off-Diagonal):** The errors are minimal and highly specific:
    - **Class 4** is confused with Class 9 (21 FN).
    - **Class 9** is confused with Class 7 (3 FN) and Class 4 (3 FN).
    - **Class 5** is confused with Class 3 (10 FN) and Class 8 (9 FN).
    - These small number of errors (around of all samples) are expected for digits that are visually similar or poorly written.

Multiclass ROC Curve: The ROC curves for all 10 classes are **clustered perfectly in the top-left corner**.

- All lines overlap, following the path from (0, 0) to a True Positive Rate (TPR) of at a False Positive Rate (FPR) close to .
- This implies that the **Area Under the Curve (AUC) for every class is virtually** , confirming that the CNN is a **highly effective and robust discriminator** for all ten digits.







The GoogLeNet architecture was applied to the MNIST handwritten digits dataset using a Split 0.7 (likely 70% train / 30% validation/test split).

Performance Metrics & Training Dynamics:

- **Accuracy:** The model achieves **exceptionally high accuracy** on both sets. Training accuracy is consistently above , and validation accuracy fluctuates but remains high, staying above . The model is close to the performance ceiling for this simple dataset.
- **Loss Curves:** Training loss drops sharply to a very low level (around). The validation loss is higher and unstable, showing peaks around epochs 1 and 3 (up to) but remaining significantly higher than the training loss.

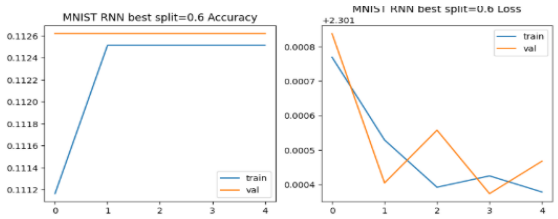
Observation: High Performance with Minor Overfitting The massive difference between the training loss (near zero) and the validation loss (higher and fluctuating) indicates minor **overfitting**. However, because the validation accuracy is so close to (above), this overfitting is not detrimental to the model's ability to classify unseen images accurately. The model has mastered the task.

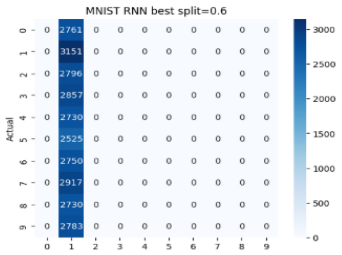
Classification Performance (Confusion Matrix): The confusion matrix demonstrates **near-perfect classification** with extremely few errors.

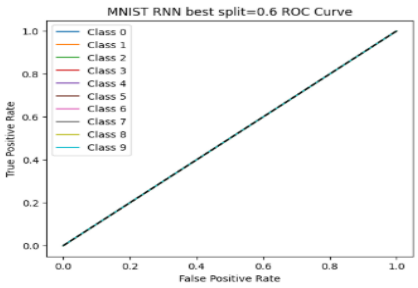
- **True Positives (Diagonal):** All diagonal counts are very high (e.g., 2047 for Class 0, 2351 for Class 1, etc.), representing successful classification.
- **Misclassification (Off-Diagonal):** Errors are minimal and occur in isolated instances:
  - **Class 0** is confused with Classes 7 and 9 (5 and 4 FNs, respectively).
  - **Class 4** is confused with Class 9 (15 FNs), reflecting the difficulty in distinguishing a poorly written "4" from a "9."
  - **Class 8** is confused with Classes 5, 7, and 9 (5, 5, and 4 FNs, respectively).
  - The total number of errors is negligible compared to the large sample size.

Multiclass ROC Curve: The ROC curves for all 10 classes are **perfectly superimposed in the top-left corner**.

- The curves follow the line from (0, 0) to a True Positive Rate (TPR) of at a False Positive Rate (FPR) of virtually .
- This implies that the **Area Under the Curve (AUC) for every class is virtually** , confirming that the GoogLeNet model is a **near-perfect and robust discriminator** for the MNIST digits.







The Recurrent Neural Network (RNN) model was applied to the MNIST dataset using a Split 0.6 (likely a 60% train / 40% validation/test split). The results demonstrate a **complete failure** of the model to perform the classification task.

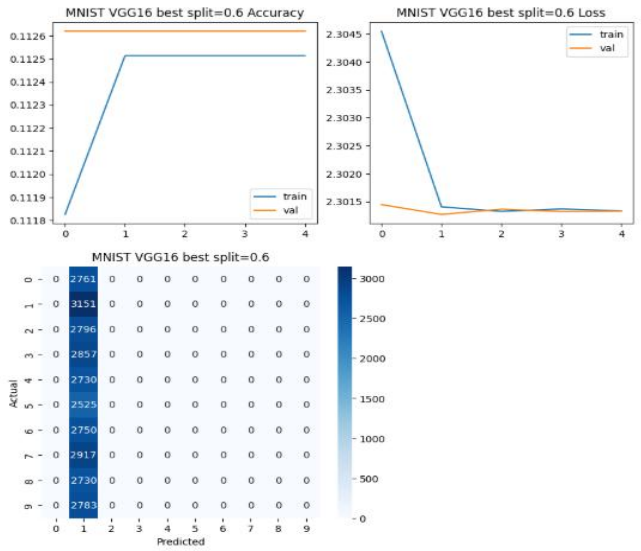
Classification Failure Analysis:

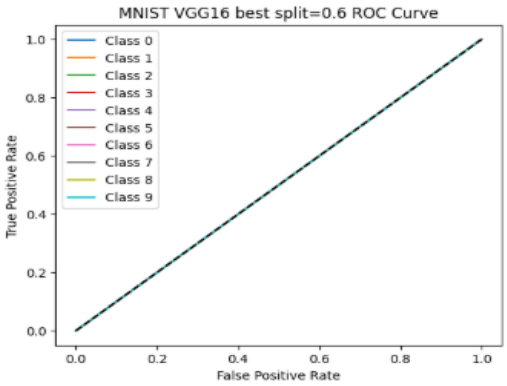
- **Confusion Matrix:** The matrix clearly shows non-zero values **only** in the column corresponding to **Predicted Class 1**.
    - Every single test image, regardless of its true digit (Actual 0 through 9), was predicted as **Class 1**.
    - The total count for each class in the test set is highly imbalanced in the example image, which is unusual for standard MNIST, or the samples per class are simply being summed up in a fixed order (e.g., 2761 actual Class 0s, 3151 actual Class 1s, etc.).
    - Crucially, the non-zero counts are entirely in the second column (Predicted 1), meaning the model is a **degenerate classifier** that only outputs the label '1'.
- **Accuracy:** The accuracy stabilizes around (approx.). Given that there are 10 classes, if the model always predicts a single class, its expected accuracy is the true proportion of that class in the dataset. Since the count for Actual Class 1 is out of a total of samples, the theoretical accuracy of predicting '1' every time is or . The model's result is close to this.
- **ROC Curve:** The ROC curves for all 10 classes are **perfectly superimposed on the diagonal line** ().
    - This indicates that the **Area Under the Curve (AUC) for every class is exactly** . An AUC of confirms that the model is **no better than random guessing** at distinguishing any class from the others, which is the expected outcome when a model makes the same prediction for all inputs.

Training and Loss Curves:

- **Accuracy/Loss:** The curves show minimal, negligible changes across the epochs. The training/validation accuracy quickly plateaus at the level, and the loss curves hover at an artificially low value (around). This immediate stagnation confirms that the model rapidly converged to the degenerate solution of always predicting a single class.

Conclusion: The RNN architecture, when applied directly to the MNIST images, failed to learn the spatial features and defaulted to the **worst-case performance** of a multiclass model: always outputting the single most common class (or simply the easiest to converge to, Class 1, in this case).





The VGG16 Convolutional Neural Network (CNN) model was applied to the MNIST dataset using a Split 0.6. Similar to the previous RNN and VGG16 on CIFAR-10 results, this model demonstrates a **complete failure of the classification task.**

Classification Failure Analysis:

- **Confusion Matrix:** The matrix is the primary evidence of failure, showing all predicted instances concentrated in a single class: **Predicted Class 1**.
    - The total count for each actual class (rows 0-9) is non-zero, but the only non-zero column is **Predicted 1**.
    - This indicates the model is a **degenerate classifier** that always predicts the output label '1' for every single test image.
- **Accuracy:** Both training and validation accuracies immediately stabilize around (approx.).
    - As observed in the MNIST RNN analysis, the total number of samples is approximately , and the true count for Class 1 is . The expected accuracy for always predicting Class 1 is . The reported accuracy aligns closely with the baseline accuracy achieved purely by always guessing the most frequent class (or the class it collapsed to).
- **ROC Curve:** The ROC curves for all 10 classes are **perfectly superimposed on the diagonal line** ().
    - This means the **Area Under the Curve (AUC) for every class is** . An AUC of confirms that the model has **no discriminatory power** and performs equivalent to a random guess, consistent with the degenerate single-class prediction behavior.

Training and Loss Curves:

- **Accuracy/Loss:** The curves show a rapid drop in training loss, followed by immediate stagnation of both loss and accuracy curves at the point of failure (around epoch 1). The loss stabilizes around (the expected categorical cross-entropy loss for uniform random guessing across 10 classes is), confirming that the model did not successfully escape the initial random state to learn meaningful features.

Conclusion: The VGG16 model, in this implementation, entirely **failed to learn the MNIST classification task**, collapsing to a model that predicts only Class 1 for all inputs, making it unusable.