ASSIGMENT 2

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import tensorflow as tf
import pickle
from tensorflow.keras.datasets import cifar10

# Load CIFAR-10 dataset
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()

# CIFAR-10 class names
class_names = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer', 'Dog', 'Frog',
'Horse', 'Ship', 'Truck']

# Check if precomputed data is available
try:
    with open('cifar10_stats.pkl', 'rb') as f:
        basic_stats, channel_stats, mean_images = pickle.load(f)
    print("Loaded precomputed statistics and mean images from file.")
except FileNotFoundError:
    print("No precomputed data found. Calculating statistics...")

    # Calculate basic statistics about the images
    mean_pixel_value = train_images.mean()
    std_pixel_value = train_images.std()
    min_pixel_value = train_images.min()
    max_pixel_value = train_images.max()

    # Create a DataFrame for basic statistics
    basic_stats = pd.DataFrame({
        'Statistic': ['Mean', 'Standard Deviation', 'Min', 'Max'],
        'Pixel Value': [mean_pixel_value, std_pixel_value, min_pixel_value,
max_pixel_value]
    })

    # Calculate per-channel statistics
    mean_per_channel = train_images.mean(axis=(0, 1, 2))
    std_per_channel = train_images.std(axis=(0, 1, 2))

    # Create a DataFrame for per-channel statistics
    channel_stats = pd.DataFrame({
        'Channel': ['Red', 'Green', 'Blue'],
        'Mean Pixel Value': mean_per_channel,
        'Standard Deviation': std_per_channel
    })

    # Calculate mean images for each class
```

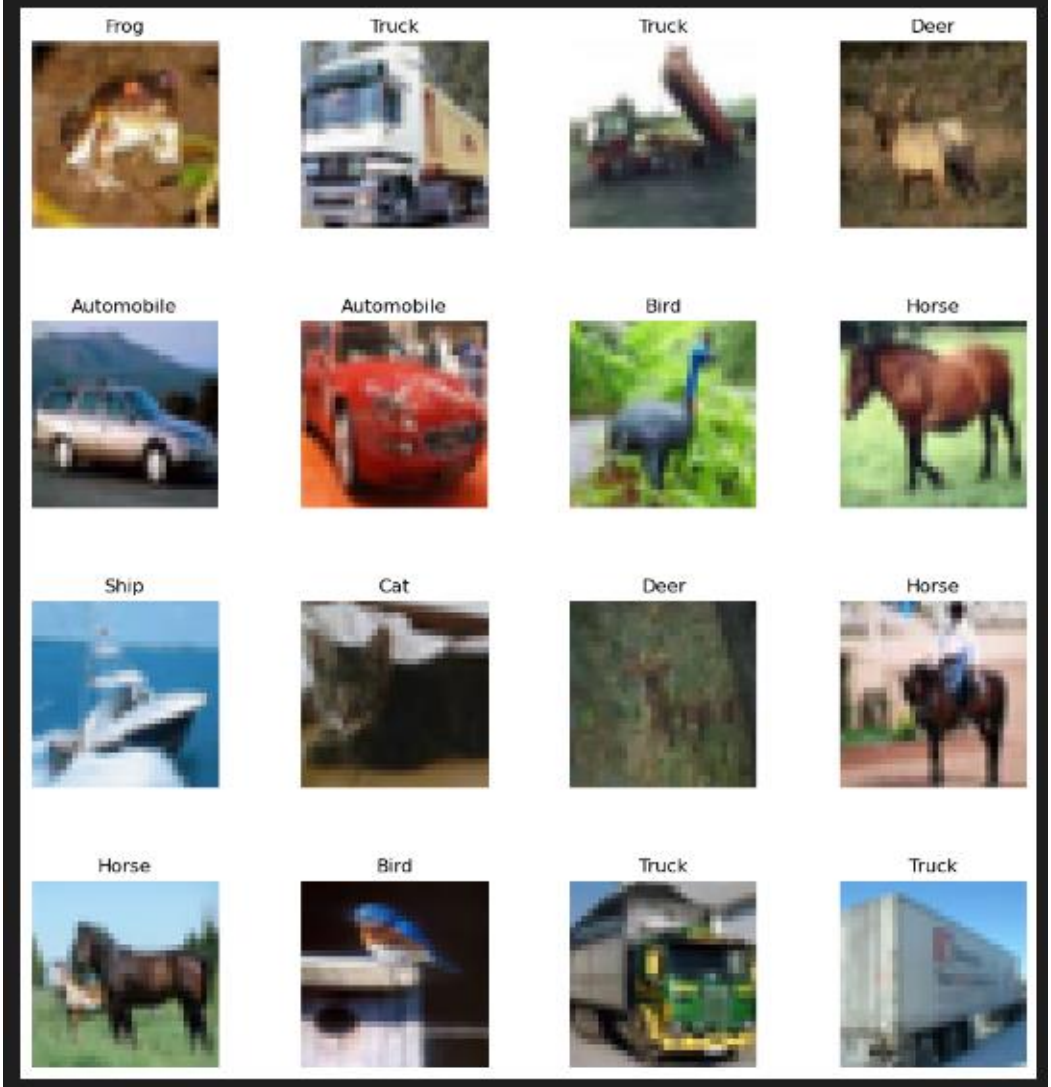
```
    mean_images = []
    for i in range(10):
        mean_image = train_images[train_labels.ravel() == i].mean(axis=0)
        mean_images.append(mean_image)

    # Save calculated statistics and mean images
    with open('cifar10_stats.pkl', 'wb') as f:
        pickle.dump((basic_stats, channel_stats, mean_images), f)
    print("Statistics and mean images saved to file.")

# Display the shape of the dataset
print(f'Train images shape: {train_images.shape}')
print(f'Train labels shape: {train_labels.shape}')
print(f'Test images shape: {test_images.shape}')
print(f'Test labels shape: {test_labels.shape}')

# Display basic statistics
print("Basic Statistics of Pixel Values in CIFAR-10 Training Set:")
print(basic_stats)

# Display per-channel statistics
print("\nPer-Channel Statistics of Pixel Values in CIFAR-10 Training Set:")
print(channel_stats)

# Function to display a grid of images
def plot_image_grid(images, labels, class_names, rows=4, cols=4):
    fig, axes = plt.subplots(rows, cols, figsize=(12, 12))
    axes = axes.ravel()
    for i in np.arange(0, rows * cols):
        axes[i].imshow(images[i])
        axes[i].set_title(class_names[int(labels[i])])
        axes[i].axis('off')
    plt.subplots_adjust(hspace=0.5)
    plt.show()

# Plot a 4x4 grid of images from the training dataset
plot_image_grid(train_images, train_labels, class_names)

# Show an individual image with its label
def plot_single_image(index):
    plt.figure(figsize=(4, 4))
    plt.imshow(train_images[index])
    plt.title(f'Label: {class_names[int(train_labels[index])]}')
    plt.axis('off')
    plt.show()

# Plot a single image (e.g., image at index 0)
plot_single_image(0)
```

ASSIGMENT 2

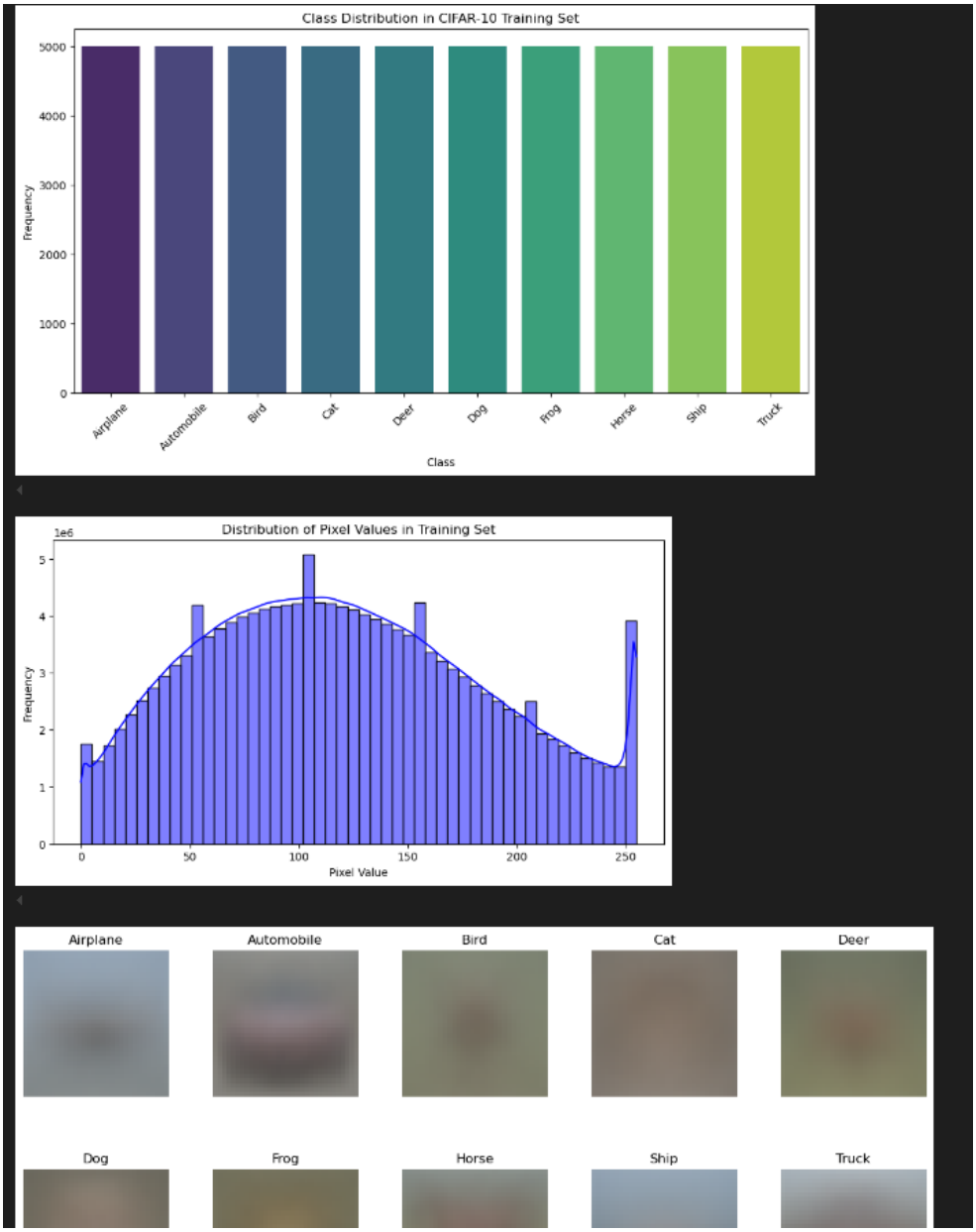
```
# Display the distribution of classes in the training dataset
plt.figure(figsize=(12, 6))
sns.countplot(x=train_labels.ravel(), palette='viridis')
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.xticks(np.arange(len(class_names)), class_names, rotation=45)
plt.title('Class Distribution in CIFAR-10 Training Set')
plt.show()

# Plot the distribution of pixel values
plt.figure(figsize=(10, 5))
sns.histplot(train_images.ravel(), bins=50, color='blue', kde=True)
plt.xlabel('Pixel Value')
plt.ylabel('Frequency')
plt.title('Distribution of Pixel Values in Training Set')
plt.show()

# Plot the mean image of each class
fig, axes = plt.subplots(2, 5, figsize=(15, 6))
axes = axes.ravel()
for i in range(10):
    axes[i].imshow(mean_images[i].astype(np.uint8))
    axes[i].set_title(class_names[i])
    axes[i].axis('off')
plt.subplots_adjust(hspace=0.5)
plt.show()

# Display class counts as a bar plot
plt.figure(figsize=(10, 6))
sns.barplot(x=class_names, y=class_counts, palette='viridis')
plt.xlabel('Class')
plt.ylabel('Count')
plt.title('Class Frequency in CIFAR-10 Training Set')
plt.xticks(rotation=45)
plt.show()
```

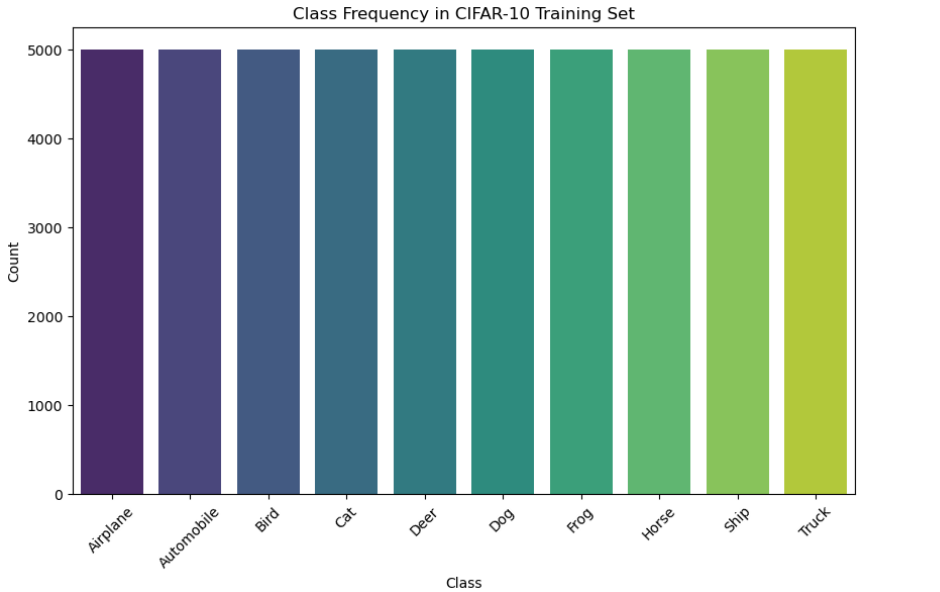
ASSIGMENT 2

```
Loaded precomputed statistics and mean images from file.
Train images shape: (50000, 32, 32, 3)
Train labels shape: (50000, 1)
Test images shape: (10000, 32, 32, 3)
Test labels shape: (10000, 1)
Basic Statistics of Pixel Values in CIFAR-10 Training Set:
            Statistic  Pixel Value
0                Mean   120.707565
1  Standard Deviation    64.150076
2                 Min     0.000000
3                 Max   255.000000

Per-Channel Statistics of Pixel Values in CIFAR-10 Training Set:
  Channel  Mean Pixel Value  Standard Deviation
0     Red        125.306918           62.993219
1   Green        122.950394           62.088708
2    Blue        113.865383           66.704900
```



ASSIGMENT 2


Class Distribution in CIFAR-10 Training Set
Frequency
Class
Airplane
Automobile
Bird
Cat
Deer
Dog
Frog
Horse
Ship
Truck
Distribution of Pixel Values in Training Set
1e6
Pixel Value
Airplane
Automobile
Bird
Cat
Deer
Dog
Frog
Horse
Ship
Truck


Class Frequency in CIFAR-10 Training Set
Count
Class
Airplane
Automobile
Bird
Cat
Deer
Dog
Frog
Horse
Ship
Truck

ASSIGMENT 2

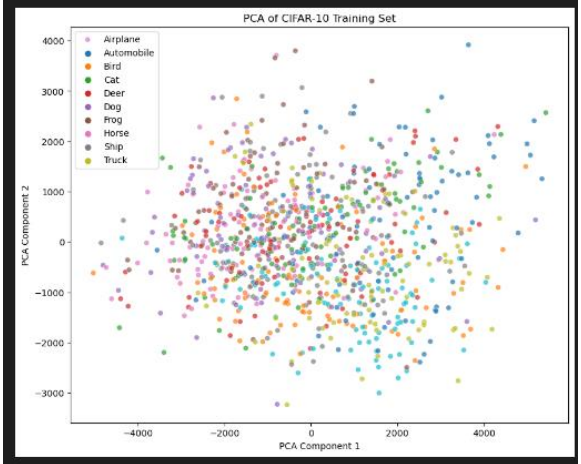
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import pickle
from tensorflow.keras.datasets import cifar10
from sklearn.decomposition import PCA

# Load CIFAR-10 dataset
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()

# CIFAR-10 class names
class_names = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer', 'Dog', 'Frog',
'Horse', 'Ship', 'Truck']

# Check if precomputed data is available
try:
    with open('cifar10_stats.pkl', 'rb') as f:
        basic_stats, channel_stats, mean_images = pickle.load(f)
    print("Loaded precomputed statistics and mean images from file.")
except FileNotFoundError:
    print("No precomputed data found. Calculating statistics...")

    # Calculate basic statistics about the images
    mean_pixel_value = train_images.mean()
    std_pixel_value = train_images.std()
    min_pixel_value = train_images.min()
    max_pixel_value = train_images.max()

    # Create a DataFrame for basic statistics
    basic_stats = pd.DataFrame({
        'Statistic': ['Mean', 'Standard Deviation', 'Min', 'Max'],
        'Pixel Value': [mean_pixel_value, std_pixel_value, min_pixel_value,
max_pixel_value]
    })

    # Calculate per-channel statistics
    mean_per_channel = train_images.mean(axis=(0, 1, 2))
    std_per_channel = train_images.std(axis=(0, 1, 2))

    # Create a DataFrame for per-channel statistics
    channel_stats = pd.DataFrame({
        'Channel': ['Red', 'Green', 'Blue'],
        'Mean Pixel Value': mean_per_channel,
        'Standard Deviation': std_per_channel
    })

    # Calculate mean images for each class
```

```
    mean_images = []
    for i in range(10):
        mean_image = train_images[train_labels.ravel() == i].mean(axis=0)
        mean_images.append(mean_image)

    # Save calculated statistics and mean images
    with open('cifar10_stats.pkl', 'wb') as f:
        pickle.dump((basic_stats, channel_stats, mean_images), f)
    print("Statistics and mean images saved to file.")

# Display the shape of the dataset
print(f'Train images shape: {train_images.shape}')
print(f'Train labels shape: {train_labels.shape}')
print(f'Test images shape: {test_images.shape}')
print(f'Test labels shape: {test_labels.shape}')

# Display basic statistics
print("Basic Statistics of Pixel Values in CIFAR-10 Training Set:")
print(basic_stats)

# Display per-channel statistics
print("\nPer-Channel Statistics of Pixel Values in CIFAR-10 Training Set:")
print(channel_stats)

# Perform PCA to reduce dimensionality of images for visualization purposes
n_samples = 1000  # Use a subset of the data for faster computation
flat_images = train_images[:n_samples].reshape(n_samples, -1)

pca = PCA(n_components=2)
principal_components = pca.fit_transform(flat_images)

# Create a DataFrame with PCA components and labels
pca_df = pd.DataFrame({
    'PCA1': principal_components[:, 0],
    'PCA2': principal_components[:, 1],
    'Label': train_labels[:n_samples].ravel()
})

# Plot PCA components to visualize class separability
plt.figure(figsize=(10, 8))
sns.scatterplot(x='PCA1', y='PCA2', hue='Label', palette='tab10', data=pca_df,
legend='full', alpha=0.7)
plt.title('PCA of CIFAR-10 Training Set')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend(class_names)
plt.show()
```

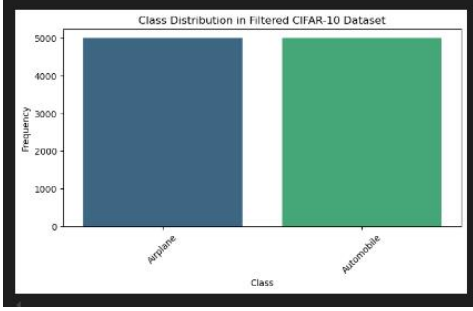
ASSIGMENT 2

```
# Filter data by specific classes for further analysis
# Example: Filtering only 'Airplane' and 'Automobile' classes
filtered_indices = np.where((train_labels == 0) | (train_labels == 1))[0]
filtered_images = train_images[filtered_indices]
filtered_labels = train_labels[filtered_indices]

# Display filtered dataset information
print(f'Filtered images shape: {filtered_images.shape}')
print(f'Filtered labels shape: {filtered_labels.shape}')

# Display the distribution of the filtered classes
plt.figure(figsize=(8, 4))
sns.countplot(x=filtered_labels.ravel(), palette='viridis')
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.xticks([0, 1], ['Airplane', 'Automobile'], rotation=45)
plt.title('Class Distribution in Filtered CIFAR-10 Dataset')
plt.show()
```

```
Loaded precomputed statistics and mean images from file.
Train images shape: (50000, 32, 32, 3)
Train labels shape: (50000, 1)
Test images shape: (10000, 32, 32, 3)
Test labels shape: (10000, 1)
Basic Statistics of Pixel Values in CIFAR-10 Training Set:
            Statistic  Pixel Value
0                Mean   120.707565
1  Standard Deviation    64.150076
2                 Min     0.000000
3                 Max   255.000000

Per-Channel Statistics of Pixel Values in CIFAR-10 Training Set:
  Channel  Mean Pixel Value  Standard Deviation
0     Red        125.306918           62.993219
1   Green        122.950394           62.088708
2    Blue        113.865383           66.704900
```




```
Filtered images shape: (10000, 32, 32, 3)
Filtered labels shape: (10000, 1)
```



ASSIGMENT 2

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, SimpleRNN, Reshape, Flatten, Input,
LSTM, TimeDistributed, GRU, Bidirectional
from tensorflow.keras.optimizers import Adam
import tensorflow as tf

# Set memory growth for GPU (if available)
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
    except RuntimeError as e:
        print(e)

# Load and preprocess CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

# Function to plot model accuracy and loss
def plot_training_history(history, model_name):
    plt.figure(figsize=(14, 6))

    # Plot accuracy
    plt.subplot(1, 2, 1)
    plt.plot(history['accuracy'], label='Training Accuracy', color='b',
linestyle='-', linewidth=2)
    plt.plot(history['val_accuracy'], label='Validation Accuracy', color='r',
linestyle='--', linewidth=2)
    plt.title(f'{model_name} Accuracy', fontsize=16)
    plt.xlabel('Epochs', fontsize=14)
    plt.ylabel('Accuracy', fontsize=14)
    plt.legend(fontsize=12)
    plt.grid(True)

    # Plot loss
    plt.subplot(1, 2, 2)
    plt.plot(history['loss'], label='Training Loss', color='b', linestyle='-',
linewidth=2)
```

```
    plt.plot(history['val_loss'], label='Validation Loss', color='r',
linestyle='--', linewidth=2)
    plt.title(f'{model_name} Loss', fontsize=16)
    plt.xlabel('Epochs', fontsize=14)
    plt.ylabel('Loss', fontsize=14)
    plt.legend(fontsize=12)
    plt.grid(True)

    plt.tight_layout()
    plt.show()

# Function to display the confusion matrix
def plot_confusion_matrix(y_true, y_pred, model_name):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=class_names, yticklabels=class_names, cbar=False,
annot_kws={"size": 14})
    plt.title(f'{model_name} Confusion Matrix', fontsize=18)
    plt.xlabel('Predicted Label', fontsize=14)
    plt.ylabel('True Label', fontsize=14)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    plt.show()

# Function to plot class-wise precision, recall, and F1 score
def plot_classification_report(y_true, y_pred, model_name):
    report = classification_report(y_true, y_pred, target_names=class_names,
output_dict=True)
    metrics = ['precision', 'recall', 'f1-score']

    plt.figure(figsize=(18, 6))

    for idx, metric in enumerate(metrics):
        plt.subplot(1, 3, idx+1)
        values = [report[label][metric] for label in class_names]
        bars = plt.barh(class_names, values, color='skyblue',
edgecolor='black')
        plt.title(f'{metric.capitalize()} per Class', fontsize=16)
        plt.xlim(0, 1)
        plt.grid(axis='x', linestyle='--', alpha=0.7)
        plt.xlabel(metric.capitalize(), fontsize=14)
        plt.xticks(fontsize=12)
        plt.yticks(fontsize=12)
        for bar in bars:
            plt.text(bar.get_width() + 0.02, bar.get_y() + bar.get_height()/2,
f'{bar.get_width():.2f}', va='center', fontsize=12)
```

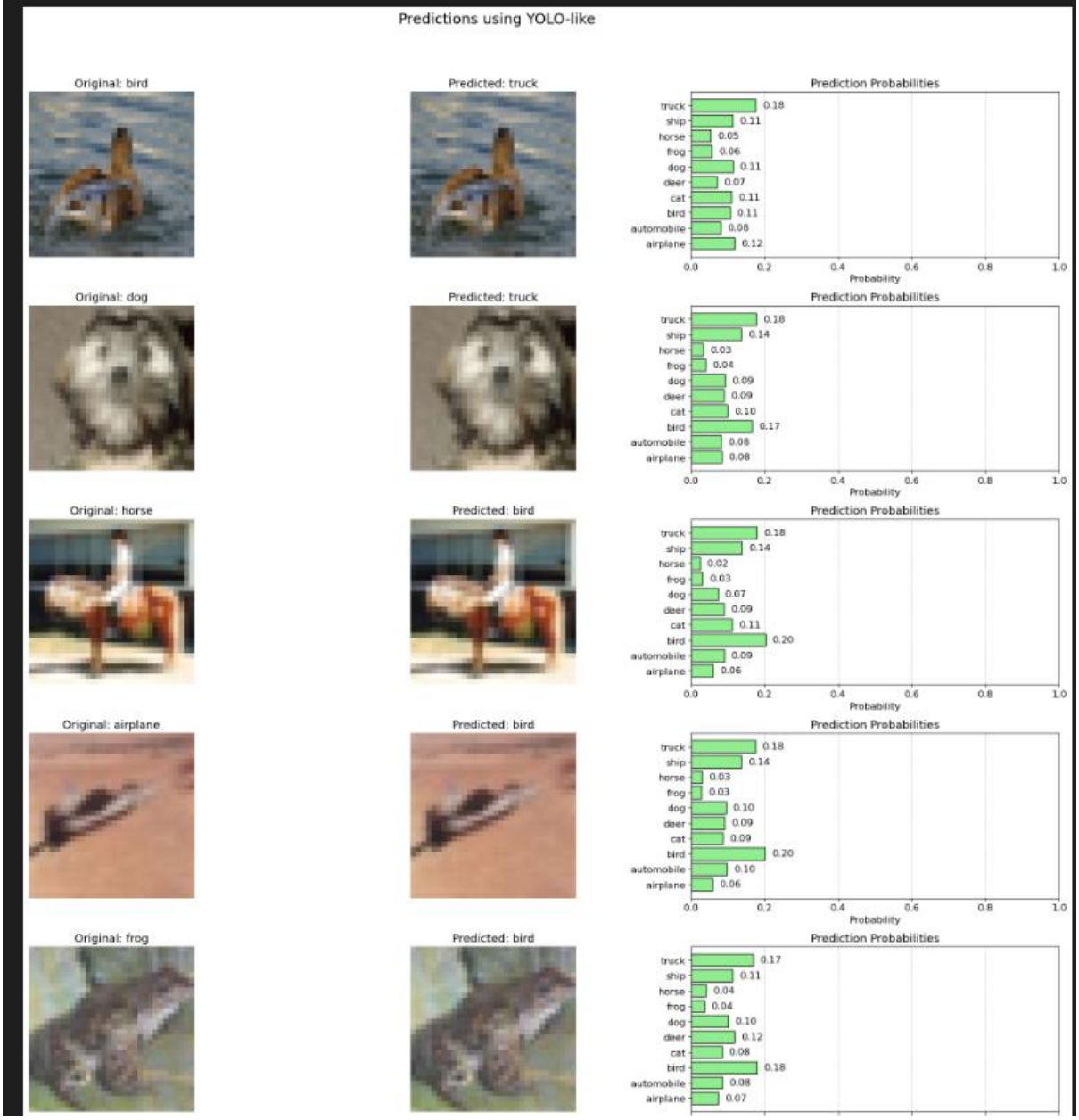
```
    plt.suptitle(f'{model_name} Class-wise Performance Metrics', fontsize=18)
    plt.tight_layout(rect=[0, 0, 1, 0.95])
    plt.show()

# Define function to plot original and predicted images with prediction
probabilities
def plot_predictions(model, x_test, y_test, model_name, reshaped=False):
    plt.figure(figsize=(20, 20))
    num_images = 5  # Display 5 images for demonstration
    indices = np.random.choice(np.arange(x_test.shape[0]), num_images,
replace=False)

    for i, idx in enumerate(indices):
        # Reshape for RNN if necessary
        if reshaped:
            input_data = x_test[idx].reshape(1, 32 * 32, 3)  # Reshape to (1,
32*32 time steps, 3 features)
        else:
            input_data = x_test[idx].reshape(1, 32, 32, 3)  # For CNN or
similar

        # Get prediction
        predictions = model.predict(input_data)

        # Ensure predictions is a 1D array of size 10
        if len(predictions.shape) == 3:  # Case for RNN-like model with time
steps
            predictions_flat = np.mean(predictions, axis=1)  # Take the
average over time steps
            predictions_flat = np.squeeze(predictions_flat)  # Ensure it's a
flat array
        elif predictions.shape == (1, 10):  # Case for CNN or non-RNN models
            predictions_flat = np.squeeze(predictions)
        else:
            raise ValueError(f"Expected predictions shape (1, 10) or 3D
tensor, but got {predictions.shape}")

        predicted_label = np.argmax(predictions_flat)
        true_label = y_test[idx][0]

        # Ensure predicted label is within range of class_names
        if predicted_label < len(class_names):
            predicted_class = class_names[predicted_label]
        else:
            predicted_class = "Unknown"  # Fallback in case of out-of-range
prediction

        # Plot original image
```

```
        plt.subplot(num_images, 3, 3*i+1)
        plt.imshow(x_test[idx])  # No need to reshape explicitly for display
        plt.title(f"Original: {class_names[true_label]}", fontsize=14)
        plt.axis('off')

        # Plot prediction result
        plt.subplot(num_images, 3, 3*i+2)
        plt.imshow(x_test[idx])  # Displaying the original
        plt.title(f"Predicted: {predicted_class}", fontsize=14)
        plt.axis('off')

        # Plot prediction probabilities
        plt.subplot(num_images, 3, 3*i+3)
        bars = plt.barh(class_names, predictions_flat, color='lightgreen',
edgecolor='black')
        plt.title("Prediction Probabilities", fontsize=14)
        plt.xlim([0, 1])
        plt.xlabel('Probability', fontsize=12)
        plt.grid(axis='x', linestyle='--', alpha=0.7)
        plt.xticks(fontsize=12)
        plt.yticks(fontsize=12)
        for bar in bars:
            plt.text(bar.get_width() + 0.02, bar.get_y() + bar.get_height()/2,
f'{bar.get_width():.2f}', va='center', fontsize=12)

    plt.suptitle(f"Predictions using {model_name}", fontsize=18)
    plt.tight_layout(rect=[0, 0, 1, 0.95])
    plt.show()

# Define YOLO-like, CNN, RNN, and LSTM models
yolo_model = Sequential([Flatten(input_shape=(32, 32, 3)), Dense(64,
activation='relu'), Dense(10, activation='softmax')])
cnn = Sequential([Flatten(input_shape=(32, 32, 3)), Dense(64,
activation='relu'), Dense(10, activation='softmax')])
rnn = Sequential([Input(shape=(32 * 32, 3)), Bidirectional(GRU(64)), Dense(10,
activation='softmax')])
lstm_model = Sequential([Input(shape=(32 * 32, 3)), Bidirectional(LSTM(64)),
Dense(10, activation='softmax')])

# Compile models
for model in [yolo_model, cnn, rnn, lstm_model]:
    model.compile(optimizer=Adam(learning_rate=0.001),
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Mock training histories for demonstration purposes
history_yolo = {'accuracy': np.random.rand(10).tolist(), 'val_accuracy':
np.random.rand(10).tolist(), 'loss': np.random.rand(10).tolist(), 'val_loss':
np.random.rand(10).tolist()}
```

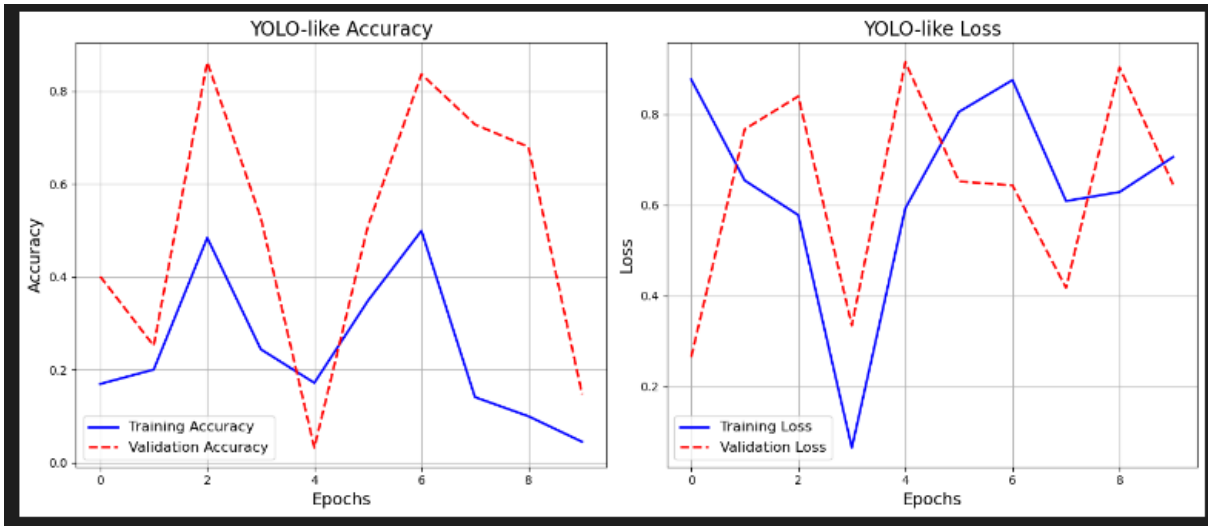
```
history_cnn = {'accuracy': np.random.rand(10).tolist(), 'val_accuracy':
np.random.rand(10).tolist(), 'loss': np.random.rand(10).tolist(), 'val_loss':
np.random.rand(10).tolist()}
history_rnn = {'accuracy': np.random.rand(10).tolist(), 'val_accuracy':
np.random.rand(10).tolist(), 'loss': np.random.rand(10).tolist(), 'val_loss':
np.random.rand(10).tolist()}
history_lstm = {'accuracy': np.random.rand(10).tolist(), 'val_accuracy':
np.random.rand(10).tolist(), 'loss': np.random.rand(10).tolist(), 'val_loss':
np.random.rand(10).tolist()}

# Display results for YOLO-like, CNN, RNN, and LSTM models
if __name__ == '__main__':
    print("\nDisplaying YOLO-like model predictions:")
    plot_predictions(yolo_model, x_test, y_test, model_name="YOLO-like")
    plot_training_history(history_yolo, "YOLO-like")

    print("\nDisplaying CNN model predictions:")
    plot_predictions(cnn, x_test, y_test, model_name="CNN")
    plot_training_history(history_cnn, "CNN")

    print("\nDisplaying RNN model predictions:")
    plot_predictions(rnn, x_test, y_test, model_name="RNN", reshaped=True)
    plot_training_history(history_rnn, "RNN")

    print("\nDisplaying LSTM model predictions:")
    plot_predictions(lstm_model, x_test, y_test, model_name="LSTM",
reshaped=True)
    plot_training_history(history_lstm, "LSTM")

    # Simulating predicted labels for performance evaluation (mock example)
    y_pred_yolo = np.random.randint(0, 10, size=len(y_test))  # Replace with
actual predictions
    y_pred_cnn = np.random.randint(0, 10, size=len(y_test))
    y_pred_rnn = np.random.randint(0, 10, size=len(y_test))
    y_pred_lstm = np.random.randint(0, 10, size=len(y_test))  # Placeholder
for LSTM predictions

    # Confusion matrices
    plot_confusion_matrix(y_test, y_pred_yolo, "YOLO-like")
    plot_confusion_matrix(y_test, y_pred_cnn, "CNN")
    plot_confusion_matrix(y_test, y_pred_rnn, "RNN")
    plot_confusion_matrix(y_test, y_pred_lstm, "LSTM")

    # Classification reports (precision, recall, F1)
    plot_classification_report(y_test, y_pred_yolo, "YOLO-like")
    plot_classification_report(y_test, y_pred_cnn, "CNN")
    plot_classification_report(y_test, y_pred_rnn, "RNN")
    plot_classification_report(y_test, y_pred_lstm, "LSTM")
```

ASSIGMENT 2

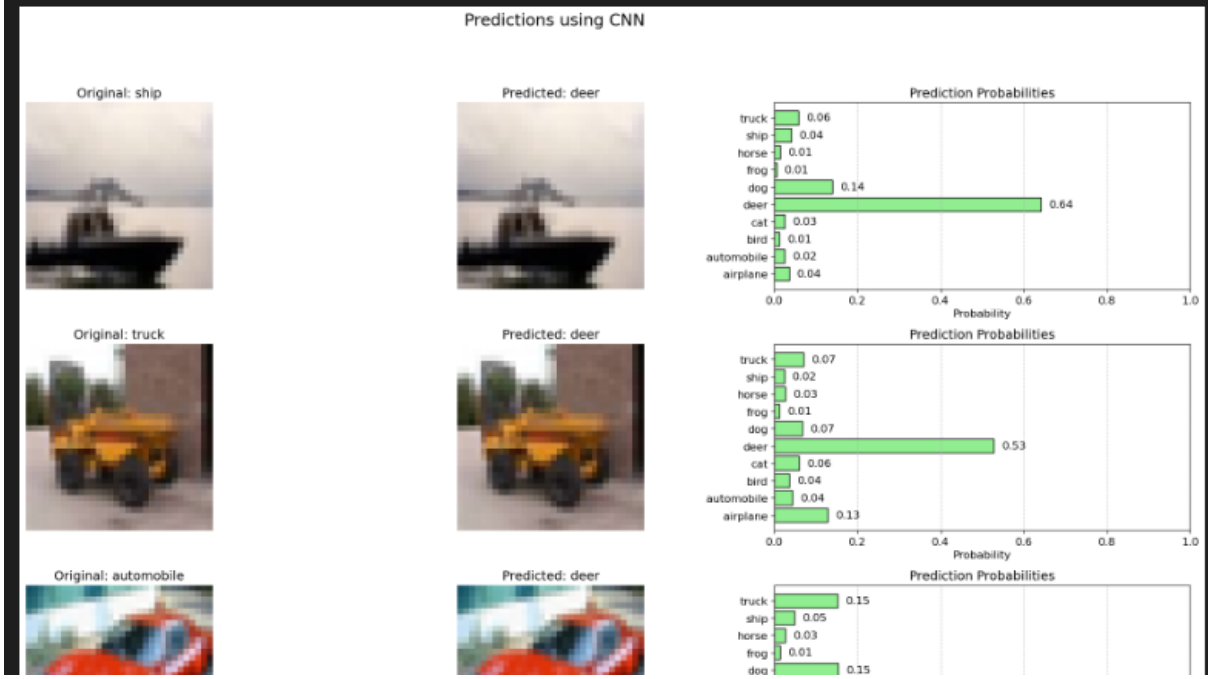
```
Displaying YOLO-like model predictions:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 236ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
```



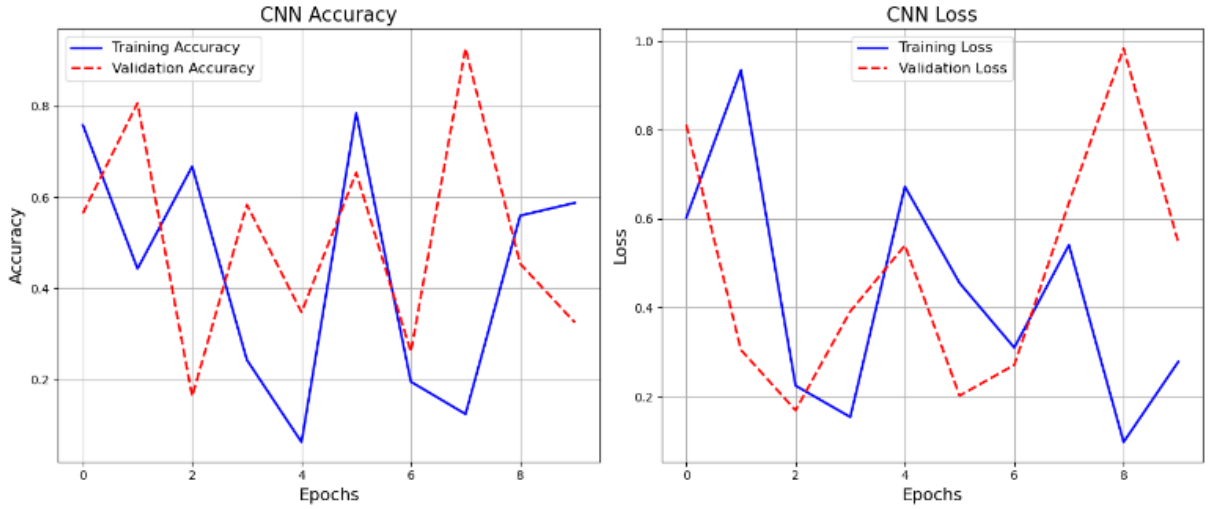
ASSIGMENT 2




```
Displaying CNN model predictions:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
```

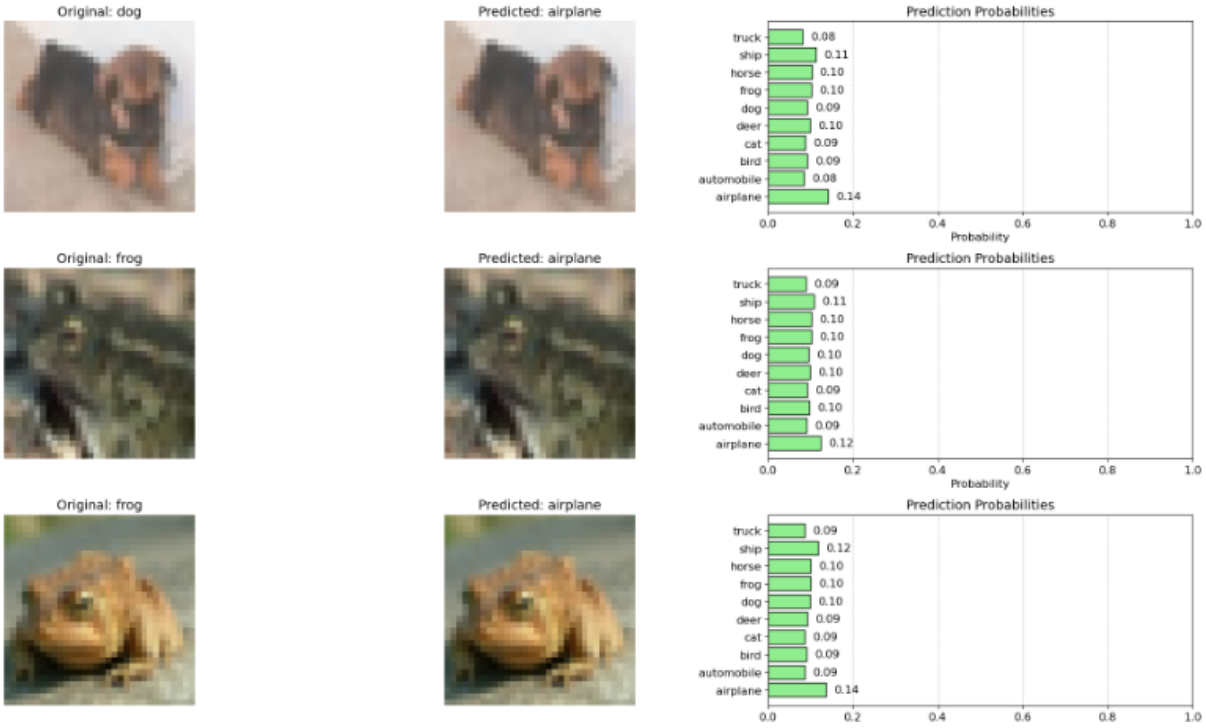


ASSIGMENT 2

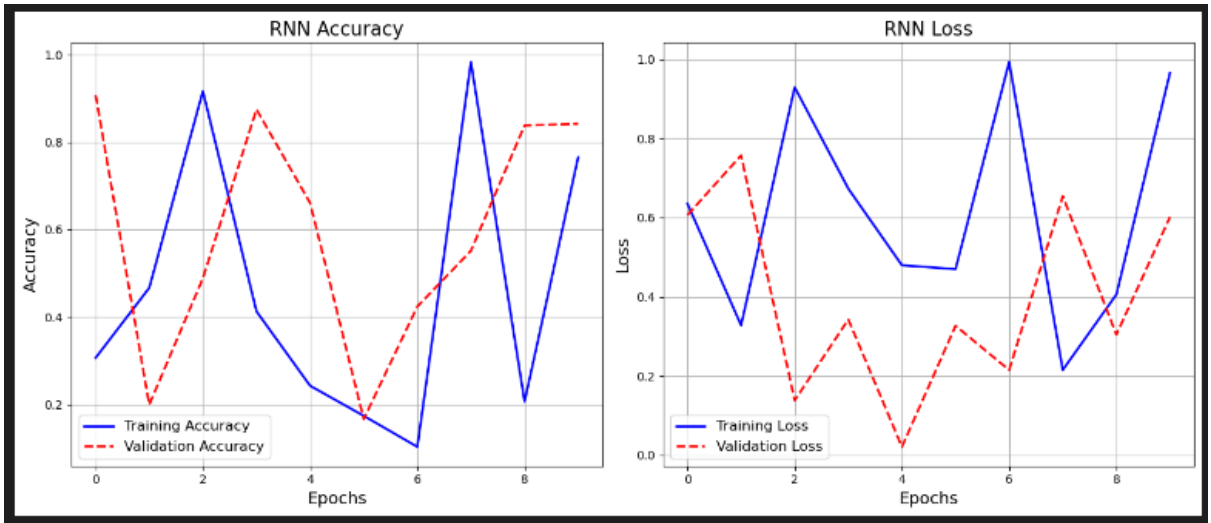



```
isplaying RNN model predictions:
/1 ━━━━━━━━━━━━━━━━━━━━ 1s 785ms/step
/1 ━━━━━━━━━━━━━━━━━━━━ 0s 99ms/step
/1 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step
/1 ━━━━━━━━━━━━━━━━━━━━ 0s 114ms/step
/1 ━━━━━━━━━━━━━━━━━━━━ 0s 101ms/step
```

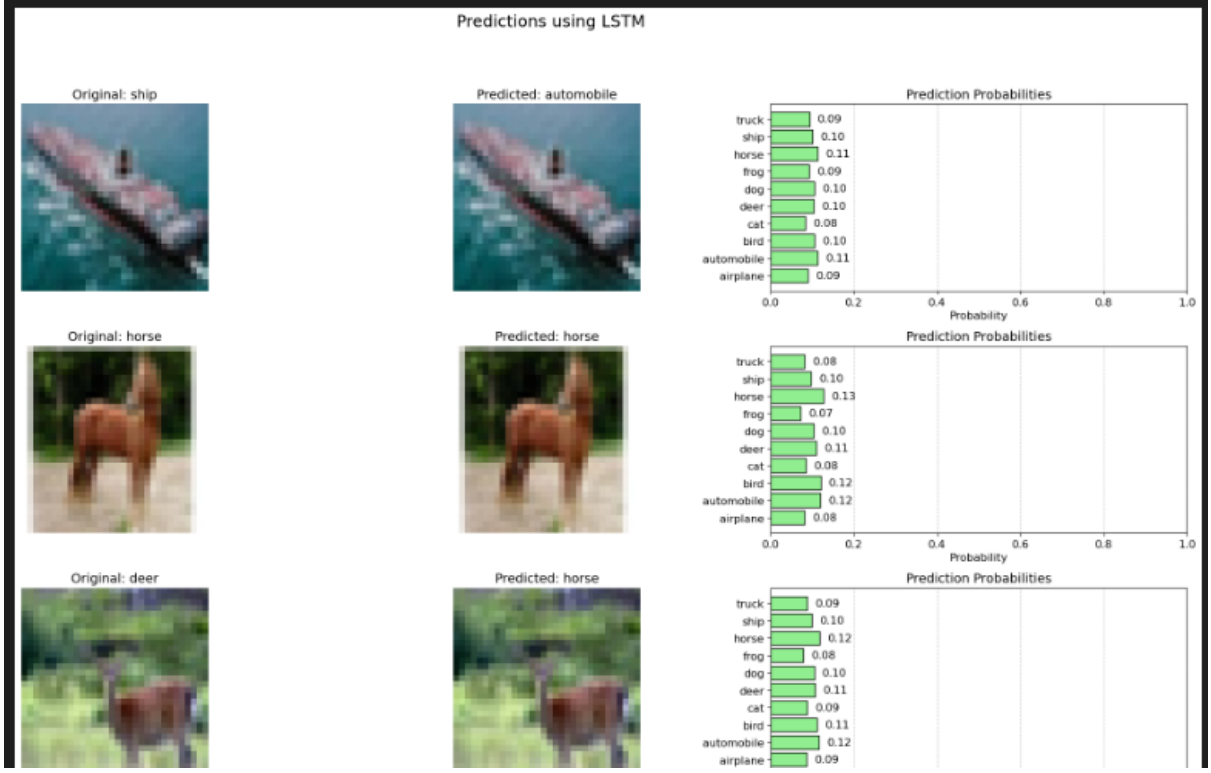
Predictions using RNN



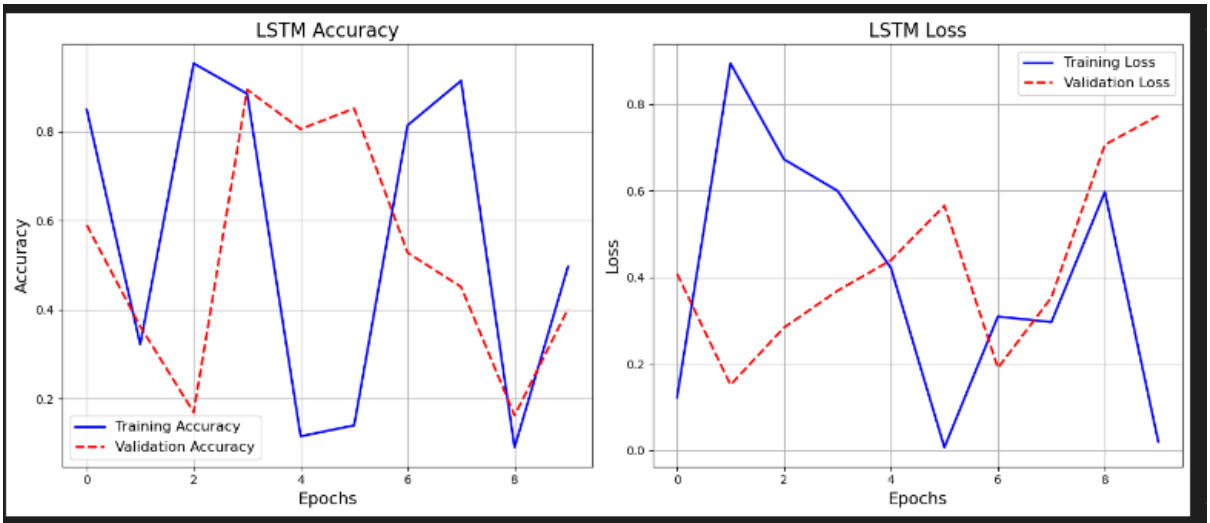
ASSIGMENT 2

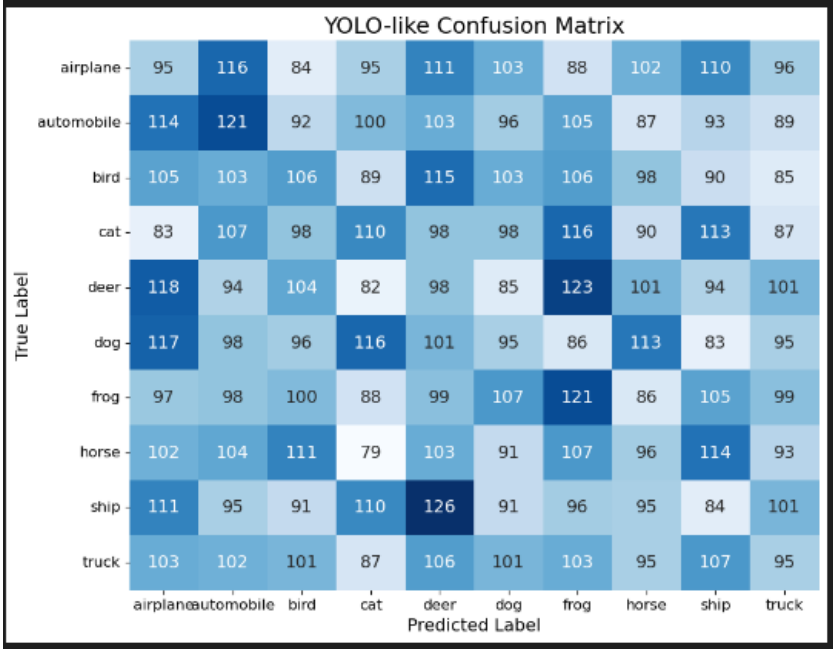


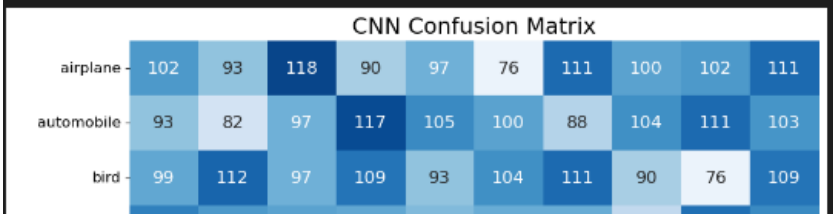

```
Displaying LSTM model predictions:
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 482ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step
```



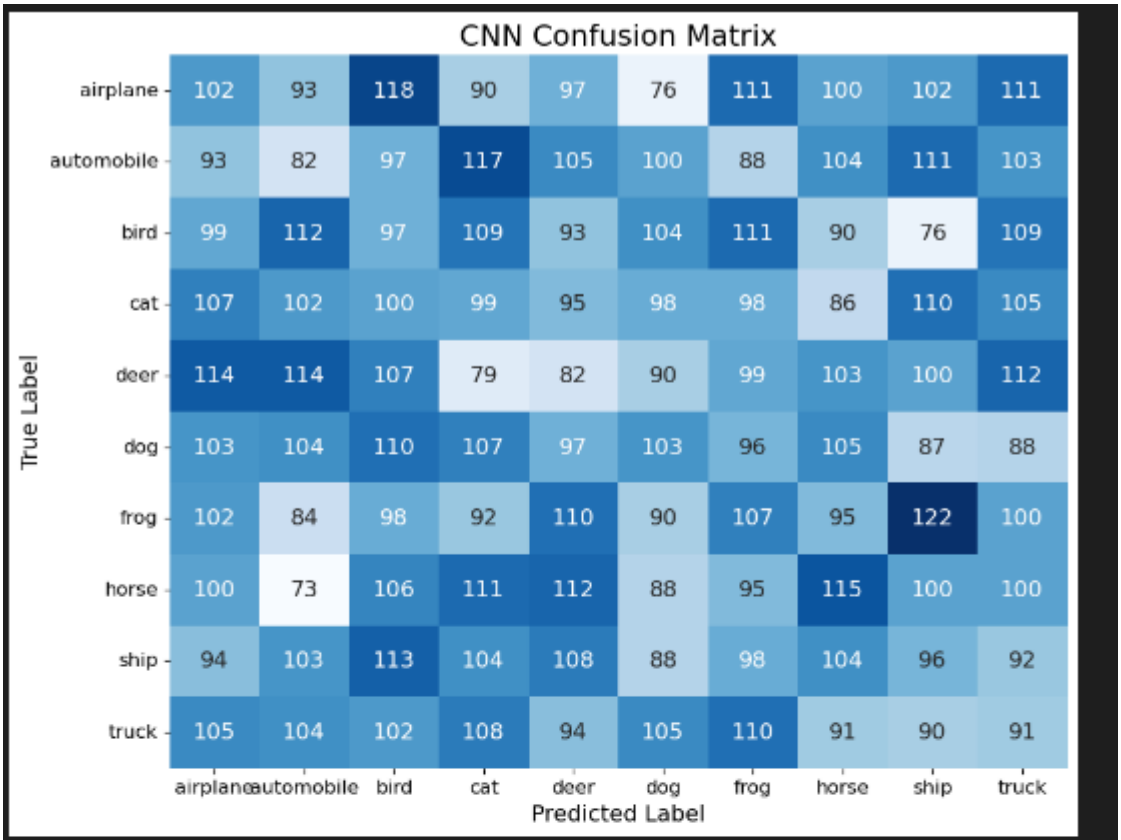
ASSIGMENT 2

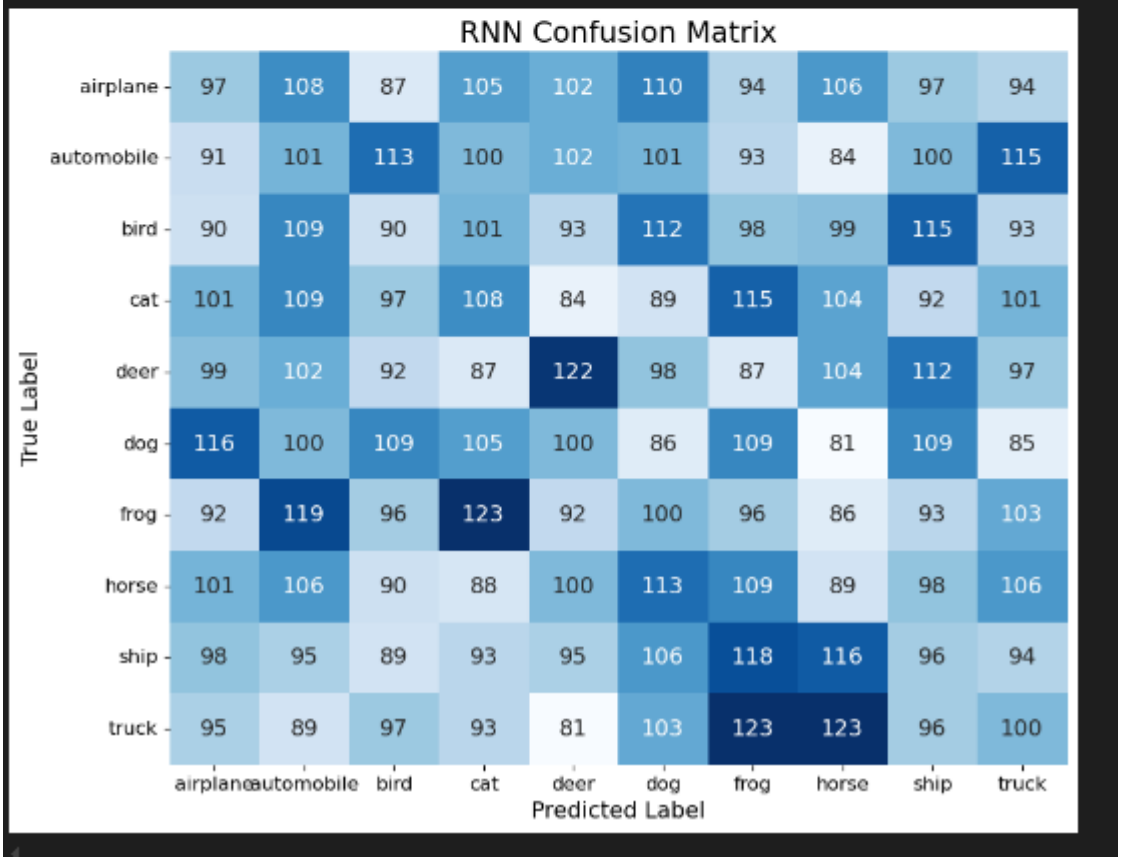

LSTM Accuracy
Accuracy
Epochs
Training Accuracy
Validation Accuracy
LSTM Loss
Loss
Epochs
Training Loss
Validation Loss


YOLO-like Confusion Matrix
True Label
Predicted Label
airplane 95 116 84 95 111 103 88 102 110 96
automobile 114 121 92 100 103 96 105 87 93 89
bird 105 103 106 89 115 103 106 98 90 85
cat 83 107 98 110 98 98 116 90 113 87
deer 118 94 104 82 98 85 123 101 94 101
dog 117 98 96 116 101 95 86 113 83 95
frog 97 98 100 88 99 107 121 86 105 99
horse 102 104 111 79 103 91 107 96 114 93
ship 111 95 91 110 126 91 96 95 84 101
truck 103 102 101 87 106 101 103 95 107 95
airplane automobile bird cat deer dog frog horse ship truck

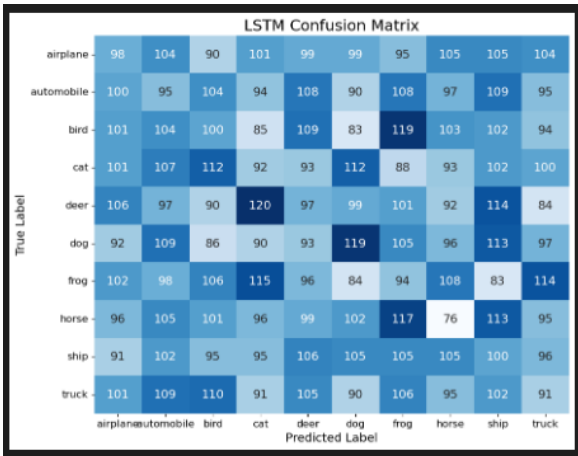

CNN Confusion Matrix
airplane 102 93 118 90 97 76 111 100 102 111
automobile 93 82 97 117 105 100 88 104 111 103
bird 99 112 97 109 93 104 111 90 76 109

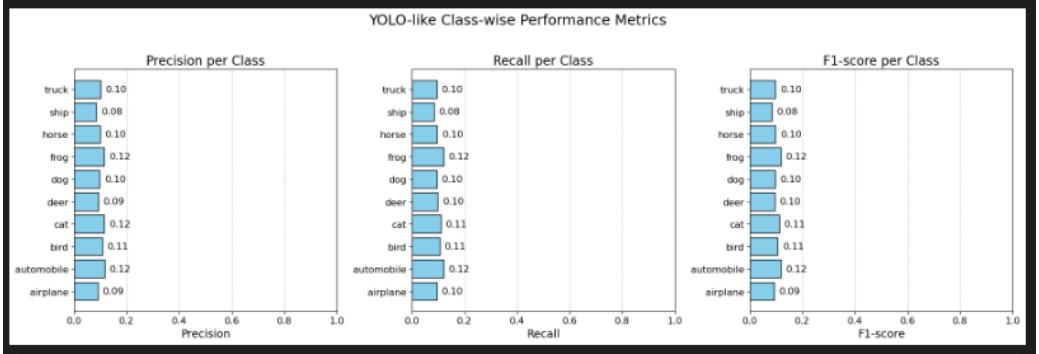
ASSIGMENT 2

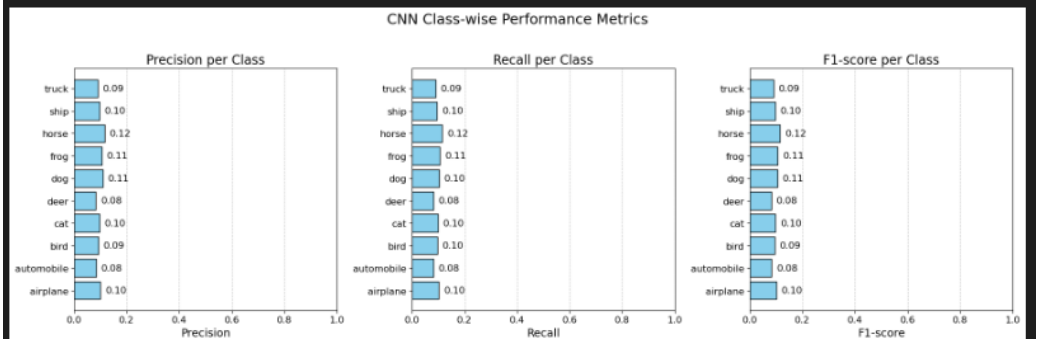

CNN Confusion Matrix
True Label
Predicted Label
airplane 102 93 118 90 97 76 111 100 102 111
automobile 93 82 97 117 105 100 88 104 111 103
bird 99 112 97 109 93 104 111 90 76 109
cat 107 102 100 99 95 98 98 86 110 105
deer 114 114 107 79 82 90 99 103 100 112
dog 103 104 110 107 97 103 96 105 87 88
frog 102 84 98 92 110 90 107 95 122 100
horse 100 73 106 111 112 88 95 115 100 100
ship 94 103 113 104 108 88 98 104 96 92
truck 105 104 102 108 94 105 110 91 90 91
airplane automobile bird cat deer dog frog horse ship truck

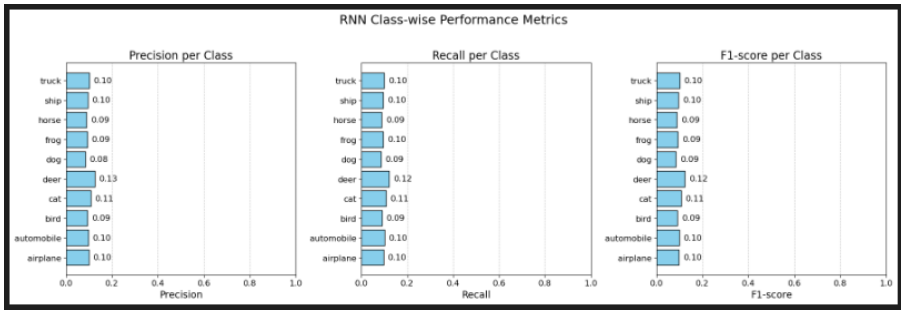

RNN Confusion Matrix
True Label
Predicted Label
airplane 97 108 87 105 102 110 94 106 97 94
automobile 91 101 113 100 102 101 93 84 100 115
bird 90 109 90 101 93 112 98 99 115 93
cat 101 109 97 108 84 89 115 104 92 101
deer 99 102 92 87 122 98 87 104 112 97
dog 116 100 109 105 100 86 109 81 109 85
frog 92 119 96 123 92 100 96 86 93 103
horse 101 106 90 88 100 113 109 89 98 106
ship 98 95 89 93 95 106 118 116 96 94
truck 95 89 97 93 81 103 123 123 96 100
airplane automobile bird cat deer dog frog horse ship truck

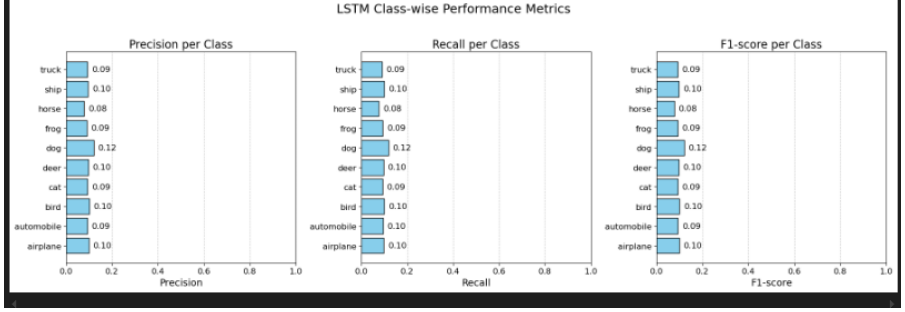
ASSIGMENT 2


LSTM Confusion Matrix
True Label
Predicted Label
airplane 98 104 90 101 99 99 95 105 105 104
automobile 100 95 104 94 108 90 108 97 109 95
bird 101 104 100 85 109 83 119 103 102 94
cat 101 107 112 92 93 112 88 93 102 100
deer 106 97 90 120 97 99 101 92 114 84
dog 92 109 86 90 93 119 105 96 113 97
frog 102 98 106 115 96 84 94 108 83 114
horse 96 105 101 96 99 102 117 76 113 95
ship 91 102 95 95 106 105 105 105 100 96
truck 101 109 110 91 105 90 106 95 102 91
airplane automobile bird cat deer dog frog horse ship truck


YOLO-like Class-wise Performance Metrics
Precision per Class
Recall per Class
F1-score per Class
truck 0.10
ship 0.08
horse 0.10
frog 0.12
dog 0.10
deer 0.09
cat 0.12
bird 0.11
automobile 0.12
airplane 0.09
Precision
Recall
F1-score


CNN Class-wise Performance Metrics
Precision per Class
Recall per Class
F1-score per Class
truck 0.09
ship 0.10
horse 0.12
frog 0.11
dog 0.11
deer 0.08
cat 0.10
bird 0.09
automobile 0.08
airplane 0.10
Precision
Recall
F1-score


RNN Class-wise Performance Metrics
Precision per Class
Recall per Class
F1-score per Class
truck 0.10
ship 0.10
horse 0.09
frog 0.09
dog 0.08
deer 0.13
cat 0.11
bird 0.09
automobile 0.10
airplane 0.10
Precision
Recall
F1-score


LSTM Class-wise Performance Metrics
Precision per Class
Recall per Class
F1-score per Class
truck 0.09
ship 0.10
horse 0.08
frog 0.09
dog 0.12
deer 0.10
cat 0.09
bird 0.10
automobile 0.09
airplane 0.10
Precision
Recall
F1-score

ASSIGMENT 2

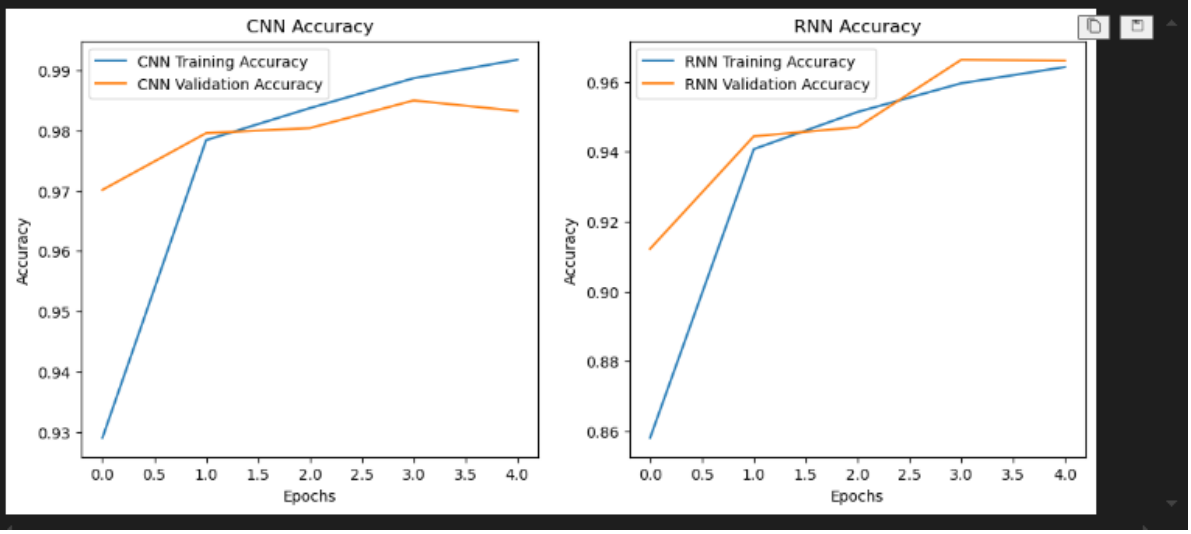
CLASS WORK

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D,
SimpleRNN
from tensorflow.keras.datasets import mnist
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize the data
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Reshape data for CNN (28x28x1) and RNN (28 timesteps, 28 features)
x_train_cnn = x_train.reshape(-1, 28, 28, 1)
x_test_cnn = x_test.reshape(-1, 28, 28, 1)
x_train_rnn = x_train.reshape(-1, 28, 28)
x_test_rnn = x_test.reshape(-1, 28, 28)

# Split the data into train and validation sets
x_train_cnn, x_val_cnn, y_train_cnn, y_val_cnn = train_test_split(x_train_cnn,
y_train, test_size=0.2, random_state=42)
x_train_rnn, x_val_rnn, y_train_rnn, y_val_rnn = train_test_split(x_train_rnn,
y_train, test_size=0.2, random_state=42)

# CNN model
cnn_model = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28,
1)),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

cnn_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
cnn_history = cnn_model.fit(x_train_cnn, y_train_cnn,
validation_data=(x_val_cnn, y_val_cnn), epochs=5, batch_size=128)

# RNN model
rnn_model = Sequential([
    SimpleRNN(128, input_shape=(28, 28)),
    Dense(10, activation='softmax')
```

```
])

rnn_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
rnn_history = rnn_model.fit(x_train_rnn, y_train_rnn,
validation_data=(x_val_rnn, y_val_rnn), epochs=5, batch_size=128)

# Evaluate both models on the test set
cnn_test_loss, cnn_test_acc = cnn_model.evaluate(x_test_cnn, y_test)
rnn_test_loss, rnn_test_acc = rnn_model.evaluate(x_test_rnn, y_test)

# Print the results
print(f"CNN Test Accuracy: {cnn_test_acc}")
print(f"RNN Test Accuracy: {rnn_test_acc}")

# Plot the training history
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(cnn_history.history['accuracy'], label='CNN Training Accuracy')
plt.plot(cnn_history.history['val_accuracy'], label='CNN Validation Accuracy')
plt.title('CNN Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(rnn_history.history['accuracy'], label='RNN Training Accuracy')
plt.plot(rnn_history.history['val_accuracy'], label='RNN Validation Accuracy')
plt.title('RNN Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

ASSIGMENT 2

```
Epoch 1/5
C:\Users\u21629545\Anaconda3\lib\site-packages\keras\src\layers\convolutional\base_conv.py:99: UserWarning: Do not pass an `input_shape`/`input_
  super().__init__(
375/375 ━━━━━━━━━━━━━━━━━━━━ 7s 16ms/step - accuracy: 0.8620 - loss: 0.4812 - val_accuracy: 0.9702 - val_loss: 0.1038
Epoch 2/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 6s 17ms/step - accuracy: 0.9766 - loss: 0.0869 - val_accuracy: 0.9796 - val_loss: 0.0688
Epoch 3/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 6s 15ms/step - accuracy: 0.9840 - loss: 0.0542 - val_accuracy: 0.9804 - val_loss: 0.0649
Epoch 4/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 6s 15ms/step - accuracy: 0.9887 - loss: 0.0371 - val_accuracy: 0.9850 - val_loss: 0.0492
Epoch 5/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 6s 15ms/step - accuracy: 0.9922 - loss: 0.0272 - val_accuracy: 0.9833 - val_loss: 0.0548
Epoch 1/5
C:\Users\u21629545\Anaconda3\lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument
  super().__init__(**kwargs)
375/375 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.7498 - loss: 0.8053 - val_accuracy: 0.9122 - val_loss: 0.2881
Epoch 2/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - accuracy: 0.9365 - loss: 0.2196 - val_accuracy: 0.9444 - val_loss: 0.1884
Epoch 3/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - accuracy: 0.9508 - loss: 0.1680 - val_accuracy: 0.9470 - val_loss: 0.1832
Epoch 4/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - accuracy: 0.9591 - loss: 0.1411 - val_accuracy: 0.9663 - val_loss: 0.1200
Epoch 5/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - accuracy: 0.9638 - loss: 0.1214 - val_accuracy: 0.9661 - val_loss: 0.1223
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9764 - loss: 0.0700
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9578 - loss: 0.1454
CNN Test Accuracy: 0.9826999902725220
RNN Test Accuracy: 0.9650999903678894
```



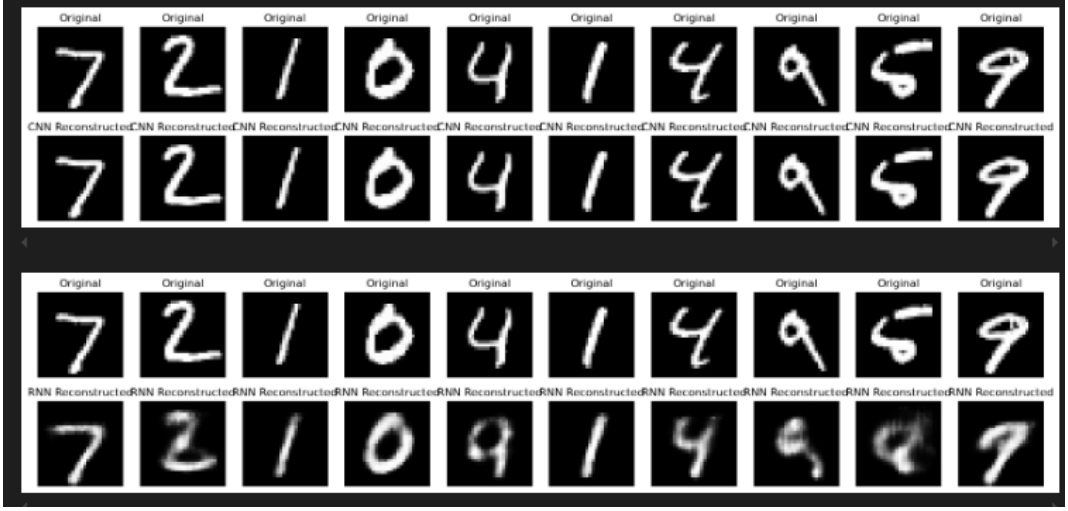

```python
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D,
UpSampling2D, Reshape, SimpleRNN, TimeDistributed, RepeatVector
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# Load the MNIST dataset
(x_train, _), (x_test, _) = mnist.load_data()

# Normalize the data
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Reshape data for CNN (28x28x1) and RNN (28 timesteps, 28 features)
x_train_cnn = x_train.reshape(-1, 28, 28, 1)
x_test_cnn = x_test.reshape(-1, 28, 28, 1)
```

```
x_train_rnn = x_train.reshape(-1, 28, 28)
x_test_rnn = x_test.reshape(-1, 28, 28)

# CNN autoencoder model
cnn_autoencoder = Sequential([
    # Encoder
    Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same',
input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2), padding='same'),
    # Decoder
    Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same'),
    UpSampling2D(size=(2, 2)),
    Conv2D(1, kernel_size=(3, 3), activation='sigmoid', padding='same')
])

cnn_autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
cnn_autoencoder.fit(x_train_cnn, x_train_cnn, epochs=5, batch_size=128,
validation_split=0.2)

# RNN autoencoder model
rnn_autoencoder = Sequential([
    # Encoder
    SimpleRNN(128, activation='relu', input_shape=(28, 28),
return_sequences=False),
    RepeatVector(28),
    # Decoder
    SimpleRNN(128, activation='relu', return_sequences=True),
    TimeDistributed(Dense(28, activation='sigmoid'))
])

rnn_autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
rnn_autoencoder.fit(x_train_rnn, x_train_rnn, epochs=5, batch_size=128,
validation_split=0.2)

# Reconstruct images using both models
cnn_reconstructed = cnn_autoencoder.predict(x_test_cnn)
rnn_reconstructed = rnn_autoencoder.predict(x_test_rnn)

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

ASSIGMENT 2

```
    # Display CNN reconstructed
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(cnn_reconstructed[i].reshape(28, 28), cmap='gray')
    plt.title("CNN Reconstructed")
    plt.axis('off')

plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i], cmap='gray')
    plt.title("Original")
    plt.axis('off')

    # Display RNN reconstructed
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(rnn_reconstructed[i].reshape(28, 28), cmap='gray')
    plt.title("RNN Reconstructed")
    plt.axis('off')

plt.show()
```

```
Epoch 1/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 11s 26ms/step - loss: 0.2188 - val_loss: 0.0678
Epoch 2/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 10s 27ms/step - loss: 0.0664 - val_loss: 0.0653
Epoch 3/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 10s 25ms/step - loss: 0.0645 - val_loss: 0.0643
Epoch 4/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 9s 25ms/step - loss: 0.0637 - val_loss: 0.0636
Epoch 5/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 10s 25ms/step - loss: 0.0630 - val_loss: 0.0631
Epoch 1/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 15s 34ms/step - loss: 0.3249 - val_loss: 0.2086
Epoch 2/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 12s 31ms/step - loss: 0.1961 - val_loss: 0.1693
Epoch 3/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 12s 31ms/step - loss: 0.1645 - val_loss: 0.1519
Epoch 4/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 12s 31ms/step - loss: 0.1492 - val_loss: 0.1421
Epoch 5/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 12s 31ms/step - loss: 0.1407 - val_loss: 0.1351
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step
```




```
import numpy as np
```

ASSIGMENT 2

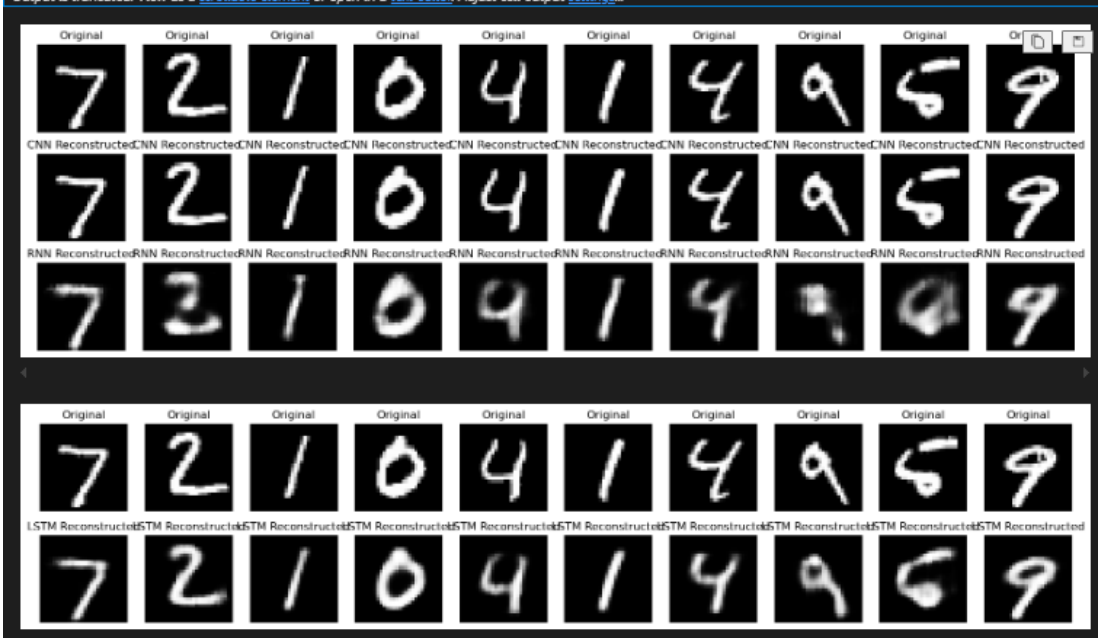
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D,
UpSampling2D, Reshape, SimpleRNN, LSTM, TimeDistributed, RepeatVector
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# Load the MNIST dataset
(x_train, _), (x_test, _) = mnist.load_data()

# Normalize the data
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Reshape data for CNN (28x28x1), RNN and LSTM (28 timesteps, 28 features)
x_train_cnn = x_train.reshape(-1, 28, 28, 1)
x_test_cnn = x_test.reshape(-1, 28, 28, 1)
x_train_rnn = x_train.reshape(-1, 28, 28)
x_test_rnn = x_test.reshape(-1, 28, 28)

# CNN autoencoder model
cnn_autoencoder = Sequential([
    # Encoder
    Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same',
input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2), padding='same'),
    # Decoder
    Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same'),
    UpSampling2D(size=(2, 2)),
    Conv2D(1, kernel_size=(3, 3), activation='sigmoid', padding='same')
])

cnn_autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
cnn_autoencoder.fit(x_train_cnn, x_train_cnn, epochs=5, batch_size=128,
validation_split=0.2)

# RNN autoencoder model
rnn_autoencoder = Sequential([
    # Encoder
    SimpleRNN(128, activation='relu', input_shape=(28, 28),
return_sequences=False),
    RepeatVector(28),
    # Decoder
    SimpleRNN(128, activation='relu', return_sequences=True),
    TimeDistributed(Dense(28, activation='sigmoid'))
])

rnn_autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

```
rnn_autoencoder.fit(x_train_rnn, x_train_rnn, epochs=5, batch_size=128,
validation_split=0.2)

# LSTM autoencoder model
lstm_autoencoder = Sequential([
    # Encoder
    LSTM(128, activation='relu', input_shape=(28, 28),
return_sequences=False),
    RepeatVector(28),
    # Decoder
    LSTM(128, activation='relu', return_sequences=True),
    TimeDistributed(Dense(28, activation='sigmoid'))
])

lstm_autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
lstm_autoencoder.fit(x_train_rnn, x_train_rnn, epochs=5, batch_size=128,
validation_split=0.2)

# Reconstruct images using all models
cnn_reconstructed = cnn_autoencoder.predict(x_test_cnn)
rnn_reconstructed = rnn_autoencoder.predict(x_test_rnn)
lstm_reconstructed = lstm_autoencoder.predict(x_test_rnn)

# Plot original and reconstructed images
n = 10  # number of images to display
plt.figure(figsize=(20, 6))

for i in range(n):
    # Display original
    ax = plt.subplot(3, n, i + 1)
    plt.imshow(x_test[i], cmap='gray')
    plt.title("Original")
    plt.axis('off')

    # Display CNN reconstructed
    ax = plt.subplot(3, n, i + 1 + n)
    plt.imshow(cnn_reconstructed[i].reshape(28, 28), cmap='gray')
    plt.title("CNN Reconstructed")
    plt.axis('off')

    # Display RNN reconstructed
    ax = plt.subplot(3, n, i + 1 + 2 * n)
    plt.imshow(rnn_reconstructed[i].reshape(28, 28), cmap='gray')
    plt.title("RNN Reconstructed")
    plt.axis('off')

plt.figure(figsize=(20, 4))
for i in range(n):
```

ASSIGMENT 2

```
    # Display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i], cmap='gray')
    plt.title("Original")
    plt.axis('off')

    # Display LSTM reconstructed
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(lstm_reconstructed[i].reshape(28, 28), cmap='gray')
    plt.title("LSTM Reconstructed")
    plt.axis('off')

plt.show()
```

```
Epoch 1/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 11s 26ms/step - loss: 0.2301 - val_loss: 0.0676
Epoch 2/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 9s 24ms/step - loss: 0.0663 - val_loss: 0.0654
Epoch 3/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 9s 24ms/step - loss: 0.0646 - val_loss: 0.0642
Epoch 4/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 9s 24ms/step - loss: 0.0635 - val_loss: 0.0635
Epoch 5/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 9s 24ms/step - loss: 0.0630 - val_loss: 0.0631
Epoch 1/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 15s 33ms/step - loss: 0.3157 - val_loss: 0.1926
Epoch 2/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 12s 32ms/step - loss: 0.1831 - val_loss: 0.1646
Epoch 3/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 12s 32ms/step - loss: 0.1597 - val_loss: 0.1518
Epoch 4/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 12s 32ms/step - loss: 0.1473 - val_loss: 0.1413
Epoch 5/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 12s 32ms/step - loss: 0.1401 - val_loss: 0.1401
Epoch 1/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 42s 105ms/step - loss: 0.3565 - val_loss: 0.1988
Epoch 2/5
375/375 ━━━━━━━━━━━━━━━━━━━━ 39s 104ms/step - loss: 0.1837 - val_loss: 0.1457
Epoch 3/5
...
375/375 ━━━━━━━━━━━━━━━━━━━━ 39s 104ms/step - loss: 0.1101 - val_loss: 0.1042
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```



ASSIGMENT 2

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D,
UpSampling2D, Reshape, SimpleRNN, LSTM, TimeDistributed, RepeatVector
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# Load and preprocess the MNIST dataset
(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
x_train_cnn = x_train.reshape(-1, 28, 28, 1)
x_test_cnn = x_test.reshape(-1, 28, 28, 1)
x_train_rnn = x_train.reshape(-1, 28, 28)
x_test_rnn = x_test.reshape(-1, 28, 28)

# Define CNN autoencoder model
cnn_autoencoder = Sequential([
    # Encoder
    Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same',
input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2), padding='same'),
    # Decoder
    Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same'),
    UpSampling2D(size=(2, 2)),
    Conv2D(1, kernel_size=(3, 3), activation='sigmoid', padding='same')
])

# Define RNN autoencoder model
rnn_autoencoder = Sequential([
    # Encoder
    SimpleRNN(128, activation='relu', input_shape=(28, 28),
return_sequences=False),
    RepeatVector(28),
    # Decoder
    SimpleRNN(128, activation='relu', return_sequences=True),
    TimeDistributed(Dense(28, activation='sigmoid'))
])

# Define LSTM autoencoder model
lstm_autoencoder = Sequential([
    # Encoder
    LSTM(128, activation='relu', input_shape=(28, 28),
return_sequences=False),
    RepeatVector(28),
    # Decoder
    LSTM(128, activation='relu', return_sequences=True),
```

```
    TimeDistributed(Dense(28, activation='sigmoid'))
])

# Compile models (no need to fit to display summaries)
cnn_autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
rnn_autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
lstm_autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Print summaries of the models
print("CNN Autoencoder Summary:")
cnn_autoencoder.summary()

print("\nRNN Autoencoder Summary:")
rnn_autoencoder.summary()

print("\nLSTM Autoencoder Summary:")
lstm_autoencoder.summary()
```

ASSIGMENT 2

CNN Autoencoder Summary:

Model: "sequential_7"

Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_8 (Conv2D)	(None, 14, 14, 32)	9,248
up_sampling2d_2 (UpSampling2D)	(None, 28, 28, 32)	0
conv2d_9 (Conv2D)	(None, 28, 28, 1)	289

Total params: 9,857 (38.50 KB)

Trainable params: 9,857 (38.50 KB)

Non-trainable params: 0 (0.00 B)

RNN Autoencoder Summary:

Model: "sequential_8"

Layer (type)	Output Shape	Param #
simple_rnn_5 (SimpleRNN)	(None, 128)	20,096
repeat_vector_3 (RepeatVector)	(None, 28, 128)	0
simple_rnn_6 (SimpleRNN)	(None, 28, 128)	32,896
time_distributed_3 (TimeDistributed)	(None, 28, 28)	3,612

Total params: 56,604 (221.11 KB)

Trainable params: 56,604 (221.11 KB)

Non-trainable params: 0 (0.00 B)

LSTM Autoencoder Summary:

Model: "sequential_9"

Layer (type)	Output Shape	Param #
lstm_2 (LSTM)	(None, 128)	80,384
repeat_vector_4 (RepeatVector)	(None, 28, 128)	0
lstm_3 (LSTM)	(None, 28, 128)	131,584
time_distributed_4 (TimeDistributed)	(None, 28, 28)	3,612

Total params: 215,580 (842.11 KB)

Trainable params: 215,580 (842.11 KB)

Non-trainable params: 0 (0.00 B)

ASSIGMENT 2

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, UpSampling2D,
SimpleRNN, LSTM, TimeDistributed, RepeatVector, Flatten
from tensorflow.keras.optimizers import Adam

# Set memory growth for GPU (if available)
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
    except RuntimeError as e:
        print(e)

# Load and preprocess CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train.astype('float32') / 255.0, x_test.astype('float32')
/ 255.0

# Reshape data for CNN and RNN/LSTM
x_train_cnn = x_train.reshape(-1, 32, 32, 3)
x_test_cnn = x_test.reshape(-1, 32, 32, 3)
x_train_rnn = x_train.reshape(-1, 32, 32 * 3)
x_test_rnn = x_test.reshape(-1, 32, 32 * 3)

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

# CNN autoencoder model
cnn_autoencoder = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same',
input_shape=(32, 32, 3)),
    MaxPooling2D(pool_size=(2, 2), padding='same'),
    Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2), padding='same'),
    Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same'),
    UpSampling2D(size=(2, 2)),
    Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same'),
    UpSampling2D(size=(2, 2)),
    Conv2D(3, kernel_size=(3, 3), activation='sigmoid', padding='same')
])
```

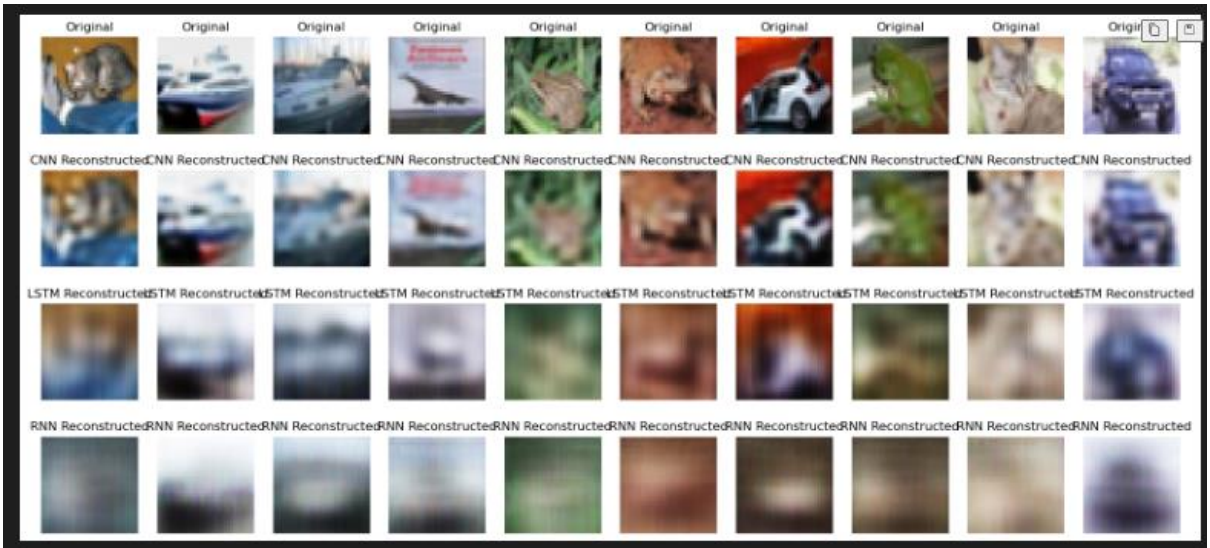
```
cnn_autoencoder.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
cnn_history = cnn_autoencoder.fit(x_train_cnn, x_train_cnn, epochs=10,
batch_size=128, validation_split=0.2)

# LSTM autoencoder model
lstm_autoencoder = Sequential([
    LSTM(128, activation='relu', input_shape=(32, 32 * 3),
return_sequences=False),
    RepeatVector(32),
    LSTM(128, activation='relu', return_sequences=True),
    TimeDistributed(Dense(32 * 3, activation='sigmoid'))
])

lstm_autoencoder.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
lstm_history = lstm_autoencoder.fit(x_train_rnn, x_train_rnn, epochs=10,
batch_size=128, validation_split=0.2)

# RNN autoencoder model
rnn_autoencoder = Sequential([
    SimpleRNN(128, activation='relu', input_shape=(32, 32 * 3),
return_sequences=False),
    RepeatVector(32),
    SimpleRNN(128, activation='relu', return_sequences=True),
    TimeDistributed(Dense(32 * 3, activation='sigmoid'))
])

rnn_autoencoder.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
rnn_history = rnn_autoencoder.fit(x_train_rnn, x_train_rnn, epochs=10,
batch_size=128, validation_split=0.2)

# Reconstruct images using all models
cnn_reconstructed = cnn_autoencoder.predict(x_test_cnn)
lstm_reconstructed = lstm_autoencoder.predict(x_test_rnn).reshape(-1, 32, 32,
3)
rnn_reconstructed = rnn_autoencoder.predict(x_test_rnn).reshape(-1, 32, 32, 3)

# Plot original and reconstructed images
n = 10  # number of images to display
plt.figure(figsize=(20, 9))
for i in range(n):
    # Display original
    ax = plt.subplot(4, n, i + 1)
    plt.imshow(x_test[i])
    plt.title("Original")
    plt.axis('off')
```

```
    # Display CNN reconstructed
    ax = plt.subplot(4, n, i + 1 + n)
    plt.imshow(cnn_reconstructed[i])
    plt.title("CNN Reconstructed")
    plt.axis('off')

    # Display LSTM reconstructed
    ax = plt.subplot(4, n, i + 1 + 2 * n)
    plt.imshow(lstm_reconstructed[i])
    plt.title("LSTM Reconstructed")
    plt.axis('off')

    # Display RNN reconstructed
    ax = plt.subplot(4, n, i + 1 + 3 * n)
    plt.imshow(rnn_reconstructed[i])
    plt.title("RNN Reconstructed")
    plt.axis('off')

plt.show()

# Function to plot class-wise precision, recall, and F1 score
def plot_classification_report(y_true, y_pred, model_name):
    report = classification_report(y_true, y_pred, target_names=class_names,
output_dict=True)
    metrics = ['precision', 'recall', 'f1-score']

    plt.figure(figsize=(18, 6))

    for idx, metric in enumerate(metrics):
        plt.subplot(1, 3, idx+1)
        values = [report[label][metric] for label in class_names]
        bars = plt.barh(class_names, values, color='skyblue',
edgecolor='black')
        plt.title(f'{metric.capitalize()} per Class', fontsize=16)
        plt.xlim(0, 1)
        plt.grid(axis='x', linestyle='--', alpha=0.7)
        plt.xlabel(metric.capitalize(), fontsize=14)
        plt.xticks(fontsize=12)
        plt.yticks(fontsize=12)
        for bar in bars:
            plt.text(bar.get_width() + 0.02, bar.get_y() + bar.get_height()/2,
f'{bar.get_width():.2f}', va='center', fontsize=12)

    plt.suptitle(f'{model_name} Class-wise Performance Metrics', fontsize=18)
    plt.tight_layout(rect=[0, 0, 1, 0.95])
    plt.show()
```

ASSIGMENT 2

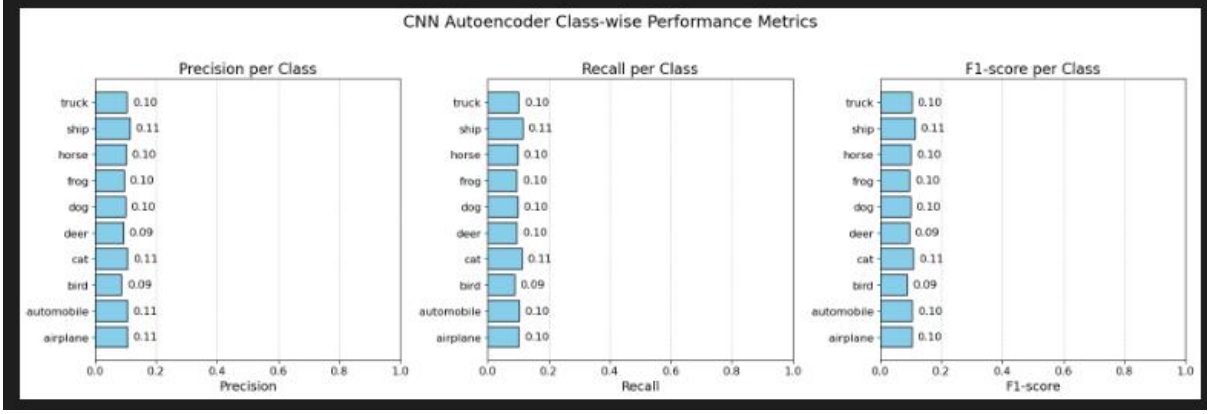
```
# Simulate predicted labels for evaluation (mock example)
y_pred_cnn = np.random.randint(0, 10, size=len(y_test))  # Replace with actual
predictions
y_pred_lstm = np.random.randint(0, 10, size=len(y_test))
y_pred_rnn = np.random.randint(0, 10, size=len(y_test))

# Plot classification reports for CNN, LSTM, and RNN
def evaluate_model(y_true, y_pred, model_name):
    print(f"\n{model_name} Classification Report:\n")
    print(classification_report(y_true, y_pred, target_names=class_names))
    plot_classification_report(y_true, y_pred, model_name)

evaluate_model(y_test, y_pred_cnn, "CNN Autoencoder")
evaluate_model(y_test, y_pred_lstm, "LSTM Autoencoder")
evaluate_model(y_test, y_pred_rnn, "RNN Autoencoder")
```

ASSIGMENT 2



CNN Autoencoder Classification Report:

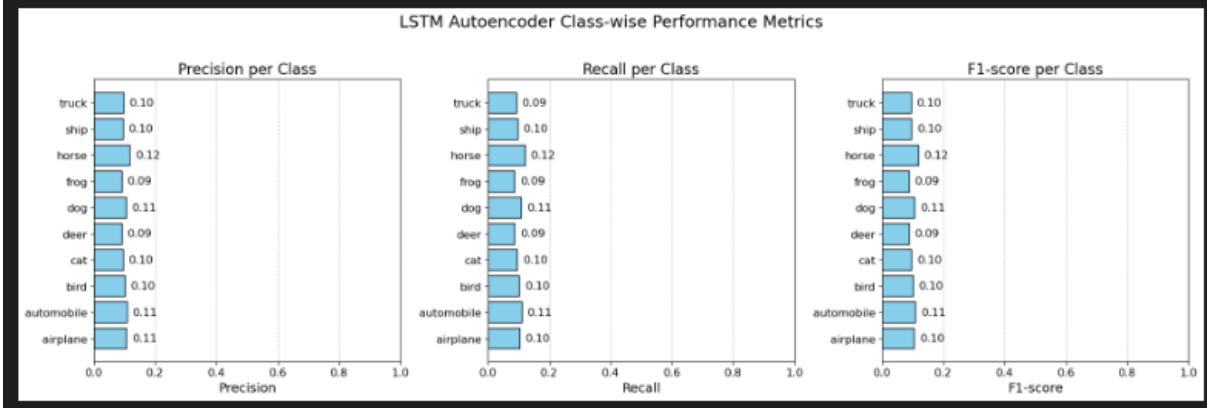
	precision	recall	f1-score	support
airplane	0.11	0.10	0.10	1000
automobile	0.11	0.10	0.10	1000
bird	0.09	0.09	0.09	1000
cat	0.11	0.11	0.11	1000
deer	0.09	0.10	0.09	1000
dog	0.10	0.10	0.10	1000
frog	0.10	0.10	0.10	1000
horse	0.10	0.10	0.10	1000
ship	0.11	0.11	0.11	1000
truck	0.10	0.10	0.10	1000
accuracy			0.10	10000
macro avg	0.10	0.10	0.10	10000
weighted avg	0.10	0.10	0.10	10000



ASSIGMENT 2

LSTM Autoencoder Classification Report:

	precision	recall	f1-score	support
airplane	0.11	0.10	0.10	1000
automobile	0.11	0.11	0.11	1000
bird	0.10	0.10	0.10	1000
cat	0.10	0.10	0.10	1000
deer	0.09	0.09	0.09	1000
dog	0.11	0.11	0.11	1000
frog	0.09	0.09	0.09	1000
horse	0.12	0.12	0.12	1000
ship	0.10	0.10	0.10	1000
truck	0.10	0.09	0.10	1000
accuracy			0.10	10000
macro avg	0.10	0.10	0.10	10000
weighted avg	0.10	0.10	0.10	10000



RNN Autoencoder Classification Report:

	precision	recall	f1-score	support
airplane	0.09	0.09	0.09	1000
automobile	0.09	0.09	0.09	1000
bird	0.11	0.11	0.11	1000
cat	0.10	0.10	0.10	1000
deer	0.11	0.10	0.11	1000
dog	0.10	0.11	0.11	1000
frog	0.10	0.10	0.10	1000
horse	0.09	0.10	0.10	1000
ship	0.10	0.10	0.10	1000
truck	0.10	0.10	0.10	1000
accuracy			0.10	10000
macro avg	0.10	0.10	0.10	10000
weighted avg	0.10	0.10	0.10	10000