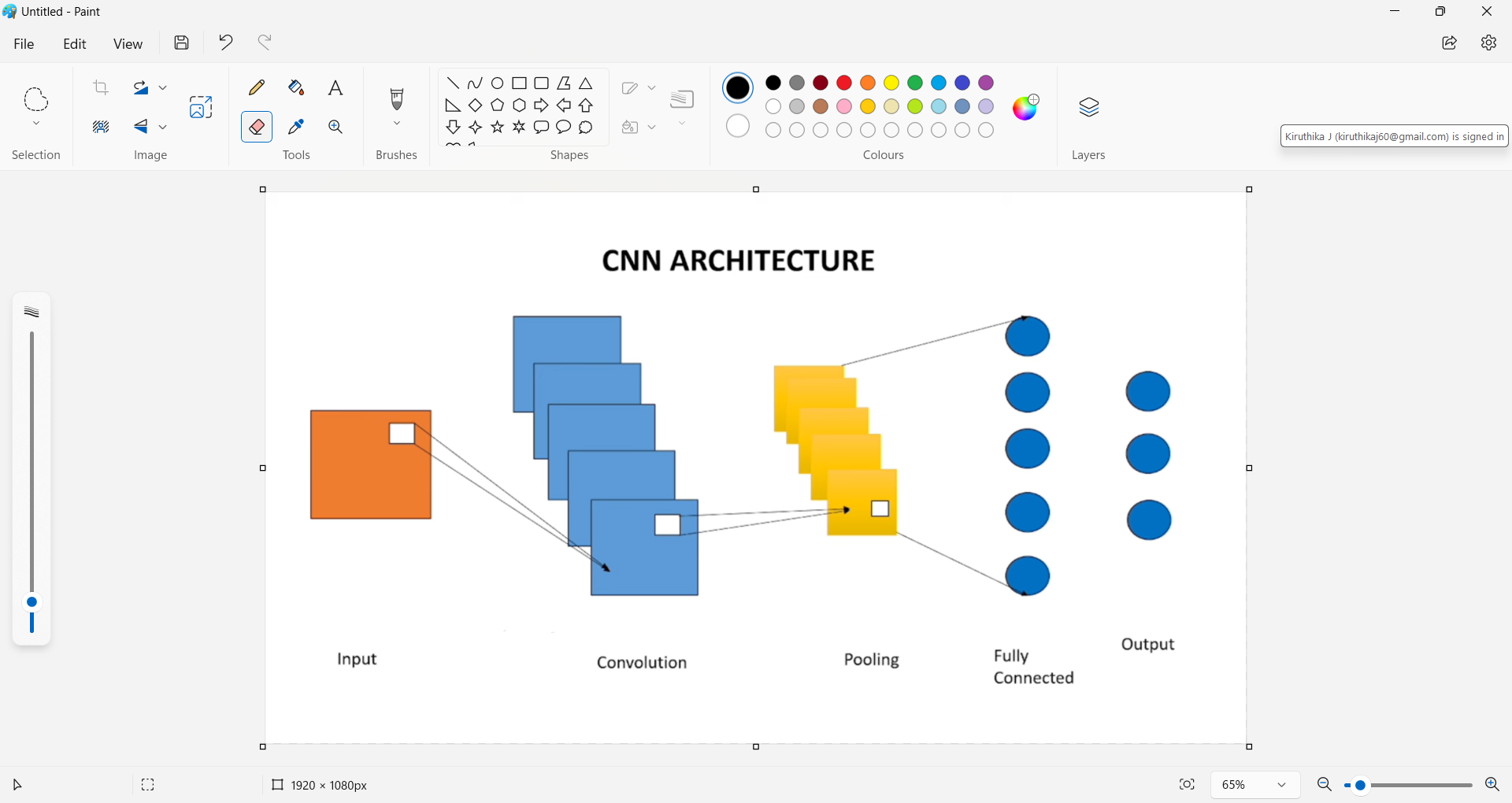
# **ADVANCED TECHNIQUES IN CNN ARCHITECTURES.**

**Convolutional Neural Network (CNN)**

In deep learning, a **Convolutional Neural Network (CNN)** is a type of artificial neural network designed specifically for processing structured grid data, such as images. CNNs are particularly effective in tasks like image classification, object detection, and segmentation.

**Key components of CNNs Architectures:**



1. **Convolutional Layers**: These layers apply convolutional filters to the input data, enabling the network to learn spatial hierarchies and local patterns (like edges and textures) in images.
2. **Pooling Layers**: Pooling reduces the spatial dimensions of the data, summarizing the features while retaining important information. Common pooling methods include max pooling and average pooling.
3. **Activation Functions**: Functions like ReLU (Rectified Linear Unit) are applied to introduce non-linearity, allowing the network to learn complex patterns.
4. **Fully Connected Layers**: After several convolutional and pooling layers, the high-level reasoning is performed in fully connected layers, where each neuron is connected to every neuron in the previous layer.
5. **Dropout and Regularization**: Techniques to prevent overfitting by randomly deactivating certain neurons during training.

**ADVANCED TECHNIQUES IN CNN ARCHITECTURES**

1. LeNet-5
2. AlexNet
3. VGGNet16
4. ResNet

**LeNet-5**

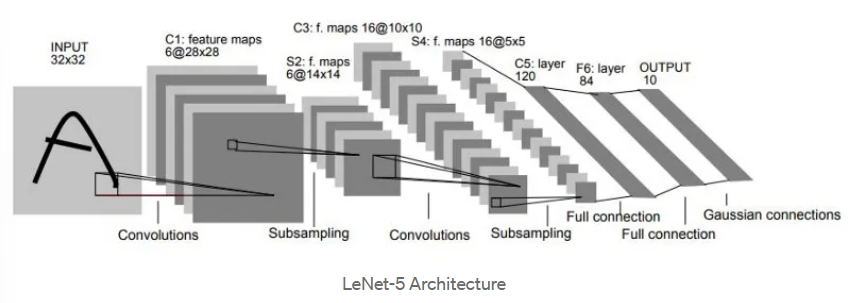
**Definition:**

LeNet-5 is a type of neural network that was designed to recognize images, specifically handwritten

digits, letters like those in the MNIST dataset. It's one of the earliest convolutional neural networks (CNNs).

**Architecture**

The LeNet-5 CNN architecture has seven layers. Three convolutional layers, two subsampling layers, and two fully linked layers make up the layer composition.



**CODE FOR LENET-5**

# Import necessary modules

from keras.models import Sequential

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

from keras import backend as K

# Clear any existing models from memory

K.clear\_session()

# Define the model only once

model = Sequential()

model.add(Conv2D(32, (5, 5), activation='relu', input\_shape=(28, 28, 1)))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(64, (5, 5), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dense(120, activation='relu'))

model.add(Dense(84, activation='relu'))

model.add(Dense(10, activation='softmax'))

# Compile the model

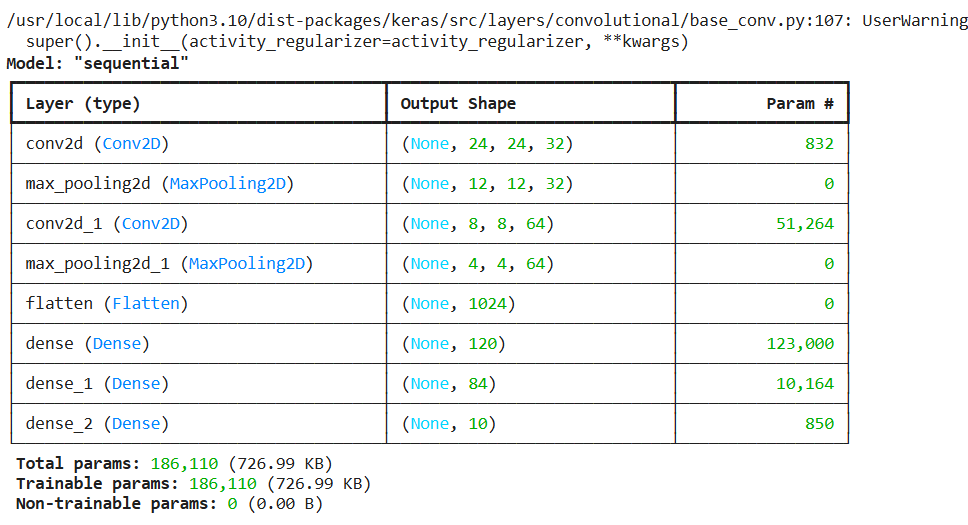
model.compile(optimizer='adam',

loss='categorical\_crossentropy',

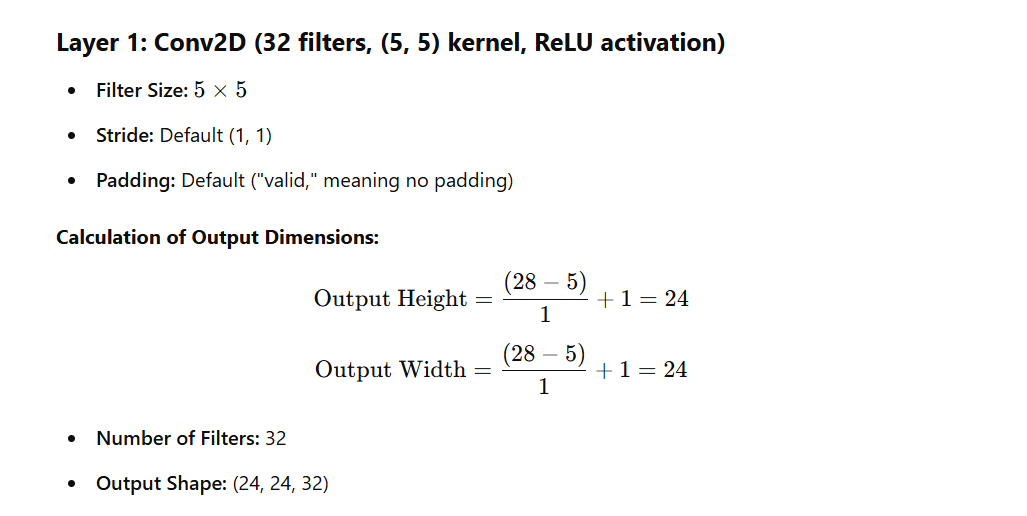
metrics=['accuracy'])

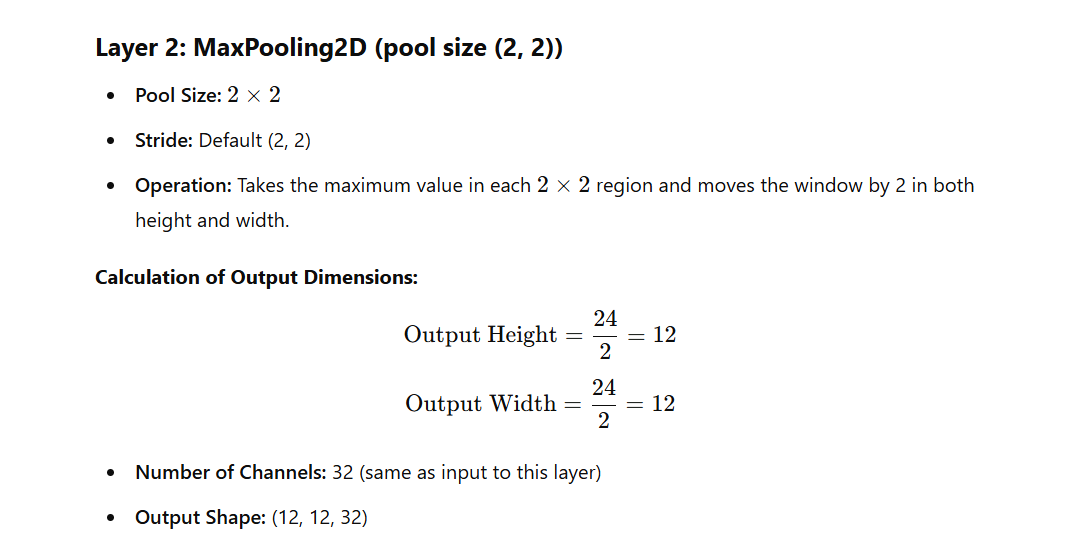
model.summary()

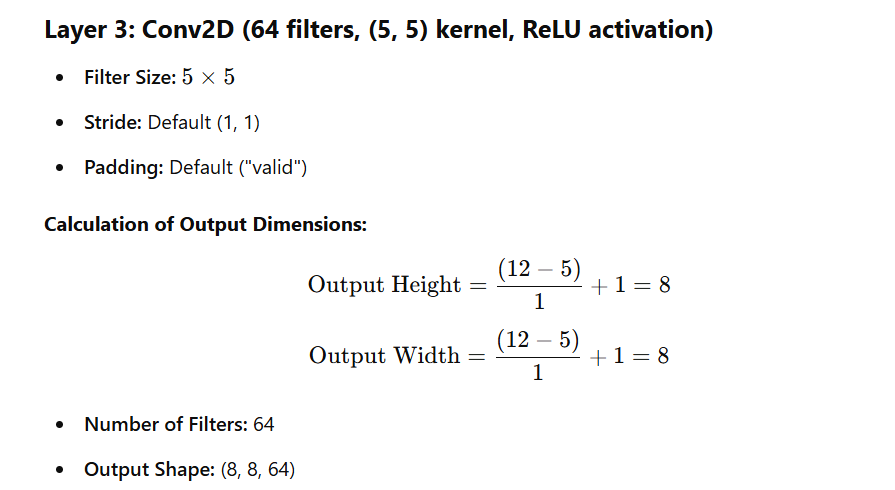
**OUTPUT:**

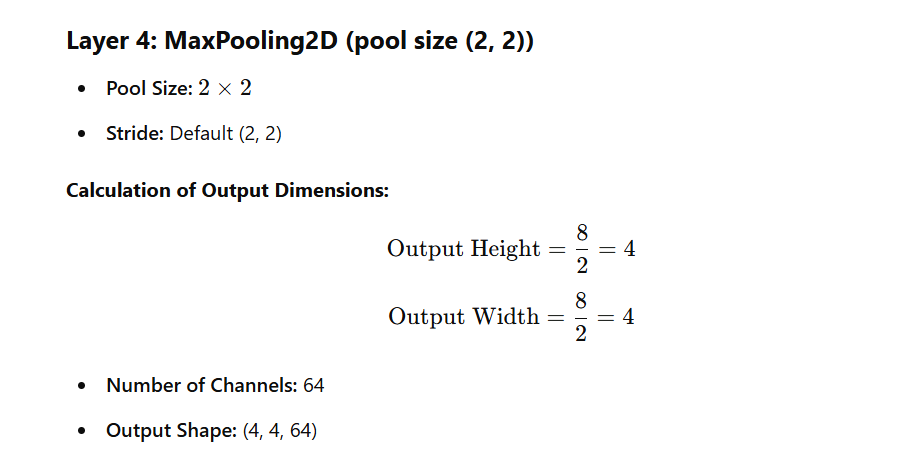


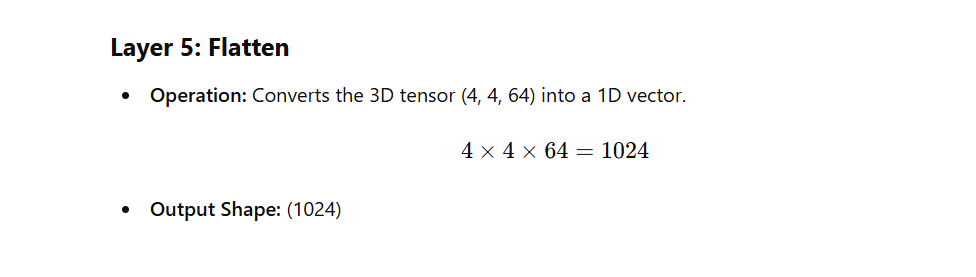
**CALCULATION**

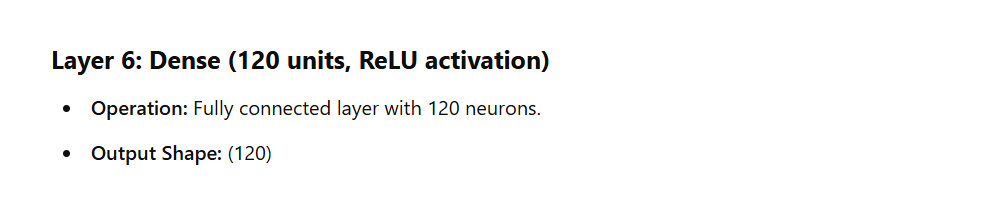


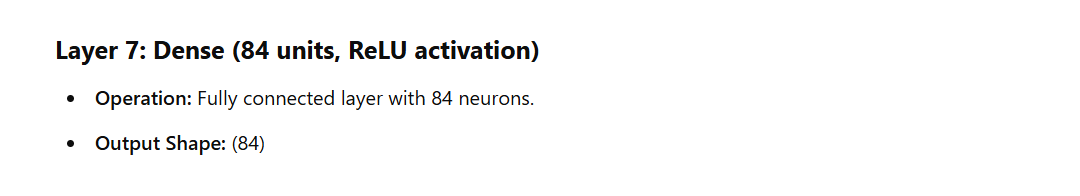


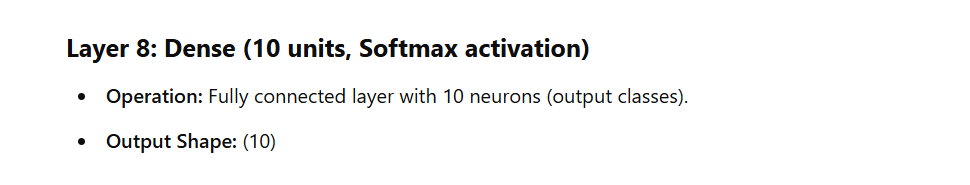










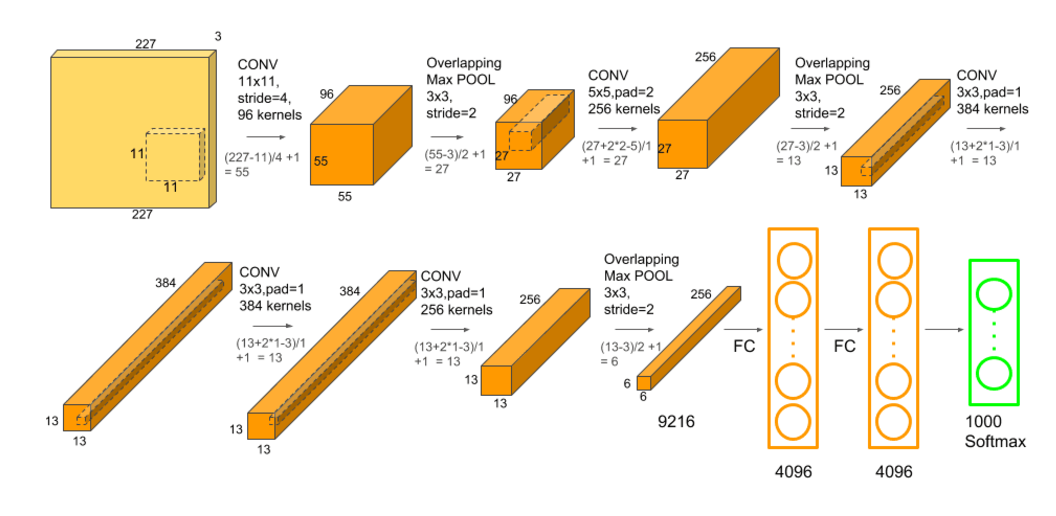


**AlexNet**

**Definition:**

AlexNet is primarily used for image classification tasks, where it classifies images into predefined categories. It is widely recognized for its success in large-scale image recognition challenges, particularly in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012, where it drastically improved performance compared to previous methods.

**Architecture:**



The **AlexNet architecture** consists of 8 layers: 5 convolutional layers (Conv) and 3 fully connected layers (FC). To understand the architecture and calculate the output size of each layer, we'll break down the details of each layer, including the size of the input and output for the convolutional and fully connected layers.

**CODE FOR ALEXNET**

# Initialize the model

model = Sequential()

# 1st Convolutional Layer

model.add(Conv2D(96, (3, 3), strides=(1, 1), activation='relu', padding='same', input\_shape=(32, 32, 3)))

model.add(MaxPooling2D(pool\_size=(2, 2), strides=(2, 2)))  # Output: 16x16

# 2nd Convolutional Layer

model.add(Conv2D(256, (3, 3), strides=(1, 1), activation='relu', padding='same'))

model.add(MaxPooling2D(pool\_size=(2, 2), strides=(2, 2)))  # Output: 8x8

# 3rd Convolutional Layer

model.add(Conv2D(384, (3, 3), strides=(1, 1), activation='relu', padding='same'))  # Output: 8x8

# 4th Convolutional Layer

model.add(Conv2D(384, (3, 3), strides=(1, 1), activation='relu', padding='same'))  # Output: 8x8

# 5th Convolutional Layer

model.add(Conv2D(256, (3, 3), strides=(1, 1), activation='relu', padding='same'))

model.add(MaxPooling2D(pool\_size=(2, 2), strides=(2, 2)))  # Output: 4x4

# Flatten the output

model.add(Flatten())

# Fully Connected Layer 1

model.add(Dense(4096, activation='relu'))

model.add(Dropout(0.5))

# Fully Connected Layer 2

model.add(Dense(4096, activation='relu'))

model.add(Dropout(0.5))

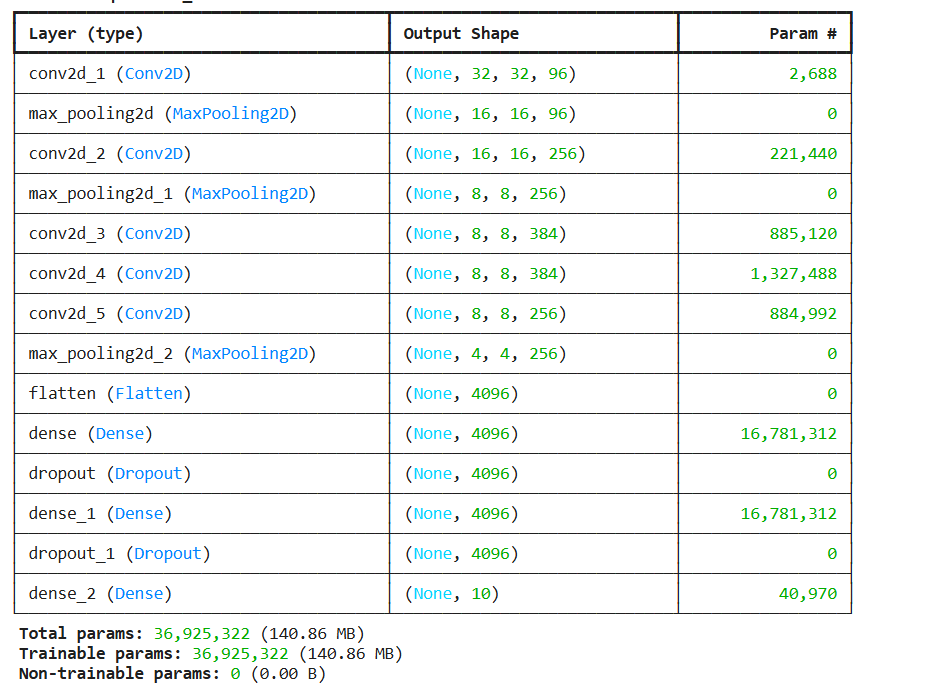
# Output Layer (10 classes for CIFAR-10)

model.add(Dense(10, activation='softmax'))

# Model Summary

model.summary()

**OUTPUT**



**VGG16**

**Definition**

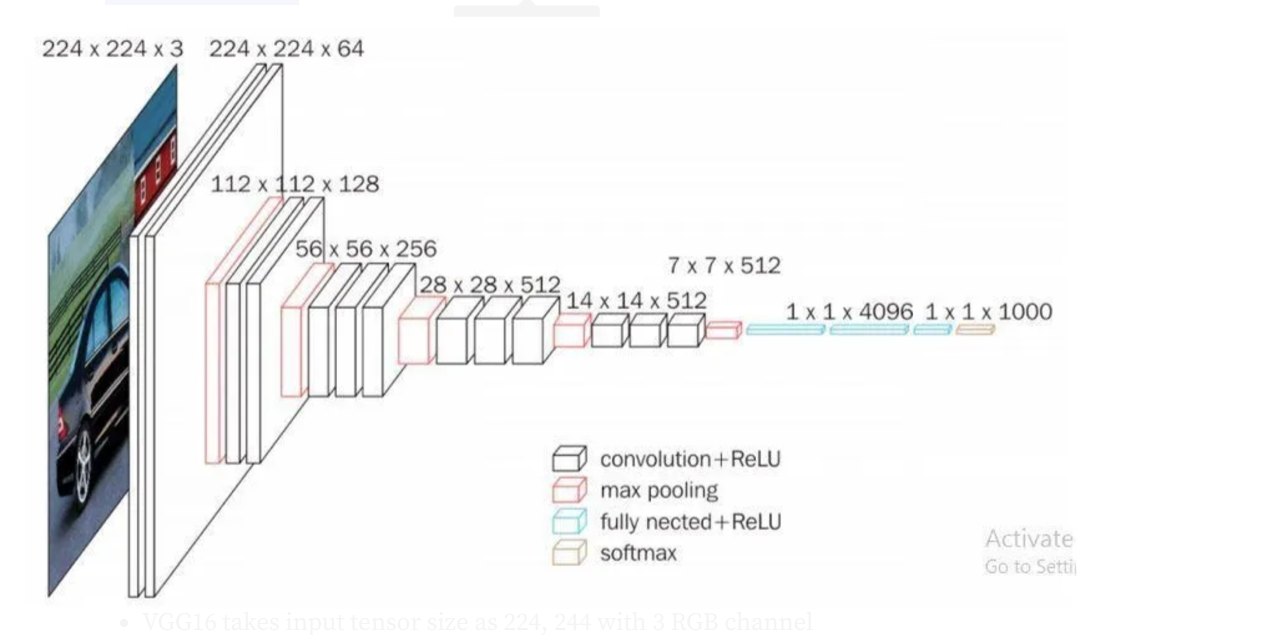
VGG16 is a well-known convolutional neural network (CNN) architecture that was proposed by the Visual Geometry Group at the University of Oxford in the paper *"Very Deep Convolutional Networks for Large-Scale Image Recognition"* (2014) by Simonyan and Zisserman. It is one of the most widely used models for image classification tasks and is part of the VGG family of networks, which are distinguished by their use of very deep architectures.

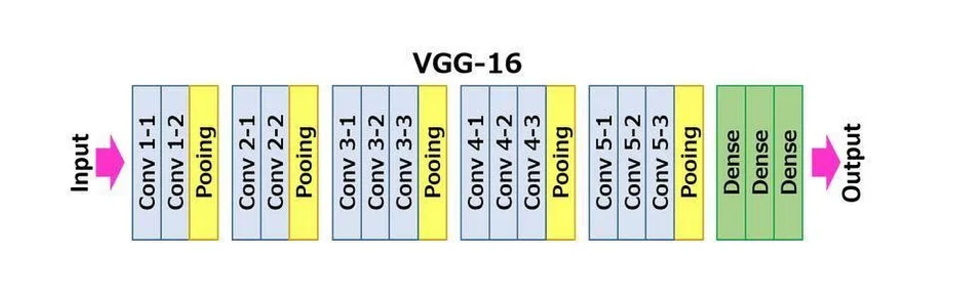
VGG16 has 16 layers that contain weights (hence the name VGG16), which include:

1. 13 Convolutional Layers
2. 3 Fully Connected Layers

VGG16 is object detection and classification algorithm which is able to classify 1000 images of 1000 different categories with 92.7% accuracy. It is one of the popular algorithms for image classification and is easy to use with transfer learning.

**ARCHITECTURE**





**CODE FOR VGG16**

from keras.models import Sequential

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

from keras import backend as K

# Initialize the model

model = Sequential()

# 1st Convolutional Block

model.add(Conv2D(64, (3, 3), strides=(1, 1), activation='relu', padding='same', input\_shape=(32, 32, 3)))

model.add(Conv2D(64, (3, 3), strides=(1, 1), activation='relu', padding='same'))

model.add(MaxPooling2D(pool\_size=(2, 2), strides=(2, 2)))  # Output: 16x16

# 2nd Convolutional Block

model.add(Conv2D(128, (3, 3), strides=(1, 1), activation='relu', padding='same'))

model.add(Conv2D(128, (3, 3), strides=(1, 1), activation='relu', padding='same'))

model.add(MaxPooling2D(pool\_size=(2, 2), strides=(2, 2)))  # Output: 8x8

# 3rd Convolutional Block

model.add(Conv2D(256, (3, 3), strides=(1, 1), activation='relu', padding='same'))

model.add(Conv2D(256, (3, 3), strides=(1, 1), activation='relu', padding='same'))

model.add(MaxPooling2D(pool\_size=(2, 2), strides=(2, 2)))  # Output: 4x4

# 4th Convolutional Block

model.add(Conv2D(512, (3, 3), strides=(1, 1), activation='relu', padding='same'))

model.add(Conv2D(512, (3, 3), strides=(1, 1), activation='relu', padding='same'))

model.add(MaxPooling2D(pool\_size=(2, 2), strides=(2, 2)))  # Output: 2x2

# 5th Convolutional Block

model.add(Conv2D(512, (3, 3), strides=(1, 1), activation='relu', padding='same'))

model.add(Conv2D(512, (3, 3), strides=(1, 1), activation='relu', padding='same'))

model.add(MaxPooling2D(pool\_size=(2, 2), strides=(2, 2)))  # Output: 1x1

# Flatten the output

model.add(Flatten())

# Fully Connected Layer 1

model.add(Dense(4096, activation='relu'))

model.add(Dropout(0.5))

# Fully Connected Layer 2

model.add(Dense(4096, activation='relu'))

model.add(Dropout(0.5))

# Output Layer (10 classes for CIFAR-10)

model.add(Dense(10, activation='softmax'))

# Model Summary

model.summary()

**OUTPUT**

/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base\_conv.py:107: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

super().\_\_init\_\_(activity\_regularizer=activity\_regularizer, \*\*kwargs)

**Model: "sequential\_1"**

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓

┃ **Layer (type)** ┃ **Output Shape** ┃ **Param #** ┃

┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩

│ conv2d\_2 (Conv2D) │ (None, 32, 32, 64) │ 1,792 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_3 (Conv2D) │ (None, 32, 32, 64) │ 36,928 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ max\_pooling2d\_2 (MaxPooling2D) │ (None, 16, 16, 64) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_4 (Conv2D) │ (None, 16, 16, 128) │ 73,856 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_5 (Conv2D) │ (None, 16, 16, 128) │ 147,584 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ max\_pooling2d\_3 (MaxPooling2D) │ (None, 8, 8, 128) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_6 (Conv2D) │ (None, 8, 8, 256) │ 295,168 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_7 (Conv2D) │ (None, 8, 8, 256) │ 590,080 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ max\_pooling2d\_4 (MaxPooling2D) │ (None, 4, 4, 256) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_8 (Conv2D) │ (None, 4, 4, 512) │ 1,180,160 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_9 (Conv2D) │ (None, 4, 4, 512) │ 2,359,808 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ max\_pooling2d\_5 (MaxPooling2D) │ (None, 2, 2, 512) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_10 (Conv2D) │ (None, 2, 2, 512) │ 2,359,808 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_11 (Conv2D) │ (None, 2, 2, 512) │ 2,359,808 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ max\_pooling2d\_6 (MaxPooling2D) │ (None, 1, 1, 512) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ flatten\_1 (Flatten) │ (None, 512) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ dense\_3 (Dense) │ (None, 4096) │ 2,101,248 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ dropout (Dropout) │ (None, 4096) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ dense\_4 (Dense) │ (None, 4096) │ 16,781,312 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ dropout\_1 (Dropout) │ (None, 4096) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ dense\_5 (Dense) │ (None, 10) │ 40,970 │

└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘

**Total params:** 28,328,522 (108.06 MB)

**Trainable params:** 28,328,522 (108.06 MB)

**Non-trainable params:** 0 (0.00 B)

**RENET-TECHNIQUES**

**ResNet (Residual Network)** is a deep learning architecture that introduces the concept of **residual learning** to solve the problem of training very deep neural networks. The main idea behind ResNet is to use **skip connections** or **shortcut connections**, which allow the input to bypass certain layers and be added directly to the output of those layers.

**Residual Block**: The core component of ResNet, where the input to the block is added to the output, creating a skip connection. Mathematically, the output y of a residual block is:

y=F(x)+xy = \mathcal{F}(x) + xy=F(x)+x

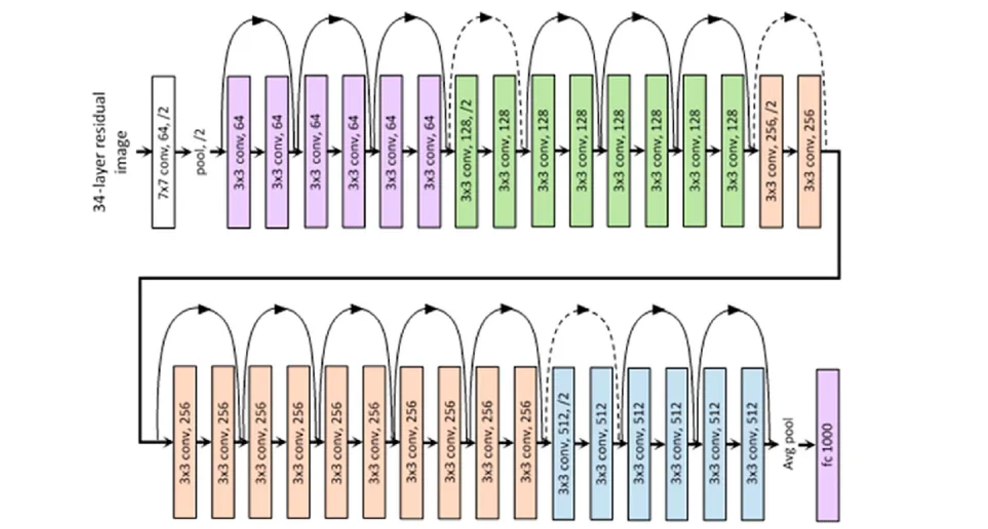
where is the input to the block, F(x)\mathcal{F}(x) is the function (set of convolutional layers) learned by the network, and yyy is the output.

**Skip Connections**: These connections enable the input to skip over one or more layers and be added directly to the output. This helps in preventing the vanishing gradient problem and makes training deeper networks easier.

**Deep Architectures**: ResNet can be trained with extremely deep networks, such as ResNet-50, ResNet-101, and even ResNet-152, without suffering from performance degradation.

**Global Average Pooling**: Instead of using fully connected layers, ResNet typically uses global average pooling to reduce spatial dimensions, which helps with overfitting and reduces computational costs.

**ARCHITECTURE**



CODE FOR RESNET

from keras.models import Model

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

from keras import backend as K

# Define the residual block with 1x1 convolution for matching dimensions

def residual\_block(x, filters, kernel\_size=(3, 3), stride=1):

shortcut = x

# First convolutional layer

x = Conv2D(filters, kernel\_size, strides=stride, activation='relu', padding='same')(x)

x = Conv2D(filters, kernel\_size, strides=stride, activation=None, padding='same')(x) # No activation

# If the input and output channels do not match, apply 1x1 convolution to the shortcut

if shortcut.shape[-1] != x.shape[-1]:

shortcut = Conv2D(filters, (1, 1), strides=stride, activation=None, padding='same')(shortcut)

# Add the shortcut connection (skip connection)

x = Add()([x, shortcut])

x = Conv2D(filters, kernel\_size, activation='relu', padding='same')(x) # Optional ReLU after adding the residual

return x

# Initialize the model

input\_layer = Input(shape=(32, 32, 3))

# 1st Convolutional Layer

x = Conv2D(64, (3, 3), activation='relu', padding='same')(input\_layer)

# 1st Residual Block

x = residual\_block(x, 64)

# Max Pooling after first block

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

# 2nd Residual Block

x = residual\_block(x, 128)

# Max Pooling after second block

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

# 3rd Residual Block

x = residual\_block(x, 256)

# Max Pooling after third block

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

# 4th Residual Block

x = residual\_block(x, 512)

# Max Pooling after fourth block

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

# 5th Residual Block

x = residual\_block(x, 512)

# Max Pooling after fifth block

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

# Flatten the output

x = Flatten()(x)

# Fully Connected Layer 1

x = Dense(4096, activation='relu')(x)

x = Dropout(0.5)(x)

# Fully Connected Layer 2

x = Dense(4096, activation='relu')(x)

x = Dropout(0.5)(x)

# Output Layer (10 classes for CIFAR-10)

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

# Define the model

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

# Model Summary

model.summary()

OUTPUT:

M **odel: "functional\_29"**

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┓

┃ **Layer (type)** ┃ **Output Shape** ┃ **Param #** ┃ **Connected to** ┃

┡━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━┩

│ input\_layer\_4 │ (None, 32, 32, 3) │ 0 │ - │

│ (InputLayer) │ │ │ │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ conv2d\_24 (Conv2D) │ (None, 32, 32, 64) │ 1,792 │ input\_layer\_4[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ conv2d\_25 (Conv2D) │ (None, 32, 32, 64) │ 36,928 │ conv2d\_24[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ conv2d\_26 (Conv2D) │ (None, 32, 32, 64) │ 36,928 │ conv2d\_25[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ add\_4 (Add) │ (None, 32, 32, 64) │ 0 │ conv2d\_26[0][0], │

│ │ │ │ conv2d\_24[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ conv2d\_27 (Conv2D) │ (None, 32, 32, 64) │ 36,928 │ add\_4[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ max\_pooling2d\_9 │ (None, 16, 16, 64) │ 0 │ conv2d\_27[0][0] │

│ (MaxPooling2D) │ │ │ │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ conv2d\_28 (Conv2D) │ (None, 16, 16, 128) │ 73,856 │ max\_pooling2d\_9[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ conv2d\_29 (Conv2D) │ (None, 16, 16, 128) │ 147,584 │ conv2d\_28[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ conv2d\_30 (Conv2D) │ (None, 16, 16, 128) │ 8,320 │ max\_pooling2d\_9[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ add\_5 (Add) │ (None, 16, 16, 128) │ 0 │ conv2d\_29[0][0], │

│ │ │ │ conv2d\_30[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ conv2d\_31 (Conv2D) │ (None, 16, 16, 128) │ 147,584 │ add\_5[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ max\_pooling2d\_10 │ (None, 8, 8, 128) │ 0 │ conv2d\_31[0][0] │

│ (MaxPooling2D) │ │ │ │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ conv2d\_32 (Conv2D) │ (None, 8, 8, 256) │ 295,168 │ max\_pooling2d\_10[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ conv2d\_33 (Conv2D) │ (None, 8, 8, 256) │ 590,080 │ conv2d\_32[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ conv2d\_34 (Conv2D) │ (None, 8, 8, 256) │ 33,024 │ max\_pooling2d\_10[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ add\_6 (Add) │ (None, 8, 8, 256) │ 0 │ conv2d\_33[0][0], │

│ │ │ │ conv2d\_34[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ conv2d\_35 (Conv2D) │ (None, 8, 8, 256) │ 590,080 │ add\_6[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ max\_pooling2d\_11 │ (None, 4, 4, 256) │ 0 │ conv2d\_35[0][0] │

│ (MaxPooling2D) │ │ │ │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ conv2d\_36 (Conv2D) │ (None, 4, 4, 512) │ 1,180,160 │ max\_pooling2d\_11[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ conv2d\_37 (Conv2D) │ (None, 4, 4, 512) │ 2,359,808 │ conv2d\_36[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ conv2d\_38 (Conv2D) │ (None, 4, 4, 512) │ 131,584 │ max\_pooling2d\_11[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ add\_7 (Add) │ (None, 4, 4, 512) │ 0 │ conv2d\_37[0][0], │

│ │ │ │ conv2d\_38[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ conv2d\_39 (Conv2D) │ (None, 4, 4, 512) │ 2,359,808 │ add\_7[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ max\_pooling2d\_12 │ (None, 2, 2, 512) │ 0 │ conv2d\_39[0][0] │

│ (MaxPooling2D) │ │ │ │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ conv2d\_40 (Conv2D) │ (None, 2, 2, 512) │ 2,359,808 │ max\_pooling2d\_12[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ conv2d\_41 (Conv2D) │ (None, 2, 2, 512) │ 2,359,808 │ conv2d\_40[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ add\_8 (Add) │ (None, 2, 2, 512) │ 0 │ conv2d\_41[0][0], │

│ │ │ │ max\_pooling2d\_12[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ conv2d\_42 (Conv2D) │ (None, 2, 2, 512) │ 2,359,808 │ add\_8[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ max\_pooling2d\_13 │ (None, 1, 1, 512) │ 0 │ conv2d\_42[0][0] │

│ (MaxPooling2D) │ │ │ │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ flatten\_2 (Flatten) │ (None, 512) │ 0 │ max\_pooling2d\_13[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ dense\_6 (Dense) │ (None, 4096) │ 2,101,248 │ flatten\_2[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ dropout\_2 (Dropout) │ (None, 4096) │ 0 │ dense\_6[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ dense\_7 (Dense) │ (None, 4096) │ 16,781,312 │ dropout\_2[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ dropout\_3 (Dropout) │ (None, 4096) │ 0 │ dense\_7[0][0] │

├───────────────────────────┼────────────────────────┼────────────────┼────────────────────────┤

│ dense\_8 (Dense) │ (None, 10) │ 40,970 │ dropout\_3[0][0] │

└───────────────────────────┴────────────────────────┴────────────────┴────────────────────────┘

**Total params:** 34,032,586 (129.82 MB)

**Trainable params:** 34,032,586 (129.82 MB)

**Non-trainable params:** 0 (0.00 B)

