**University of Central Missouri**

**Department of Computer Science & Cybersecurity**

**CS5720 Neural network and Deep learning**

**Spring 2025**

**Home Assignment 2. (Cover Ch 4,5)**

**Student name: Yaswanth Kumar Polarouthu**

**Submission Requirements:**

* Total Points: 100
* Once finished your assignment push your source code to your repo (GitHub) and explain the work through the ReadMe file properly. Make sure you add your student info in the ReadMe file.
* Submit your GitHub link and video on the BB.
* Comment your code appropriately ***IMPORTANT.***
* Make a simple video about 2 to 3 minutes which includes demonstration of your home assignment and explanation of code snippets.
* Any submission after provided deadline is considered as a late submission.

**Question 1: Cloud Computing for Deep Learning (20 points)**

Cloud computing offers significant advantages for deep learning applications.

(a) Define **elasticity** and **scalability** in the context of cloud computing for deep learning. (10 points)  
(b) Compare **AWS SageMaker**, **Google Vertex AI**, and **Microsoft Azure Machine Learning Studio** in terms of their deep learning capabilities. (10 points)

**Expected Output**

Write the definition and comparison for (a) and (b). No code needed.

Ans:

### (a) Elasticity and Scalability in Cloud Computing for Deep Learning (10 points)

* **Elasticity**: The ability of a cloud system to dynamically allocate or deallocate resources based on real-time demand. In deep learning, this ensures efficient use of compute resources, scaling up for high workloads (e.g., training large models) and scaling down when demand decreases to optimize costs.
* **Scalability**: The ability to handle increasing workloads by adding more resources (horizontal scaling) or upgrading existing resources (vertical scaling). In deep learning, scalability ensures that models can be trained on larger datasets or with more complex architectures without performance degradation.

(b) Comparison of AWS SageMaker, Google Vertex AI, and Azure Machine Learning Studio (10 points)

A screenshot of a black and white screen

AI-generated content may be incorrect.

Each platform excels in different aspects:

* **AWS SageMaker** is strong for enterprise-grade ML with deep AWS integration.
* **Google Vertex AI** offers the best TPU support and seamless AI automation.
* **Azure ML Studio** is great for hybrid cloud AI and enterprise AI solutions.

**Question 2: Convolution Operations with Different Parameters (20 points)**

**Task: Implement Convolution with Different Stride and Padding (10 points)**

Write a Python script using **NumPy and TensorFlow/Keras** to perform **convolution** on a **5×5 input matrix** using a **3×3 kernel** with varying parameters.

1. Define the following **5×5 input matrix**:

A number with black text

AI-generated content may be incorrect.

1. Define the following **3×3 kernel**:

A number lines with numbers

AI-generated content may be incorrect.

1. Perform **convolution operations** with:
   * **Stride = 1, Padding = ‘VALID’**
   * **Stride = 1, Padding = ‘SAME’**
   * **Stride = 2, Padding = ‘VALID’**
   * **Stride = 2, Padding = ‘SAME’**
2. Print the **output feature maps** for each case.

**Expected Output**

Print the output feature maps for

* + **Stride = 1, Padding = ‘VALID’**
  + **Stride = 1, Padding = ‘SAME’**
  + **Stride = 2, Padding = ‘VALID’**
  + **Stride = 2, Padding = ‘SAME’**

**Ans:**

**import numpy as np**

**import tensorflow as tf**

**from tensorflow.keras.layers import Conv2D**

**from tensorflow.keras.models import Sequential**

**# Define 5x5 input matrix**

**input\_matrix = np.array([**

**[1, 2, 3, 4, 5],**

**[6, 7, 8, 9, 10],**

**[11, 12, 13, 14, 15],**

**[16, 17, 18, 19, 20],**

**[21, 22, 23, 24, 25]**

**], dtype=np.float32)**

**# Reshape input to match Conv2D input shape (batch\_size, height, width, channels)**

**input\_matrix = input\_matrix.reshape((1, 5, 5, 1))**

**# Define 3x3 kernel**

**kernel = np.array([**

**[1, 0, -1],**

**[1, 0, -1],**

**[1, 0, -1]**

**], dtype=np.float32).reshape((3, 3, 1, 1)) # Reshape to (height, width, in\_channels, out\_channels)**

**# Function to perform convolution with given stride and padding**

**def apply\_convolution(stride, padding):**

**model = Sequential([**

**Conv2D(filters=1, kernel\_size=(3, 3), strides=stride, padding=padding, use\_bias=False, input\_shape=(5, 5, 1))**

**])**

**model.layers[0].set\_weights([kernel]) # Set custom kernel**

**output = model.predict(input\_matrix)**

**return output.squeeze() # Remove batch and channel dimensions**

**# Perform convolutions with different parameters**

**output\_valid\_1 = apply\_convolution(1, 'valid')**

**output\_same\_1 = apply\_convolution(1, 'same')**

**output\_valid\_2 = apply\_convolution(2, 'valid')**

**output\_same\_2 = apply\_convolution(2, 'same')**

**# Print results**

**print("Stride = 1, Padding = 'VALID'\n", output\_valid\_1)**

**print("Stride = 1, Padding = 'SAME'\n", output\_same\_1)**

**print("Stride = 2, Padding = 'VALID'\n", output\_valid\_2)**

**print("Stride = 2, Padding = 'SAME'\n", output\_same\_2)**

**Output:**

**A computer screen shot of a black screen

AI-generated content may be incorrect.**

**Question 3: CNN Feature Extraction with Filters and Pooling (30 points)**

**Task 1: Implement Edge Detection Using Convolution (15 points)**

Write a Python script using **NumPy and OpenCV (cv2)** to apply **edge detection** on an image using a **Sobel filter**.

* Load a grayscale image (you can use any sample image).
* Apply the **Sobel filter** for **edge detection** in the **x-direction** and **y-direction**.
* Display the original image and the filtered images.

Use the following Sobel filters:

A diagram of a mathematical equation

AI-generated content may be incorrect.

**Task 2: Implement Max Pooling and Average Pooling (15 points)**

Write a Python script using **TensorFlow/Keras** to demonstrate **Max Pooling** and **Average Pooling**.

* Create a **random 4x4 matrix** as an input image.
* Apply a **2x2 Max Pooling** operation.
* Apply a **2x2 Average Pooling** operation.
* Print the original matrix, max-pooled matrix, and average-pooled matrix.

**Expected Output**

**Task1: Edge Detection using Sobel Filter**

* Display **three images**:
  1. **Original Image**
  2. **Edge detection using Sobel-X**
  3. **Edge detection using Sobel-Y**

**Task2: Pooling Operations on Random 4×4 Matrix**

* Printed **original matrix, max pooled matrix, and average pooled matrix**.

Ans:

Code:

import numpy as np

import tensorflow as tf

import cv2

import matplotlib.pyplot as plt

from tensorflow.keras.layers import MaxPooling2D, AveragePooling2D

from tensorflow.keras.models import Sequential

# Task 1: Edge Detection Using Sobel Filter

def apply\_sobel\_filter(image\_path):

image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

sobel\_x = cv2.Sobel(image, cv2.CV\_64F, 1, 0, ksize=3)

sobel\_y = cv2.Sobel(image, cv2.CV\_64F, 0, 1, ksize=3)

plt.figure(figsize=(10, 4))

plt.subplot(1, 3, 1)

plt.title("Original Image")

plt.imshow(image, cmap='gray')

plt.subplot(1, 3, 2)

plt.title("Sobel X")

plt.imshow(sobel\_x, cmap='gray')

plt.subplot(1, 3, 3)

plt.title("Sobel Y")

plt.imshow(sobel\_y, cmap='gray')

plt.show()

# Task 2: Max Pooling and Average Pooling

def apply\_pooling():

input\_matrix = np.random.rand(1, 4, 4, 1)

model\_max = Sequential([MaxPooling2D(pool\_size=(2, 2), input\_shape=(4, 4, 1))])

model\_avg = Sequential([AveragePooling2D(pool\_size=(2, 2), input\_shape=(4, 4, 1))])

max\_pooled = model\_max.predict(input\_matrix).squeeze()

avg\_pooled = model\_avg.predict(input\_matrix).squeeze()

print("Original Matrix:\n", input\_matrix.squeeze())

print("Max Pooled Matrix:\n", max\_pooled)

print("Average Pooled Matrix:\n", avg\_pooled)

# Example usage

# apply\_sobel\_filter("sample\_image.jpg") # Provide a valid image path

apply\_pooling()

Output:

A computer screen with white text

AI-generated content may be incorrect.

**Question 4: Implementing and Comparing CNN Architectures (30 points)**

**Task 1: Implement AlexNet Architecture (15 points)**

Write a **Python script** using **TensorFlow/Keras** to implement a simplified **AlexNet** model with the following layers:

* **Conv2D Layer**: 96 filters, kernel size = (11,11), stride = 4, activation = ReLU
* **MaxPooling Layer**: pool size = (3,3), stride = 2
* **Conv2D Layer**: 256 filters, kernel size = (5,5), activation = ReLU
* **MaxPooling Layer**: pool size = (3,3), stride = 2
* **Conv2D Layer**: 384 filters, kernel size = (3,3), activation = ReLU
* **Conv2D Layer**: 384 filters, kernel size = (3,3), activation = ReLU
* **Conv2D Layer**: 256 filters, kernel size = (3,3), activation = ReLU
* **MaxPooling Layer**: pool size = (3,3), stride = 2
* **Flatten Layer**
* **Fully Connected (Dense) Layer**: 4096 neurons, activation = ReLU
* **Dropout Layer**: 50%
* **Fully Connected (Dense) Layer**: 4096 neurons, activation = ReLU
* **Dropout Layer**: 50%
* **Output Layer**: 10 neurons, activation = Softmax

Print the **model summary** after defining it.

**Task 2: Implement a Residual Block and ResNet (15 points)**

Write a Python script to define a **Residual Block** and use it to build a simple **ResNet-like model**.

1. Implement a function residual\_block(input\_tensor, filters) that:
   * Takes an **input tensor**.
   * Applies **two Conv2D layers** (each with 64 filters, kernel size = (3,3), activation = ReLU).
   * Includes a **skip connection** that adds the input tensor to the output before activation.
2. Create a **ResNet model** that:
   * Uses an **initial Conv2D layer** (64 filters, kernel size = (7,7), stride = 2).
   * Applies **two residual blocks**.
   * Ends with a **Flatten layer, Dense layer (128 neurons), and Output layer (Softmax)**.

Print the **model summary** after defining it.

**Expected Output**

The output should display:

1. The **model summary** for **AlexNet**.
2. The **model summary** for the **ResNet-like model**.

Ans:

Code:

import numpy as np

import tensorflow as tf

import cv2

import matplotlib.pyplot as plt

from tensorflow.keras.layers import MaxPooling2D, AveragePooling2D, Conv2D, Flatten, Dense, Dropout, Input

from tensorflow.keras.models import Sequential, Model

# Task 1: Edge Detection Using Sobel Filter

def apply\_sobel\_filter(image\_path):

image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

sobel\_x = cv2.Sobel(image, cv2.CV\_64F, 1, 0, ksize=3)

sobel\_y = cv2.Sobel(image, cv2.CV\_64F, 0, 1, ksize=3)

plt.figure(figsize=(10, 4))

plt.subplot(1, 3, 1)

plt.title("Original Image")

plt.imshow(image, cmap='gray')

plt.subplot(1, 3, 2)

plt.title("Sobel X")

plt.imshow(sobel\_x, cmap='gray')

plt.subplot(1, 3, 3)

plt.title("Sobel Y")

plt.imshow(sobel\_y, cmap='gray')

plt.show()

# Task 2: Max Pooling and Average Pooling

def apply\_pooling():

input\_matrix = np.random.rand(1, 4, 4, 1)

model\_max = Sequential([MaxPooling2D(pool\_size=(2, 2), input\_shape=(4, 4, 1))])

model\_avg = Sequential([AveragePooling2D(pool\_size=(2, 2), input\_shape=(4, 4, 1))])

max\_pooled = model\_max.predict(input\_matrix).squeeze()

avg\_pooled = model\_avg.predict(input\_matrix).squeeze()

print("Original Matrix:\n", input\_matrix.squeeze())

print("Max Pooled Matrix:\n", max\_pooled)

print("Average Pooled Matrix:\n", avg\_pooled)

# Task 3: Implement AlexNet

def build\_alexnet():

model = Sequential([

Conv2D(96, (11, 11), strides=4, activation='relu', input\_shape=(227, 227, 3)),

MaxPooling2D((3, 3), strides=2),

Conv2D(256, (5, 5), activation='relu', padding='same'),

MaxPooling2D((3, 3), strides=2),

Conv2D(384, (3, 3), activation='relu', padding='same'),

Conv2D(384, (3, 3), activation='relu', padding='same'),

Conv2D(256, (3, 3), activation='relu', padding='same'),

MaxPooling2D((3, 3), strides=2),

Flatten(),

Dense(4096, activation='relu'),

Dropout(0.5),

Dense(4096, activation='relu'),

Dropout(0.5),

Dense(10, activation='softmax')

])

model.summary()

return model

# Task 4: Implement Residual Block and ResNet

def residual\_block(input\_tensor, filters):

x = Conv2D(filters, (3, 3), padding='same', activation='relu')(input\_tensor)

x = Conv2D(filters, (3, 3), padding='same')(x)

x = tf.keras.layers.Add()([x, input\_tensor])

x = tf.keras.layers.Activation('relu')(x)

return x

def build\_resnet():

inputs = Input(shape=(224, 224, 3))

x = Conv2D(64, (7, 7), strides=2, padding='same', activation='relu')(inputs)

x = residual\_block(x, 64)

x = residual\_block(x, 64)

x = Flatten()(x)

x = Dense(128, activation='relu')(x)

outputs = Dense(10, activation='softmax')(x)

model = Model(inputs, outputs)

model.summary()

return model

# Example usage

# apply\_sobel\_filter("sample\_image.jpg") # Provide a valid image path

apply\_pooling()

alexnet\_model = build\_alexnet()

resnet\_model = build\_resnet()

Output:  
A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.