

```

import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import datasets, layers, models, optimizers
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import confusion_matrix, classification_report
from tensorflow.keras.callbacks import EarlyStopping
import pandas as pd

# Set random seed
np.random.seed(42)
tf.random.set_seed(42)

# Load dataset (MNIST)
(train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data()

# Preprocess data
train_images = train_images.astype('float32') / 255.0
test_images = test_images.astype('float32') / 255.0
train_images = train_images.reshape(-1, 28, 28, 1)
test_images = test_images.reshape(-1, 28, 28, 1)
train_labels = to_categorical(train_labels, 10)
test_labels = to_categorical(test_labels, 10)

# Split for validation
from sklearn.model_selection import train_test_split
train_images, val_images, train_labels, val_labels = train_test_split(
    train_images, train_labels, test_size=0.2, random_state=42
)

# CNN model builder function
def create_model(filters=[32, 64], kernel_size=3, learning_rate=0.001,
                 dropout=0.3, optimizer='adam'):
    model = models.Sequential([
        # First conv block
        layers.Conv2D(filters[0], kernel_size, activation='relu', padding='same',
                      input_shape=(28, 28, 1)),
        layers.MaxPooling2D(2),

        # Second conv block
        layers.Conv2D(filters[1], kernel_size, activation='relu', padding='same'),
        layers.MaxPooling2D(2),

        # Classifier
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(dropout),
        layers.Dense(10, activation='softmax')
    ])

    # Configure optimizer
    if optimizer == 'adam':
        opt = optimizers.Adam(learning_rate=learning_rate)
    elif optimizer == 'sgd':
        opt = optimizers.SGD(learning_rate=learning_rate)
    else:
        opt = optimizers.RMSprop(learning_rate=learning_rate)

    model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Define hyperparameter configurations
configs = [
    {'name': 'baseline', 'filters': [32, 64], 'kernel_size': 3,
     'learning_rate': 0.001, 'dropout': 0.3, 'optimizer': 'adam'},
    {'name': 'higher_filters', 'filters': [64, 128], 'kernel_size': 3,
     'learning_rate': 0.001, 'dropout': 0.3, 'optimizer': 'adam'},
    {'name': 'larger_kernel', 'filters': [32, 64], 'kernel_size': 5,
     'learning_rate': 0.001, 'dropout': 0.3, 'optimizer': 'adam'},
    {'name': 'lower_lr', 'filters': [32, 64], 'kernel_size': 3,
     'learning_rate': 0.0001, 'dropout': 0.3, 'optimizer': 'adam'},
    {'name': 'higher_dropout', 'filters': [32, 64], 'kernel_size': 3,
     'learning_rate': 0.001, 'dropout': 0.5, 'optimizer': 'adam'},
    {'name': 'rmsprop', 'filters': [32, 64], 'kernel_size': 3,
     'learning_rate': 0.001, 'dropout': 0.3, 'optimizer': 'rmsprop'},
]

# Set up for storing results
results = []
histories = {}
best_acc = 0

```

```

best_model = None

# Run experiments
print("Running hyperparameter optimization...")
for i, config in enumerate(configs):
    print(f"\nExperiment {i+1}/{len(configs)}: {config['name']}")

    # Build model with current config
    model = create_model(
        filters=config['filters'],
        kernel_size=config['kernel_size'],
        learning_rate=config['learning_rate'],
        dropout=config['dropout'],
        optimizer=config['optimizer']
    )

    # Train model
    early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
    history = model.fit(
        train_images, train_labels,
        validation_data=(val_images, val_labels),
        epochs=10, batch_size=128,
        callbacks=[early_stop],
        verbose=1
    )

    # Evaluate model
    _, test_acc = model.evaluate(test_images, test_labels, verbose=0)
    print(f"Test accuracy: {test_acc:.4f}")

    # Store results
    results.append({
        'config': config['name'],
        'test_accuracy': test_acc,
        'parameters': config
    })
    histories[config['name']] = history.history

    # Keep track of best model
    if test_acc > best_acc:
        best_acc = test_acc
        best_model = model

    # Generate confusion matrix for best model
    y_pred = np.argmax(model.predict(test_images), axis=1)
    y_true = np.argmax(test_labels, axis=1)
    cm = confusion_matrix(y_true, y_pred)

    # Save confusion matrix plot
    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap='Blues')
    plt.title(f'Confusion Matrix - {config["name"]}')
    plt.colorbar()
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.savefig(f'best_cm_{config["name"]}.png')
    plt.close()

# Convert results to DataFrame and sort
results_df = pd.DataFrame(results)
results_df = results_df.sort_values('test_accuracy', ascending=False)

# Display and save results
print("\nResults summary (ranked by test accuracy):")
print(results_df[['config', 'test_accuracy']])
results_df.to_csv('hyperparameter_results.csv', index=False)

# Plot accuracy comparison
plt.figure(figsize=(10, 6))
plt.bar(results_df['config'], results_df['test_accuracy'], color='skyblue')
plt.axhline(y=np.mean(results_df['test_accuracy']), color='red',
            linestyle='--', label='Average')
plt.title('Test Accuracy by Configuration')
plt.xlabel('Configuration')
plt.ylabel('Accuracy')
plt.ylim(0.96, 1.0) # Adjust for better visualization
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()
plt.savefig('accuracy_comparison.png')

# Plot learning curves for best model
best_config = results_df.iloc[0]['config']


```

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

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

plt.tight_layout()
plt.savefig('best_model_learning_curves.png')

print(f"\nBest configuration: {best_config} with accuracy: {best_acc:.4f}")
print("Optimization complete. Results saved to CSV and plots generated.")
```

 Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>
 11490434/11490434 0s 0us/step
 Running hyperparameter optimization...

Experiment 1/6: baseline
 /usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/
 super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Epoch 1/10
 375/375 82s 206ms/step - accuracy: 0.8239 - loss: 0.5732 - val_accuracy: 0.9783 - val_loss: 0.0697
 Epoch 2/10
 375/375 61s 151ms/step - accuracy: 0.9726 - loss: 0.0859 - val_accuracy: 0.9856 - val_loss: 0.0473
 Epoch 3/10
 375/375 82s 152ms/step - accuracy: 0.9821 - loss: 0.0590 - val_accuracy: 0.9864 - val_loss: 0.0456
 Epoch 4/10
 375/375 80s 147ms/step - accuracy: 0.9862 - loss: 0.0451 - val_accuracy: 0.9887 - val_loss: 0.0390
 Epoch 5/10
 375/375 56s 150ms/step - accuracy: 0.9891 - loss: 0.0344 - val_accuracy: 0.9891 - val_loss: 0.0413
 Epoch 6/10
 375/375 80s 146ms/step - accuracy: 0.9906 - loss: 0.0303 - val_accuracy: 0.9893 - val_loss: 0.0409
 Epoch 7/10
 375/375 87s 160ms/step - accuracy: 0.9922 - loss: 0.0247 - val_accuracy: 0.9887 - val_loss: 0.0440
 Test accuracy: 0.9893
 313/313 4s 13ms/step

Experiment 2/6: higher_filters
 /usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/
 super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Epoch 1/10
 375/375 153s 403ms/step - accuracy: 0.8498 - loss: 0.4918 - val_accuracy: 0.9804 - val_loss: 0.0628
 Epoch 2/10
 375/375 200s 398ms/step - accuracy: 0.9783 - loss: 0.0727 - val_accuracy: 0.9871 - val_loss: 0.0442
 Epoch 3/10
 375/375 149s 398ms/step - accuracy: 0.9849 - loss: 0.0490 - val_accuracy: 0.9881 - val_loss: 0.0396
 Epoch 4/10
 375/375 203s 400ms/step - accuracy: 0.9883 - loss: 0.0378 - val_accuracy: 0.9898 - val_loss: 0.0369
 Epoch 5/10
 375/375 200s 395ms/step - accuracy: 0.9920 - loss: 0.0262 - val_accuracy: 0.9892 - val_loss: 0.0367
 Epoch 6/10
 375/375 203s 397ms/step - accuracy: 0.9923 - loss: 0.0241 - val_accuracy: 0.9885 - val_loss: 0.0464
 Epoch 7/10
 375/375 202s 397ms/step - accuracy: 0.9936 - loss: 0.0206 - val_accuracy: 0.9883 - val_loss: 0.0475
 Epoch 8/10
 375/375 202s 396ms/step - accuracy: 0.9950 - loss: 0.0160 - val_accuracy: 0.9908 - val_loss: 0.0408
 Test accuracy: 0.9916
 313/313 9s 28ms/step

Experiment 3/6: larger_kernel
 /usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/
 super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Epoch 1/10
 375/375 120s 317ms/step - accuracy: 0.8463 - loss: 0.5047 - val_accuracy: 0.9817 - val_loss: 0.0619
 Epoch 2/10
 375/375 139s 309ms/step - accuracy: 0.9758 - loss: 0.0786 - val_accuracy: 0.9828 - val_loss: 0.0588
 Epoch 3/10
 375/375 117s 313ms/step - accuracy: 0.9834 - loss: 0.0542 - val_accuracy: 0.9886 - val_loss: 0.0403
 Epoch 4/10
 375/375 145s 321ms/step - accuracy: 0.9876 - loss: 0.0394 - val_accuracy: 0.9885 - val_loss: 0.0418
 Epoch 5/10
 375/375 117s 313ms/step - accuracy: 0.9901 - loss: 0.0316 - val_accuracy: 0.9895 - val_loss: 0.0392
 Epoch 6/10
 375/375 117s 312ms/step - accuracy: 0.9919 - loss: 0.0259 - val_accuracy: 0.9899 - val_loss: 0.0404
 Epoch 7/10
 375/375 144s 318ms/step - accuracy: 0.9929 - loss: 0.0215 - val_accuracy: 0.9908 - val_loss: 0.0347
 Epoch 8/10
 375/375 139s 311ms/step - accuracy: 0.9946 - loss: 0.0170 - val_accuracy: 0.9918 - val_loss: 0.0382
 Epoch 9/10
 375/375 146s 321ms/step - accuracy: 0.9946 - loss: 0.0164 - val_accuracy: 0.9918 - val_loss: 0.0366
 Epoch 10/10
 375/375 118s 314ms/step - accuracy: 0.9947 - loss: 0.0155 - val_accuracy: 0.9914 - val_loss: 0.0427
 Test accuracy: 0.9926
 313/313 7s 21ms/step

Experiment 4/6: lower_lr
 /usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/
 super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Epoch 1/10
 375/375 63s 161ms/step - accuracy: 0.6221 - loss: 1.3767 - val_accuracy: 0.9288 - val_loss: 0.2588
 Epoch 2/10
 375/375 81s 159ms/step - accuracy: 0.9148 - loss: 0.2842 - val_accuracy: 0.9559 - val_loss: 0.1547
 Epoch 3/10
 375/375 81s 156ms/step - accuracy: 0.9474 - loss: 0.1813 - val_accuracy: 0.9698 - val_loss: 0.1113
 Epoch 4/10
 375/375 60s 160ms/step - accuracy: 0.9603 - loss: 0.1346 - val_accuracy: 0.9741 - val_loss: 0.0901
 Epoch 5/10
 375/375 60s 161ms/step - accuracy: 0.9686 - loss: 0.1081 - val_accuracy: 0.9782 - val_loss: 0.0756
 Epoch 6/10
 375/375 82s 160ms/step - accuracy: 0.9721 - loss: 0.0930 - val_accuracy: 0.9807 - val_loss: 0.0670
 Epoch 7/10
 375/375 83s 163ms/step - accuracy: 0.9756 - loss: 0.0826 - val_accuracy: 0.9818 - val_loss: 0.0622
 Epoch 8/10

```

375/375 ----- 81s 160ms/step - accuracy: 0.9772 - loss: 0.0759 - val_accuracy: 0.9827 - val_loss: 0.0581
Epoch 9/10
375/375 ----- 82s 160ms/step - accuracy: 0.9798 - loss: 0.0691 - val_accuracy: 0.9848 - val_loss: 0.0528
Epoch 10/10
375/375 ----- 82s 162ms/step - accuracy: 0.9821 - loss: 0.0608 - val_accuracy: 0.9854 - val_loss: 0.0515
Test accuracy: 0.9850

```

Experiment 5/6: higher_dropout

```

Epoch 1/10
375/375 ----- 63s 163ms/step - accuracy: 0.7928 - loss: 0.6522 - val_accuracy: 0.9747 - val_loss: 0.0804
Epoch 2/10
375/375 ----- 59s 158ms/step - accuracy: 0.9669 - loss: 0.1120 - val_accuracy: 0.9848 - val_loss: 0.0504
Epoch 3/10
375/375 ----- 81s 154ms/step - accuracy: 0.9776 - loss: 0.0775 - val_accuracy: 0.9863 - val_loss: 0.0433
Epoch 4/10
375/375 ----- 84s 159ms/step - accuracy: 0.9814 - loss: 0.0621 - val_accuracy: 0.9877 - val_loss: 0.0443
Epoch 5/10
375/375 ----- 58s 155ms/step - accuracy: 0.9842 - loss: 0.0517 - val_accuracy: 0.9893 - val_loss: 0.0378
Epoch 6/10
375/375 ----- 83s 158ms/step - accuracy: 0.9864 - loss: 0.0479 - val_accuracy: 0.9899 - val_loss: 0.0364
Epoch 7/10
375/375 ----- 81s 157ms/step - accuracy: 0.9879 - loss: 0.0407 - val_accuracy: 0.9905 - val_loss: 0.0369
Epoch 8/10
375/375 ----- 82s 158ms/step - accuracy: 0.9878 - loss: 0.0372 - val_accuracy: 0.9897 - val_loss: 0.0368
Epoch 9/10
375/375 ----- 83s 160ms/step - accuracy: 0.9897 - loss: 0.0342 - val_accuracy: 0.9910 - val_loss: 0.0374
Test accuracy: 0.9899

```

Experiment 6/6: rmsprop

```

Epoch 1/10
375/375 ----- 60s 157ms/step - accuracy: 0.8246 - loss: 0.5533 - val_accuracy: 0.9757 - val_loss: 0.0816
Epoch 2/10
375/375 ----- 82s 158ms/step - accuracy: 0.9758 - loss: 0.0806 - val_accuracy: 0.9847 - val_loss: 0.0513
Epoch 3/10
375/375 ----- 82s 159ms/step - accuracy: 0.9835 - loss: 0.0543 - val_accuracy: 0.9862 - val_loss: 0.0442
Epoch 4/10
375/375 ----- 83s 162ms/step - accuracy: 0.9870 - loss: 0.0422 - val_accuracy: 0.9872 - val_loss: 0.0435
Epoch 5/10
375/375 ----- 79s 154ms/step - accuracy: 0.9901 - loss: 0.0333 - val_accuracy: 0.9887 - val_loss: 0.0421
Epoch 6/10
375/375 ----- 82s 155ms/step - accuracy: 0.9923 - loss: 0.0270 - val_accuracy: 0.9892 - val_loss: 0.0407
Epoch 7/10
375/375 ----- 83s 160ms/step - accuracy: 0.9928 - loss: 0.0220 - val_accuracy: 0.9887 - val_loss: 0.0411
Epoch 8/10
375/375 ----- 60s 159ms/step - accuracy: 0.9941 - loss: 0.0207 - val_accuracy: 0.9903 - val_loss: 0.0377
Epoch 9/10
375/375 ----- 83s 163ms/step - accuracy: 0.9945 - loss: 0.0169 - val_accuracy: 0.9900 - val_loss: 0.0428
Epoch 10/10
375/375 ----- 79s 155ms/step - accuracy: 0.9952 - loss: 0.0149 - val_accuracy: 0.9903 - val_loss: 0.0442
Test accuracy: 0.9920

```

Results summary (ranked by test accuracy):

	config	test_accuracy
2	larger_kernel	0.9926
5	rmsprop	0.9920
1	higher_filters	0.9916
4	higher_dropout	0.9899
0	baseline	0.9893
3	lower_lr	0.9850

Best configuration: larger_kernel with accuracy: 0.9926

Optimization complete. Results saved to CSV and plots generated.

