DSA4050A: DEEP LEARNING FOR COMPUTER VISION

INSTRUCTOR: DR. EDWARD OMBUI

Lesson Notes: Convolutional Neural Networks (CNNs)

Overview

In Week 3, the focus is on Convolutional Neural Networks (CNNs), a pivotal technology in computer vision that allows machines to accurately recognize and classify images. CNNs excel at identifying patterns in images, such as edges, textures, and shapes, enabling machines to classify and recognize images with remarkable accuracy. This lesson will cover the architecture of CNNs, including convolutional layers, pooling layers, and fully connected layers. Students will engage in hands-on activities to design and implement a basic CNN for image classification using frameworks such as TensorFlow or PyTorch.

Learning Outcomes

By the end of this session, students should be able to:

1. Discuss the structure and functionality of convolutional layers, pooling layers, and fully connected layers.
2. Design and implement a simple CNN for image classification using TensorFlow or PyTorch.
3. Evaluate the performance of the CNN using metrics like accuracy and loss on a validation dataset.

Key Components of CNNs

1. Convolutional Layers:
    - These layers are responsible for feature extraction from images.
    - They utilize filters (or kernels) that slide over the input image to detect patterns such as edges, corners, and textures in the early layers and more complex structures (e.g., objects) in deeper layers.
    - The output is a feature map that highlights the presence of specific features in the input data.

    Key Parameters in Convolutional Layers:

- **Kernel size:** Dimensions of the filter (e.g., 3x3, 5x5).
- **Stride:** Steps the filter moves across the input image.
- **Padding:** Adds borders to the input image to control the output size.

2. Pooling Layers:
    - Pooling layers reduce the spatial dimensions of feature maps, which helps in minimizing computational load and controlling overfitting.
    - Two common pooling types:
        - **Max Pooling:** Retains the maximum value in a patch of the feature map.
        - **Average Pooling:** Computes the average value in a patch.

3. Fully Connected Layers:
    - These layers connect every neuron from one layer to every neuron in the next layer.
    - Flatten the output of the convolutional and pooling layers into a vector.
    - Combine the detected features to make predictions, such as classifying the input image into categories.
    - The final layer typically uses an activation function like **Softmax** to output class probabilities.
    - They are typically used at the end of the network to combine features learned by previous layers and make predictions.

Practical Implementation

Designing and Implementing a CNN

This week, you'll design a simple CNN for image classification. The process involves:

1. **Dataset Preparation**:
    - Load and preprocess the dataset (e.g., normalize pixel values, resize images).
    - Common datasets: MNIST, CIFAR-10, or custom datasets.
2. **Model Architecture**:

Example CNN Design:
plaintext
CopyEdit
```
Input Layer → Conv Layer (ReLU) → Pooling Layer → Conv Layer
(ReLU) → Pooling Layer → Fully Connected Layer → Output Layer
```

Using TensorFlow (Example):
python
CopyEdit
```
import tensorflow as tf

from tensorflow.keras import layers, models

model = models.Sequential([

    layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(64, 64, 3)),

    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),

    layers.Dense(64, activation='relu'),

    layers.Dense(10, activation='softmax')  # For 10 classes

])
```

3. **Training the Model**:
    - **Steps**:
        - Compile the model with an optimizer (e.g., Adam), loss function (e.g., categorical cross-entropy), and metrics (e.g., accuracy).
        - Train the model using a training dataset.
        - Monitor validation metrics to assess performance.

Example:
python
CopyEdit
```
model.compile(optimizer='adam',

              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history = model.fit(train_images, train_labels, epochs=10,
validation_data=(val_images, val_labels))
```

   ○
4. **Evaluating the Model**:
   ○ Use **metrics like accuracy and loss** on a validation dataset to assess the model's performance on unseen data.
   ○ Plot learning curves to visualize the training and validation performance over epochs.

Example:
python
CopyEdit
```
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val_accuracy'], label='Validation
Accuracy')

plt.legend()

plt.show()
```

Evaluation Metrics

1. **Accuracy**: Measures the percentage of correct predictions out of total predictions.
   ○ High accuracy indicates a well-performing model.
2. **Loss**: Measures the error between predicted and actual outputs.
   ○ Reducing loss over epochs indicates that the model is learning effectively. lower loss indicates better performance

Conclusion

This week's lesson on Convolutional Neural Networks equips you with foundational knowledge about CNN architecture and practical skills in implementing and evaluating image classification models using deep learning frameworks. By understanding these concepts, you can further explore advanced topics in computer vision and machine learning applications.

Below is a step-by-step **Google Colab Notebook** for building and training a CNN using the **MNIST dataset**. This notebook includes data preprocessing, model creation, training, and evaluation. You can copy and run this code directly in Colab.

The MNIST Dataset (Handwritten Digits)

- Contains 70,000 grayscale images of digits (0–9), each 28x28 pixels.
- Great for beginners to understand image classification.
- How to Load in TensorFlow

Step 1: Setting Up Google Colab

1. Open Google Colab.
2. Create a new notebook or open an existing one.
3. Copy the following code and paste it into your notebook cells.

Code: Building and Training a CNN for Image Classification

1. Import Libraries

```
import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import mnist

import matplotlib.pyplot as plt
```

2. Load and Preprocess the Dataset

```
# Load MNIST dataset

(train_images, train_labels), (test_images, test_labels) =
mnist.load_data()

# Normalize pixel values (0 to 1)

train_images = train_images / 255.0
```

```
test_images = test_images / 255.0


# Add a channel dimension (grayscale image has 1 channel)

train_images = train_images.reshape(-1, 28, 28, 1)

test_images = test_images.reshape(-1, 28, 28, 1)


# Display a sample image

plt.imshow(train_images[0].reshape(28, 28), cmap='gray')

plt.title(f"Label: {train_labels[0]}")

plt.show()
```

3. Build the CNN Model

```
# Create CNN model

model = models.Sequential([

    layers.Conv2D(32, (3, 3), activation='relu', 
input_shape=(28, 28, 1)),

    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),

    layers.Dense(64, activation='relu'),
    layers.Dropout(0.5),  # Regularization

    layers.Dense(10, activation='softmax')  # 10 classes for
digits 0-9

])

# Compile the model

model.compile(optimizer='adam',

              loss='sparse_categorical_crossentropy',

              metrics=['accuracy'])



# Print the model summary

model.summary()
```

4. Train the Model

```
# Train the model

history = model.fit(train_images, train_labels,

                    epochs=10,

                    validation_split=0.2,  # Use 20% of
training data for validation

                    batch_size=64)
```

5. Evaluate the Model

```
# Evaluate on the test dataset
```

```
test_loss, test_accuracy = model.evaluate(test_images,
test_labels)

print(f"Test Accuracy: {test_accuracy * 100:.2f}%")

# Plot training and validation accuracy

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()

plt.show()
```

6. Save and Load the Model

```
# Save the model

model.save('mnist_cnn_model.h5')

print("Model saved as mnist_cnn_model.h5")

# Load the model

loaded_model =
tf.keras.models.load_model('mnist_cnn_model.h5')
```

7. Test Predictions

```
# Make predictions
```

```
import numpy as np

predictions = loaded_model.predict(test_images)

# Display a test image and predicted label

index = 0  # Change this to view other test images

plt.imshow(test_images[index].reshape(28, 28), cmap='gray')

plt.title(f"True Label: {test_labels[index]} | Predicted:
{np.argmax(predictions[index])}")

plt.show()
```

Running the Notebook

1. Dataset: The MNIST dataset is downloaded automatically by TensorFlow.
2. Hardware Acceleration: In Colab, go to `Runtime > Change runtime type` and select GPU for faster training.
3. Results: After training, you'll see accuracy plots and test results.

Resources to Learn More

1. **Documentation and Tutorials**:
   - TensorFlow CNN Tutorial
   - PyTorch CNN Guide
2. **Books**:
   - **Deep Learning with Python** by François Chollet (author of Keras)
   - **Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow** by Aurélien Géron
3. **Interactive Platforms**:
   - Google Colab: Free environment for coding with GPUs.
   - Kaggle: Practice coding and participate in challenges.