🧬 Skin Cancer Detection CNN Model

# Model Design and Layer Functions with Implementation Code

## ✅ Model Overview

This model is a Convolutional Neural Network (CNN) designed for binary classification of skin lesion images (benign cancerous vs. malignant cancerous). It follows a sequential architecture, composed of multiple convolutional layers, pooling layers, and dense layers, with increasing feature complexity across its depth.

## 🧱 Layer-by-Layer Architecture

### 🔹 1st Layer: Convolution Block (Feature Extraction)

Job:  
Convolute 32 different weighted (3x3 kernel) With the input (224, 224) RGB (3 Channel) Image for activition Function ‘relu’.

Code Responsible:

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(224, 224, 3))

### 🔹Max Pooling Block

Job:  
Downsamples the image using a 2×2 window (2 stride) to reduce spatial resolution while preserving most dominant features.

Code Responsible:

layers.MaxPooling2D((2, 2))

### 🔹 2nd Layer: Convolution Block

Job:  
Applies 64 different weighted (3x3 kernel) to extract deeper and more abstract spatial features.

Code Responsible:

layers.Conv2D(64, (3, 3), activation='relu')

### 🔹 Max Pooling Block

Job:  
Further reduces the resolution by performing another 2×2 (2 stride) downsampling operation.

Code Responsible:

layers.MaxPooling2D((2, 2))

### 🔹 3rd Layer: Convolution Block

Job:  
Applies 128 different weighted (3x3 kernel) to extract high-level semantic features.

Code Responsible:

layers.Conv2D(128, (3, 3), activation='relu')

### 🔹 Max Pooling Block

Job:  
Reduces the image feature map again with a 2×2 (2 stride)downsampling window.

Code Responsible:

layers.MaxPooling2D((2, 2))

### 🔹 4th Layer: Flattening

Job:  
Transforms the 3D feature maps into a 1D vector for dense layer processing.

Code Responsible:

layers.Flatten()

### 🔹 5th Layer: Dense Layer (Fully Connected)

Job:  
 **l**earn complex relationships and combinations of the extracted Layers by connected 512 neuron with the **flattened previous layer.**

#### ****How it works:****

Assume that the previous layer (Flatten()) outputs a vector of length N (say 100,352 values). Then:

* You define a Dense layer with **512 neurons**.
* Each of those 512 neurons will have **N weights** + 1 bias(parameter helps the model learn **shifts** in the activation, allowing it to fit the training data better.).
* So this layer computes:

y=ReLU(W⋅x+b)

where:

* + x: input vector (flattened)
  + W: weights (size 512 × N)
  + b: bias vector (size 512)
  + ReLU: activation function applied element-wise

Code Responsible:

layers.Dense(512, activation='relu')

### 🔹 Dropout

Job:  
Randomly disables 50% of the neurons during training to prevent overfitting.

Code Responsible:

layers.Dropout(0.5)

### 🔹 Output Layer (Classification)

Job:  
Outputs a single probability score using the sigmoid function to classify the image.

Code Responsible:

layers.Dense(1, activation='sigmoid')

## ⚙️ Model Compilation

Job:  
Configures the model for training.  
  
- Loss Function: binary\_crossentropy is used for binary classification problems.  
- Optimizer: Adam adjusts learning rate adaptively to minimize the loss.  
- Metric: accuracy is used to monitor performance during training and validation.

Code Responsible:

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

## 📊 Model Summary

- Total Layers: 5 (including Conv, Pooling, Flatten, Dense, and Output)  
- Total Parameters: ~44.5 million  
- Input Size: 224×224×3 (RGB image)  
- Output: Binary prediction (benign cancerous vs. malignant cancerous)