```
#PROGRAM-3
import tensorflow as tf
from tensorflow.keras import layers, datasets, models, callbacks
import matplotlib.pyplot as plt

# Load and preprocess data
(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

# Data Augmentation
data_augmentation = tf.keras.Sequential([
    layers.RandomRotation(0.1),
    layers.RandomTranslation(0.1, 0.1),
    layers.RandomZoom(0.1),
])

# Build the model
model = models.Sequential()

model.add(data_augmentation)  # Add data augmentation as the first layer
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.2))  # Add dropout layer

model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.3))

model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.4))

model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(10, activation='softmax'))

# Learning Rate Scheduler
def lr_schedule(epoch, lr):
  if epoch > 5:
    return lr * 0.5  # Reduce learning rate by half after 5 epochs
  return lr

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Train the model with learning rate scheduler callback
history = model.fit(train_images, train_labels, epochs=15, validation_data=(test_images,
test_labels),
          callbacks=[callbacks.LearningRateScheduler(lr_schedule)])

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

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()

# Plot training & validation loss values

plt.plot(history.history['loss'])

plt.plot(history.history['val_loss'])

plt.title('Model loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()
```

To improve and modify the CNN for MNIST classification, we can introduce a few more advanced techniques:

1. **Add Dropout Layers**: Dropout can help reduce overfitting by randomly turning off neurons during training.
2. **Batch Normalization**: Batch normalization can help stabilize and speed up the training process by normalizing the inputs for each layer.
3. **Increased Epochs and Learning Rate Scheduler**: Train for more epochs and add a learning rate scheduler to reduce the learning rate gradually, improving convergence.
4. **Data Augmentation**: Adding data augmentation can improve generalization, as it creates variations in the input images by applying random transformations like rotation, shift, or zoom.

Explanation of the Modifications:

1. **Data Augmentation**: We added random transformations such as rotation, translation, and zoom to help the model generalize better by seeing different variations of the images.
2. **Batch Normalization**: Adding batch normalization after each convolution layer helps stabilize and accelerate training by normalizing the output of the previous layer.
3. **Dropout Layers**: Dropout layers are added after each max pooling layer and dense layer to reduce overfitting. Each dropout layer has an increasing dropout rate (0.2, 0.3, 0.4, and 0.5), randomly turning off a portion of neurons during training.
4. **Learning Rate Scheduler**: A learning rate scheduler reduces the learning rate by half after the 5th epoch. This helps the optimizer converge more gradually in later epochs.

5. **Increased Epochs**: Training for 15 epochs (instead of 5) helps ensure that the model has enough time to learn with the added complexity.

Expected Outcome

The modifications, especially data augmentation, dropout, and batch normalization, should result in improved accuracy and generalization. You should observe:

- A more stable training process with smoother loss and accuracy curves.
- Higher test accuracy and lower validation loss due to reduced overfitting.