Practical Work 3: Convolutional Neural Networks and Computer Vision

Louis Fippo Fitime, Claude Tinku, Kerolle Sonfack
Department of Computer Engineering, ENSPY

October 8, 2025

Abstract

This third Practical Work document is dedicated to mastering **Convolutional Neural Networks (CNNs)** and their fundamental applications in Computer Vision. Students will learn how to design robust architectures, implement advanced techniques like residual networks, and explore tasks ranging from image classification to style transfer.

Learning Objectives

- Understand the fundamental principles of **convolution** and **pooling** operations.
- Build, train, and evaluate a CNN for image classification.
- Know how to integrate and understand the utility of **residual blocks** (ResNets).
- Apply CNNs to **visual recognition** and **style transfer** tasks.
- Master the implementation of these concepts using the **Keras** library.

1 Part 1: CNN Fundamentals (1.5h)

1.1 Theoretical Concepts

- **Convolution ($*$):** Explain the role of the filter (kernel) and the stride in a convolution operation. What is the main objective of a convolutional layer in a CNN?
- **Pooling:** Describe the two main types of pooling (Max Pooling, Average Pooling) and their role in dimensionality reduction and robust feature extraction.
- **From Image to Classification:** Explain how the spatial features of an image are transformed into flat vectors for the input of fully connected (Dense) layers.
- **Residual Networks (ResNets):** What problem do the residual connections (*skip connections*) seek to solve in very deep networks?

1.2 Practical: Preparing the CIFAR-10 Data

For the practical exercises, we will use the **CIFAR-10** dataset (10 classes of 32×32 color images), which is more complex than MNIST.

Instructions:

1. Create a new file `cnn_classification.py`.
2. Implement the data preparation.

```python
import tensorflow as tf
from tensorflow import keras
import numpy as np

1. Load the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.
    load_data()

Number of classes
NUM_CLASSES = 10
INPUT_SHAPE = x_train.shape[1:] # (32, 32, 3)

2. Normalize pixel values to [0, 1]
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

3. Convert labels to One-Hot Encoding format
y_train = keras.utils.to_categorical(y_train, num_classes=
    NUM_CLASSES)
y_test = keras.utils.to_categorical(y_test, num_classes=NUM_CLASSES)

print(f"Input data shape: {INPUT_SHAPE}")

TODO: Print the shape of the labels after conversion
```

Listing 1: Loading and Pre-processing CIFAR-10

2 Part 2: Basic CNN Implementation (2.5h)

2.1 Exercise 1: Classic CNN Architecture

You will build a simple but effective CNN.

Instructions:

1. Complete the `build_basic_cnn` function below.
2. Train the model for at least 10 epochs.

```

... (previous code)
def build_basic_cnn(input_shape, num_classes):
model = keras.Sequential([
# Convolutional Layer 1: 32 filters, 3x3 size, ReLU activation
keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same',
input_shape=input_shape),
# Pooling Layer 1: Max Pooling 2x2
# TODO: Add the Max Pooling layer
keras.layers.MaxPooling2D(pool_size=(2, 2)), # Added Max Pooling

    # Convolutional Layer 2: 64 filters, 3x3 size, ReLU activation
    # TODO: Add the second Conv2D layer
    keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same
'), # Added Conv2D layer

    # Pooling Layer 2: Max Pooling 2x2
    keras.layers.MaxPooling2D(pool_size=(2, 2)),

    # Flatten Layer to transition to Dense layers
    keras.layers.Flatten(),

    # Dense Layer 1: 512 units, ReLU activation
    keras.layers.Dense(512, activation='relu'),
    # Output Layer: num_classes units, Softmax activation
    keras.layers.Dense(num_classes, activation='softmax')
])
return model

model = build_basic_cnn(INPUT_SHAPE, NUM_CLASSES)
model.compile(optimizer='adam', loss='categorical_crossentropy',
   metrics=['accuracy'])

Train the model (use a small part of the dataset for Dev/Validation)
history = model.fit(
x_train, y_train,
batch_size=64,
epochs=10,
validation_split=0.1 # 10% of training data for validation
)
```

```

TODO: Evaluate the model on x_test, y_test and display the accuracy.
...
```

Listing 2: Building and Training a Basic CNN

2.2 Exercise 2: Introduction to Residual Blocks (ResNets)

Residual networks (ResNets) use skip connections to help the gradient propagate through deep networks.

Instructions:

1. Analyze the role of the simplified `residual_block` function below.
2. **Question:** Explain the advantage of adding the input `x` to the output of the convolutional path.

```
import tensorflow as tf
from tensorflow import keras

def residual_block(x, filters, kernel_size=(3, 3), stride=1):
# Main path
y = keras.layers.Conv2D(filters, kernel_size, strides=stride,
   padding='same', activation='relu')(x)
y = keras.layers.Conv2D(filters, kernel_size, padding='same')(y)

# Skip Connection path
if stride > 1:
    # The skip connection must reduce dimensions if stride > 1
    x = keras.layers.Conv2D(filters, (1, 1), strides=stride)(x)

# TODO: Complete the addition of the skip path with the main path
z = keras.layers.Add()([x, y])

z = keras.layers.Activation('relu')(z)
return z

TODO: Build a small architecture using 3 consecutive residual blocks
input_layer = keras.Input(shape=INPUT_SHAPE)
x = residual_block(input_layer, 32)
x = residual_block(x, 64, stride=2)
x = residual_block(x, 64)
```

Listing 3: Simplified Residual Block Function

3 Part 3: Advanced Applications (3h)

3.1 Exercise 3: Application to Recognition and Detection

Although full implementation is too long, the student must understand the concept of the task.

Instructions:

1. **Recognition/Classification (Completed):** Exercise 1 covers image classification (Recognition).

2. **Image Segmentation (Concept):** Research the **U-Net** architecture (often used for segmentation). Briefly describe:

   - What is the output of an image segmentation model (as opposed to classification)?
   - What is the role of the upsampling steps in the U-Net architecture?

3. **Object Detection (Concept):** Research the concept of **Bounding Boxes**. Explain how a CNN, in addition to classifying an image, can predict the position of an object (coordinates (x,y,w,h)).

3.2 Exercise 4: Neural Style Transfer

Style transfer uses a pre-trained CNN (usually **VGG16**) to separate the content of one image and the style of another.

Instructions:

1. **VGG16 Loading:** Complete the code to load the VGG16 model pre-trained on ImageNet, excluding the fully connected layers.

2. **Definition of Losses:** What is the role of **content losses** (*content loss*) and **style losses** (*style loss*) in the optimization of the generated image?

```

Requires matplotlib and pil for image processing
1. Load a content image and a style image (to be provided by the
   student)
TODO: Load and pre-process two images of your choice (e.g., sizes
   512x512)
content_image = ...
style_image = ...
2. Load the pre-trained VGG16 model
vgg = keras.applications.VGG16(include_top=False, weights='imagenet'
   )
vgg.trainable = False # Important: the VGG model is not being
   trained here

Content and style layers for feature extraction
content_layers = ['block5_conv2']
style_layers = ['block1_conv1', 'block2_conv1', 'block3_conv1', '
   block4_conv1', 'block5_conv1']
```

```

Function to create an extractor model
def create_extractor(model, style_layers, content_layers):
# The model outputs will be the activations of the selected layers
outputs = [model.get_layer(name).output for name in style_layers +
    content_layers]
return keras.Model(inputs=model.input, outputs=outputs)

extractor = create_extractor(vgg, style_layers, content_layers)

TODO: The next step is to define the target image and optimize
the pixels to minimize BOTH content loss AND style loss.
This process uses a custom optimization loop (not shown here).
```

Listing 4: Preparation for Style Transfer

4 Conclusion

To submit:

- The link to your GitHub repository with the complete `cnn_classification.py` files.
- A short report (`.pdf` or Overleaf link) answering the theoretical questions, notably the analysis of architectures (ResNets) and the concepts of segmentation/detection.