FNN Classifier on CIFAR-10 Dataset

Loading Libraries

In [ ]:
```python
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 [ ]:
```python
# 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

# Flatten the images
x_train_flattened = x_train.reshape((x_train.shape[0], -1))
x_test_flattened = x_test.reshape((x_test.shape[0], -1))
```

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

In [ ]:
```python
# 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 [ ]:
```python
# Build the FNN model
model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(x_train_flattened.shape[1],)))
```

```
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(128, 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_flattened, y_train, epochs=10,
                    validation_data=(x_test_flattened, y_test))
```

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 25s 15ms/step - accuracy: 0.2683 - loss: 2.0066 - val_accuracy: 0.3939 - val_loss: 1.6992
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 23s 15ms/step - accuracy: 0.3800 - loss: 1.7112 - val_accuracy: 0.4298 - val_loss: 1.6117
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 23s 15ms/step - accuracy: 0.4217 - loss: 1.6183 - val_accuracy: 0.4220 - val_loss: 1.6272
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 23s 15ms/step - accuracy: 0.4431 - loss: 1.5517 - val_accuracy: 0.4330 - val_loss: 1.5913
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 23s 15ms/step - accuracy: 0.4617 - loss: 1.5114 - val_accuracy: 0.4758 - val_loss: 1.4808
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 23s 15ms/step - accuracy: 0.4739 - loss: 1.4675 - val_accuracy: 0.4580 - val_loss: 1.5275
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 23s 15ms/step - accuracy: 0.4857 - loss: 1.4391 - val_accuracy: 0.4797 - val_loss: 1.4631
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 23s 15ms/step - accuracy: 0.4947 - loss: 1.4130 - val_accuracy: 0.4706 - val_loss: 1.4824
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 23s 15ms/step - accuracy: 0.5006 - loss: 1.3912 - val_accuracy: 0.4780 - val_loss: 1.4732
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 23s 15ms/step - accuracy: 0.5081 - loss: 1.3676 - val_accuracy: 0.4991 - val_loss: 1.4223
```

Saving the model

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

# Save the weights with the correct filename
model.save_weights("fnn_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.
```

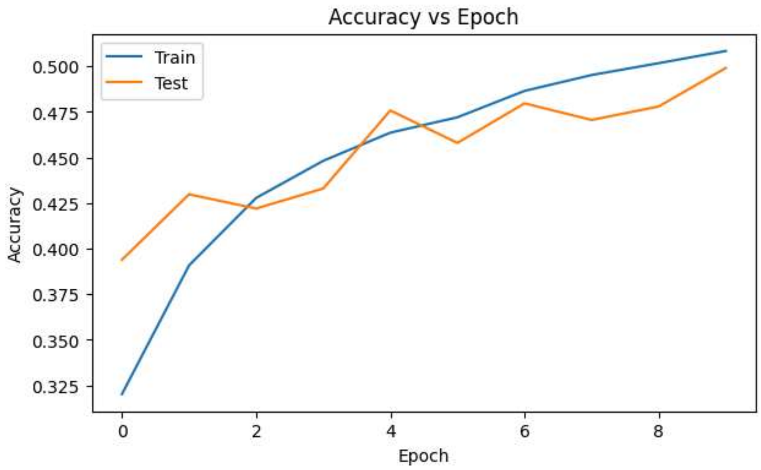
Evaluating the Model Predictions

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

```
313/313 - 1s - 2ms/step - accuracy: 0.4991 - loss: 1.4223
Test accuracy: 49.91%
```

Plot: Accuracy vs Epoch

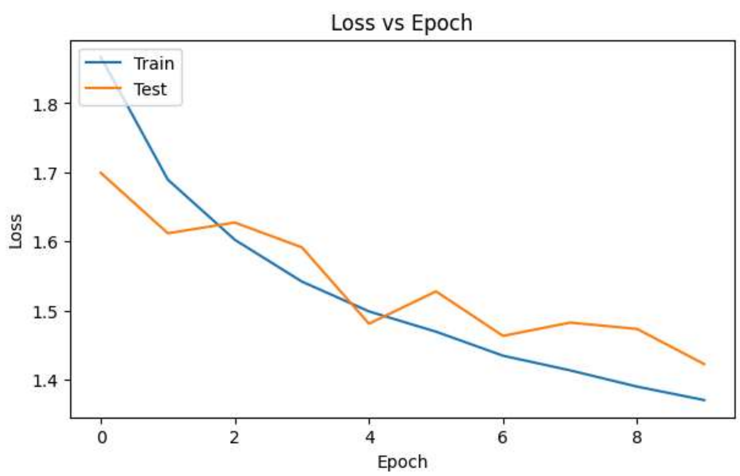
In [ ]:
```
# Plot training & validation accuracy values
plt.figure(figsize=(7, 4))
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Accuracy vs Epoch')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Test'], loc='upper left')
plt.savefig('accuracy_vs_epoch_FNN.png')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



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_FNN.png')


plt.show()
```



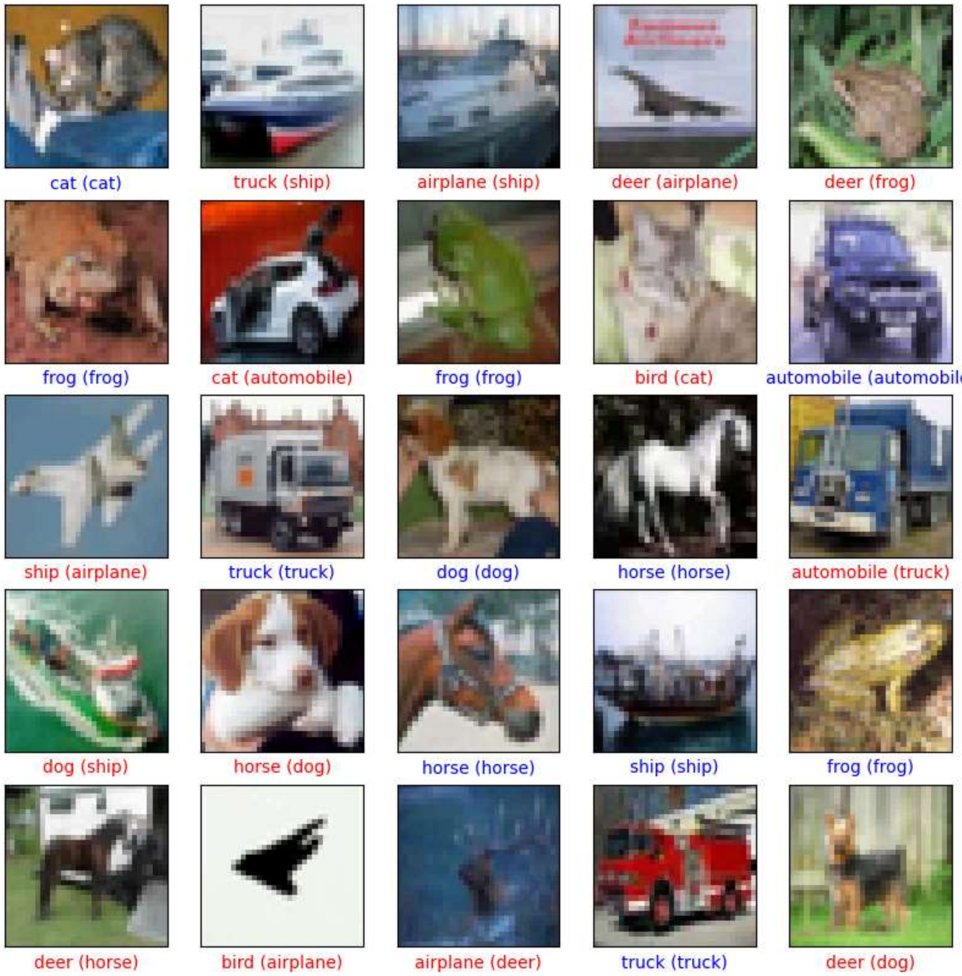
Visualising the Predictions

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

# 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_FNN.png')
plt.show()
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/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:.2f}")

# 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: 49.91
```

Display the Table

In [ ]:
```
# 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_FNN.csv', index=True)
```

	accuracy	precision	recall	f1-score
airplane	52	59.5	52	55.5
automobile	61.3	63.59	61.3	62.42
bird	32.5	39.63	32.5	35.71
cat	30.1	37.07	30.1	33.22
deer	43	39.23	43	41.03
dog	37.8	43.75	37.8	40.56
frog	61.3	46.33	61.3	52.78
horse	63.2	50.04	63.2	55.86
ship	58.7	64.36	58.7	61.4
truck	59.2	55.22	59.2	57.14

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 (FNN)', 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_FNN.png', bbox_inches='tight', dpi=300)

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

Classification Report (FNN)

Class Names / Scores	accuracy	precision	recall	f1-score
airplane	52.0	59.5	52.0	55.5
automobile	61.3	63.59	61.3	62.42
bird	32.5	39.63	32.5	35.71
cat	30.1	37.07	30.1	33.22
deer	43.0	39.23	43.0	41.03
dog	37.8	43.75	37.8	40.56
frog	61.3	46.33	61.3	52.78
horse	63.2	50.04	63.2	55.86
ship	58.7	64.36	58.7	61.4
truck	59.2	55.22	59.2	57.14