# Home Assignment 2 Solutions

CS5720 Neural Network and Deep Learning  
Spring 2025

## Question 1: Cloud Computing for Deep Learning

(a) Define elasticity and scalability in the context of cloud computing for deep learning.

Elasticity in cloud computing refers to the ability of a system to dynamically adjust its resources to match changing workload demands. In the context of deep learning, elasticity allows for the allocation of more resources (e.g., GPUs, CPUs, memory) when needed to handle large datasets or complex models, and deallocation when resources are no longer needed.

Scalability in cloud computing refers to the ability of a system to handle increased workload by adding more resources. In deep learning, scalability allows for the distribution of workload across multiple machines, enabling faster training and inference times.

(b) Compare AWS SageMaker, Google Vertex AI, and Microsoft Azure Machine Learning Studio in terms of their deep learning capabilities.

All three platforms provide managed services for building, training, and deploying machine learning models, including deep learning models. Here's a brief comparison:

• AWS SageMaker: Supports popular deep learning frameworks like TensorFlow, PyTorch, and MXNet. Offers automatic model tuning, hyperparameter optimization, and distributed training.

• Google Vertex AI: Supports TensorFlow, PyTorch, and scikit-learn. Offers AutoML, hyperparameter tuning, and distributed training.

• Microsoft Azure Machine Learning Studio: Supports TensorFlow, PyTorch, and scikit-learn. Offers automated machine learning, hyperparameter tuning, and distributed training.

## Question 2: Convolution Operations with Different Parameters

Below is the Python code to perform convolution operations with different stride and padding parameters:

import numpy as np  
import tensorflow as tf  
  
# Define the 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]])  
  
# Define the kernel  
kernel = np.array([[1, 2, 3],  
 [4, 5, 6],  
 [7, 8, 9]])  
  
# Perform convolution operations  
output\_valid = tf.nn.conv2d(input\_matrix.reshape(1, 5, 5, 1), kernel.reshape(3, 3, 1, 1), strides=[1, 1, 1, 1], padding='VALID')  
output\_same = tf.nn.conv2d(input\_matrix.reshape(1, 5, 5, 1), kernel.reshape(3, 3, 1, 1), strides=[1, 1, 1, 1], padding='SAME')  
output\_valid\_stride2 = tf.nn.conv2d(input\_matrix.reshape(1, 5, 5, 1), kernel.reshape(3, 3, 1, 1), strides=[1, 2, 2, 1], padding='VALID')  
output\_same\_stride2 = tf.nn.conv2d(input\_matrix.reshape(1, 5, 5, 1), kernel.reshape(3, 3, 1, 1), strides=[1, 2, 2, 1], padding='SAME')  
  
# Print the output feature maps  
print("Output (VALID):")  
print(output\_valid.numpy().reshape(3, 3))  
print("Output (SAME):")  
print(output\_same.numpy().reshape(5, 5))  
print("Output (VALID, stride=2):")  
print(output\_valid\_stride2.numpy().reshape(2, 2))  
print("Output (SAME, stride=2):")  
print(output\_same\_stride2.numpy().reshape(3, 3))

**Output :**

**A screenshot of a computer program

AI-generated content may be incorrect.**

## Question 3: CNN Feature Extraction with Filters and Pooling

### Task 1: Implement Edge Detection Using Convolution

import cv2

import numpy as np

import matplotlib.pyplot as plt

from urllib.request import urlopen

# Function to download and load a sample image

def load\_sample\_image():

# URL of a sample image (you can replace with your own image path)

url = "https://raw.githubusercontent.com/opencv/opencv/master/samples/data/lena.jpg"

img = cv2.imdecode(np.asarray(bytearray(urlopen(url).read()), dtype=np.uint8), cv2.IMREAD\_GRAYSCALE)

return img

# Load a grayscale image

image = load\_sample\_image()

# Apply the Sobel filter for edge detection in the x-direction

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

# Convert to absolute values and then to uint8 for display

sobel\_x\_abs = cv2.convertScaleAbs(sobel\_x)

# Apply the Sobel filter for edge detection in the y-direction

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

# Convert to absolute values and then to uint8 for display

sobel\_y\_abs = cv2.convertScaleAbs(sobel\_y)

# Display the original image and the filtered images

plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)

plt.title('Original Image')

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

plt.axis('off')

plt.subplot(1, 3, 2)

plt.title('Edge Detection using Sobel-X')

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

plt.axis('off')

plt.subplot(1, 3, 3)

plt.title('Edge Detection using Sobel-Y')

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

plt.axis('off')

plt.tight\_layout()

plt.show()

print("Task 1: Edge Detection using Sobel Filter completed.")A screenshot of a computer

AI-generated content may be incorrect.

### Task 2: Implement Max Pooling and Average Pooling

import numpy as np

import tensorflow as tf

# Create a random 4x4 matrix as an input image

np.random.seed(42) # For reproducibility

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

# Reshape the input matrix for TensorFlow operations

input\_tensor = tf.reshape(tf.constant(input\_matrix, dtype=tf.float32), [1, 4, 4, 1])

# Apply a 2x2 Max Pooling operation

max\_pooled = tf.nn.max\_pool2d(

input\_tensor,

ksize=[1, 2, 2, 1],

strides=[1, 2, 2, 1],

padding='VALID'

)

# Apply a 2x2 Average Pooling operation

avg\_pooled = tf.nn.avg\_pool2d(

input\_tensor,

ksize=[1, 2, 2, 1],

strides=[1, 2, 2, 1],

padding='VALID'

)

# Print the original matrix, max-pooled matrix, and average-pooled matrix

print("Original Matrix:")

print(input\_matrix)

print("\nMax Pooled Matrix:")

print(tf.squeeze(max\_pooled).numpy())

print("\nAverage Pooled Matrix:")

print(tf.squeeze(avg\_pooled).numpy())

print("\nTask 2: Pooling Operations on Random 4×4 Matrix completed.")A screenshot of a computer program

AI-generated content may be incorrect.

## Question 4: Implementing and Comparing CNN Architectures

### Task 1: Implement AlexNet Architecture

import tensorflow as tf

from tensorflow.keras.models import Sequential

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

# Define the AlexNet model

def create\_alexnet():

model = Sequential([

# First Convolutional Layer

Conv2D(96, kernel\_size=(11, 11), strides=4, activation='relu',

input\_shape=(227, 227, 3), padding='valid'),

MaxPooling2D(pool\_size=(3, 3), strides=2),

# Second Convolutional Layer

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

MaxPooling2D(pool\_size=(3, 3), strides=2),

# Third Convolutional Layer

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

# Fourth Convolutional Layer

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

# Fifth Convolutional Layer

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

MaxPooling2D(pool\_size=(3, 3), strides=2),

# Flatten Layer

Flatten(),

# First Fully Connected Layer

Dense(4096, activation='relu'),

Dropout(0.5),

# Second Fully Connected Layer

Dense(4096, activation='relu'),

Dropout(0.5),

# Output Layer

Dense(10, activation='softmax')

])

return model

# Create and print the AlexNet model summary

alexnet\_model = create\_alexnet()

print("AlexNet Model Summary:")

alexnet\_model.summary()

A screenshot of a computer

AI-generated content may be incorrect.

Task 2: Implement a Residual Block and ResNet

import tensorflow as tf

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Conv2D, Add, Activation, Flatten, Dense

# Define a residual block function

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

# First convolutional layer

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

# Second convolutional layer

x = Conv2D(filters, kernel\_size=(3, 3), padding='same')(x)

# Skip connection (add the input to the output)

x = Add()([x, input\_tensor])

# Apply activation after the addition

x = Activation('relu')(x)

return x

# Create a simple ResNet-like model

def create\_resnet():

# Input layer

input\_tensor = Input(shape=(224, 224, 3))

# Initial convolutional layer

x = Conv2D(64, kernel\_size=(7, 7), strides=2, padding='same', activation='relu')(input\_tensor)

# Apply two residual blocks

x = residual\_block(x, 64)

x = residual\_block(x, 64)

# Flatten layer

x = Flatten()(x)

# Dense layer

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

# Output layer

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

# Create the model

model = Model(inputs=input\_tensor, outputs=output)

return model

# Create and print the ResNet model summary

resnet\_model = create\_resnet()

print("\nResNet-like Model Summary:")

resnet\_model.summary()A screenshot of a computer program

AI-generated content may be incorrect.