**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:Sushma Veluru  
700765390**

**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.

**(a) Elasticity and Scalability in Cloud Computing for Deep Learning**

* **Elasticity** refers to the ability of a cloud system to dynamically allocate or deallocate resources based on workload demand. In deep learning, this means provisioning more GPUs or TPUs during model training and releasing them when no longer needed, optimizing cost and performance.
* **Scalability** is the ability of a cloud system to handle increased workload by adding resources (scale up) or distributing the load across multiple resources (scale out). In deep learning, scalability ensures efficient handling of large datasets and complex models by leveraging distributed computing.

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

| **Feature** | **AWS SageMaker** | **Google Vertex AI** | **Microsoft Azure ML Studio** |
| --- | --- | --- | --- |
| **Ease of Use** | Fully managed with automation | Integrated with Google services | Drag-and-drop and code-based |
| **Compute Options** | Supports CPUs, GPUs, and custom accelerators (Inferentia) | Uses TPUs and GPUs | Supports multiple VM types, GPUs, and FPGAs |
| **Integration** | Integrates with AWS services (S3, Lambda, etc.) | Deep integration with Google Cloud (BigQuery, DataFlow) | Integrates with Azure services (Blob Storage, Synapse) |
| **Model Deployment** | Real-time and batch inference with auto-scaling | AutoML and scalable deployment options | Managed endpoints for deployment |
| **AutoML Capabilities** | Built-in AutoML tools | Strong AutoML capabilities | Supports AutoML for model training |
| **Pricing** | Pay-per-use, Spot instances for cost savings | Flexible pricing with sustained discounts | Consumption-based pricing with reserved instances |

**Summary**:

* **AWS SageMaker** is ideal for enterprises needing deep integration with AWS and scalable deployment.
* **Google Vertex AI** excels in AutoML and TPUs, making it great for automated and high-performance deep learning.
* **Azure ML Studio** is user-friendly with strong enterprise integration, suitable for both no-code and code-based ML development.

**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’**

**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**.

**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**.