**input\_shape=(28,28,1) model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=input\_shape)) model.add(MaxPooling2D(pool\_size=(2, 2))) model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu')) model.add(MaxPooling2D(pool\_size=(2, 2))) model.add(Flatten()) model.add(Dense(128, activation='relu')) model.add(Dense(64, activation='relu'))**

**In the provided code snippet, a convolutional neural network (CNN) model is being built using Keras. Let's analyze the transformation of the input shape through each layer:**

**1. Input Layer:**

**- Input shape: (28, 28, 1)**

**- The input image has a height and width of 28 pixels and a single channel (grayscale).**

**2. Convolutional Layer 1:**

**- Number of filters: 32**

**- Kernel size: (3, 3)**

**- Activation function: ReLU**

**- Padding: Default padding (which is 'valid')**

**- Stride: Default stride (which is (1, 1))**

**- Output shape: The output shape of a convolutional layer can be calculated using the formula:**

**output\_shape = (input\_shape - kernel\_size + 2 \* padding) / stride + 1**

**- For the first convolutional layer:**

**output\_shape = (28 - 3 + 2 \* 0) / 1 + 1 = 26**

**- So, the output shape is (26, 26, 32) since we have 32 filters.**

**3. MaxPooling Layer 1:**

**- Pool size: (2, 2)**

**- Output shape: The output shape of a max pooling layer can be calculated by halving the input shape in each dimension.**

**- For the first max pooling layer, the output shape becomes (13, 13, 32).**

**4. Convolutional Layer 2:**

**- Number of filters: 32**

**- Kernel size: (3, 3)**

**- Activation function: ReLU**

**- Padding: Default padding (which is 'valid')**

**- Stride: Default stride (which is (1, 1))**

**- Output shape: Following the same formula as above:**

**output\_shape = (13 - 3 + 2 \* 0) / 1 + 1 = 11**

**- So, the output shape is (11, 11, 32) since we have 32 filters.**

**5. MaxPooling Layer 2:**

**- Pool size: (2, 2)**

**- Output shape: The output shape becomes (5, 5, 32) after halving the input shape.**

**6. Flatten Layer:**

**- This layer flattens the 3D output into a 1D vector.**

**- Output shape: The output shape becomes (5 \* 5 \* 32) = 800.**

**7. Dense Layer 1:**

**- Number of neurons: 128**

**- Activation function: ReLU**

**- Output shape: The output shape becomes (128).**

**8. Dense Layer 2:**

**- Number of neurons: 64**

**- Activation function: ReLU**

**- Output shape: The output shape becomes (64).**

**In summary, the output shape changes as follows:**

**(28, 28, 1) -> (26, 26, 32) -> (13, 13, 32) -> (11, 11, 32) -> (5, 5, 32) -> (800) -> (128) -> (64)**

**The number of pixels in the output varies across the layers, starting from 28x28 in the input and ending with a vector of size 64 in the last Dense layer.**

**Parameter calculations:**

**input\_shape = (28, 28, 1)**

**model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=input\_shape))**

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

**model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu'))**

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

**model.add(Flatten())**

**model.add(Dense(128, activation='relu'))**

**model.add(Dense(64, activation='relu'))**

**```**

**1. Convolutional Layer 1:**

**- Number of filters: 32**

**- Kernel size: (3, 3)**

**- Trainable parameters = (3 \* 3 \* 1 + 1) \* 32 = 320**

**2. MaxPooling Layer 1:**

**- No trainable parameters**

**3. Convolutional Layer 2:**

**- Number of filters: 32**

**- Kernel size: (3, 3)**

**- Trainable parameters = (3 \* 3 \* 32 + 1) \* 32 = 9,248**

**4. MaxPooling Layer 2:**

**- No trainable parameters**

**5. Flatten Layer:**

**- No trainable parameters**

**6. Dense Layer 1:**

**- Number of neurons: 128**

**- Number of inputs from previous layer = 800 (flattened shape)**

**- Trainable parameters = (800 + 1) \* 128 = 102,528**

**7. Dense Layer 2:**

**- Number of neurons: 64**

**- Number of inputs from previous layer = 128**

**- Trainable parameters = (128 + 1) \* 64 = 8,256**

**Total Trainable Parameters: 320 + 9,248 + 102,528 + 8,256 = 120,352**

**Total Parameters (including non-trainable parameters): Same as trainable parameters, since there are no non-trainable parameters mentioned in the code snippet.**

**So, the calculated number of trainable parameters for the model is 120,352.**

**break down the calculation for a convolutional layer as an example:**

**Number of filters: 32**

**Kernel size: (3, 3)**

**Input channels: 1**

**The trainable parameters for the convolutional layer can be calculated as:**

**(trainable\_params\_per\_filter \* number\_of\_filters) + biases**

**For each filter, the trainable parameters consist of the weights of the convolutional kernel and a bias term. The weights of the convolutional kernel can be calculated as the product of the kernel size and the number of input channels: (3 \* 3 \* 1 = 9).**

**Therefore, the trainable parameters per filter would be (9 + 1) = 10 (weights + bias).**

**For the given example with 32 filters, the total trainable parameters for the convolutional layer would be (10 \* 32) = 320.**