CNN Classifier on CIFAR-10 Dataset

Loading Libraries

In [ ]:
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import classification_report, accuracy_score
import pandas as pd
from tabulate import tabulate
```

Loading Dataset

In [ ]:
```
# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0
```

In [ ]:
```
# Define class names for CIFAR-10 dataset
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

In [ ]:
```
# Display some images from the dataset (optional)
plt.figure(figsize=(8,8))
for i in range(16):
    plt.subplot(4,4,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_train[i])
    plt.xlabel(class_names[np.argmax(y_train[i][0])])
plt.show()
```

Defining the CNN Model

In [ ]:
```
# Build the CNN model
model = models.Sequential()
```

```
# model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
```

In [ ]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Training the Model

In [ ]:
```
history = model.fit(x_train, y_train, epochs=10,
                    validation_data=(x_test, y_test))
```

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 15s 9ms/step - accuracy: 0.3500 - loss: 1.7656 - val_accuracy: 0.5402 - val_loss: 1.2928
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - accuracy: 0.5656 - loss: 1.2180 - val_accuracy: 0.6207 - val_loss: 1.0896
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - accuracy: 0.6296 - loss: 1.0450 - val_accuracy: 0.6411 - val_loss: 1.0168
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 13s 9ms/step - accuracy: 0.6753 - loss: 0.9280 - val_accuracy: 0.6689 - val_loss: 0.9583
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - accuracy: 0.7039 - loss: 0.8409 - val_accuracy: 0.6649 - val_loss: 0.9567
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - accuracy: 0.7239 - loss: 0.7749 - val_accuracy: 0.6803 - val_loss: 0.9308
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - accuracy: 0.7452 - loss: 0.7223 - val_accuracy: 0.6971 - val_loss: 0.8802
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - accuracy: 0.7601 - loss: 0.6837 - val_accuracy: 0.6833 - val_loss: 0.9464
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - accuracy: 0.7776 - loss: 0.6292 - val_accuracy: 0.7001 - val_loss: 0.8998
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 13s 8ms/step - accuracy: 0.7947 - loss: 0.5859 - val_accuracy: 0.6920 - val_loss: 0.9144
```

Saving the model

In [ ]:
```
# Save the model architecture as JSON
model_json = model.to_json()
with open("cnn_model.json", "w") as json_file:
    json_file.write(model_json)

# Save the weights with the correct filename
model.save_weights("cnn_model_weights.weights.h5")

print("Model weights saved to disk.")


# # To Load Model ::
# # Load the JSON file that contains the model architecture
# with open('fnn_model.json', 'r') as json_file:
#     loaded_model_json = json_file.read()

# # Reconstruct the model from the JSON file
# loaded_model = tf.keras.models.model_from_json(loaded_model_json)

# # Load the saved weights into the model
# loaded_model.load_weights("fnn_model_weights.h5")

# print("Model loaded from disk.")
```

```
Model weights saved to disk.
```

Evaluating the Model Predictions

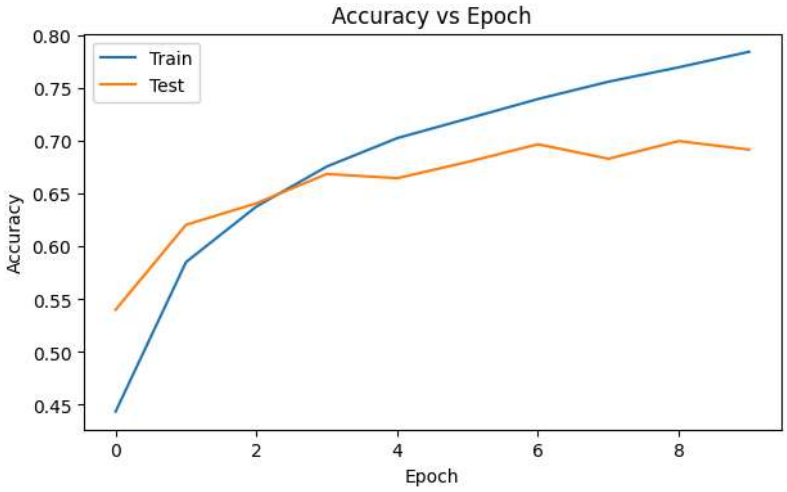
In [ ]:
```
# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f"Test accuracy: {test_acc*100:.2f}%")
```

```
313/313 - 1s - 3ms/step - accuracy: 0.6920 - loss: 0.9144
Test accuracy: 69.20%
Test accuracy: 69.20%
```

Plot: Accuracy vs Epoch

In [ ]:
```
# Plot training & validation accuracy values
plt.figure(figsize=(7, 4))
plt.plot(history.history['accuracy'])
```

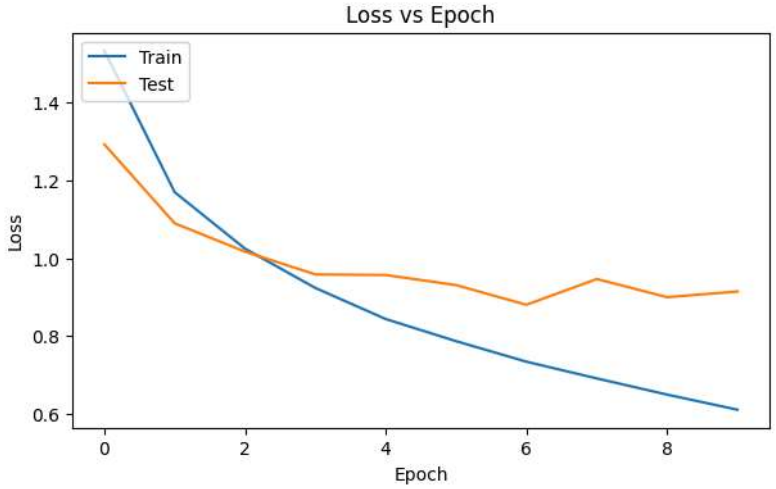
```
plt.plot(history.history['val_accuracy'])
plt.title('Accuracy vs Epoch')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Test'], loc='upper left')
plt.savefig('accuracy_vs_epoch_CNN.png')
```



Plot: Loss vs Epoch

In [ ]:
```
# Plot training & validation loss values
plt.figure(figsize=(7, 4))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Loss vs Epoch')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Test'], loc='upper left')
plt.savefig('loss_vs_epoch_CNN.png')


plt.show()
```



Visualising the Predictions

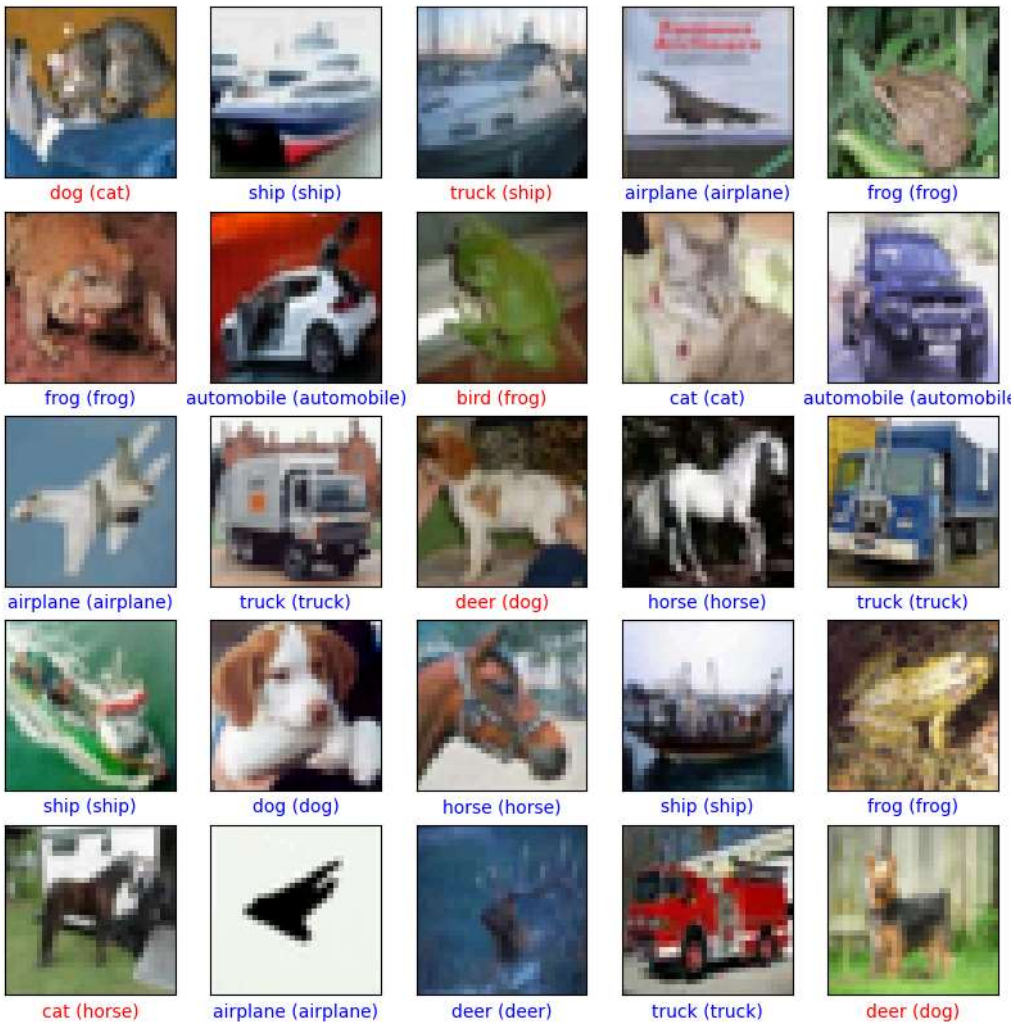
In [ ]:
```
# Make predictions
predictions = model.predict(x_test)

# Display some predictions
plt.figure(figsize=(10, 10))

for i in range(25):
    plt.subplot(5, 5, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_test[i])
    predicted_label = class_names[np.argmax(predictions[i])]
    true_label = class_names[y_test[i][0]]
```

```
    color = 'blue' if predicted_label == true_label else 'red'
    plt.xlabel(f"{predicted_label} ({true_label})", color=color)
    plt.savefig('Predictions_CNN.png')
plt.show()
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
```



Tabulating Classification Report

In [ ]:
```
# One-hot encode the labels
y_train, y_test = to_categorical(y_train), to_categorical(y_test)

# Convert predictions to class labels
y_pred = np.argmax(predictions, axis=1)
y_true = np.argmax(y_test, axis=1)
```

In [ ]:
```
# Calculate accuracy
accuracy = accuracy_score(y_true, y_pred)
print(f"Accuracy: {accuracy*100:.3f}")

# Generate classification report
report = classification_report(y_true, y_pred, target_names=class_names, output_dict=True)

# Convert classification report to DataFrame
report_df = pd.DataFrame(report).transpose()*100

# Calculate accuracy for each class
report_df['accuracy'] = report_df.apply(lambda row: row['support'] * row['recall'] / row['support']
    if row.name in class_names else np.nan, axis=1)

# Remove accuracy, macro avg, and weighted avg rows
report_df = report_df.loc[class_names]

# Select and reorder columns
report_df = report_df[['accuracy', 'precision', 'recall', 'f1-score']]


# Round the DataFrame to 2 decimal places
report_df = report_df.round(2)
```

```
Accuracy: 69.200
```

Display the Table

In [ ]:
```python
# Display the classification report in a box format
print(tabulate(report_df, headers='keys', tablefmt='grid'))

# Optionally, save the table to a CSV file
report_df.to_csv('classification_report_CNN.csv', index=True)
```

```
+------------+------------+-------------+----------+------------+
|            |   accuracy |   precision |   recall |   f1-score |
+============+============+=============+==========+============+
| airplane   |       76.5 |       70.64 |     76.5 |      73.45 |
+------------+------------+-------------+----------+------------+
| automobile |       86.1 |       76.67 |     86.1 |      81.11 |
+------------+------------+-------------+----------+------------+
| bird       |       60.1 |       58.52 |     60.1 |      59.3  |
+------------+------------+-------------+----------+------------+
| cat        |       42.4 |       54.22 |     42.4 |      47.59 |
+------------+------------+-------------+----------+------------+
| deer       |       63.7 |       65.4  |     63.7 |      64.54 |
+------------+------------+-------------+----------+------------+
| dog        |       61.7 |       56.4  |     61.7 |      58.93 |
+------------+------------+-------------+----------+------------+
| frog       |       83   |       70.94 |     83   |      76.5  |
+------------+------------+-------------+----------+------------+
| horse      |       65.7 |       83.91 |     65.7 |      73.7  |
+------------+------------+-------------+----------+------------+
| ship       |       73.1 |       85.1  |     73.1 |      78.64 |
+------------+------------+-------------+----------+------------+
| truck      |       79.7 |       72.13 |     79.7 |      75.72 |
+------------+------------+-------------+----------+------------+
```

In [ ]:
```python
# Create a matplotlib figure
fig, ax = plt.subplots(figsize=(7, 6))  # Adjust the size as needed

# Hide axes
ax.xaxis.set_visible(False)
ax.yaxis.set_visible(False)
ax.set_frame_on(False)

# Create the table
table = ax.table(cellText=report_df.values,
                 colLabels=report_df.columns,
                 rowLabels=report_df.index,
                 cellLoc='center',
                 loc='center')

# Adjust table properties
table.auto_set_font_size(True)
# table.set_fontsize(11)
table.scale(1.2, 1.2)

# Add corner label
table.add_cell(0, -1, width=0.15, height=0.045)
table[0, -1].set_text_props(text='Class Names / Scores', weight='bold')

# Add a title to the plot
plt.title('Classification Report (CNN)', x=0.3, y=0.95, fontsize=16, fontweight='bold', ha='center')

# Adjust plot layout
# plt.subplots_adjust(top=1)

# Save the table as an image
plt.savefig('classification_report_CNN.png', bbox_inches='tight', dpi=300)

# Show the plot
plt.show()
```

Classification Report (CNN)

Class Names / Scores	accuracy	precision	recall	f1-score
airplane	76.5	70.64	76.5	73.45
automobile	86.1	76.67	86.1	81.11
bird	60.1	58.52	60.1	59.3
cat	42.4	54.22	42.4	47.59
deer	63.7	65.4	63.7	64.54
dog	61.7	56.4	61.7	58.93
frog	83.0	70.94	83.0	76.5
horse	65.7	83.91	65.7	73.7
ship	73.1	85.1	73.1	78.64
truck	79.7	72.13	79.7	75.72