```python
import tensorflow as tf
from tensorflow.keras import layers, datasets, models, callbacks
import matplotlib.pyplot as plt

def load_and_preprocess_data():
    """Load and preprocess MNIST data with normalization."""
    (train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data()
    train_images = train_images.reshape((train_images.shape[0], 28, 28, 1)).astype('float32') / 255
    test_images = test_images.reshape((test_images.shape[0], 28, 28, 1)).astype('float32') / 255
    return train_images, train_labels, test_images, test_labels

def build_model():
    """Build a CNN model for MNIST classification with batch normalization."""
    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.BatchNormalization(),
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dropout(0.5),  # Add dropout to prevent overfitting
        layers.Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

def plot_training_history(history):
```

```
    """Plot accuracy and loss graphs for training history."""
    # Plot accuracy
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train')
    plt.plot(history.history['val_accuracy'], label='Test')
    plt.title(f'Model accuracy (final test accuracy: {history.history["val_accuracy"][-1]:.2f})')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend(loc='upper left')

    # Plot loss
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train')
    plt.plot(history.history['val_loss'], label='Test')
    plt.title(f'Model loss (final test loss: {history.history["val_loss"][-1]:.2f})')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend(loc='upper left')

    plt.tight_layout()
    plt.show()

def main():
    # Load and preprocess data
    train_images, train_labels, test_images, test_labels = load_and_preprocess_data()

    # Data augmentation
    data_gen = tf.keras.preprocessing.image.ImageDataGenerator(
        rotation_range=10,
        zoom_range=0.1,
```

```
        width_shift_range=0.1,
        height_shift_range=0.1
    )

    # Build model
    model = build_model()

    # Define callbacks
    early_stopping = callbacks.EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)
    reduce_lr = callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2,
min_lr=0.00001)

    # Train model
    history = model.fit(
        data_gen.flow(train_images, train_labels, batch_size=64),
        epochs=20,
        validation_data=(test_images, test_labels),
        callbacks=[early_stopping, reduce_lr]
    )

    # Evaluate model
    test_loss, test_acc = model.evaluate(test_images, test_labels)
    print(f"Test accuracy: {test_acc}")

    # Plot training history
    plot_training_history(history)

    # Save model weights
    model.save('mnist_cnn_model.h5')
    print("Model saved as 'mnist_cnn_model.h5'")
```

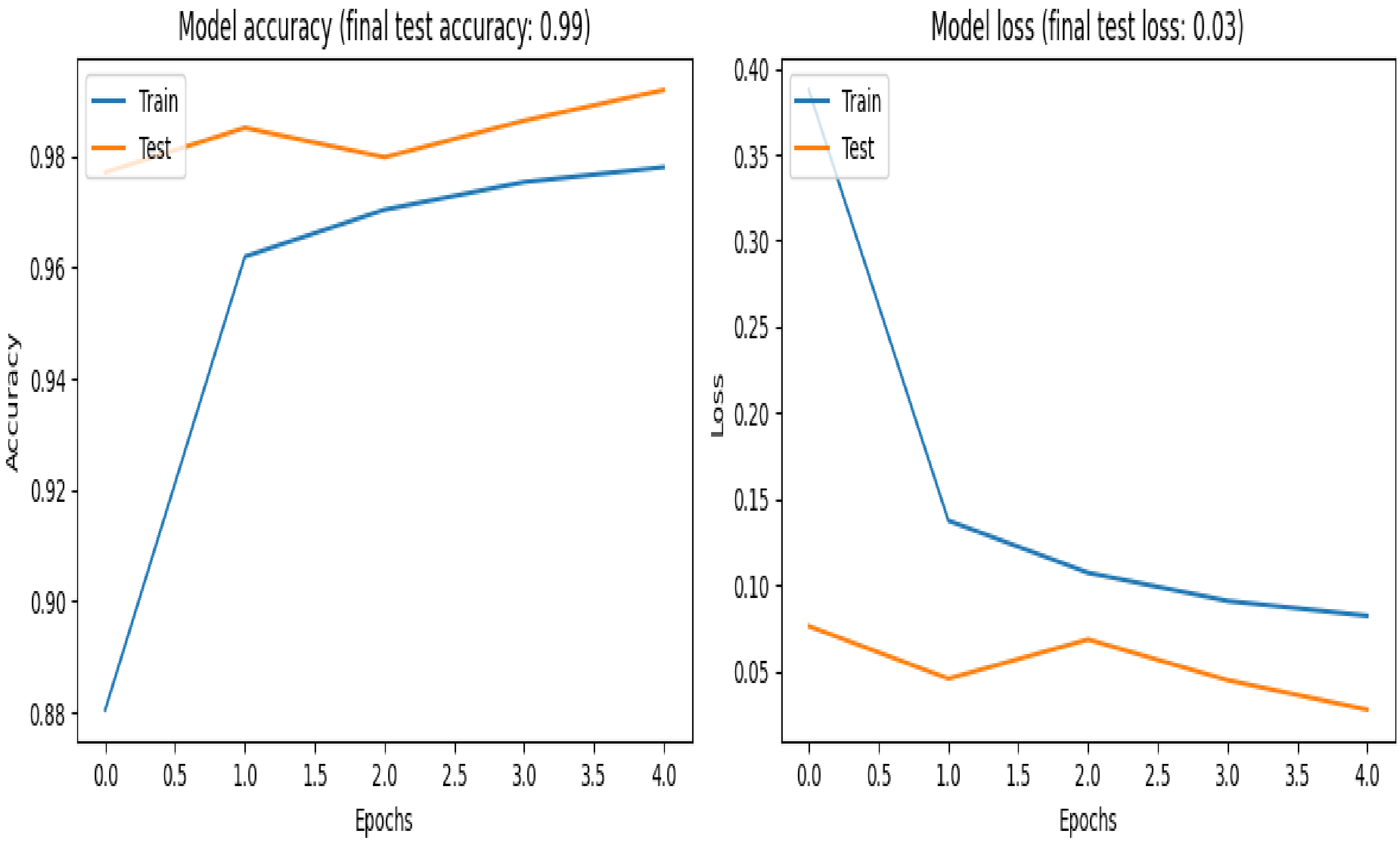
```
if __name__ == "__main__":
    main()
```

- **Data Augmentation:**
    - Defined a `data_gen` generator using `ImageDataGenerator` with parameters to rotate, zoom, and shift images. This improves generalization by adding variation to the training images.
- **Early Stopping:**
    - Defined an `EarlyStopping` callback with `patience=3` to stop training if the validation loss does not improve for 3 consecutive epochs. This helps prevent overfitting.
- **Reduce Learning Rate on Plateau:**
    - Used `ReduceLROnPlateau` callback to halve the learning rate if the validation loss plateaus for 2 epochs. This helps the model converge to a better minimum.
- **Batch Normalization:**
    - Added `BatchNormalization` layers after each convolution layer to improve training stability and convergence speed.

**Output:**

```
Epoch 1/5
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
938/938 ━━━━━━━━━━━━━━━━━━━━ 102s 105ms/step - accuracy: 0.7656 - loss: 0.7411 - val_accuracy: 0.9771 - val_loss: 0.0761 - learning_rate: 0.0010
Epoch 2/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 94s 100ms/step - accuracy: 0.9574 - loss: 0.1515 - val_accuracy: 0.9851 - val_loss: 0.0456 - learning_rate: 0.0010
Epoch 3/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 99s 105ms/step - accuracy: 0.9687 - loss: 0.1134 - val_accuracy: 0.9798 - val_loss: 0.0683 - learning_rate: 0.0010
Epoch 4/5
```

```
938/938 ━━━━━━━━━━━━━━━━━━━━ 95s 101ms/step - accuracy:
0.9752 - loss: 0.0923 - val_accuracy: 0.9864 - val_loss: 0.0447 -
learning_rate: 0.0010
Epoch 5/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 142s 101ms/step - accuracy:
0.9766 - loss: 0.0833 - val_accuracy: 0.9919 - val_loss: 0.0275 -
learning_rate: 0.0010
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 13ms/step - accuracy: 0.9909
- loss: 0.0340
Test accuracy: 0.9919000267982483
```



```
import tensorflow as tf
from tensorflow.keras import layers, models, optimizers
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt


def create_data(seed=42):
    """Create standardized random data for model training."""
    np.random.seed(seed)
    X = np.random.randn(1000, 10)
    y = np.random.randn(1000, 1)
    return X, y


def create_model():
    """Build a simple feedforward neural network model with an Input layer."""
    model = models.Sequential([
        layers.Input(shape=(10,)),
        layers.Dense(50, activation="relu"),
        layers.Dense(20, activation="relu"),
        layers.Dense(1)
    ])
    return model


def train_model_with_history(model, optimizer, X_train, y_train, X_val, y_val, batch_size, epochs,
optimizer_name, learning_rates):
    """Train the model with dynamically changing learning rates and track MSE and MAE for both train
and validation."""

    # Initialize dictionary to store history
    history = {
        "loss": [],
        "val_loss": [],
```

```
        "mae": [],
        "val_mae": []
    }

    # Loop over epochs with dynamic learning rate updates
    for epoch in range(epochs):
        # Update learning rate dynamically, ensuring it is a float
        lr = float(learning_rates[epoch % len(learning_rates)])
        optimizer.learning_rate.assign(lr)

        # Compile model with the updated optimizer
        model.compile(optimizer=optimizer, loss="mse", metrics=["mae"])

        # Train the model for one epoch and capture history
        hist = model.fit(X_train, y_train, batch_size=batch_size, epochs=1, verbose=0, validation_data=(X_val, y_val))

        # Append epoch metrics to history dictionary
        history["loss"].append(hist.history["loss"][0])
        history["val_loss"].append(hist.history["val_loss"][0])
        history["mae"].append(hist.history["mae"][0])
        history["val_mae"].append(hist.history["val_mae"][0])

        # Print learning rate and losses for the epoch
        print(f"Epoch {epoch+1}/{epochs} - {optimizer_name} | LR: {optimizer.learning_rate.numpy():.6f} | Loss: {history['loss'][-1]:.4f} | Val Loss: {history['val_loss'][-1]:.4f}")

    return history

def plot_metric_history(history, metric="loss", metric_label="MSE"):
    """Plot training and validation metric history."""
```

```python
    plt.plot(history[metric], label=f'Train {metric_label}', color='blue')
    plt.plot(history[f'val_{metric}'], label=f'Validation {metric_label}', linestyle='dashed', color='orange')

    plt.xlabel("Epochs")
    plt.ylabel(metric_label)
    plt.title(f'Train vs Validation {metric_label}')
    plt.legend()
    plt.grid(True)
    plt.show()

# Create data and split into training and validation sets
X, y = create_data()
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize model
model = create_model()

# Define optimizer with an initial learning rate
optimizer = optimizers.Adam(learning_rate=0.01)

# Define learning rates to cycle through
learning_rates = [0.01, 0.005, 0.001]

# Define training parameters
epochs = 30
batch_size = 32

# Train the model
print("\nTraining with Dynamic Learning Rates:")
history = train_model_with_history(model, optimizer, X_train, y_train, X_val, y_val, batch_size, epochs, "Adam", learning_rates)
```

# Plot MSE and MAE for both train and validation

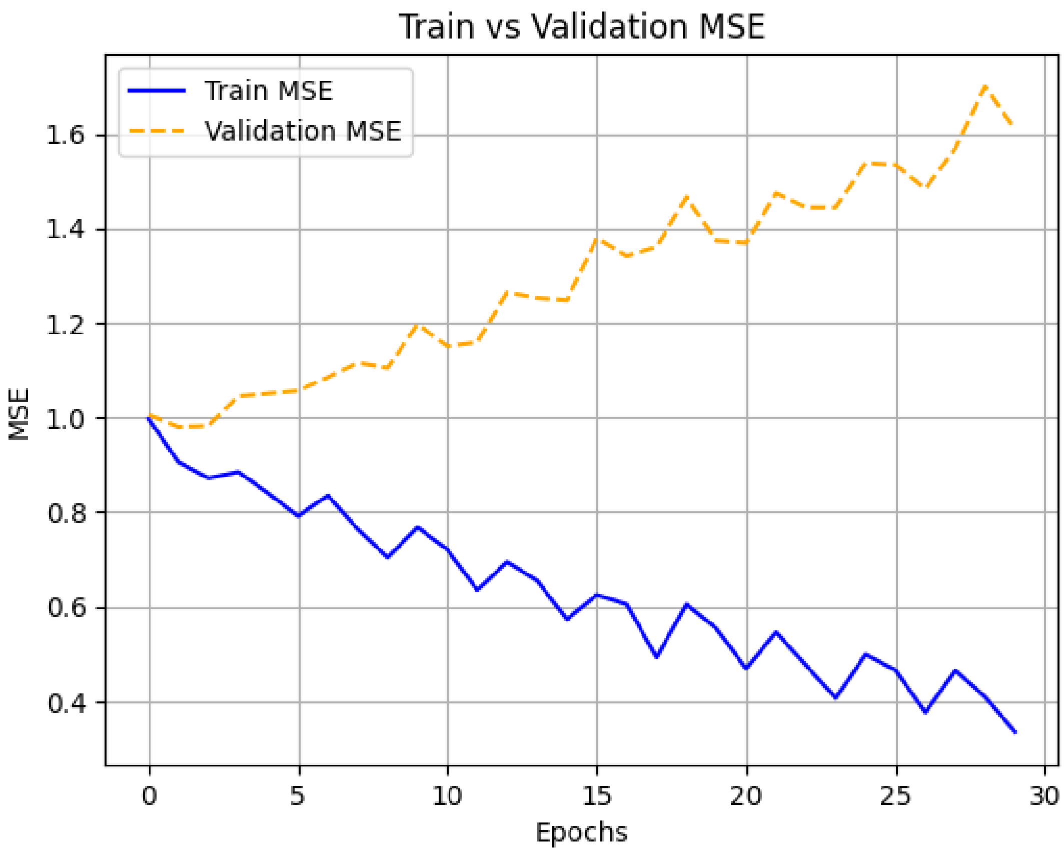
plot_metric_history(history, metric="loss", metric_label="MSE")

plot_metric_history(history, metric="mae", metric_label="MAE")

- **Variable Learning Rates:** We define a list `learning_rates` with different rates. Each epoch cycles through these rates by updating the optimizer's learning rate.
- **Track Both Training and Validation Loss:** We split the data into training and validation sets to track `loss` and `val_loss` to monitor both overfitting and generalization.
- **Add MAE as an Additional Metric:** MAE is tracked as an additional metric by adding it to the model's metrics list during compilation, and both `mae` and `val_mae` values are stored in `history` for plotting.

**Output:**

```
Training with Dynamic Learning Rates:
Epoch 1/30 - Adam | LR: 0.010000 | Loss: 0.9969 | Val Loss: 1.0054
Epoch 2/30 - Adam | LR: 0.005000 | Loss: 0.9051 | Val Loss: 0.9803
Epoch 3/30 - Adam | LR: 0.001000 | Loss: 0.8715 | Val Loss: 0.9827
Epoch 4/30 - Adam | LR: 0.010000 | Loss: 0.8847 | Val Loss: 1.0452
Epoch 5/30 - Adam | LR: 0.005000 | Loss: 0.8396 | Val Loss: 1.0507
Epoch 6/30 - Adam | LR: 0.001000 | Loss: 0.7915 | Val Loss: 1.0566
Epoch 7/30 - Adam | LR: 0.010000 | Loss: 0.8349 | Val Loss: 1.0849
Epoch 8/30 - Adam | LR: 0.005000 | Loss: 0.7637 | Val Loss: 1.1152
Epoch 9/30 - Adam | LR: 0.001000 | Loss: 0.7037 | Val Loss: 1.1048
Epoch 10/30 - Adam | LR: 0.010000 | Loss: 0.7674 | Val Loss: 1.1969
Epoch 11/30 - Adam | LR: 0.005000 | Loss: 0.7203 | Val Loss: 1.1504
Epoch 12/30 - Adam | LR: 0.001000 | Loss: 0.6347 | Val Loss: 1.1591
Epoch 13/30 - Adam | LR: 0.010000 | Loss: 0.6944 | Val Loss: 1.2639
Epoch 14/30 - Adam | LR: 0.005000 | Loss: 0.6551 | Val Loss: 1.2525
Epoch 15/30 - Adam | LR: 0.001000 | Loss: 0.5730 | Val Loss: 1.2479
Epoch 16/30 - Adam | LR: 0.010000 | Loss: 0.6244 | Val Loss: 1.3787
Epoch 17/30 - Adam | LR: 0.005000 | Loss: 0.6049 | Val Loss: 1.3416
Epoch 18/30 - Adam | LR: 0.001000 | Loss: 0.4927 | Val Loss: 1.3605
Epoch 19/30 - Adam | LR: 0.010000 | Loss: 0.6048 | Val Loss: 1.4661
Epoch 20/30 - Adam | LR: 0.005000 | Loss: 0.5539 | Val Loss: 1.3734
Epoch 21/30 - Adam | LR: 0.001000 | Loss: 0.4687 | Val Loss: 1.3688
Epoch 22/30 - Adam | LR: 0.010000 | Loss: 0.5460 | Val Loss: 1.4739
Epoch 23/30 - Adam | LR: 0.005000 | Loss: 0.4758 | Val Loss: 1.4444
Epoch 24/30 - Adam | LR: 0.001000 | Loss: 0.4064 | Val Loss: 1.4443
Epoch 25/30 - Adam | LR: 0.010000 | Loss: 0.4988 | Val Loss: 1.5380
Epoch 26/30 - Adam | LR: 0.005000 | Loss: 0.4653 | Val Loss: 1.5339
Epoch 27/30 - Adam | LR: 0.001000 | Loss: 0.3762 | Val Loss: 1.4840
Epoch 28/30 - Adam | LR: 0.010000 | Loss: 0.4651 | Val Loss: 1.5686
Epoch 29/30 - Adam | LR: 0.005000 | Loss: 0.4089 | Val Loss: 1.6999
Epoch 30/30 - Adam | LR: 0.001000 | Loss: 0.3349 | Val Loss: 1.6090
```


Train vs Validation MSE
Train MSE
Validation MSE
MSE
Epochs
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0
5
10
15
20
25
30