```
# TensorFlow and tf.keras
import tensorflow as tf

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt
import time

print(f"TensorFlow version: {tf.__version__}")

# Load the Fashion MNIST dataset
fashion_mnist = tf.keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_dat

# Define class names for visualization
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

print(f"Training data shape: {train_images.shape}")
print(f"Number of training labels: {len(train_labels)}")
print(f"Test data shape: {test_images.shape}")
print(f"Number of test labels: {len(test_labels)}")

# Normalize pixel values to be between 0 and 1
train_images_normalized = train_images / 255.0
test_images_normalized = test_images / 255.0

# Define function to display model performance
def evaluate_model(model, test_images, test_labels, experiment_name):
    start_time = time.time()

    history = model.fit(
        train_images_reshaped,
        train_labels,
        epochs=10,
        validation_data=(test_images_reshaped, test_labels),
        verbose=1
    )

    end_time = time.time()
    training_time = end_time - start_time

    test_loss, test_acc = model.evaluate(test_images_reshaped, test_labels, verbo

    # Create a probability model
    probability_model = tf.keras.Sequential([model, tf.keras.layers.Softmax()])
    predictions = probability_model.predict(test_images_reshaped)
```

```
    # Plot training history
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.title(f'{experiment_name} - Accuracy')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title(f'{experiment_name} - Loss')
    plt.legend()
    plt.tight_layout()
    plt.show()

    # Plot confusion matrix
    cm = tf.math.confusion_matrix(test_labels, np.argmax(predictions, axis=1))
    plt.figure(figsize=(10, 8))
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title(f'Confusion Matrix - {experiment_name}')
    plt.colorbar()
    tick_marks = np.arange(len(class_names))
    plt.xticks(tick_marks, class_names, rotation=45)
    plt.yticks(tick_marks, class_names)
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.tight_layout()
    plt.show()

    # Display sample predictions
    num_rows = 5
    num_cols = 3
    num_images = num_rows * num_cols
    plt.figure(figsize=(2*2*num_cols, 2*num_rows))
    for i in range(num_images):
        plt.subplot(num_rows, 2*num_cols, 2*i+1)
        plot_image(i, predictions[i], test_labels, test_images_normalized)
        plt.subplot(num_rows, 2*num_cols, 2*i+2)
        plot_value_array(i, predictions[i], test_labels)
    plt.tight_layout()
    plt.suptitle(f'Sample Predictions - {experiment_name}', y=1.02)
    plt.show()

    return {
        "experiment": experiment_name,
```

```
        "accuracy": test_acc,
        "loss": test_loss,
        "training_time": training_time,
        "history": history.history
    }

# Define visualization functions
def plot_image(i, predictions_array, true_label, img):
    true_label, img = true_label[i], img[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])

    plt.imshow(img, cmap=plt.cm.binary)

    predicted_label = np.argmax(predictions_array)
    if predicted_label == true_label:
        color = 'blue'
    else:
        color = 'red'

    plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                    100*np.max(predictions_array),
                                    class_names[true_label]),
                                    color=color)

def plot_value_array(i, predictions_array, true_label):
    true_label = true_label[i]
    plt.grid(False)
    plt.xticks(range(10))
    plt.yticks([])
    thisplot = plt.bar(range(10), predictions_array, color="#777777")
    plt.ylim([0, 1])
    predicted_label = np.argmax(predictions_array)

    thisplot[predicted_label].set_color('red')
    thisplot[true_label].set_color('blue')

# Display sample images before preprocessing
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(train_images_normalized[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.suptitle('Sample Training Images', y=0.92)
plt.show()
```

```
# Reshape data for CNN input (adding channel dimension)
train_images_reshaped = train_images_normalized.reshape(
    train_images_normalized.shape[0], 28, 28, 1)
test_images_reshaped = test_images_normalized.reshape(
    test_images_normalized.shape[0], 28, 28, 1)

print(f"Training data shape after reshaping: {train_images_reshaped.shape}")
print(f"Test data shape after reshaping: {test_images_reshaped.shape}")

# --------------------------------
# EXPERIMENT 1: Basic CNN Model
# --------------------------------
print("\n========== EXPERIMENT 1: Basic CNN Model ==========")

model1 = tf.keras.Sequential([
    # Input layer
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)
    tf.keras.layers.MaxPooling2D((2, 2)),

    # Flatten layer
    tf.keras.layers.Flatten(),

    # Dense layers
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)  # Output layer (10 classes)
])

# Compile model
model1.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy']
)

# Display model summary
model1.summary()

# Evaluate first model
results1 = evaluate_model(model1, test_images_reshaped, test_labels, "Experiment

# --------------------------------
# EXPERIMENT 2: Deeper CNN with Dropout
# --------------------------------
print("\n========== EXPERIMENT 2: Deeper CNN with Dropout ==========")

model2 = tf.keras.Sequential([
    # First convolutional layer
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_s
    tf.keras.layers.BatchNormalization(),
```

```
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Dropout(0.25),

    # Second convolutional layer
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Dropout(0.25),

    # Flatten layer
    tf.keras.layers.Flatten(),

    # Dense layers
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10)  # Output layer (10 classes)
])

# Compile model with learning rate scheduler
initial_learning_rate = 0.001
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate, decay_steps=10000, decay_rate=0.9, staircase=True)

model2.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=lr_schedule),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy']
)

# Display model summary
model2.summary()

# Evaluate second model
results2 = evaluate_model(model2, test_images_reshaped, test_labels, "Experiment

# --------------------------------
# EXPERIMENT 3: Data Augmentation
# --------------------------------
print("\n========== EXPERIMENT 3: Data Augmentation ==========")

# Create data augmentation layer
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomRotation(0.1),
    tf.keras.layers.RandomZoom(0.1),
```

```
    tf.keras.layers.RandomTranslation(0.1, 0.1)
])

# Display examples of augmented images
plt.figure(figsize=(10, 10))
for i in range(9):
    augmented_image = data_augmentation(train_images_reshaped[i:i+1])
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_image[0, :, :, 0], cmap='gray')
    plt.title(class_names[train_labels[i]])
    plt.axis('off')
plt.suptitle('Examples of Augmented Images', y=0.92)
plt.show()

# Define model with data augmentation
model3 = tf.keras.Sequential([
    # Data augmentation layer (only applied during training)
    data_augmentation,

    # First convolutional layer
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_s
    tf.keras.layers.MaxPooling2D((2, 2)),

    # Second convolutional layer
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.MaxPooling2D((2, 2)),

    # Third convolutional layer
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.MaxPooling2D((2, 2)),

    # Flatten layer
    tf.keras.layers.Flatten(),

    # Dense layers
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10)  # Output layer (10 classes)
])

# Compile model
model3.compile(
    optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.001),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy']
)

# Display model summary
model3.summary()
```

```
# Evaluate third model
results3 = evaluate_model(model3, test_images_reshaped, test_labels, "Experiment

# --------------------------------
# RESULTS COMPARISON
# --------------------------------
print("\n========== RESULTS COMPARISON ==========")

# Prepare results for comparison
results = [results1, results2, results3]
experiment_names = [r["experiment"] for r in results]
accuracies = [r["accuracy"] for r in results]
losses = [r["loss"] for r in results]
training_times = [r["training_time"] for r in results]

# Display comparison in table format
from tabulate import tabulate

table_data = []
for i, r in enumerate(results):
    table_data.append([
        r["experiment"],
        f"{r['accuracy']:.4f}",
        f"{r['loss']:.4f}",
        f"{r['training_time']:.2f}s"
    ])

print(tabulate(table_data,
               headers=["Experiment", "Accuracy", "Loss", "Training Time"],
               tablefmt="grid"))

# Plot comparison
plt.figure(figsize=(15, 5))

# Accuracy comparison
plt.subplot(1, 3, 1)
plt.bar(experiment_names, accuracies, color='skyblue')
plt.title('Accuracy Comparison')
plt.ylabel('Accuracy')
plt.ylim(0.8, 1.0)  # Adjust as needed
plt.xticks(rotation=45)

# Loss comparison
plt.subplot(1, 3, 2)
plt.bar(experiment_names, losses, color='salmon')
plt.title('Loss Comparison')
plt.ylabel('Loss')
plt.xticks(rotation=45)

# Training time comparison
```

```
# Training time comparison
plt.subplot(1, 3, 3)
plt.bar(experiment_names, training_times, color='lightgreen')
plt.title('Training Time Comparison')
plt.ylabel('Time (seconds)')
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()

print("\nExperiment complete. Please refer to the visualizations and comparison t
```

```
•••  TensorFlow version: 2.18.0
     Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dat
     29515/29515 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
     Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dat
     26421880/26421880 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
     Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dat
     5148/5148 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
     Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dat
     4422102/4422102 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
     Training data shape: (60000, 28, 28)
     Number of training labels: 60000
     Test data shape: (10000, 28, 28)
     Number of test labels: 10000
```


```
     Training data shape after reshaping: (60000, 28, 28, 1)
     Test data shape after reshaping: (10000, 28, 28, 1)

     ========== EXPERIMENT 1: Basic CNN Model ==========
     /usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_
```