**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: Jyostna Marella(700761880)**

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

1. **Elasticity and Scalability in Cloud Computing for Deep Learning (10 points)**
   * **Elasticity** refers to the ability of a cloud computing system to automatically adjust resources (such as compute power, storage, and networking) based on demand. For deep learning, this means dynamically scaling up GPUs/TPUs during model training and scaling down when the demand decreases, optimizing cost efficiency.
   * **Scalability** is the capacity of a system to handle an increasing workload by adding more resources, either **vertically** (upgrading existing hardware) or **horizontally** (adding more machines). In deep learning, scalability ensures that models can be trained on massive datasets efficiently by distributing workloads across multiple cloud instances.

# Comparison of AWS Sage Maker, Google Vertex AI, and Microsoft Azure Machine Learning Studio (10 points)

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **AWS Sage Maker** | **Google Vertex AI** | **Microsoft Azure ML Studio** |
| **Ease of Use** | Provides pre-built Jupyter notebooks and automated ML capabilities. | Simplifies end-to-end AI workflows with AutoML and pipelines. | Offers a no-code UI along with support for advanced ML pipelines. |
| **Compute Support** | Supports CPU, GPU, and AWS Inferentia chips for cost-effective inference. | Offers TPUs, GPUs, and CPUs for AI model training and deployment. | Provides flexible compute options, including NVIDIA GPUs and FPGAs. |
| **AutoML Capabilities** | Includes built-in AutoML for training without deep expertise. | Strong AutoML support with hyperparameter tuning. | Advanced AutoML with drag-and-drop features. |
| **Model Deployment** | One-click deployment with endpoints for real- time inference. | Seamless model deployment via AI pipelines. | Supports real-time and batch inference with MLOps integration. |
| **Integration & Ecosystem** | Deep integration with AWS services (S3, Lambda, EC2, etc.). | Native support for Google Cloud services (BigQuery, Dataflow, etc.). | Well-integrated with Azure services (Power BI, Cognitive Services, etc.). |
| **Pricing** | Pay-as-you-go pricing with spot instance discounts. | Flexible pricing based on usage; discounts for sustained use. | Consumption-based pricing with enterprise discounts. |

Each platform has strengths depending on specific needs:

* + **AWS SageMaker**: Best for users already in the AWS ecosystem. It provides end-to-end machine learning services, including built-in algorithms, automated model tuning, and easy AWS storage and compute integration.
  + **Google Vertex AI**: Optimized for large-scale AI workloads and deep learning applications, especially with TPUs (Tensor Processing Units). It offers seamless integration with Google Cloud services and a unified AI platform for model training, deployment, and monitoring.
  + **Azure Machine Learning Studio**: Well-suited for enterprises using Microsoft services. It provides robust MLOps capabilities, AutoML features, and strong integration with tools like Power BI and Azure DevOps.

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

import numpy as np import tensorflow as tf

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Conv2D

# 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 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 to match kernel shape (height, width, input\_channels, output\_channels) kernel = kernel.reshape(3, 3, 1, 1)

# Function to perform convolution

def apply\_convolution(stride, padding): input\_layer = Input(shape=(5, 5, 1))

conv\_layer = Conv2D(filters=1, kernel\_size=(3, 3), strides=stride, padding=padding, kernel\_initializer=tf.keras.initializers.Constant(kernel),

use\_bias=False)(input\_layer)

model = Model(inputs=input\_layer, outputs=conv\_layer) output = model.predict(input\_matrix)

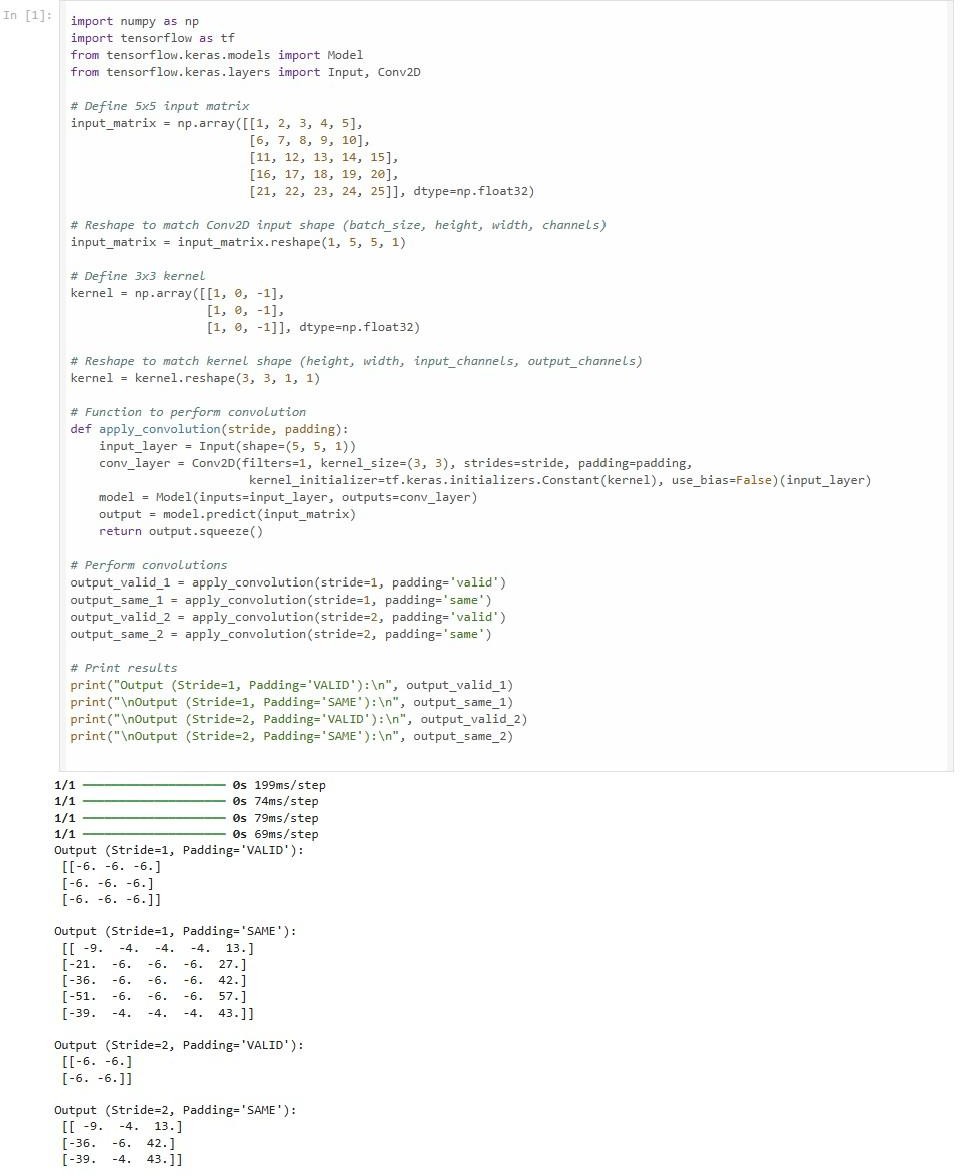
return output.squeeze()

# Perform convolutions

output\_valid\_1 = apply\_convolution(stride=1, padding='valid') output\_same\_1 = apply\_convolution(stride=1, padding='same') output\_valid\_2 = apply\_convolution(stride=2, padding='valid') output\_same\_2 = apply\_convolution(stride=2, padding='same')

# Print results

print("Output (Stride=1, Padding='VALID'):\n", output\_valid\_1) print("\nOutput (Stride=1, Padding='SAME'):\n", output\_same\_1) print("\nOutput (Stride=2, Padding='VALID'):\n", output\_valid\_2) print("\nOutput (Stride=2, Padding='SAME'):\n", output\_same\_2)

This script applies a 3×3 kernel to a 5×5 input matrix using different stride and padding configurations, then prints the resulting feature maps

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

import numpy as np import cv2

import tensorflow as tf

from tensorflow.keras.layers import MaxPooling2D, AveragePooling2D

# 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)

# Show images inline in Google Colab import matplotlib.pyplot as plt

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

plt.subplot(1, 3, 1) plt.title("Original Image") plt.imshow(image, cmap="gray") plt.axis("off")

plt.subplot(1, 3, 2)

plt.title("Sobel-X Edge Detection") plt.imshow(sobel\_x, cmap="gray") plt.axis("off")

plt.subplot(1, 3, 3)

plt.title("Sobel-Y Edge Detection") plt.imshow(sobel\_y, cmap="gray") plt.axis("off")

plt.show()

# Example usage (Uncomment and provide a valid image path in Colab) # apply\_sobel\_filter('/content/sample\_image.jpg')

# Task 2: Implement Max Pooling and Average Pooling def apply\_pooling():

# Generate a random 4x4 matrix with integer values (TensorFlow expects float inputs) input\_matrix = np.random.randint(0, 256, (1, 4, 4, 1), dtype=np.int32)

# Convert to a TensorFlow tensor with float32 dtype

input\_tensor = tf.convert\_to\_tensor(input\_matrix, dtype=tf.float32)

# Define pooling layers

max\_pool = MaxPooling2D(pool\_size=(2, 2), strides=2)(input\_tensor) avg\_pool = AveragePooling2D(pool\_size=(2, 2), strides=2)(input\_tensor)

# Print results

print("Original Matrix:\n", input\_matrix.squeeze()) # Remove extra dimensions for readability print("\nMax Pooled Matrix:\n", max\_pool.numpy().squeeze())

print("\nAverage Pooled Matrix:\n", avg\_pool.numpy().squeeze())

# Example usage apply\_pooling()



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

import numpy as np import cv2

import tensorflow as tf

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

Add

from tensorflow.keras.models import Model, Sequential

# Task 1: Implement AlexNet Architecture def 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 2: Implement a 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 = Add()([x, input\_tensor]) # Skip connection x = tf.keras.layers.Activation('relu')(x)

return x

def 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 alexnet\_model = alexnet() resnet\_model = resnet()



