Extended CNN Example with Modifications

This document provides detailed guidance for students to modify and experiment with Convolutional Neural Networks (CNNs) using TensorFlow. It explains how to change the kernel size, add more filters, modify dense layers, increase epochs, and try different activation functions. These experiments will help students understand the impact of these changes on model performance.

1. Change the Kernel Size

The kernel (or filter) size in convolutional layers determines how much of the input the layer looks at when detecting features. Smaller kernels (e.g., 3x3) focus on finer details, while larger kernels (e.g., 5x5) capture more global features.

Example modification:

```
# Original:
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))

# Modified:
model.add(layers.Conv2D(32, (5, 5), activation='relu', input_shape=(28, 28, 1)))
```

2. Add More Filters

The number of filters in a convolutional layer determines how many feature maps the layer generates. Adding more filters allows the model to learn more complex features, but it also increases the computational load.

Example modification:

```
# Original:
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))

# Modified:
model.add(layers.Conv2D(64, (3, 3), activation='relu', input_shape=(28, 28, 1)))
```

3. Change the Dense Layer Size

The Dense (fully connected) layers combine the learned features and are responsible for making the final classification decision. Increasing the number of neurons in a Dense layer can improve the model's ability to learn complex features.

Example modification:

```
# Original:
model.add(layers.Dense(64, activation='relu'))

# Modified:
model.add(layers.Dense(128, activation='relu'))
```

4. Increase Epochs

Training for more epochs gives the model more time to learn patterns from the data. However, training for too long can lead to overfitting, where the model becomes too specialized on the training data and performs poorly on unseen data.

Example modification:

```
# Original:
history = model.fit(train_images, train_labels, epochs=5, validation_data=(test_images, test_labels))

# Modified:
history = model.fit(train_images, train_labels, epochs=10, validation_data=(test_images, test_labels))
```

5. Use Different Activation Functions

Activation functions introduce non-linearity to the model, enabling it to learn more complex patterns. The default function in CNNs is ReLU, but you can experiment with alternatives like tanh or sigmoid to see how they affect the model's performance.

Example modification:

```
# Original (using ReLU):
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))

# Modified (using tanh or sigmoid):
model.add(layers.Conv2D(32, (3, 3), activation='tanh', input_shape=(28, 28, 1)))
```

Full Example Combining Modifications

Below is a complete example that includes multiple modifications, such as changing kernel size, adding filters, increasing Dense layer size, increasing epochs, and using different activation functions.

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

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

# Preprocess the data: reshape and normalize
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

# Define the CNN model
model = models.Sequential()

# Modify kernel size and filters
model.add(layers.Conv2D(64, (5, 5), activation='tanh', input_shape=(28, 28, 1)))  # Changed
to 64 filters, kernel size 5x5, and tanh activation
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), activation='relu'))  # Increased filters to 128
model.add(layers.MaxPooling2D((2, 2)))

# Flatten the 2D image to 1D before feeding into Dense layers
model.add(layers.Flatten())

# Modify the Dense layer
model.add(layers.Dense(128, activation='relu'))  # Increased number of neurons to 128

# Output layer
model.add(layers.Dense(10, activation='softmax'))  # Output layer for 10 classes

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

# Train the model
history = model.fit(train_images, train_labels, epochs=10, validation_data=(test_images,
test_labels))  # Increased epochs to 10

# Evaluate the model
```

```
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f"Test accuracy: {test_acc}")

# Plot training history
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```