Cancer Medicine Journal

Review Article | Vol 7 Issue 1

Convolutional Neural Networks in Dermatology: Skin Cancer Detection and Analysis

Manoj Ganthya

The University of Burdwan, West Bengal, India

Correspondence should be addressed to Manoj Ganthya, The University of Burdwan, West Bengal, India

Received: August 30, 2024; Accepted: October 20, 2024; Published: October 27, 2024

ABSTRACT

The application of Convolutional Neural Networks (CNNs) in the automated detection and classification of skin cancer has been investigated to give a non-invasive and reliable alternative to the traditional diagnostic methods, including visual examination and biopsies. An in-depth overview of CNN architecture, covering key components such as convolutional layers, pooling layers, activation functions, and the backpropagation algorithm has been presented. Moreover, critical aspects of dataset preparation, including image pre-processing, augmentation techniques, and the use of large, annotated datasets to enhance model performance have been addressed.

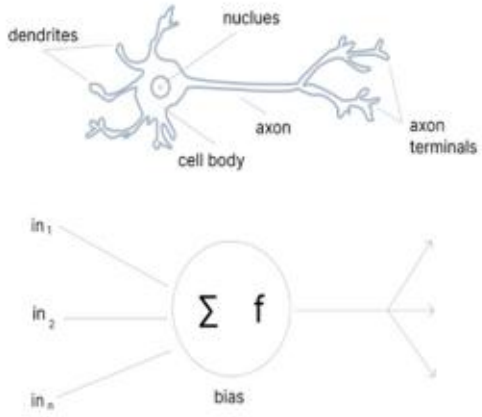
KEYWORDS

Skin cancer detection; Convolution neural network; Neural network architecture; Image classification

INRODUCTION

Deep Learning, a subset of machine learning inspired by the structure and function of the human brain, has transformed the landscape of artificial intelligence (AI) over the past few decades [1-3]. The concept of artificial neural networks, which are the backbone of deep learning, dates back to the 1940s and 1950s with early attempts to mimic brain functions. However, it was not until the 1980s that significant progress was made, particularly with the introduction of backpropagation, a method for training multi-layer networks (Figure 1 and Figure 2).

The 2000s saw deep learning gain momentum with the advent of more sophisticated architectures such as convolutional neural networks (CNNs) for image processing and recurrent neural networks (RNNs) for sequential data. These innovations, coupled with the rise of powerful computing resources like GPUs, allowed for the training of deeper and more complex networks on large datasets.



Citation: Manoj Ganthya, Convolutional Neural Networks in Dermatology: Skin Cancer Detection and Analysis. Cancer Med J 7(1): 100-113.

Figure 1: The concept of artificial neural networks.

Figure 2: A subset of machine learning inspired by the structure and function of the human brain, has transformed the landscape of artificial intelligence.

A pivotal moment came in 2012 with the success of AlexNet in the ImageNet competition, showcasing the potential of deep learning in achieving unprecedented accuracy in image recognition tasks. This breakthrough spurred a surge in research and applications across various domains, from natural language processing to autonomous vehicles.

In recent years, the field has been revolutionized by the development of advanced architectures like transformers, which have set new benchmarks in tasks such as language translation and text generation. Deep learning continues to evolve, driving significant advancements in AI and reshaping industries with its ability to process and learn from vast amounts of data.

Brief Explanation of CNN

Convolutional Neural Networks (CNNs) are a class of deep learning models that are particularly effective for analyzing visual data. They have revolutionized the field of computer vision, enabling advancements in image recognition, object detection, and more. Let's delve into the basics of CNNs, their history, and their key components.

Basic concept of CNNs

CNNs are designed to automatically and adaptively learn spatial hierarchies of features from input images. They mimic the way humans perceive images, focusing on the spatial relationships among pixels (Figure 3).

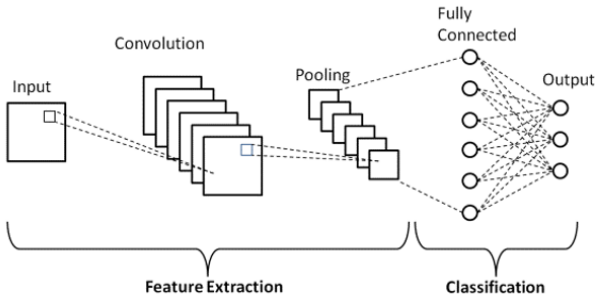


Figure 3: Basic concept of CNNs.

Key components of CNNs

Convolutional Layers: These layers apply filters to the input image to create feature maps. The filters, or kernels, slide over the image, detecting features such as edges, textures, and patterns.

- ***Activation Functions:*** Non-linear functions like ReLU (Rectified Linear Unit) are applied to the feature maps to introduce non-linearity, allowing the network to learn complex patterns.\
- ***Pooling Layers:*** These layers reduce the spatial dimensions of the feature maps, retaining important information while decreasing computational complexity. Common pooling techniques include max pooling and average pooling.
- ***Fully Connected Layers:*** After several convolutional and pooling layers, the final feature maps are flattened and passed through fully connected layers to make predictions or classifications.
- ***Output Layer:*** This layer generates the final predictions, often using softmax for classification tasks.

History of CNNs

Early concepts and development (1980s-1990s)

The concept of CNNs dates back to the 1980s when Kunihiko Fukushima developed the Neocognitron, a neural network inspired by the visual cortex. It introduced the idea of using hierarchical layers to detect increasingly complex features. In the late 1980s and early 1990s, Yann LeCun and his colleagues at AT&T Bell Labs made significant contributions by developing the first practical CNN, known as LeNet, which was used for digit recognition tasks such as reading handwritten digits on checks.

The emergence of modern CNNs (2000s)

The early 2000s saw limited success for CNNs due to computational constraints and limited availability of large datasets. However, researchers continued to refine the architecture and techniques. In 2006, Geoffrey Hin-ton's work on deep learning and unsupervised pre-training brought renewed interest in neural networks, setting the stage for more complex models.

Breakthrough with AlexNet (2012)

A major breakthrough came in 2012 when Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton introduced AlexNet, a deep CNN that dramatically improved performance in the ImageNet Large Scale Visual Recognition Challenge. AlexNet significantly outperformed traditional methods in image classification, demonstrating the power of deep CNNs with large datasets and powerful GPUs (Figure 4).

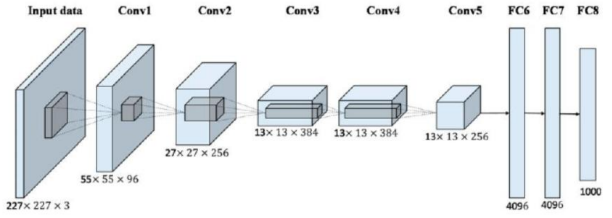


Figure 4: Breakthrough with AlexNet.

Continued advancements (2010s-Present)

Following AlexNet's success, deeper and more complex CNN architectures emerged, such as VGGNet, GoogleNet (Inception), and ResNet. These models pushed the boundaries of what CNNs could achieve, setting new benchmarks in various computer vision tasks. CNNs have since been widely adopted in numerous applications, including medical imaging, autonomous driving, facial recognition, and more.

Application of CNN

Convolutional Neural Networks (CNNs) have become indispensable tools in the field of artificial intelligence, particularly due to their ability to handle large-scale data and learn hierarchical features. They have made significant contributions across various domains, extending beyond their initial application in computer vision. Below, we explore the importance of CNNs in several key fields.

Computer Vision

Image classification

CNNs have set new benchmarks in identifying objects and features within images. Models like AlexNet, VGGNet, and ResNet have excelled in competitions like ImageNet, showcasing their ability to classify images with high accuracy.

Object detection

CNNs power systems like YOLO (You Only Look Once) and R-CNN (Region-based Convolutional Neural Networks), which can detect and localize objects in real-time. These technologies are crucial in applications ranging from security surveillance to autonomous driving.

Image segmentation

CNNs enable precise segmentation of images, allowing for tasks such as medical image analysis where accurate delineation of structures is essential. Models like U-Net have proven effective in segmenting complex medical images, aiding in disease diagnosis and treatment planning.

Applications

Medical imaging: Detecting anomalies like tumors in MRI scans, facilitating early diagnosis and treatment (Figure 5).

Autonomous vehicles: Enabling vehicles to recognize and navigate around obstacles, pedestrians, and other vehicles.

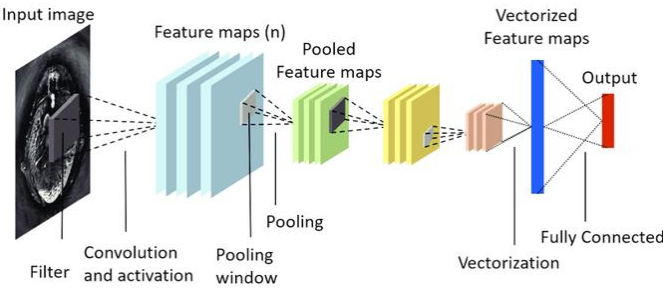


Figure 5: Medical imaging.

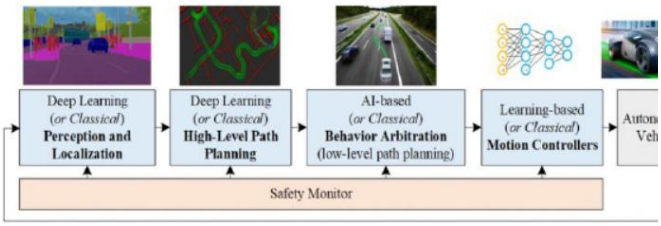


Figure 6: Autonomous vehicles.

Natural Language Processing (NLP)

Text classification: CNNs can process text data for tasks like sentiment analysis, spam detection, and document classification by treating text sequences similarly to one-dimensional images. They can capture local correlations in text through convolutional operations.

Named entity recognition: CNNs help in identifying entities such as names, locations, and dates within text, crucial for information extraction and knowledge management.

Sentence modeling: CNNs can effectively model sentences for tasks such as language translation and text summarization by capturing the hierarchical nature of linguistic features.

Applications

Spam detection

Identifying spam emails by analyzing the textual content.

Customer support

Classifying and routing customer inquiries to appropriate departments based on content.

Healthcare

Medical image analysis

CNNs are used for detecting and diagnosing diseases from medical images, including X-rays, MRIs, and CT scans. They can identify patterns and anomalies that might be missed by human experts.

Genomics

CNNs analyze DNA sequences to identify genetic markers associated with diseases, facilitating personalized medicine.

Applications

Disease detection

Identifying diseases like cancer or pneumonia from radiographic images with high accuracy.

Drug discovery

Analyzing biological data to discover new drugs and predict their efficacy.

Robotics

Object recognition

CNNs enable robots to identify and interact with objects in their environment, essential for tasks like assembly line work and autonomous navigation.

Motion analysis

CNNs help in analyzing and predicting the movements of robots, improving their ability to perform complex tasks autonomously.

Applications

Manufacturing

Robots using CNNs to inspect products for defects and ensure quality control.

Autonomous navigation

Enabling drones and robots to navigate through complex environments by recognizing obstacles and planning paths.

UNDERSTANDING CNN

Basic Architecture of CNN

The basic architecture of CNN consists of several key layers, each serving a specific function to progressively extract and learn features from the input data. Here's an overview of the fundamental components of a CNN:

Input layer

The input layer receives the raw data, typically in the form of an image represented by a matrix of pixel values. For instance, a grayscale image is represented by a 2D matrix, while a color image is represented by a 3D matrix with dimensions corresponding to width, height, and color channels (e.g., RGB). Example: For a 28 × 28 grayscale image, the input layer would be a matrix of size 28 × 28.

Convolutional layer

A convolutional layer is a fundamental building block of CNNs. It performs a convolution operation, which is a specialized kind of linear operation. Convolutional layers are primarily used for feature extraction from input data, such as images, by applying convolutional filters (kernels) to the input.

Filters/Kernels

Kernels are fundamental for feature extraction in CNNs. Each kernel learns to detect specific features from the input data (Figure 7).

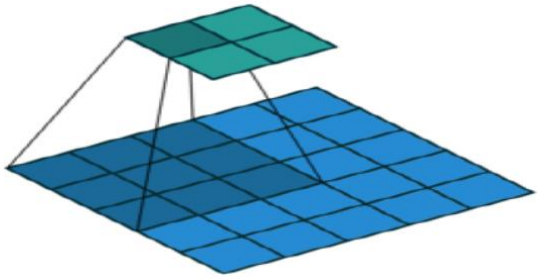


Figure 7: Kernels.

Here the green matrix is the input image, where we apply kernel which is moving over matrix to learn the different features of the original image. The kernel firstly, move over x-axis, then shift down and again move over x-axis. The sum of the dot product of the image pixel value gives the output matrix.

If a m × m image convolved with n × n kernel, the output image is of size (m - n + 1) × (m - n + 1).

$$\text{Input} = \begin{pmatrix} a & \cdots & b \\ \vdots & \ddots & \vdots \\ c & \cdots & d \end{pmatrix}_{mxm} \quad \text{Kernel} = \begin{pmatrix} p & \cdots & q \\ \vdots & \ddots & \vdots \\ r & \cdots & s \end{pmatrix}_{nxn}$$

$$\text{Output} = \begin{pmatrix} k & \cdots & l \\ \vdots & \ddots & \vdots \\ x & \cdots & y \end{pmatrix}_{m-n+1xm-n+1}$$

Standard type of kernels/filters

We can use any matrix as kernels in CNNs. In CNNs, basically we follow some standard kernels for optimizing output.

Identity kernel

This kernel does not alter the image, it is often used as a starting padding.

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Edge detection kernels

These kernels are used to highlight the edge in images (Figure 8) (Table 1).

Sobel Kernel	$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$
Prewitt Kernel	$\begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$
Laplacian Kernel	$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 6 & -1 \\ 0 & -1 & 0 \end{pmatrix}$

Table 1: Edge detection kernels.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Load an image
image = cv2.imread('img/bird_img.jpg', 0)

# Define Sobel kernel for edge detection
sobel_kernel = np.array([[-1, 0, 1],
                         [-2, 0, 2],
                         [-1, 0, 1]])

# Apply the kernel to the image
sobel_edge = cv2.filter2D(image, -1, sobel_kernel)

# Display the original and edge-detected images
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')

plt.subplot(1, 2, 2)
plt.title('Sobel Edge Detection')
plt.imshow(sobel_edge, cmap='gray')

plt.show()
```

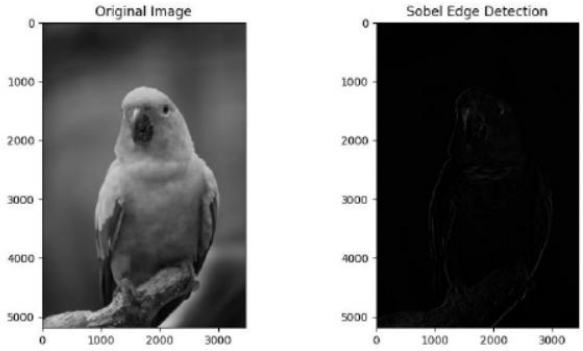


Figure 8: Original image *vs.* sobel edge detection.

Blurring kernels

These kernels are used to smooth out the image, reducing noise and details (Figure 9) (Table 2).

Box Blur Kernel	$\begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}$
Gaussian Blur Kernel	$\begin{pmatrix} 1/16 & 2/16 & 1/16 \\ 2/16 & 4/16 & 2/16 \\ 1/16 & 2/16 & 1/16 \end{pmatrix}$

Table 2: Blurring kernels.

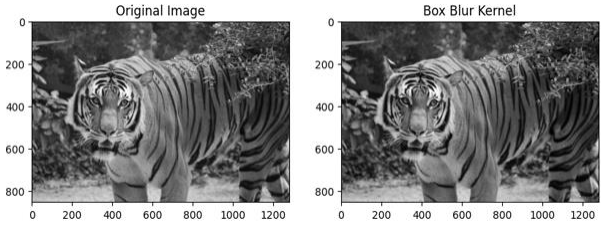


Figure 9: Original image *vs.* box blur kernel.

Emboss kernel

This kernel has a 3D shading effect (Figure 10).

$$\begin{pmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$

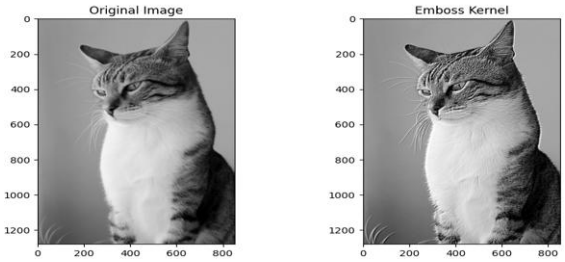


Figure 10: Original *vs.* emboss kernel.

Small matrices (e.g., 3 × 3 or 5 × 5) that detect local patterns such as edges, textures, and shapes.

Padding

Padding refers to the practice of adding extra pixels to the borders of an input image before applying the convolution operation. Basically (Figure 11), we face two problems without padding:

Problem 1: In image classification task, we need to apply multiple time convolution. If m × m is original image with n × n kernel, then our output matrix will be m - n + 1, m - n + 1. So, after multiple convolutions, our original image will be very small.

Problem 2: When kernel moves over original matrix, it touches the edge less than middle part of image. So, the corner features of any image or edges are not used much in the output.

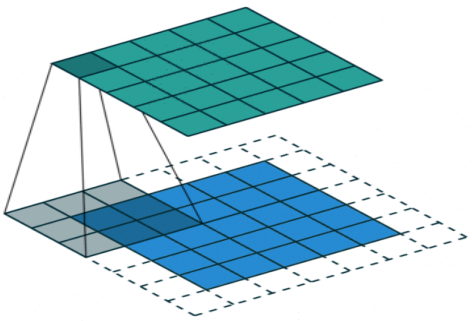


Figure 11: Padding.

To fixed, this problem we introduce Padding to preserve the size of original image with essential features. So, if m × m original matrix with n × n kernel with p padding, then output will be (m + 2p - n + 1)×(m + 2p - n + 1) matrix.

Stride

Stride is the number of pixels shifts over the input matrix. Stride affects the spatial dimensions of the output feature maps and the computational efficiency of the network. When the stride is set to 1, the kernel moves one pixel at a time. This results in overlapping applications of the kernel and produces an output feature map with the maximum possible spatial dimensions. When the stride is set to a value greater than 1, the kernel moves by that many pixels at a time. This reduces the spatial dimensions of the output feature map and increases computational efficiency, but it may result in loss of spatial detail (Figure 12).

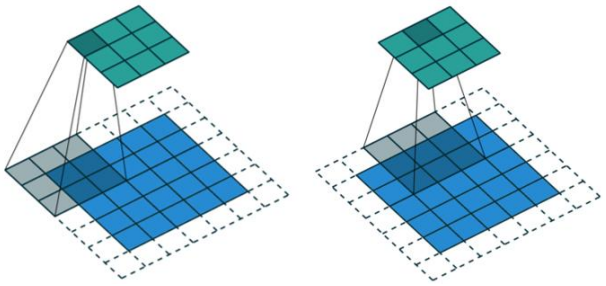


Figure 12: Stride.

For padding p, input image $m \times m$, kernel $n \times n$, with stride s, then output matrix will be $(\frac{m+2p-n}{s} + 1) \times (\frac{m+2p-n}{s} + 1)$.

Activation function

In Convolutional Neural Networks (CNNs), an activation function is a crucial component that introduces non-linearity into the network. This allows the network to learn and model complex patterns. The activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it.

The purpose of the activation function is to introduce non-linearity into the output of a neuron.

A neural network without a function is essentially just a linear regression model. The activation function does the non-linear transformation to the input making it capable of learning and performing more complex tasks.

Most useful activation function are:

- Sigmoid Function
- Tanh
- ReLU
- SoftMax

Linear vs. non-linear functions

Linear functions

A linear function is of the form f(x) = ax + b, which can only capture linear relationships between inputs and outputs. If all layers in a neural network were linear, the entire network could be reduced to a single linear transformation. No matter how many layers we stack, the final transformation would still be linear. This limitation severely restricts the ability of the network to model complex patterns in the data.

Non-linear functions

Non-linear functions can capture a wide range of behaviors and relationships that linear functions cannot. Examples of non-linear functions include polynomials, exponential functions, and trigonometric functions. Non-linearity allows the network to create complex mappings from inputs to outputs, enabling it to learn and model intricate patterns in data. Importunacy of Non-Linear Function in CNNs are:

Non-linear activation functions allow neural networks to approximate any continuous function. This is known as the Universal Approximation Theorem, which states that a neural network with at least one hidden layer and a non-

linear activation function can approximate any continuous function to any de-sired accuracy given sufficient neurons.

In CNNs, stacking layers helps in creating hierarchical feature representations. Early layers might detect simple features like edges and textures, while deeper layers can capture more complex features like shapes and objects. Non-linearity between these layers allows each layer to build upon the previous one in a meaningful way.

Sigmoid Function

The sigmoid function is a type of activation function that maps any real-valued number into a range between 0 and 1. It is a function which is plotted as 'S' shaped graph (Figure 13). This is express by

$$\sigma = \frac{1}{1 + e^{-x}}$$

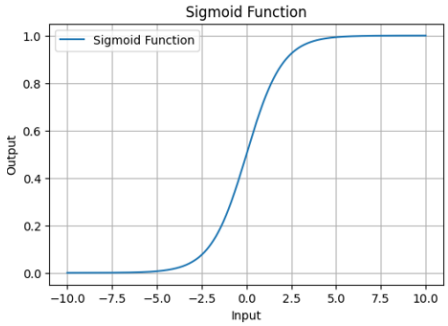


Figure 13: Sigmoid functions.

Properties

- Domain: (-∞, +∞)
- Range: (0, 1)
- Continuous, Differentiable, monotonically increasing everywhere.

The sigmoid function is commonly used in the output layer of a binary classification model, as its output range is [0,1], which can be interpreted as a probability. A threshold (e.g., 0.5) is often used to decide the class label (1 if the value is greater than 0.5, otherwise 0). It is derivative $\sigma'(x) = \sigma(x)[1-\sigma(x)]$.

Tanh Function

The hyperbolic tangent (tanh) function is another activation function that maps any real-valued number into a range between -1 and 1 (Figure 14). This express by

$$\tanh x = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

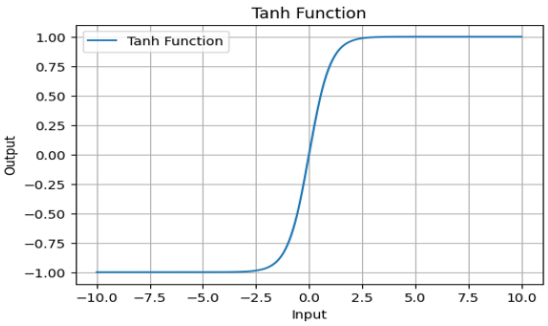


Figure 14: Tanh function.

Properties

- Domain: $(-\infty, +\infty)$
- Range: $(-1, +1)$
- Continuous: The function is continuous everywhere.
- Differentiable: The function is differentiable.
- Zero-Centered: Unlike the sigmoid function, the tanh function is zero-centered, which can help with faster convergence. Zero-centered means that the activation function's output values range symmetrically around zero. For the tanh function, the range is $(-1, +1)$ with a mean value of 0.

It's derivative $\tanh`(x) = 1 - \tanh^2(x)$

ReLU Function

The Rectified Linear Unit (ReLU) is an activation function that outputs the input directly if it is positive, otherwise, it outputs zero (Figure 15). This express by

$$ReLU(x) = max(0, x)$$

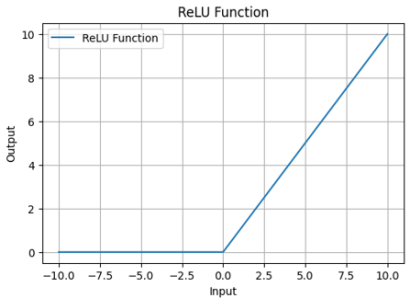


Figure 15: ReLU function.

Properties

- Domain: (-∞,+∞)
- Range: [0, +∞)
- Continuous: The function is continuous but not differentiable at
- Non-Saturating: Unlike sigmoid and tanh, ReLU does not saturate for positive values, which helps mitigate the vanishing gradient problem.

Softmax Function

The softmax function is an activation function used in the output layer of a neural network for multi-class classification problems (Figure 16). This express by:

$$softmax(x_i) = \frac{e^{(x_i)}}{\sum_j e^{(x_j)}}$$

where x_i is the i-th element of the input vector.

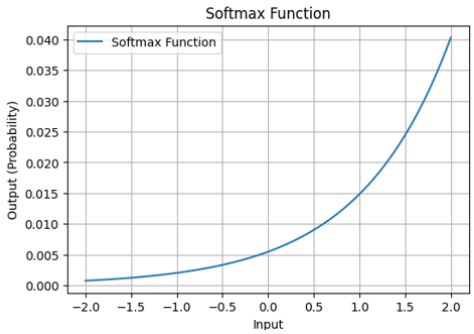


Figure 16: Softmax function.

Properties

- Domain: $(-\infty, +\infty)$ for each element of the input vector.
- Range: (0, 1) for each element of the output vector.
- Continuous: The function is continuous.
- Differentiable: The function is differentiable.
- Probabilistic Interpretation: The output values can be interpreted as probabilities that sum to 1.

Pooling Layer

The pooling layer reduces the spatial dimensions of the feature maps, making the computation more efficient and reducing the likelihood of overfitting.

Max pooling

Selects the maximum value from each region of the feature map.

Average pooling

Computes the average value of each region.

Stride and pool size

Similar to the convolutional layer, these determine how the pooling window moves and its size.

Example

Max Pooling with 2 × 2 window and stride of 2: Reduces a 4 × 4 feature map to 2 × 2 by taking the maximum value in each 2 × 2 window.

Fully Connected Layer (Dense Layer)

After several convolutional and pooling layers, the high-level reasoning is done by fully connected layers.

Flattening

The output from the final pooling or convolutional layer is flattened into a 1D vector.

Dense layers

Each neuron in a dense layer is connected to every neuron in the previous layer, allowing for high-level feature integration and decision making.

Example: If the flattened vector has 128 features, each neuron in the dense layer will have 128 weights.

Output Layer

The output layer provides the final predictions of the network. For classification tasks, this layer typically uses a softmax activation function to output prob-abilities for each class.

Forward Propagation

Feedforward propagation is to pass input data through the network and produce an output. The input image or data is fed into the network and apply convolution operations using kernels to detect various features in the input data. After each convolution operation, an activation function (like ReLU, sigmoid, or tanh) is applied to introduce non-linearity. Then reduce the spatial dimensions (width and height) of the feature maps while retaining the most important information, typically using operations like max pooling or average pooling. The reduced feature maps are flattened into a single vector and passed through one or more fully connected (dense) layers. These layers combine the features to predict the final output.

Loss Calculation

The loss calculation in Convolutional Neural Networks (CNNs) is a crucial step in training the network. It measures the discrepancy between the network's predictions and the actual target values. The objective is to minimize this loss during training, leading to better model performance.

Types of loss functions

The choice of loss function depends on the type of task (e.g., classification, regression). Here are some common loss functions:

Cross-Entropy Loss (Log Loss)

$$\iota = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic}\log(\hat{y}_{ic})$$

Where N is the number of samples, C is the number of classes, y_{ic} is the binary indicator (o or 1), $\hat{y}_{ic}$ is the predicted probability that sample i belong to class c.

Mean squared error (MSE)

$$\iota = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Where N is the number of samples, y_i is the true value for the i-th sample, $\hat{y}_i$ is the predicted value for i-th sample.

Binary cross-entropy loss

$$\iota = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)]$$

Where N is the number of samples, C is the number of classes, $y_i c$ is the binary indicator (0 or 1), $\hat{y}_{ic}$ is the predicated probability that sample i belong to class C.

Categorical cross-entropy

$$\iota = -(1/N)\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic}\log(\hat{y}_{ic})$$

Where N is the number of samples, C is the number of classes, y_{ic} is the binary indicator (0 or 1), $\hat{y}_{ic}$ is the predicated probability that sample i-th belong to class C.

Backpropagation

The purpose of backpropagation is to update the weights of the network to minimize the loss function. It adjusts the parameters of the network based on the error between the predicted output and the actual target. Compute the loss (error) using a loss function. Then we apply backward pass, Compute the gradient of the loss with respect to the output of the network. Propagate the gradient back through the fully connected layers, updating the weights and biases using gradient descent. Pass the gradient through the pooling layers. In the case of max pooling, the gradient is routed to the location that had the maximum value in the

forward pass. the gradient of the loss with respect to the filters and the input feature maps. Update the filters (weights) using gradient descent. The gradients are passed through the activation functions using the chain rule. For functions like ReLU, this involves simple operations, while for sigmoid or tanh, it involves computing derivatives. Adjust the weights and biases of the network using the computed gradients. This is typically done using an optimization algorithm like stochastic gradient descent (SGD), Adam, or RMSprop.

Iteration

The entire process of forward propagation, loss calculation, and backpropagation is repeated for many epochs (iterations over the training dataset) until the model converges to a solution (i.e., the loss is minimized).

SKIN CANCER CLASSIFICATION USING CNN

Skin Cancer Classification project is an exciting exploration into developing a Convolutional Neural Network (CNN) model designed to classify various types of skin cancer from images. To assist medical professionals by providing an accurate, automated tool that aids in the early detection and diagnosis of skin can-cer, potentially saving lives through early intervention.

Objective of the Project

Skin cancer is one of the most common types of cancer, but early detection significantly improves the chances of successful treatment. However, identifying the type of skin cancer can be challenging even for experienced dermatologists. Our objective is to create a CNN model that can reliably classify skin cancer im-ages into distinct categories, providing a valuable second opinion that can enhance diagnostic accuracy and speed.

Dataset used for Training and Testing

For this ambitious endeavor, we used the International Skin Imaging Collaboration (ISIC) dataset (Table 3). This comprehensive dataset comprises thousands of dermatoscope images representing common pigmented skin lesions, including:

- Actinic Keratosis
- Basal Cell Carcinoma
- Dermatofibroma
- Melanoma
- Nevus
- Pigmented Benign Keratosis
- Seborrheic Keratosis
- Squamous Cell Carcinoma
- Vascular Lesion

Number of Images on ISIC Skin Dataset	Train (Files)	Test (Files)
Actinic Keratosis	114	16
Basal Cell Carcinoma	376	16
Dermatofibroma	95	16
Melanoma	438	17
Nevus	357	16
Pigmented Benign Keratosis	462	16
Seborrheic Keratosis	77	3
Squamous Cell Carcinoma	181	16
Vascular Lesion	139	3

Table 3: Dataset used for training and testing.

We divided the dataset into training and testing sets, ensuring our model was trained on a wide variety of images and evaluated rigorously to assess its performance.

Architecture of the CNN Model Implemented

Our CNN model is a carefully crafted network designed to extract and learn from the complex patterns within skin lesion images. Here's a closer look at its architecture:

Input layer:

The model starts by accepting images of size 128 × 128 × 3 (height, width, and color channels).

Convolutional layers:

- The first Conv2D layer uses 32 filters with a 3x3 kernel size and ReLU activation to detect edges and textures.

- A MaxPooling2D layer follows, reducing the spatial dimensions and emphasizing the most critical features.
- The second Conv2D layer expands to 64 filters, deepening the network's understanding of the image.
- Another MaxPooling2D layer to consolidate learning.
- The third Conv2D layer, with 128 filters, captures even more detailed patterns.
- A final MaxPooling2D layer further distills the feature maps.

Flatten layer:

This layer transforms the 3D feature maps into a 1D vector, making it suitable for dense layers.

Dense layers:

A Dense layer with 128 units and ReLU activation processes the combined features. A Dropout layer with a rate of 0.5 helps prevent overfitting by randomly disabling neurons during training. The output layer is a Dense layer with 9 units (one for each class), using softmax activation to produce probabilities for each class.

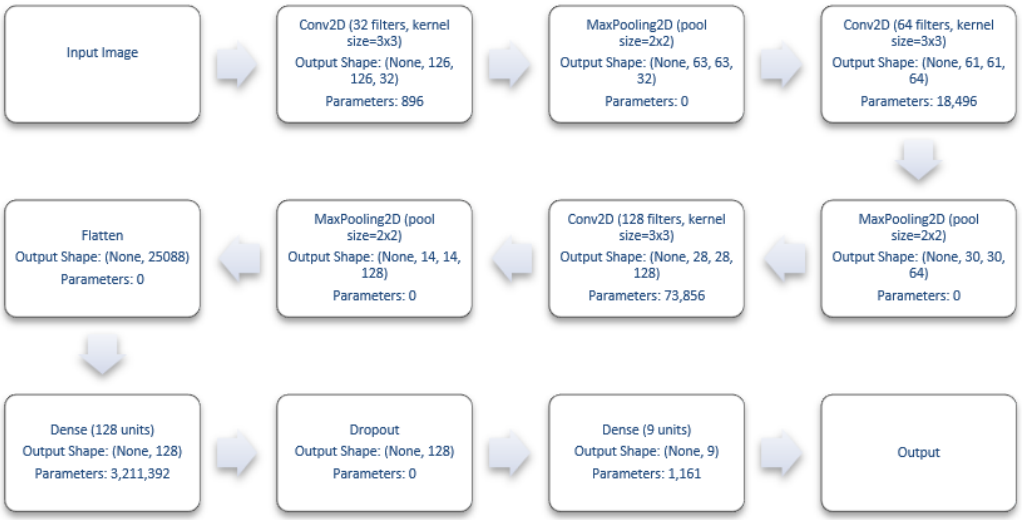
```
# Install necessary libraries

import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import
train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.preprocessing.image import
ImageDataGenerator, load_img, img_to_array
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Max-
Pooling2D, Flatten, Dense, Dropout
from tensorflow.keras.models import load_model
from flask import Flask, request, jsonify

# Define the path to the dataset
base_dir = 'ISIC_Skin_Cancer_Dataset/Train'

# Define image dimensions
img_width, img_height = 128, 128

# Initialize lists for images and labels
images = []
labels = []

# Load and preprocess images
for label in os.listdir(base_dir):
    label_dir = os.path.join(base_dir, label)
    if os.path.isdir(label_dir):
        for img_name in os.listdir(label_dir):
            img_path = os.path.join(label_dir,
img_name)
            img = load_img(img_path, tar-
get_size=(img_width, img_height))
            img_array = img_to_array(img)
            images.append(img_array)
            labels.append(label)

# Convert lists to numpy arrays
images = np.array(images)
labels = np.array(labels)

# Encode labels
label_encoder = LabelEncoder()
labels = label_encoder.fit_transform(labels)

# Split data into training and testing sets
X_train, X_test, y_train, y_test =
train_test_split(images, labels, test_size=0.2,
random_state=42)

# Display some images with their labels to verify
def display_images(images, labels, label_encoder,
num_images=5):
    plt.figure(figsize=(20, 10))
    for i in range(num_images):
        plt.subplot(1, num_images, i+1)
        plt.imshow(images[i].astype('uint8'))
        plt.title(label_encoder.inverse_trans-
form([labels[i]])[0])
        plt.axis('off')
    plt.show()

display_images(X_train, y_train, label_encoder)

# Data augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

```
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow(X_train,
y_train, batch_size=32)
test_generator = test_datagen.flow(X_test, y_test,
batch_size=32)

# Build the model
model = Sequential()

# Add convolutional layers
model.add(Conv2D(32, (3, 3), activation='relu',
input_shape=(img_width, img_height, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Flatten the output
model.add(Flatten())

# Add fully connected layers
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(len(np.unique(labels)), activa-
tion='softmax'))

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

# Summary of the model
model.summary()

# Train the model
history=model.fit(train_generator,epochs=25,vali-
dation_data=test_generator)

# Evaluate the model
```

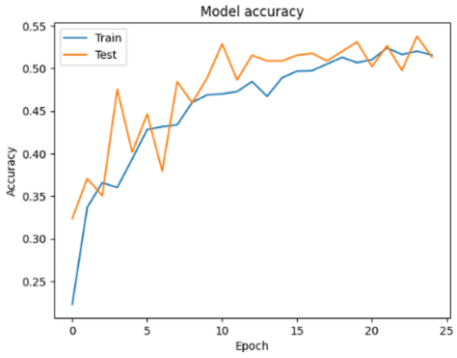
```
test_loss, test_accuracy = model.evalu-
ate(test_generator)
print(f'Test Accuracy: {test_accuracy:.2f}')

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# Save the model
model.save('skin_cancer_model.h5')
```

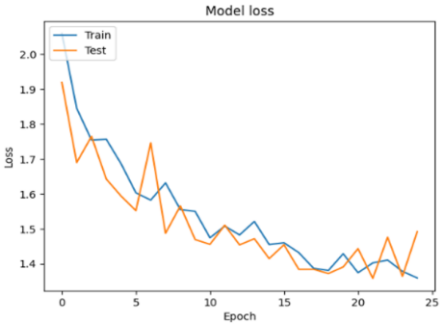
RESULTS ACHIEVED

Layer (Type)	Output Shape	Param
Conv2d	(None, 126, 126, 32)	896
Max_pooling2d	(None, 63, 63, 63)	0
Conv2d_1	(None, 61, 61, 64)	18,496
Max_pooling2d_1	(None, 30, 30, 64)	0
Conv2d_2	(None, 28, 28, 128)	73,856
Max_pooling2d_2	(None, 14, 14, 128)	0
Flatten	(None, 25088)	0
Dense	(None, 128)	32,11,392
Dropout	(None, 128)	0
Dense_1	(None, 9)	1,161



After training our model for 25 epochs, we evaluated its performance on the test dataset using accuracy and loss as key metrics. Here's a summary of our findings:





	Accuracy	Loss
Training	48.96%	1.38
Validation	51.34%	1.49
Test	47.35%	1.57

These results, while promising, indicate that there is room for improvement. By experimenting with more advanced architectures, fine-tuning hyperparameters, and employing data augmentation techniques, we believe the model's accuracy can be significantly enhanced.

CONCLUSION

In this project, we have taken significant strides towards harnessing the power of deep learning to aid in the early detection and classification of skin cancer. Our CNN model, trained on the extensive ISIC dataset, showcases the potential of AI in medical diagnostics. While the journey is ongoing, and there are challenges to overcome, the progress made thus far is encouraging. With further refinement, this technology could become an invaluable tool in the fight against skin cancer, ultimately saving lives through timely and accurate diagnosis.

REFERENCES

1. Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press.
2. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553): 436-444.
3. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems 25: 1097-1105.
4. Esteva A, Kuprel B, Novoa RA, et al. (2017) Dermatologist-level classification of skin cancer with deep neural networks. Nature 542(7639): 115-118.
5. Codella NC, Gutman D, Celebi ME, et al. (2018) Skin lesion analysis toward melanoma detection: A challenge at the 2017 international symposium on biomedical imaging (isbi), hosted by the international skin imaging collaboration (isic). In 2018 IEEE 15th international symposium on biomedical imaging (ISBI 2018): 168-172.
6. He K, Zhang X, Ren S, et al. (2016) Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition: 770-778.
7. Lopez A (2021) Melanoma and nonmelanoma skin cancer classification using deep learning-based data fusion: A comparative study. PLOS ONE 16(8): e0256041.
8. Tschandl P, Rinner C, Apalla Z, et al. (2020) Human–computer collaboration for skin cancer recognition. Nature Medicine 26(8): 1229-1234.
9. Zhang J, Xie Y, Xia Y, et al. (2019) Attention residual learning for skin lesion classification. IEEE Transactions on Medical Imaging 38(9): 2092-2103.