Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

Assignment – CNN Architecture

1. What is a Convolutional Neural Network (CNN), and why is it used for image processing?

➔ A Convolutional Neural Network (CNN) is a type of deep learning model designed specifically for processing data with a grid-like topology, such as images. It is particularly effective in image-related tasks due to its ability to capture spatial hierarchies and patterns.

Why CNN is used for image processing –

- CNNs excel in tasks such as object detection, image classification, semantic segmentation, and image generation.
- Unlike traditional methods that require manual feature engineering, CNNs learn the most important features directly from the data.
- Through feature maps and pooling, CNNs are robust to changes in the position of objects within an image.

2. What are the key components of a CNN architecture?

➔ The key components of a Convolutional Neural Network (CNN) architecture are designed to process and analyze grid-like data, such as images. Each component plays a specific role in extracting and interpreting features from the input data. Here's an overview of these components:

- Input Layer: Accepts the input data, typically an image represented as a tensor.
- Convolutional Layers: Perform convolution operations using filters (kernels) to extract local features.
- Activation Functions: Introduce non-linearity to the model, enabling it to learn complex patterns.
- Pooling Layers: Downsample the feature maps, reducing their spatial dimensions and computation requirements.
- Fully-Connected Layers: Connect all neurons from the previous layer to the next layer, acting as a classifier.
- Dropout Layers: Randomly deactivate neurons during training to prevent overfitting.
- Output Layers: Produces the final predictions or probabilities for the target output.

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

3. What is the role of the convolutional layer in CNNs?

➔ The convolutional layer is the foundational component of Convolutional Neural Networks (CNNs). Its primary role is to extract features from the input data by applying convolution operations. It transforms the input image or data into feature maps that highlight relevant patterns, making it easier for the network to understand and classify the data.

- Feature Extraction: Identifies local patterns such as edges, textures, or corners in the input data, early layers detect simple features (e.g., edges or lines), while deeper layers identify more complex patterns (e.g., shapes, objects).
- Creating Feature Maps: Produces feature maps by sliding filters over the input and computing dot products.
- Weight Sharing: Filters (or kernels) have shared weights across the input image, making the model parameter-efficient compared to fully connected layers.
- Reducing Dimensionality with Strides: Uses strides (steps taken by the filter) to control how much the input data is downsampled.

4. What is a filter (kernel) in CNNs?

➔ A filter (also known as a kernel) in Convolutional Neural Networks (CNNs) is a small matrix of weights used to extract features from the input data through the convolution operation. Filters are the fundamental building blocks of the convolutional layer, enabling the network to detect patterns like edges, textures, and complex shapes in images.

How Filter Works –

- The filter slides across the input image (or feature map) with a defined stride.
- The result of the convolution operation is stored in a feature map.
- Each filter produces a separate feature map, which is stacked together to form the output of the layer.

5. What is pooling in CNNs, and why is it important?

➔ Pooling is a downsampling operation used in Convolutional Neural Networks (CNNs) to reduce the spatial dimensions (height and width) of feature maps while retaining the most important information. Pooling layers follow convolutional layers in a CNN architecture and play a vital role in improving the model's efficiency and robustness.

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

Importance of Pooling in CNNs –

- Dimensionality Reduction: Reduces the spatial size of feature maps, leading to lower computational requirements and memory usage.
- Prevention of Overfitting: Acts as a regularizer by simplifying the representation of feature maps.
- Feature Invariance: Makes the model less sensitive to small translations or distortions in the input, improving generalization.
- Enhancing Computational Efficiency: By reducing the size of the data, pooling speeds up training and inference without significantly losing important information.

6. What are the common types of pooling used in CNNs?

➔ The common types of pooling used in **Convolutional Neural Networks (CNNs)** are techniques for downsampling feature maps, reducing their spatial dimensions while retaining essential information. Each type of pooling has a unique approach to summarizing features.

- Max Pooling: Select the maximum value from each pooling window, captures the most prominent features in a region. Makes the model focus on the strongest activations, often representing the most significant patterns.
- Average Pooling: Computes the average of all values in the pooling window, smoothens features, and is less aggressive than max pooling.
- Global Pooling: Reduces each feature map to a single value by applying a pooling operation (max or average) across the entire spatial dimensions.

7. How does the backpropagation algorithm work in CNNs?

➔ The backpropagation algorithm in Convolutional Neural Networks (CNNs) is a process of updating the weights and biases of the network to minimize the error between the predicted and actual outputs. This algorithm works by calculating the gradient of the loss function with respect to each parameter through a series of steps involving the chain rule of calculus.

Steps in Backpropagation for CNNs-

- Forward Pass: Input data passes through the network, layer by layer, producing an output. The loss function computes the error by comparing the output to the actual labels.
-

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

- Backward Pass: Gradients of the loss function with respect to each parameter (weights and biases) are calculated, layer by layer, starting from the output layer and moving backward to the input layer.
- Fully Connected Layers: Compute the gradient of the loss function with respect to the layer's outputs. Use the chain rule to calculate gradients for the weights and biases. Update the parameters using these gradients.
- Convolutional Layers: The gradient of the loss with respect to the filter weights is calculated using the output error from the current layer and the input from the previous layer. The filters (kernels) are updated based on the gradients to better detect relevant features. The error is propagated backward through the convolution operation to earlier layers.
- Pooling Layers: Pooling layers do not have trainable parameters, During backpropagation, the error is routed to the locations of the maximum values (in max pooling) or distributed evenly (in average pooling).

8. What is the role of activation functions in CNNs?

➔ Activation functions play a crucial role in **Convolutional Neural Networks (CNNs)** by introducing non-linearity into the model. This non-linearity enables the network to learn and model complex patterns in the data, making it possible to solve real-world problems such as image classification, object detection, and more.

Roles of activation function in CNNs-

- Introducing Non Linearity
- Enabling Hierarchical Feature Learning
- Controlling Information Flow
- Ensuring Differentiability
- Preventing Gradient Vanishing or Exploding

9. What is the concept of receptive fields in CNNs?

➔ The receptive field in a Convolutional Neural Network (CNN) refers to the region of the input image that influences the activation of a particular neuron in a given layer. It represents how much of the input data a neuron "sees" and responds to during processing.

Significance of Receptive Fields-

- Local Feature Detection

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

- Global Context Understanding
- Trade-offs
- Architecture Design

Receptive fields are foundational to understanding how CNNs process spatial information in images. By controlling the size of the receptive field, CNNs achieve a balance between capturing fine details and understanding global patterns, enabling them to perform tasks like image classification, object detection, and segmentation effectively.

10. Explain the concept of tensor space in CNNs.

➔ The tensor space in Convolutional Neural Networks (CNNs) refers to the multi-dimensional data representation used to process and transform inputs (e.g., images) as they pass through the layers of the network. Tensors are generalizations of scalars, vectors, and matrices to higher dimensions, and they are central to the mathematical framework of CNNs.

Tensor space in CNNs provides a structured, multi-dimensional representation of data as it flows through the network. It enables efficient data transformation, hierarchical feature extraction, and scalable computation, which are essential for the success of CNNs in tasks like image and video processing.

11. What is LeNet-5, and how does it contribute to the development of CNNs?

➔ LeNet-5 is a pioneering Convolutional Neural Network (CNN) architecture developed by Yann LeCun and his collaborators in the late 1990s. It is one of the first deep learning models specifically designed for image recognition and laid the groundwork for the development of modern CNNs. LeNet-5's success demonstrated the potential of CNNs for complex visual tasks and sparked further research and development in the field of deep learning.

Contribution to the development of CNNs-

- Demonstrating the Potential of CNNs: LeNet-5 demonstrated that deep, convolutional networks could effectively learn complex patterns and achieve high accuracy in image classification tasks, setting a precedent for future research.

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

- Pioneering Layer Design: Introduced the combination of convolutional and pooling layers, which became standard for extracting features and down-sampling data in subsequent CNNs.
- End-to-End Learning: LeNet-5 was one of the first models to use backpropagation effectively for training deep architectures, demonstrating that it was possible to train networks with multiple convolutional and fully connected layers using gradient-based optimization.

12. What is AlexNet, and why was it a breakthrough in deep learning?

➔ AlexNet consists of 8 layers: 5 convolutional layers followed by 3 fully connected layers, The architecture is deeper than the previous CNNs like LeNet-5, which helped it learn more complex features. Max pooling is used after certain convolutional layers to reduce the spatial size of the feature maps and control overfitting.

AlexNet's architecture and approach demonstrated that deep neural networks could achieve state-of-the-art performance in image classification tasks by utilizing deeper structures, GPU acceleration, and advanced techniques like ReLU, dropout, and data augmentation. This marked a pivotal moment in the evolution of machine learning, cementing deep learning as a dominant paradigm in AI research and applications.

13. What is VGGNet, and how does it differ from AlexNet?

➔ VGGNet is a deep convolutional neural network architecture, it gained significant attention after achieving top performance in the 2014 ImageNet Large Scale Visual Recognition Challenge (ILSVRC). VGGNet is known for its simplicity and effectiveness, which has made it a benchmark in the study of deep learning and CNN architectures.

Differences Between VGGNet and AlexNet-

- Depth: AlexNet has 8 layers (5 convolutional and 3 fully connected), while VGGNet is deeper, with 16 to 19 layers. The increased depth of VGGNet allows it to learn more complex representations.
- Kernal Size: AlexNet uses larger kernels whereas VGGNet exclusively uses small convolutional filters. This design choice results in more layers and a deeper network with the same receptive field as larger kernels, but with fewer parameters.
- Number of Parameters: AlexNet has approximately 60 million parameters, while VGGNet has 138 million parameters for VGG16 and 143 million for VGG19. Despite VGGNet's higher number of parameters, it still demonstrates superior performance due to its deeper architecture and use of small filters.

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

14. What is GoogLeNet, and what is its main innovation?

➔ GoogLeNet is particularly noted for its unique architectural innovation, which dramatically improves computational efficiency while maintaining or improving accuracy.

The main innovation behind **GoogLeNet** is the **Inception module**. This module is designed to efficiently extract features at multiple scales within a single layer by using a combination of various filter sizes.

GoogLeNet is a deep CNN architecture known for its innovative Inception modules that allow parallel convolutional operations at different filter sizes, reducing the number of parameters and improving computational efficiency. With auxiliary classifiers and global average pooling, GoogLeNet set a new standard in CNN design, leading to a more efficient use of resources and better performance on large-scale image classification tasks. Its principles have inspired the development of more advanced networks and have had a lasting impact on the field of computer vision.

15. What is ResNet, and what problem does it solve?

➔ ResNet (Residual Networks) is a deep convolutional neural network architecture, which help address the problem of training deep neural networks effectively.

Problems solve by ResNet-

- Vanishing Gradient Problem: One of the main challenges in training deep neural networks is the vanishing gradient problem, where gradients become extremely small as they are backpropagated through many layers. This can lead to very slow training or complete failure to learn, especially in very deep networks.
- Degradation Problem: As networks become deeper, adding more layers does not necessarily improve performance and can even make it worse. This is known as the degradation problem, where deeper networks perform worse than their shallower counterparts, despite having more capacity.

16. What is DenseNet, and how does it differ from ResNet?

➔ DenseNet (Densely Connected Convolutional Networks) is an advanced deep learning architecture that was introduced in the paper **“**Densely Connected Convolutional Networks”, This architecture aims to improve upon the flow of information and gradients during training, making it more efficient and effective for deep network training.

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

Differences Between DenseNet and ResNet-

- Connection Pattern: ResNet uses residual connections where the input to a block is added to the output of that block. This allows for the direct flow of gradients but doesn't provide the same level of feature sharing as DenseNet.
- Gradient Flow: Both architectures help with the vanishing gradient problem, but DenseNet improves upon ResNet by allowing more direct paths for gradient flow due to its layer-wise connections.
- Parameter Efficiency: DenseNet can achieve better parameter efficiency than ResNet. The feature maps from earlier layers are reused extensively, which reduces the number of parameters needed for the network. This is in contrast to ResNet, where each layer learns its own set of features independently.

17. What are the main steps involved in training a CNN from scratch?

➔ Training a Convolutional Neural Network (CNN) from scratch involves several important steps. These steps encompass data preparation, model design, training, evaluation, and tuning. Here's a detailed breakdown of the main steps involved:

- Data Collection and Preparation: Collect or access a dataset relevant to the task (e.g., images for image classification or object detection). Divide the dataset into training, validation, and test sets (e.g., 70% training, 15% validation, and 15% testing).
- Model Design and Architecture: Design a custom CNN architecture or start with an established architecture like LeNet-5, AlexNet, VGGNet, etc.
- Compilation and Configuration: Choose an appropriate loss function based on the task. For example, categorical cross-entropy for multi-class classification, binary cross-entropy for binary classification, or mean squared error for regression.

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

Practical

1. Implement a basic convolution operation using a filter and a 5x5 image.

➔ CODE:

```python
#1.Implement a basic convolution operation using a filter and a 5x5 image (matrix).
import numpy as np

# Define the 5x5 image matrix
image = np.array([
    [1, 2, 3, 0, 1],
    [4, 5, 6, 1, 2],
    [7, 8, 9, 2, 3],
    [1, 3, 5, 3, 4],
    [0, 2, 4, 4, 5]
])

# Define the 3x3 filter (kernel)
kernel = np.array([
    [1, 0, -1],
    [1, 0, -1],
    [1, 0, -1]
])

# Perform convolution without padding and stride of 1
output_shape = (image.shape[0] - kernel.shape[0] + 1, image.shape[1] - kernel.shape[1] + 1)
output = np.zeros(output_shape)

for i in range(output_shape[0]):
    for j in range(output_shape[1]):
        region = image[i:i+kernel.shape[0], j:j+kernel.shape[1]]
        output[i, j] = np.sum(region * kernel)

print("Output of the convolution:")
print(output)
```

1.1s

OUTPUT:

```
1.1s
Output of the convolution:
[[ -6.  12.  12.]
 [ -8.  10.  11.]
 [-10.   4.   6.]]
```

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

2. Implement max pooling on a 4*4 feature map with a 2*2 window.

→ CODE:

```python
#2.Implement max pooling on a 4x4 feature map with a 2x2 window.
import numpy as np

# Define a 4x4 feature map
feature_map = np.array([
    [1, 3, 2, 4],
    [5, 6, 7, 8],
    [4, 2, 3, 1],
    [9, 5, 6, 7]
])

# Define the size of the pooling window
pool_size = 2

# Calculate the output size
output_size = (feature_map.shape[0] // pool_size, feature_map.shape[1] // pool_size)
output = np.zeros(output_size)

# Perform max pooling
for i in range(0, feature_map.shape[0], pool_size):
    for j in range(0, feature_map.shape[1], pool_size):
        # Get the current window
        window = feature_map[i:i+pool_size, j:j+pool_size]
        # Get the maximum value in the window
        output[i//pool_size, j//pool_size] = np.max(window)

# Print the output
print("Output after max pooling:")
print(output)
```

OUTPUT:

```
Output after max pooling:
[[6. 8.]
 [9. 7.]]
```

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

3. Implement the RELU activation function on a feature map.

→ CODE:

```python
#3.Implement the ReLU activation function on a feature map
import numpy as np

# Define a sample feature map
feature_map = np.array([
    [[1, -2, 3, -4],
     [-5, 6, -7, 8],
     [9, -10, 11, -12],
     [-13, 14, -15, 16]]
])

# Apply ReLU activation function
relu_output = np.maximum(0, feature_map)

# Print the output
print("Output after applying ReLU:")
print(relu_output)
```

✓ 0.0s

OUTPUT:

```
Output after applying ReLU:
[[[ 1  0  3  0]
  [ 0  6  0  8]
  [ 9  0 11  0]
  [ 0 14  0 16]]]
```

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

4. Create a simple CNN model with one convolutional layer and a fully connected layer, using random data.

➔ CODE:

```python
#4.Create a simple CNN model with one convolutional layer and a fully connected layer, using random data.
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models

# Create random data for the input (e.g., 10 samples of 32x32 images with 3 color channels)
input_data = np.random.rand(10, 32, 32, 3).astype(np.float32)

# Create a simple CNN model with one convolutional layer and one fully connected layer
model = models.Sequential([
    # Convolutional layer with 8 filters, a 3x3 kernel, and ReLU activation
    layers.Conv2D(8, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    # Flatten layer to convert the 2D output into 1D for the dense layer
    layers.Flatten(),
    # Fully connected (dense) layer with 10 units (e.g., for classification)
    layers.Dense(10, activation='softmax')
])

# Print the summary of the model to verify its structure
model.summary()

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Generate random labels for training (one-hot encoded)
labels = np.random.randint(0, 2, size=(10, 10))  # Random one-hot labels for 10 samples

# Train the model with the random data (this step is just for demonstration)
model.fit(input_data, labels, epochs=1)
```

✓ 15.1s

OUTPUT:

```
c:\Users\Mansi Dobriyal\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `in
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 8)	224
flatten (Flatten)	(None, 7200)	0
dense (Dense)	(None, 10)	72,010

```
 Total params: 72,234 (282.16 KB)

 Trainable params: 72,234 (282.16 KB)

 Non-trainable params: 0 (0.00 B)

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 871ms/step - accuracy: 0.0000e+00 - loss: 17.0557

<keras.src.callbacks.history.History at 0x26aada4c1d0>
```

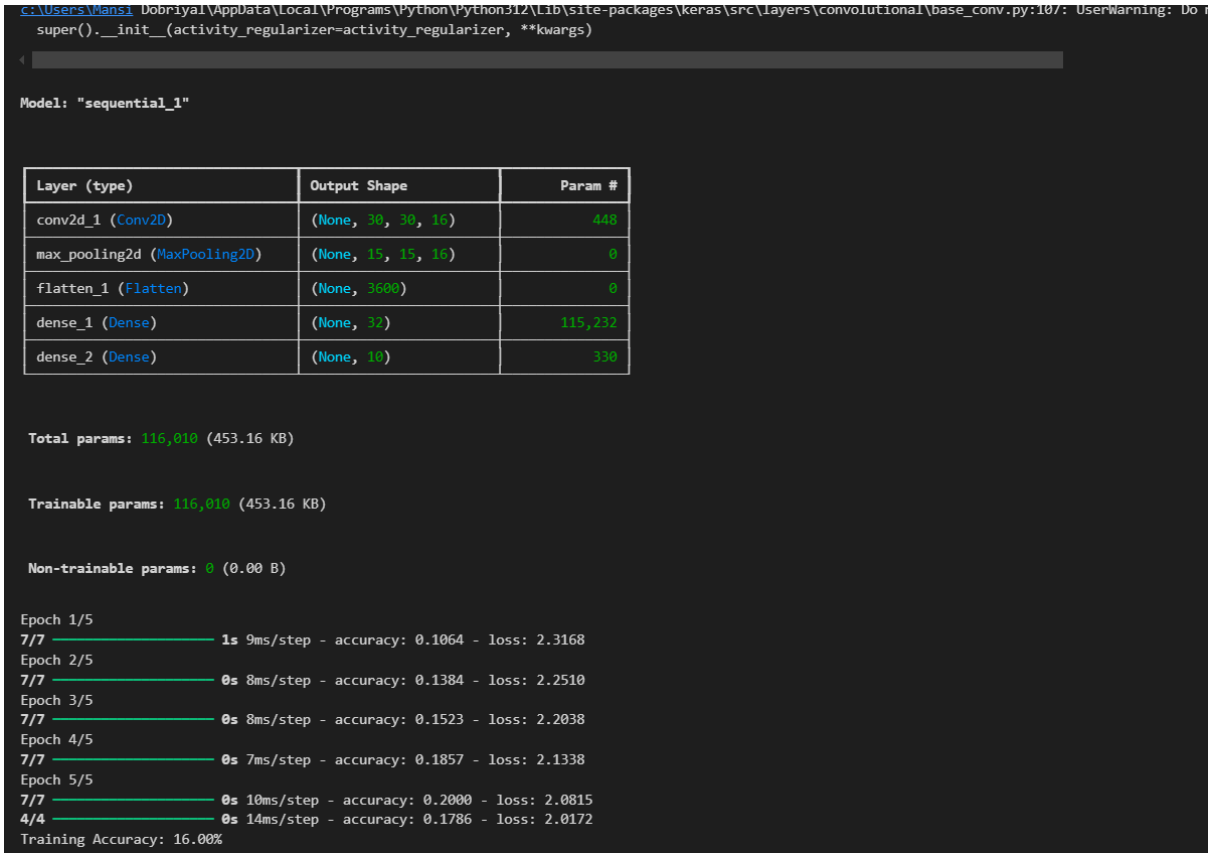
Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

5. Generate a synthetic dataset using random noise and train a simple CNN model on it.

➔ CODE:

```
#5.Generate a synthetic dataset using random noise and train a simple CNN model on it
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
# Step 1: Generate synthetic dataset
# Create random images (e.g., 100 samples of 32x32 RGB images)
num_samples = 100
image_height, image_width, num_channels = 32, 32, 3
num_classes = 10
# Random noise as image data
x_data = np.random.rand(num_samples, image_height, image_width, num_channels).astype(np.float32)
# Random labels (one-hot encoded)
y_data = tf.keras.utils.to_categorical(np.random.randint(0, num_classes, num_samples), num_classes)
# Step 2: Define a simple CNN model
model = models.Sequential([
    layers.Conv2D(16, (3, 3), activation='relu', input_shape=(image_height, image_width, num_channels)),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(32, activation='relu'),
    layers.Dense(num_classes, activation='softmax')
])
# Step 3: Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Print model summary
model.summary()
# Step 4: Train the model
history = model.fit(x_data, y_data, epochs=5, batch_size=16)
# Step 5: Evaluate the model
loss, accuracy = model.evaluate(x_data, y_data)
print(f"Training Accuracy: {accuracy * 100:.2f}%")
```

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903
OUTPUT:



Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

6. Create a simple CNN using keras with one convolutional layer and a max-pooling layer.

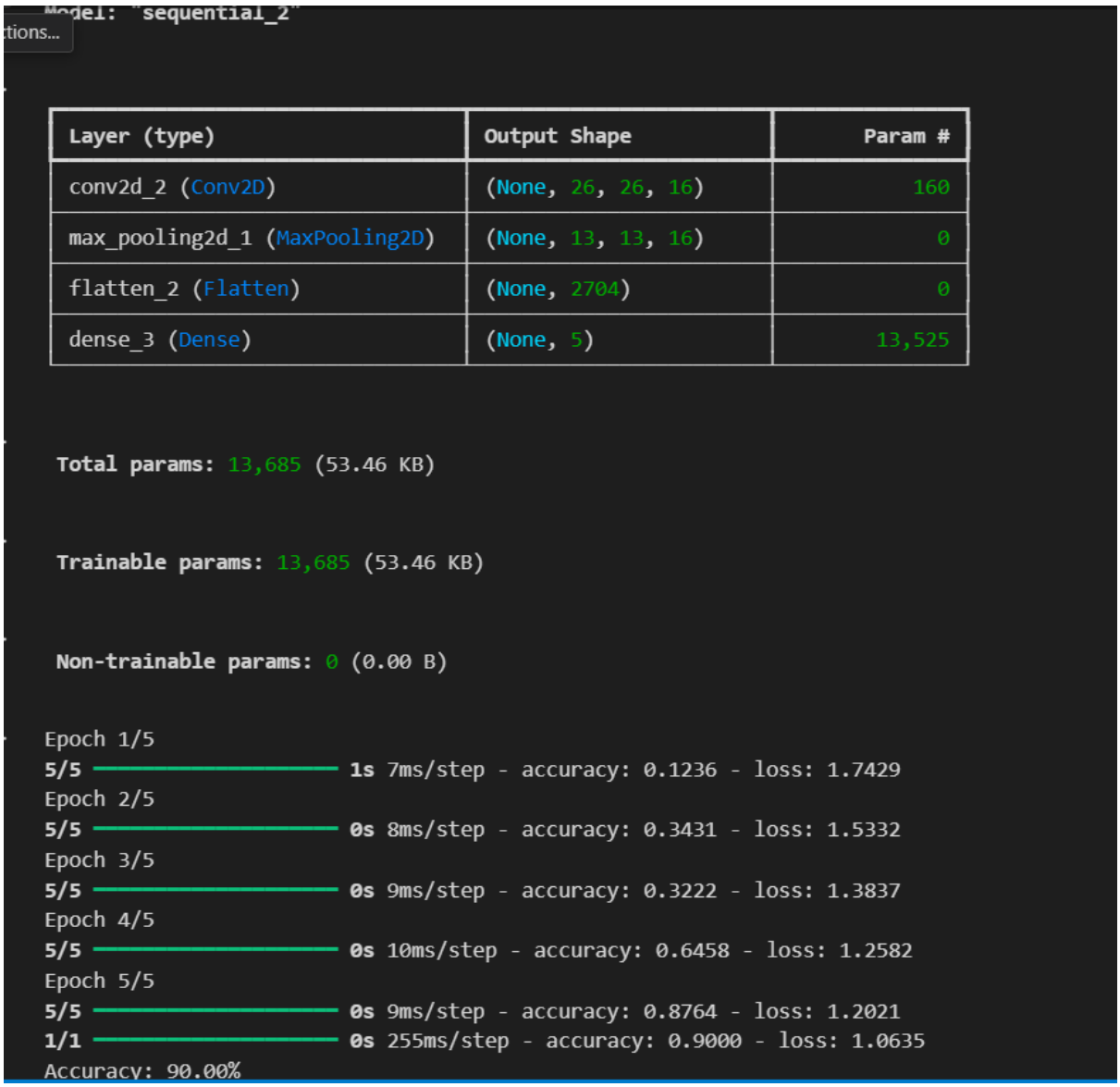
➔ CODE:

```python
#6.Create a simple CNN using Keras with one convolution layer and a max-pooling layer
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
# Step 1: Generate random data for demonstration
# 10 samples of 28x28 grayscale images (e.g., like MNIST images)
num_samples = 10
image_height, image_width, num_channels = 28, 28, 1
# Random image data
x_data = np.random.rand(num_samples, image_height, image_width, num_channels).astype(np.float32)
# Random labels (one-hot encoded) for 5 classes
num_classes = 5
y_data = tf.keras.utils.to_categorical(np.random.randint(0, num_classes, num_samples), num_classes)
# Step 2: Define a simple CNN model
model = models.Sequential([
    # Convolutional layer with 16 filters, 3x3 kernel, ReLU activation
    layers.Conv2D(16, (3, 3), activation='relu', input_shape=(image_height, image_width, num_channels)),
    # Max-pooling layer with 2x2 pool size
    layers.MaxPooling2D(pool_size=(2, 2)),
    # Flatten layer to prepare for dense layer
    layers.Flatten(),
    # Fully connected layer with 5 output units (for classification)
    layers.Dense(num_classes, activation='softmax')
])
# Step 3: Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Print the model summary
model.summary()
# Step 4: Train the model (using synthetic data)
history = model.fit(x_data, y_data, epochs=5, batch_size=2)
# Step 5: Evaluate the model
loss, accuracy = model.evaluate(x_data, y_data)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

✓ 1.3s

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

OUTPUT:



Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 26, 26, 16)	160
max_pooling2d_1 (MaxPooling2D)	(None, 13, 13, 16)	0
flatten_2 (Flatten)	(None, 2704)	0
dense_3 (Dense)	(None, 5)	13,525

Total params: 13,685 (53.46 KB)

Trainable params: 13,685 (53.46 KB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/5
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.1236 - loss: 1.7429
Epoch 2/5
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.3431 - loss: 1.5332
Epoch 3/5
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.3222 - loss: 1.3837
Epoch 4/5
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.6458 - loss: 1.2582
Epoch 5/5
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.8764 - loss: 1.2021
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 255ms/step - accuracy: 0.9000 - loss: 1.0635
Accuracy: 90.00%
```

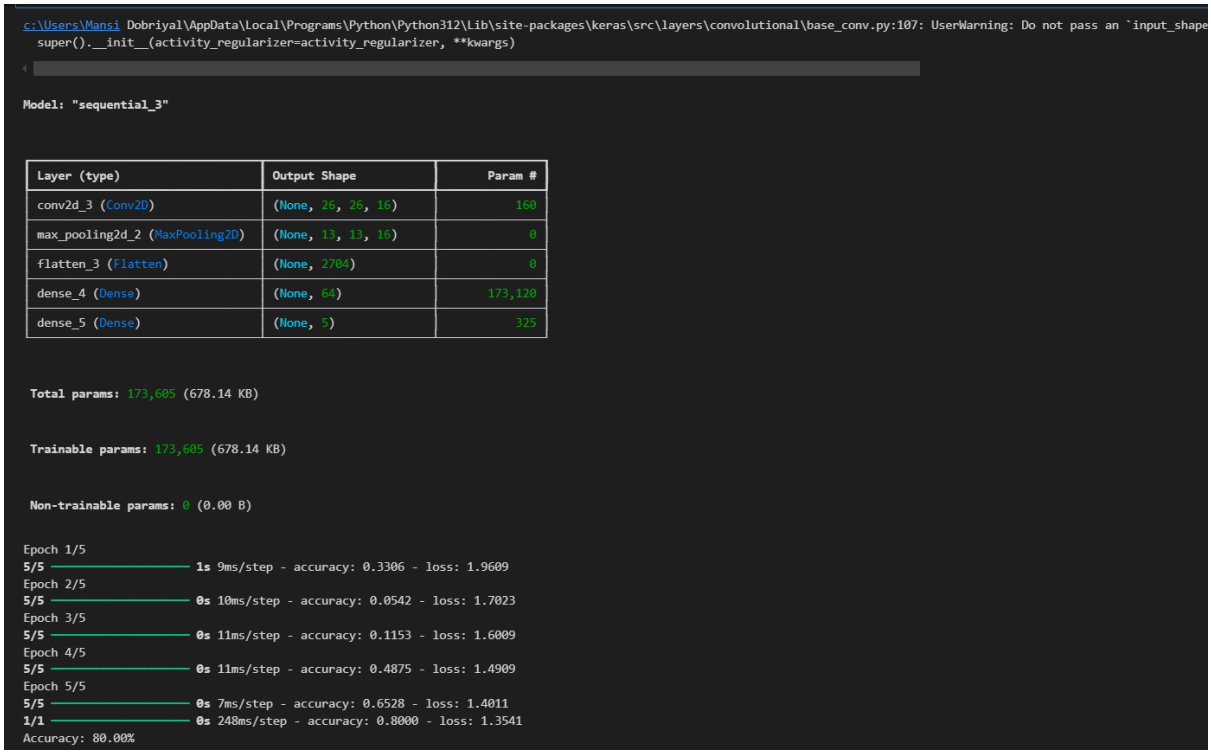
Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

7. Write a code to add a fully connected layer after the convolution and max-pooling layers in a CNN.

→ CODE:

```python
#7.Write a code to add a fully connected layer after the convolution and max-pooling layers in a CNN
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
# Step 1: Generate random data for demonstration
# 10 samples of 28x28 grayscale images (e.g., like MNIST images)
num_samples = 10
image_height, image_width, num_channels = 28, 28, 1
# Random image data
x_data = np.random.rand(num_samples, image_height, image_width, num_channels).astype(np.float32)
# Random labels (one-hot encoded) for 5 classes
num_classes = 5
y_data = tf.keras.utils.to_categorical(np.random.randint(0, num_classes, num_samples), num_classes)
# Step 2: Define a CNN model
model = models.Sequential([
    # Convolutional layer with 16 filters, 3x3 kernel, ReLU activation
    layers.Conv2D(16, (3, 3), activation='relu', input_shape=(image_height, image_width, num_channels)),
    # Max-pooling layer with 2x2 pool size
    layers.MaxPooling2D(pool_size=(2, 2)),
    # Flatten layer to prepare for dense layer
    layers.Flatten(),
    # Fully connected layer with 64 units
    layers.Dense(64, activation='relu'),
    # Output layer with 5 units (for classification)
    layers.Dense(num_classes, activation='softmax')
])
# Step 3: Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Print the model summary
model.summary()
# Step 4: Train the model (using synthetic data)
history = model.fit(x_data, y_data, epochs=5, batch_size=2)
# Step 5: Evaluate the model
loss, accuracy = model.evaluate(x_data, y_data)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903
OUTPUT:



```
c:\Users\Mansi Dobriyal\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `input_shape
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 26, 26, 16)	160
max_pooling2d_2 (MaxPooling2D)	(None, 13, 13, 16)	0
flatten_3 (Flatten)	(None, 2704)	0
dense_4 (Dense)	(None, 64)	173,120
dense_5 (Dense)	(None, 5)	325

```
 Total params: 173,605 (678.14 KB)

 Trainable params: 173,605 (678.14 KB)

 Non-trainable params: 0 (0.00 B)

Epoch 1/5
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.3306 - loss: 1.9609
Epoch 2/5
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.0542 - loss: 1.7023
Epoch 3/5
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.1153 - loss: 1.6009
Epoch 4/5
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.4875 - loss: 1.4909
Epoch 5/5
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6528 - loss: 1.4011
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 248ms/step - accuracy: 0.8000 - loss: 1.3541
Accuracy: 80.00%
```

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

8. Write a code to add batch normalization to a simple CNN model.

→ CODE:

```python
#8.Write a code to add  batch normalization to a simple CNN model
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
# Step 1: Generate random data for demonstration(10 samples of 28x28 grayscale images)
num_samples = 10
image_height, image_width, num_channels = 28, 28, 1
# Random image data
x_data = np.random.rand(num_samples, image_height, image_width, num_channels).astype(np.float32)
# Random labels (one-hot encoded) for 5 classes
num_classes = 5
y_data = tf.keras.utils.to_categorical(np.random.randint(0, num_classes, num_samples), num_classes)
# Step 2: Define a CNN model with batch normalization
model = models.Sequential([
    # Convolutional layer
    layers.Conv2D(16, (3, 3), activation='relu', input_shape=(image_height, image_width, num_channels)),
    # Batch normalization after convolution
    layers.BatchNormalization(),
    # Max-pooling layer
    layers.MaxPooling2D(pool_size=(2, 2)),
    # Another convolutional layer
    layers.Conv2D(32, (3, 3), activation='relu'),
    # Batch normalization after the second convolution
    layers.BatchNormalization(),
    # Max-pooling layer
    layers.MaxPooling2D(pool_size=(2, 2)),
    # Flatten layer to prepare for dense layer
    layers.Flatten(),
    # Fully connected layer with 64 units
    layers.Dense(64, activation='relu'),
    # Batch normalization before the output layer
    layers.BatchNormalization(),
    # Output layer with 5 units (for classification)
    layers.Dense(num_classes, activation='softmax')
])
# Step 3: Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Print the model summary
model.summary()
# Step 4: Train the model (using synthetic data)
history = model.fit(x_data, y_data, epochs=5, batch_size=2)
# Step 5: Evaluate the model
loss, accuracy = model.evaluate(x_data, y_data)
```

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903
OUTPUT:

```
Model: "sequential_4"
```

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 26, 26, 16)	160
batch_normalization (BatchNormalization)	(None, 26, 26, 16)	64
max_pooling2d_3 (MaxPooling2D)	(None, 13, 13, 16)	0
conv2d_5 (Conv2D)	(None, 11, 11, 32)	4,640
batch_normalization_1 (BatchNormalization)	(None, 11, 11, 32)	128
max_pooling2d_4 (MaxPooling2D)	(None, 5, 5, 32)	0
flatten_4 (Flatten)	(None, 800)	0
dense_6 (Dense)	(None, 64)	51,264
batch_normalization_2 (BatchNormalization)	(None, 64)	256
dense_7 (Dense)	(None, 5)	325

```
Total params: 56,837 (222.02 KB)

Trainable params: 56,613 (221.14 KB)

Non-trainable params: 224 (896.00 B)

Epoch 1/5
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - accuracy: 0.2069 - loss: 2.1114
Epoch 2/5
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.5833 - loss: 1.3493
Epoch 3/5
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.2736 - loss: 1.5277
Epoch 4/5
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8222 - loss: 0.8682
Epoch 5/5
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.7681 - loss: 0.8427
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 209ms/step - accuracy: 0.1000 - loss: 1.5273
Accuracy: 10.00%
```

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

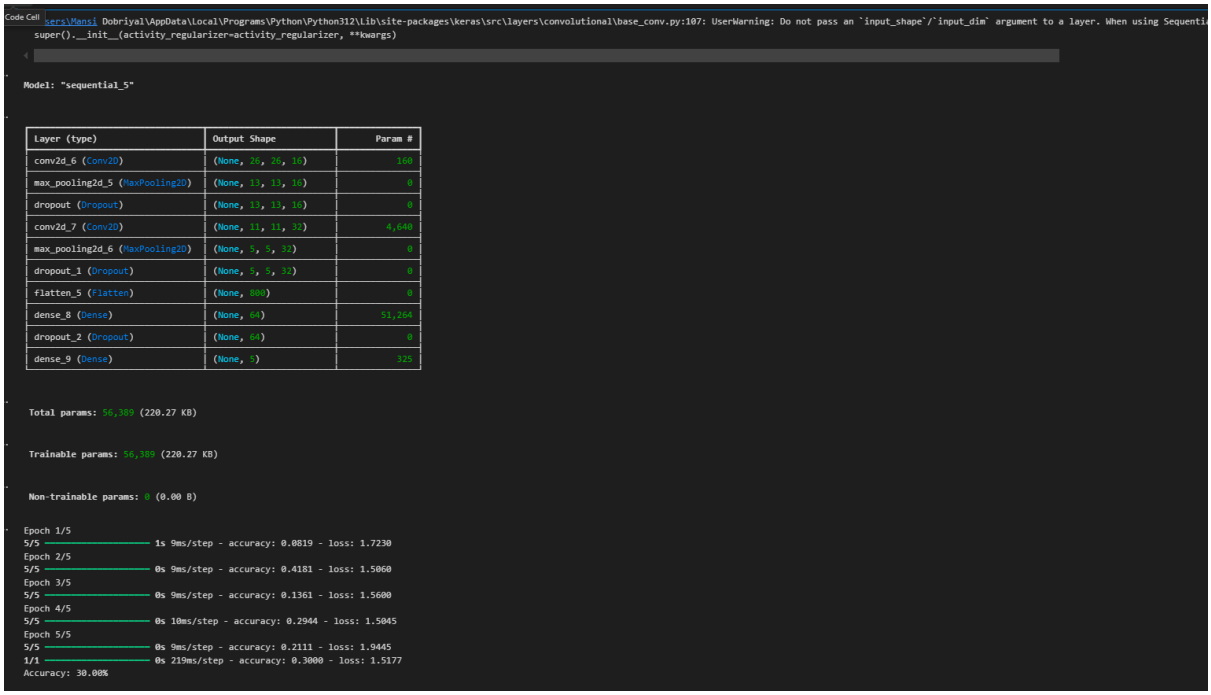
9. Write a code to add dropout regularization to a simple CNN model.

→ CODE:

```python
#9.Write a code to add dropout regularization to a simple CNN model
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
# Step 1: Generate random data for demonstration
# 10 samples of 28x28 grayscale images
num_samples = 10
image_height, image_width, num_channels = 28, 28, 1
# Random image data
x_data = np.random.rand(num_samples, image_height, image_width, num_channels).astype(np.float32)
# Random labels (one-hot encoded) for 5 classes
num_classes = 5
y_data = tf.keras.utils.to_categorical(np.random.randint(0, num_classes, num_samples), num_classes)
# Step 2: Define a CNN model with dropout regularization
model = models.Sequential([
    # Convolutional layer
    layers.Conv2D(16, (3, 3), activation='relu', input_shape=(image_height, image_width, num_channels)),
    # Max-pooling layer
    layers.MaxPooling2D(pool_size=(2, 2)),
    # Dropout layer to regularize
    layers.Dropout(0.25),  # 25% dropout

    # Second convolutional layer
    layers.Conv2D(32, (3, 3), activation='relu'),
    # Max-pooling layer
    layers.MaxPooling2D(pool_size=(2, 2)),
    # Dropout layer
    layers.Dropout(0.25),  # 25% dropout
    # Flatten layer to prepare for dense layers
    layers.Flatten(),
    # Fully connected layer with 64 units
    layers.Dense(64, activation='relu'),
    # Dropout layer before the output
    layers.Dropout(0.5),  # 50% dropout
    # Output layer with 5 units (for classification)
    layers.Dense(num_classes, activation='softmax')
])
# Step 3: Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Print the model summary
model.summary()
# Step 4: Train the model (using synthetic data)
history = model.fit(x_data, y_data, epochs=5, batch_size=2)
# Step 5: Evaluate the model
loss, accuracy = model.evaluate(x_data, y_data)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

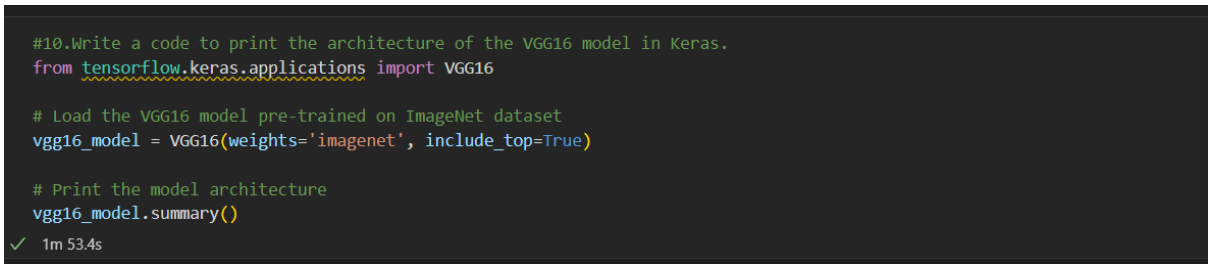
✓ 2.1s

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903
OUTPUT:



10. Write a code to print the architecture of the VGG16 model in Keras.

CODE:



Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903
OUTPUT:

Layer (type)	Output Shape	Param #
input_layer_6 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102,764,544
fc2 (Dense)	(None, 4096)	16,781,312
predictions (Dense)	(None, 1000)	4,097,000

Total params: 138,357,544 (527.79 MB)

Trainable params: 138,357,544 (527.79 MB)

Non-trainable params: 0 (0.00 B)

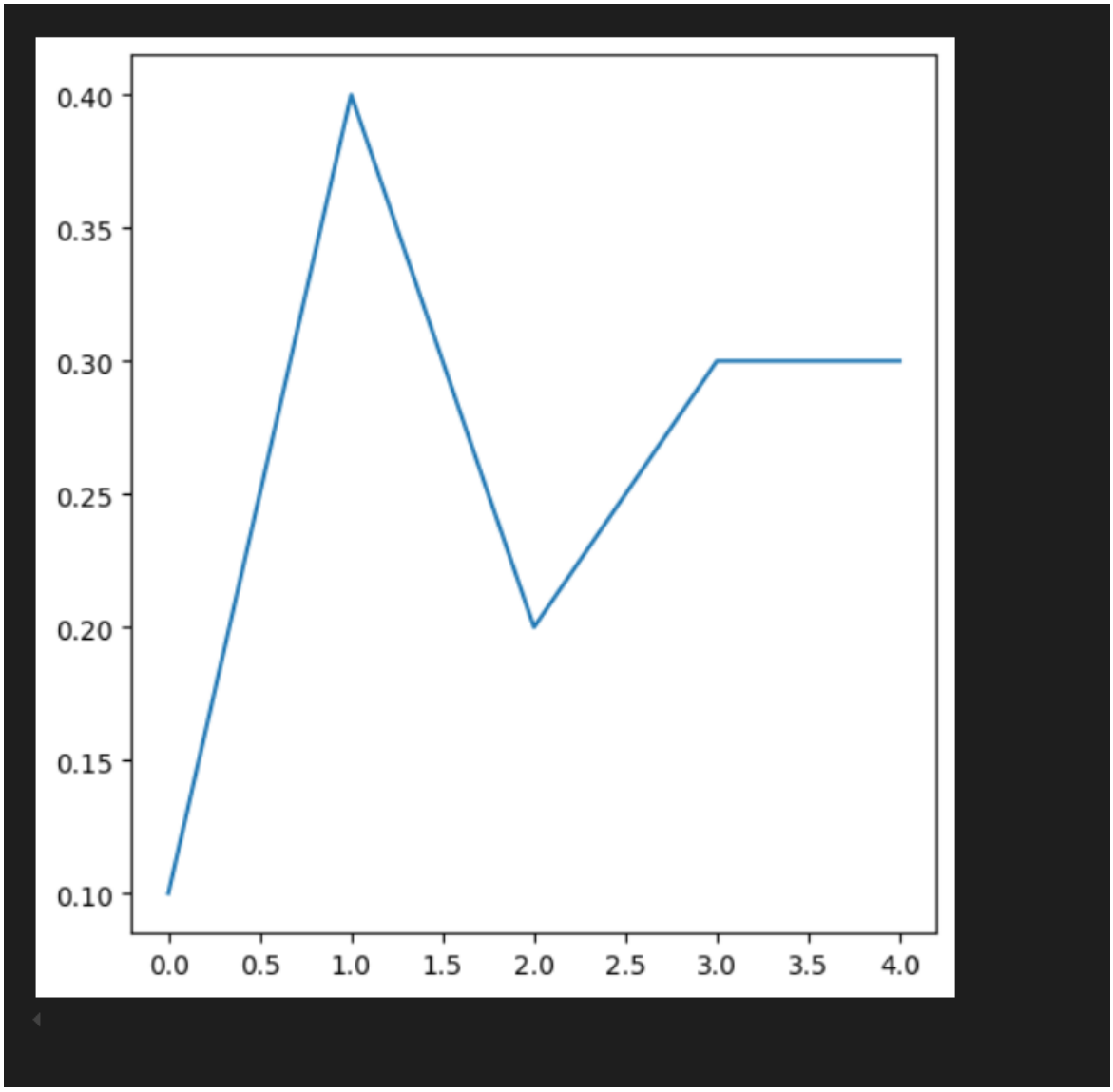
Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

11. Write a code to plot the accuracy and loss graphs after training a CNN model.

→ CODE:

```
#11.Write a code to plot the accuracy and loss graphs after training a CNN model.
import matplotlib.pyplot as plt

# Assuming you have trained your model and have the history object
# history = model.fit(...)

# Plot Accuracy
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy', linestyle='--')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid()

# Plot Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss', linestyle='--')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid()

# Show the plots
plt.tight_layout()
plt.show()
```

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903
OUTPUT:



12. Write a code to print the architecture of the ResNet50 model in Keras.

```python
#12.Write a code to print the architecture of the ResNet50 model in Keras.
from tensorflow.keras.applications import ResNet50

# Load the ResNet50 model pre-trained on ImageNet dataset
resnet50_model = ResNet50(weights='imagenet', include_top=True)

# Print the model architecture
resnet50_model.summary()
resnet50_custom = ResNet50(weights='imagenet', include_top=False, input_shape=(128, 128, 3))
resnet50_custom.summary()
```

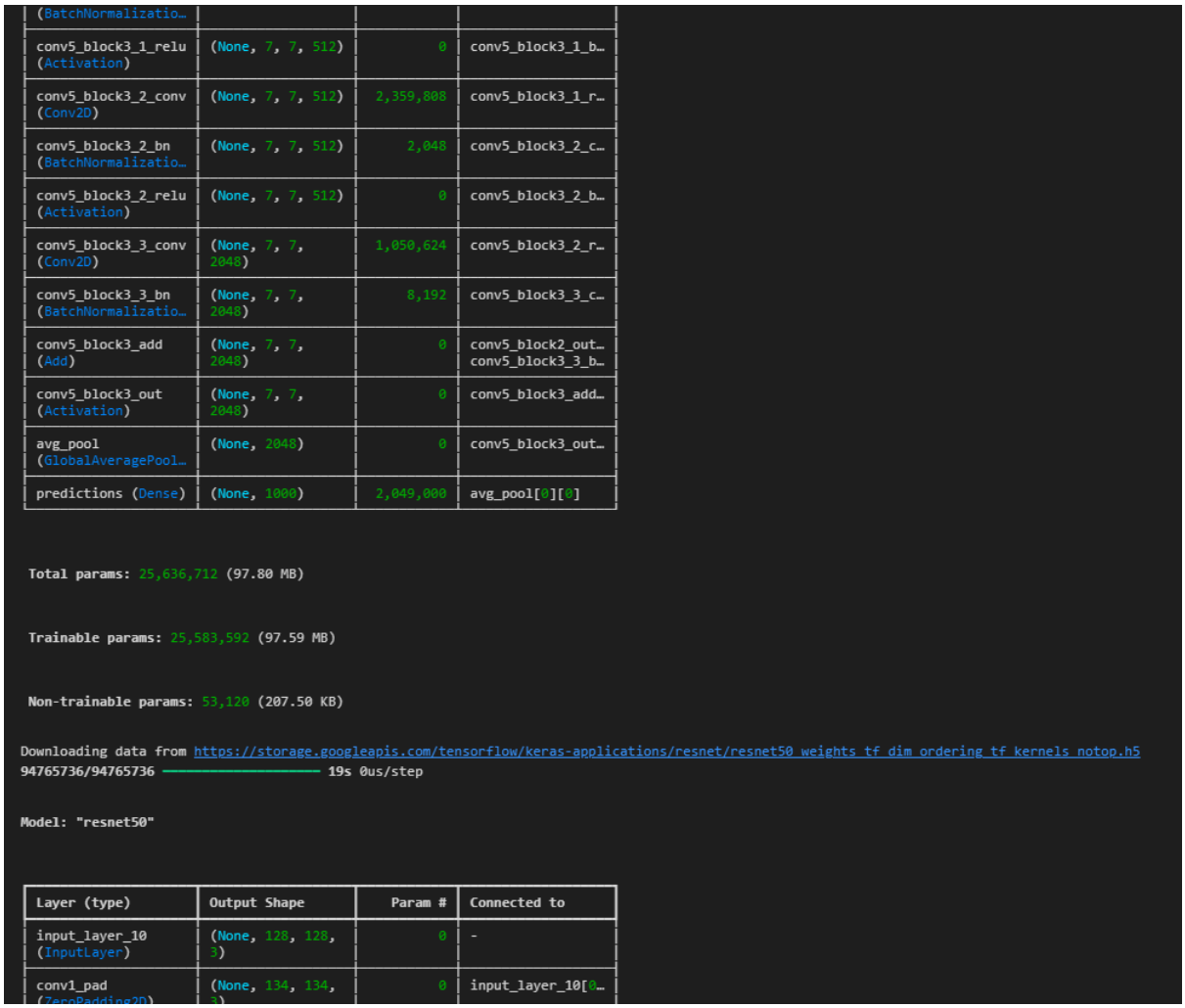
✓ 25.5s

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903
OUTPUT:

Model: "resnet50"

Layer (type)	Output Shape	Param #	Connected to
input_layer_9 (InputLayer)	(None, 224, 224, 3)	0	-
conv1_pad (ZeroPadding2D)	(None, 230, 230, 3)	0	input_layer_9[0]…
conv1_conv (Conv2D)	(None, 112, 112, 64)	9,472	conv1_pad[0][0]
conv1_bn (BatchNormalizatio…	(None, 112, 112, 64)	256	conv1_conv[0][0]
conv1_relu (Activation)	(None, 112, 112, 64)	0	conv1_bn[0][0]
pool1_pad (ZeroPadding2D)	(None, 114, 114, 64)	0	conv1_relu[0][0]
pool1_pool (MaxPooling2D)	(None, 56, 56, 64)	0	pool1_pad[0][0]
conv2_block1_1_conv (Conv2D)	(None, 56, 56, 64)	4,160	pool1_pool[0][0]
conv2_block1_1_bn (BatchNormalizatio…	(None, 56, 56, 64)	256	conv2_block1_1_c…
conv2_block1_1_relu (Activation)	(None, 56, 56, 64)	0	conv2_block1_1_b…
conv2_block1_2_conv (Conv2D)	(None, 56, 56, 64)	36,928	conv2_block1_1_r…
conv2_block1_2_bn (BatchNormalizatio…	(None, 56, 56, 64)	256	conv2_block1_2_c…
conv2_block1_2_relu (Activation)	(None, 56, 56, 64)	0	conv2_block1_2_b…
conv2_block1_0_conv (Conv2D)	(None, 56, 56, 256)	16,640	pool1_pool[0][0]
conv2_block1_3_conv (Conv2D)	(None, 56, 56, 256)	16,640	conv2_block1_2_r…
conv2_block1_0_bn (BatchNormalizatio…	(None, 56, 56, 256)	1,024	conv2_block1_0_c…
conv2_block1_3_bn (BatchNormalizatio…	(None, 56, 56, 256)	1,024	conv2_block1_3_c…
conv2_block1_add (Add)	(None, 56, 56, 256)	0	conv2_block1_0_b… conv2_block1_3_b…

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903



(BatchNormalizatio…			
conv5_block3_1_relu (Activation)	(None, 7, 7, 512)	0	conv5_block3_1_b…
conv5_block3_2_conv (Conv2D)	(None, 7, 7, 512)	2,359,808	conv5_block3_1_r…
conv5_block3_2_bn (BatchNormalizatio…	(None, 7, 7, 512)	2,048	conv5_block3_2_c…
conv5_block3_2_relu (Activation)	(None, 7, 7, 512)	0	conv5_block3_2_b…
conv5_block3_3_conv (Conv2D)	(None, 7, 7, 2048)	1,050,624	conv5_block3_2_r…
conv5_block3_3_bn (BatchNormalizatio…	(None, 7, 7, 2048)	8,192	conv5_block3_3_c…
conv5_block3_add (Add)	(None, 7, 7, 2048)	0	conv5_block2_out… conv5_block3_3_b…
conv5_block3_out (Activation)	(None, 7, 7, 2048)	0	conv5_block3_add…
avg_pool (GlobalAveragePool…	(None, 2048)	0	conv5_block3_out…
predictions (Dense)	(None, 1000)	2,049,000	avg_pool[0][0]

Total params: 25,636,712 (97.80 MB)

Trainable params: 25,583,592 (97.59 MB)

Non-trainable params: 53,120 (207.50 KB)

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 19s 0us/step

Model: "resnet50"

Layer (type)	Output Shape	Param #	Connected to
input_layer_10 (InputLayer)	(None, 128, 128, 3)	0	-
conv1_pad (ZeroPadding2D)	(None, 134, 134, 3)	0	input_layer_10[0…

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

conv5_block2_2_bn (BatchNormalizatio…	(None, 4, 4, 512)	2,048	conv5_block2_2_c…
conv5_block2_2_relu (Activation)	(None, 4, 4, 512)	0	conv5_block2_2_b…
conv5_block2_3_conv (Conv2D)	(None, 4, 4, 2048)	1,050,624	conv5_block2_2_r…
conv5_block2_3_bn (BatchNormalizatio…	(None, 4, 4, 2048)	8,192	conv5_block2_3_c…
conv5_block2_add (Add)	(None, 4, 4, 2048)	0	conv5_block1_out… conv5_block2_3_b…
conv5_block2_out (Activation)	(None, 4, 4, 2048)	0	conv5_block2_add…
conv5_block3_1_conv (Conv2D)	(None, 4, 4, 512)	1,049,088	conv5_block2_out…
conv5_block3_1_bn (BatchNormalizatio…	(None, 4, 4, 512)	2,048	conv5_block3_1_c…
conv5_block3_1_relu (Activation)	(None, 4, 4, 512)	0	conv5_block3_1_b…
conv5_block3_2_conv (Conv2D)	(None, 4, 4, 512)	2,359,808	conv5_block3_1_r…
conv5_block3_2_bn (BatchNormalizatio…	(None, 4, 4, 512)	2,048	conv5_block3_2_c…
conv5_block3_2_relu (Activation)	(None, 4, 4, 512)	0	conv5_block3_2_b…
conv5_block3_3_conv (Conv2D)	(None, 4, 4, 2048)	1,050,624	conv5_block3_2_r…
conv5_block3_3_bn (BatchNormalizatio…	(None, 4, 4, 2048)	8,192	conv5_block3_3_c…
conv5_block3_add (Add)	(None, 4, 4, 2048)	0	conv5_block2_out… conv5_block3_3_b…
conv5_block3_out (Activation)	(None, 4, 4, 2048)	0	conv5_block3_add…

Total params: 23,587,712 (89.98 MB)

Trainable params: 23,534,592 (89.78 MB)

Non-trainable params: 53,120 (207.50 KB)

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

13. Write a code to train a basic CNN model and print the training loss and accuracy after each epoch.

→ CODE:

```python
#13.Write a code to train a basic CNN model and print the training loss and accuracy after each epoch?
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape((x_train.shape[0], 28, 28, 1)).astype('float32') / 255.0
x_test = x_test.reshape((x_test.shape[0], 28, 28, 1)).astype('float32') / 255.0
# One-hot encode the labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
# Build a basic CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Define a custom callback to print loss and accuracy after each epoch
class PrintMetricsCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        print(f"Epoch {epoch + 1}: Loss = {logs['loss']:.4f}, Accuracy = {logs['accuracy']:.4f}")
# Train the model
model.fit(x_train, y_train, epochs=5, batch_size=64, callbacks=[PrintMetricsCallback()], validation_data=(x_test, y_test))
```

✓ 55.1s

OUTPUT:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step
c:\Users\Mansi Dobriyal\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pa
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8872 - loss: 0.3975Epoch 1: Loss = 0.1984, Accuracy = 0.9425
938/938 ━━━━━━━━━━━━━━━━━━━━ 12s 11ms/step - accuracy: 0.8873 - loss: 0.3973 - val_accuracy: 0.9740 - val_loss: 0.0851
Epoch 2/5
936/938 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9795 - loss: 0.0696Epoch 2: Loss = 0.0668, Accuracy = 0.9799
938/938 ━━━━━━━━━━━━━━━━━━━━ 9s 9ms/step - accuracy: 0.9795 - loss: 0.0696 - val_accuracy: 0.9770 - val_loss: 0.0736
Epoch 3/5
935/938 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9861 - loss: 0.0464Epoch 3: Loss = 0.0465, Accuracy = 0.9859
938/938 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - accuracy: 0.9861 - loss: 0.0464 - val_accuracy: 0.9828 - val_loss: 0.0477
Epoch 4/5
936/938 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9905 - loss: 0.0319Epoch 4: Loss = 0.0346, Accuracy = 0.9893
938/938 ━━━━━━━━━━━━━━━━━━━━ 9s 10ms/step - accuracy: 0.9905 - loss: 0.0319 - val_accuracy: 0.9846 - val_loss: 0.0436
Epoch 5/5
934/938 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9932 - loss: 0.0245Epoch 5: Loss = 0.0268, Accuracy = 0.9920
938/938 ━━━━━━━━━━━━━━━━━━━━ 9s 10ms/step - accuracy: 0.9932 - loss: 0.0245 - val_accuracy: 0.9838 - val_loss: 0.0420

<keras.src.callbacks.history.History at 0x26aeea06450>
```