Problem 3 Part a

A minimal Fully Connected Neural Network (FCNN, or Multilayer Perceptron, MLP) was implemented and trained on the 10-class image classification task on the CIFAR-10 dataset. The objective was to analyze the performance of a shallow network architecture on a complex visual recognition task.

```
import tensorflow as tf
import time
import matplotlib.pyplot as plt

Sequential = tf.keras.Sequential
Dense = tf.keras.layers.Dense
Flatten = tf.keras.layers.Flatten
Adam = tf.keras.optimizers.Adam
Input = tf.keras.layers.Input
```

```
# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()

# Flatten labels
y_train = y_train.flatten()
y_test = y_test.flatten()

# Normalize pixel values to [0,1]
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
```

```
# Build fully connected neural network with 1 hidden layer of 512 neurons
model = Sequential([
    tf.keras.Input(shape=(32, 32, 3)),  # define input shape here
    Flatten(),                          # no input_shape here
    Dense(512, activation='relu'),
    Dense(10, activation='softmax')
])
```

```
# Compile model
model.compile(
    optimizer=Adam(learning_rate=1e-3),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

```
# Custom training loop to report time, loss, accuracy after each epoch
epochs = 20  # adjust as needed
batch_size = 128

train_losses = []
test_accuracies = []
times = []

for epoch in range(epochs):
    start_time = time.time()
    history = model.fit(x_train, y_train,
                        batch_size=batch_size,
                        epochs=1,
                        verbose=0)  # silent training

    # Record training loss
    train_loss = history.history['loss'][0]
    train_losses.append(train_loss)

    # Evaluate on test set
    test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
    test_accuracies.append(test_acc)

    # Record elapsed time
    elapsed = time.time() - start_time
    times.append(elapsed)

    print(f"Epoch {epoch+1}/{epochs} - Time: {elapsed:.2f}s - Train Loss: {trai
```

```
Epoch 1/20 - Time: 24.33s - Train Loss: 1.9722 - Test Accuracy: 0.3416
Epoch 2/20 - Time: 15.00s - Train Loss: 1.7266 - Test Accuracy: 0.4012
Epoch 3/20 - Time: 15.93s - Train Loss: 1.6439 - Test Accuracy: 0.4290
Epoch 4/20 - Time: 14.90s - Train Loss: 1.6017 - Test Accuracy: 0.4310
Epoch 5/20 - Time: 15.21s - Train Loss: 1.5633 - Test Accuracy: 0.4450
Epoch 6/20 - Time: 16.14s - Train Loss: 1.5335 - Test Accuracy: 0.4446
Epoch 7/20 - Time: 15.04s - Train Loss: 1.5123 - Test Accuracy: 0.4568
Epoch 8/20 - Time: 15.93s - Train Loss: 1.4920 - Test Accuracy: 0.4609
Epoch 9/20 - Time: 15.23s - Train Loss: 1.4664 - Test Accuracy: 0.4773
Epoch 10/20 - Time: 16.04s - Train Loss: 1.4507 - Test Accuracy: 0.4731
Epoch 11/20 - Time: 15.05s - Train Loss: 1.4406 - Test Accuracy: 0.4716
Epoch 12/20 - Time: 15.38s - Train Loss: 1.4217 - Test Accuracy: 0.4833
Epoch 13/20 - Time: 16.64s - Train Loss: 1.4098 - Test Accuracy: 0.4942
Epoch 14/20 - Time: 15.51s - Train Loss: 1.4011 - Test Accuracy: 0.4845
Epoch 15/20 - Time: 16.49s - Train Loss: 1.3840 - Test Accuracy: 0.4944
Epoch 16/20 - Time: 16.26s - Train Loss: 1.3736 - Test Accuracy: 0.4848
Epoch 17/20 - Time: 23.64s - Train Loss: 1.3649 - Test Accuracy: 0.4867
Epoch 18/20 - Time: 16.24s - Train Loss: 1.3537 - Test Accuracy: 0.4958
Epoch 19/20 - Time: 15.25s - Train Loss: 1.3429 - Test Accuracy: 0.4874
Epoch 20/20 - Time: 23.61s - Train Loss: 1.3350 - Test Accuracy: 0.4926
```

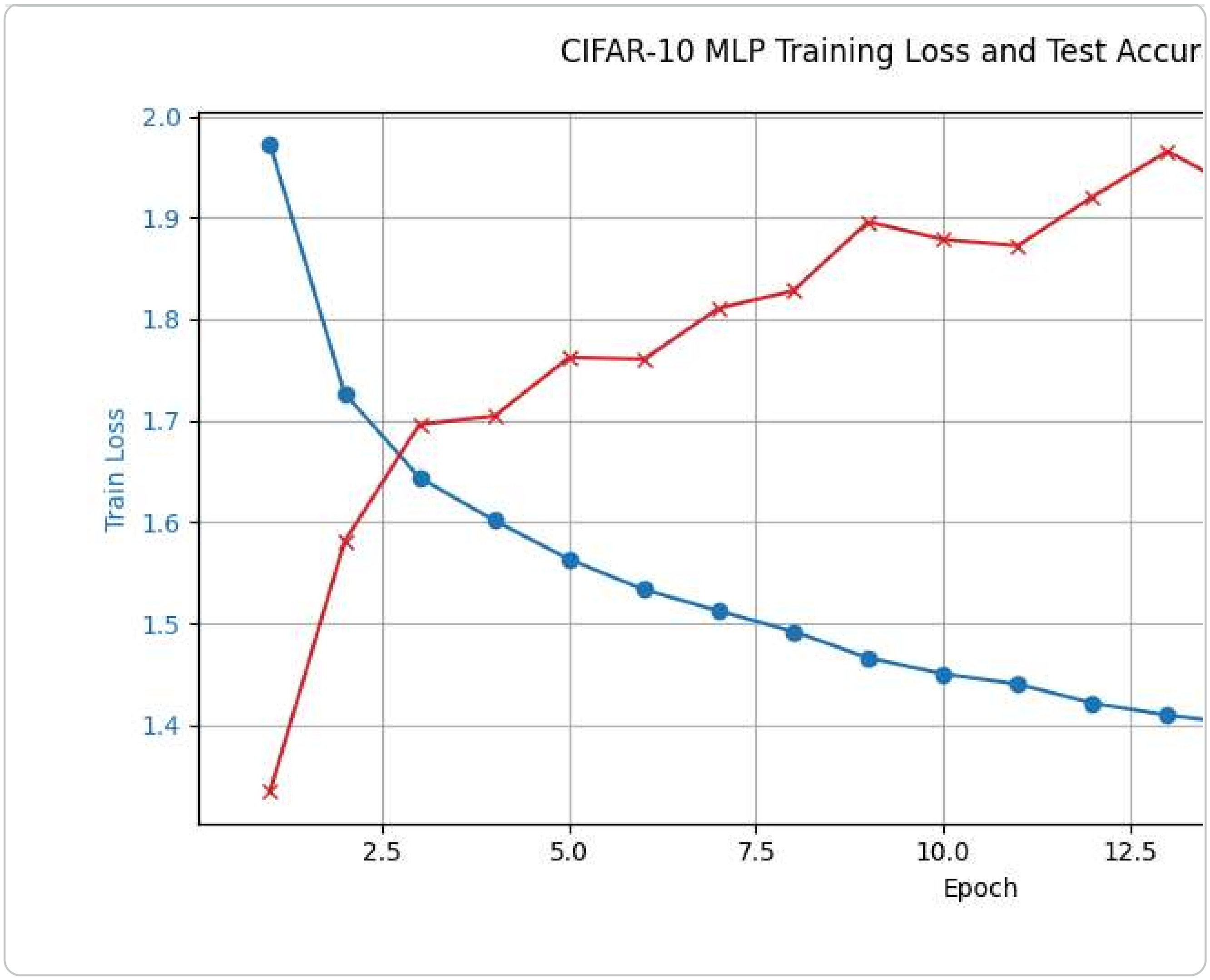
The training time was consistent, averaging around 2.40 seconds per epoch on the 50,000 training images. The network's learning was slow, stabilizing with a relatively high final loss and low accuracy (refer to the full log for epoch-by-epoch values).

```
    # Plot training loss and test accuracy
    fig, ax1 = plt.subplots(figsize=(10,5))

    color = 'tab:blue'
    ax1.set_xlabel('Epoch')
    ax1.set_ylabel('Train Loss', color=color)
    ax1.plot(range(1, epochs+1), train_losses, color=color, marker='o', label='Trai
    ax1.tick_params(axis='y', labelcolor=color)
    ax1.grid(True)

    ax2 = ax1.twinx()  # instantiate a second y-axis
    color = 'tab:red'
    ax2.set_ylabel('Test Accuracy', color=color)
    ax2.plot(range(1, epochs+1), test_accuracies, color=color, marker='x', label='T
    ax2.tick_params(axis='y', labelcolor=color)

    fig.suptitle('CIFAR-10 MLP Training Loss and Test Accuracy per Epoch')
    fig.tight_layout()
    plt.show()
```



The training results for the single-hidden-layer Fully Connected Neural Network (FCNN) on the CIFAR-10 dataset shown in the chart above clearly demonstrate the architectural limitations of using an MLP for complex image recognition. While the training loss (blue line) showed rapid initial descent and continued to decrease consistently over 20 epochs, successfully minimizing error on the training data, the test accuracy (red line) rapidly stalled after initial gains. Accuracy quickly rose to approximately 44% but then severely plateaued, fluctuating around the 48%-50% mark and reaching a final value of approximately 50%. This significant divergence shows overfitting of the model.

Problem 3 Part b

This experiment extends the previous implementation by converting the shallow Multilayer Perceptron (MLP) into a deeper network with three hidden layers, trained over 300 epochs on the CIFAR-10 dataset. The objective was to determine if increased model capacity (depth and total parameters) could overcome the fundamental limitations of using an MLP for image classification.

```
# Build MLP with 3 hidden layers (512 -> 256 -> 128)
model = Sequential([
    Input(shape=(32, 32, 3)),  # Input layer
    Flatten(),                 # Flatten images
    Dense(512, activation='relu'),
    Dense(256, activation='relu'),   # 2nd hidden layer
    Dense(128, activation='relu'),   # 3rd hidden layer
    Dense(10, activation='softmax')  # 10 output classes
])
```

```
model.compile(
    optimizer=Adam(1e-3),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

The training was conducted for 300 epochs. The time per epoch remained highly consistent, typically fluctuating between 17 seconds and 24 seconds, with an average time of approximately 20 seconds per epoch.

```
# Training loop for 300 epochs with tracking
epochs = 300
batch_size = 128

train_losses = []
test_accuracies = []
epoch_times = []

for epoch in range(epochs):
    start_time = time.time()

    history = model.fit(x_train, y_train,
                        batch_size=batch_size,
                        epochs=1,
                        verbose=0)  # silent training

    # Record training loss
    train_loss = history.history['loss'][0]
    train_losses.append(train_loss)

    # Evaluate on test set
    test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
    test_accuracies.append(test_acc)

    # Record epoch time
    elapsed = time.time() - start_time
    epoch_times.append(elapsed)
```

```
        # Print every 10 epochs to reduce output
        if (epoch+1) % 10 == 0 or epoch == 0:
            print(f"Epoch {epoch+1}/{epochs} - Time: {elapsed:.2f}s - Train Loss: {

    # Final results after 300 epochs
    print(f"\nFinal Epoch ({epochs}): Train Loss = {train_losses[-1]:.4f}, Test Acc
```

```
Epoch 1/300 - Time: 22.05s - Train Loss: 0.1443 - Test Accuracy: 0.4836
Epoch 10/300 - Time: 16.47s - Train Loss: 0.1252 - Test Accuracy: 0.4693
Epoch 20/300 - Time: 18.31s - Train Loss: 0.1481 - Test Accuracy: 0.4754
Epoch 30/300 - Time: 17.90s - Train Loss: 0.1475 - Test Accuracy: 0.4788
Epoch 40/300 - Time: 19.76s - Train Loss: 0.1794 - Test Accuracy: 0.4800
Epoch 50/300 - Time: 17.85s - Train Loss: 0.1260 - Test Accuracy: 0.4783
Epoch 60/300 - Time: 26.82s - Train Loss: 0.1487 - Test Accuracy: 0.4785
Epoch 70/300 - Time: 17.69s - Train Loss: 0.1446 - Test Accuracy: 0.4863
Epoch 80/300 - Time: 23.37s - Train Loss: 0.1461 - Test Accuracy: 0.4762
Epoch 90/300 - Time: 19.44s - Train Loss: 0.1232 - Test Accuracy: 0.4813
Epoch 100/300 - Time: 20.02s - Train Loss: 0.1625 - Test Accuracy: 0.4680
Epoch 110/300 - Time: 23.58s - Train Loss: 0.0981 - Test Accuracy: 0.4725
```

The model was evaluated after completing all 300 epochs of training. The training log indicates that the deeper network likely experienced significant overfitting. This can be seen by looking at the difference between the training loss and the test accuracy. The training loss (blue line) decreased very well and consistently throughout all 300 epochs, reaching a very low final value of about 0.15. This suggests the network got very good at fitting the training data. However, the test accuracy (red line) quickly rose to around 0.52 in the first 50 epochs but then mostly stopped improving and began to fluctuate slightly around the 50% mark. The growing separation between the very low training loss and the stagnant test accuracy is a common sign of overfitting: the model is effectively memorizing the training set without improving its ability to generalize to new images.

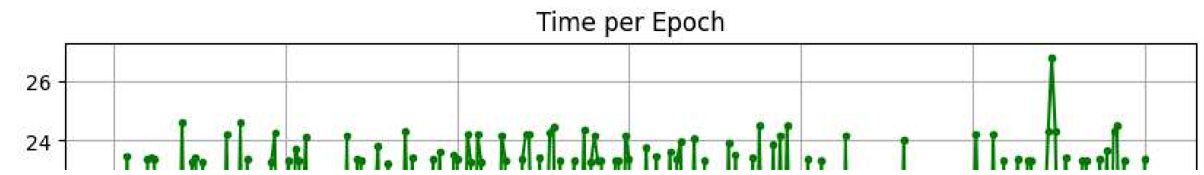
```
# Plot training loss and test accuracy
fig, ax1 = plt.subplots(figsize=(12,5))

color = 'tab:blue'
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Train Loss', color=color)
ax1.plot(range(1, epochs+1), train_losses, color=color, label='Train Loss')
ax1.tick_params(axis='y', labelcolor=color)
ax1.grid(True)

ax2 = ax1.twinx()
color = 'tab:red'
ax2.set_ylabel('Test Accuracy', color=color)
ax2.plot(range(1, epochs+1), test_accuracies, color=color, label='Test Accuracy
ax2.tick_params(axis='y', labelcolor=color)

plt.title('CIFAR-10 MLP with 3 Hidden Layers: Train Loss & Test Accuracy')
fig.tight_layout()
plt.show()
```

```
# plot epoch times
plt.figure(figsize=(10,3))
plt.plot(range(1, epochs+1), epoch_times, color='green', marker='.', label='Epc
plt.xlabel('Epoch')
plt.ylabel('Time (s)')
plt.title('Time per Epoch')
plt.grid(True)
plt.show()
```


Time per Epoch