```
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.models import Sequential

# 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]
]).reshape(1, 5, 5, 1)  # Reshape for Conv2D (batch_size, height, width, channels

# Define the kernel
kernel = np.array([
    [0, 1, 0],
    [1, -4, 1],
    [0, 1, 0]
]).reshape(3, 3, 1, 1)  # Reshape for Conv2D (height, width, in_channels, out_cha

# Function to perform convolution and print the output
def perform_convolution(input_matrix, kernel, stride, padding):
    model = Sequential()
    model.add(Conv2D(1, kernel_size=(3, 3), strides=(stride, stride), padding=pad
    model.layers[0].set_weights([kernel])
    output = model.predict(input_matrix)
    print(f"Stride = {stride}, Padding = '{padding}':\n{output.squeeze()}\n")

# Perform convolution with different parameters
perform_convolution(input_matrix, kernel, stride=1, padding='valid')
perform_convolution(input_matrix, kernel, stride=1, padding='same')
perform_convolution(input_matrix, kernel, stride=2, padding='valid')
perform_convolution(input_matrix, kernel, stride=2, padding='same')
```

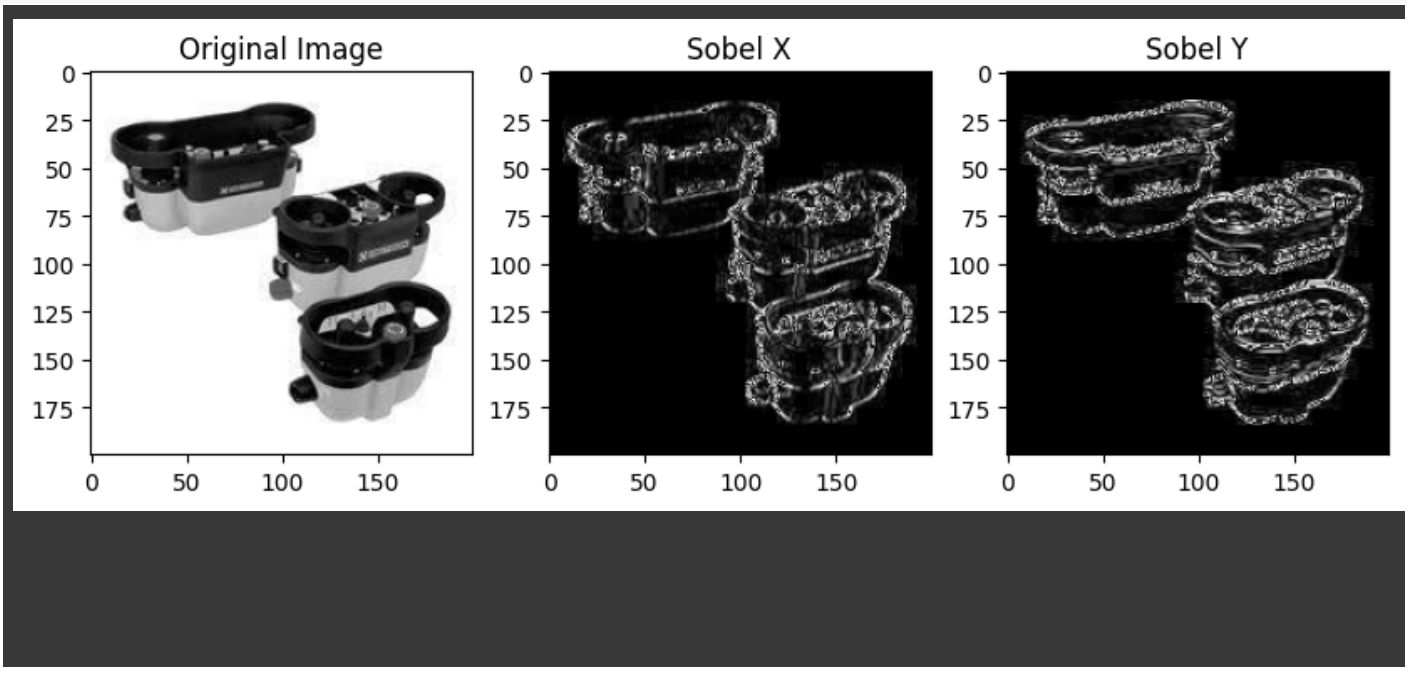
```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_c
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
Stride = 1, Padding = 'valid':
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 185ms/step
Stride = 1, Padding = 'same':
[[  4.   3.   2.   1.  -6.]
 [ -5.   0.   0.   0. -11.]
 [-10.   0.   0.   0. -16.]
 [-15.   0.   0.   0. -21.]
 [-46. -27. -28. -29. -56.]]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 193ms/step
Stride = 2, Padding = 'valid':
[[0. 0.]
 [0. 0.]]

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 144ms/step
Stride = 2, Padding = 'same':
[[  4.   2.  -6.]
 [-10.   0. -16.]
 [-46. -28. -56.]]
```

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

# Load a grayscale image
image_path = 'kick1.jpeg'  # Ensure this path is correct
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Check if the image was loaded correctly
if image is None:
    print(f"Error: Unable to load image at {image_path}")
    exit()
```

```
# Apply Sobel filter in x and y directions
sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)

# Convert the filtered images to absolute values and then to uint8
sobel_x = np.uint8(np.absolute(sobel_x))
sobel_y = np.uint8(np.absolute(sobel_y))

# Display the images
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: Implement Max Pooling and Average Pooling

```
import tensorflow as tf
import numpy as np

# Create a random 4x4 matrix
input_matrix = np.random.rand(1, 4, 4, 1)  # Reshape for Conv2D (batch_size, heig

# Define Max Pooling and Average Pooling
max_pool = tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2))
avg_pool = tf.keras.layers.AveragePooling2D(pool_size=(2, 2), strides=(2, 2))

# Apply pooling operations
max_pooled = max_pool(input_matrix)
avg_pooled = avg_pool(input_matrix)

# Print the results
print("Original Matrix:\n", input_matrix.squeeze())
print("Max Pooled Matrix:\n", max_pooled.numpy().squeeze())
print("Average Pooled Matrix:\n", avg_pooled.numpy().squeeze())
```

```
Original Matrix:
 [[0.34520768 0.32855385 0.97149048 0.40264211]
 [0.46554152 0.73863914 0.22548396 0.18768545]
 [0.53973444 0.15917988 0.6801603  0.908617  ]
 [0.46584542 0.28585449 0.86393516 0.42549066]]
Max Pooled Matrix:
 [[0.7386391 0.9714905]
 [0.5397344 0.908617 ]]
Average Pooled Matrix:
 [[0.46948555 0.4468255 ]
 [0.36265355 0.7195508 ]]
```

Q.4 Implementing and Comparing CNN Architectures

Task 1: Implement AlexNet Architecture

```
import tensorflow as tf
from tensorflow.keras import layers, models

# Define the AlexNet model
def build_alexnet(input_shape=(227, 227, 3), num_classes=10):
    model = models.Sequential()

    # Conv2D Layer: 96 filters, kernel size = (11,11), stride = 4, activation = R
```

```
    model.add(layers.Conv2D(96, (11, 11), strides=(4, 4), activation='relu', inpu

    # MaxPooling Layer: pool size = (3,3), stride = 2
    model.add(layers.MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))

    # Conv2D Layer: 256 filters, kernel size = (5,5), activation = ReLU
    model.add(layers.Conv2D(256, (5, 5), activation='relu'))

    # MaxPooling Layer: pool size = (3,3), stride = 2
    model.add(layers.MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))

    # Conv2D Layer: 384 filters, kernel size = (3,3), activation = ReLU
    model.add(layers.Conv2D(384, (3, 3), activation='relu'))

    # Conv2D Layer: 384 filters, kernel size = (3,3), activation = ReLU
    model.add(layers.Conv2D(384, (3, 3), activation='relu'))

    # Conv2D Layer: 256 filters, kernel size = (3,3), activation = ReLU
    model.add(layers.Conv2D(256, (3, 3), activation='relu'))

    # MaxPooling Layer: pool size = (3,3), stride = 2
    model.add(layers.MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))

    # Flatten Layer
    model.add(layers.Flatten())

    # Fully Connected (Dense) Layer: 4096 neurons, activation = ReLU
    model.add(layers.Dense(4096, activation='relu'))

    # Dropout Layer: 50%
    model.add(layers.Dropout(0.5))

    # Fully Connected (Dense) Layer: 4096 neurons, activation = ReLU
    model.add(layers.Dense(4096, activation='relu'))

    # Dropout Layer: 50%
    model.add(layers.Dropout(0.5))

    # Output Layer: 10 neurons, activation = Softmax
    model.add(layers.Dense(num_classes, activation='softmax'))

    return model

# Build the model
alexnet_model = build_alexnet()
```

```
# Print the model summary
alexnet_model.summary()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_co
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
```

Layer (type)	Output Shape	
conv2d (Conv2D)	(None, 55, 55, 96)	
max_pooling2d (MaxPooling2D)	(None, 27, 27, 96)	
conv2d_1 (Conv2D)	(None, 23, 23, 256)	
max_pooling2d_1 (MaxPooling2D)	(None, 11, 11, 256)	
conv2d_2 (Conv2D)	(None, 9, 9, 384)	
conv2d_3 (Conv2D)	(None, 7, 7, 384)	1
conv2d_4 (Conv2D)	(None, 5, 5, 256)	
max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 256)	
flatten (Flatten)	(None, 1024)	
dense (Dense)	(None, 4096)	4
dropout (Dropout)	(None, 4096)	
dense_1 (Dense)	(None, 4096)	16
dropout_1 (Dropout)	(None, 4096)	
dense_2 (Dense)	(None, 10)	

```
Total params: 24,767,882 (94.48 MB)
Trainable params: 24,767,882 (94.48 MB)
Non-trainable params: 0 (0.00 B)
```

Task 2: Implement a Residual Block and ResNet

```
import tensorflow as tf
from tensorflow.keras import layers, models

# Define a Residual Block
def residual_block(input_tensor, filters):
```

```
def residual_block(input_tensor, filters):
    x = layers.Conv2D(filters, (3, 3), padding='same', activation='relu')(input_ten
    x = layers.Conv2D(filters, (3, 3), padding='same')(x)
    x = layers.add([input_tensor, x])
    x = layers.Activation('relu')(x)
    return x

# Define the ResNet-like model
def build_resnet(input_shape=(224, 224, 3), num_classes=10):
    inputs = layers.Input(shape=input_shape)

    # Initial Conv2D layer
    x = layers.Conv2D(64, (7, 7), strides=(2, 2), padding='same', activation='relu'
    x = layers.MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding='same')(x)

    # Apply two residual blocks
    x = residual_block(x, 64)
    x = residual_block(x, 64)

    # Flatten layer
    x = layers.Flatten()(x)

    # Dense layer
    x = layers.Dense(128, activation='relu')(x)

    # Output layer
    outputs = layers.Dense(num_classes, activation='softmax')(x)

    # Create the model
    model = models.Model(inputs, outputs)
    return model

# Build the model
resnet_model = build_resnet()

# Print the model summary
resnet_model.summary()
```

Model: "functional_14"

Layer (type)	Output Shape	Param #	Connec
input_layer_1 (InputLayer)	(None, 224, 224, 3)	0	–
conv2d_5 (Conv2D)	(None, 112, 112, 64)	9,472	input_
max_pooling2d_3 (MaxPooling2D)	(None, 56, 56, 64)	0	conv2c
conv2d_6 (Conv2D)	(None, 56, 56, 64)	36,928	max_po
conv2d_7 (Conv2D)	(None, 56, 56, 64)	36,928	conv2c
add (Add)	(None, 56, 56, 64)	0	max_po conv2c
activation (Activation)	(None, 56, 56, 64)	0	add[0]
conv2d_8 (Conv2D)	(None, 56, 56, 64)	36,928	activa
conv2d_9 (Conv2D)	(None, 56, 56, 64)	36,928	conv2c
add_1 (Add)	(None, 56, 56, 64)	0	activa conv2c
activation_1 (Activation)	(None, 56, 56, 64)	0	add_1
flatten_1 (Flatten)	(None, 200704)	0	activa
dense_3 (Dense)	(None, 128)	25,690,240	flatte
dense_4 (Dense)	(None, 10)	1,290	dense_

Total params: 25,848,714 (98.61 MB)
Trainable params: 25,848,714 (98.61 MB)
Non-trainable params: 0 (0.00 B)