**IMPLEMENTATION**

**MODULES:**

* Dataset
* Importing the necessary libraries
* Retrieving the images
* Splitting the dataset
* Building the model
* Apply the model and plot the graphs for accuracy and loss
* Accuracy on test set
* Saving the Trained Model

**MODULES DESCSRIPTION:**

**Dataset:**

In the first module of Skin Cancer Prediction Using Deep Learning Techniques, we developed the system to get the input dataset. Data collection process is the first real step towards the real development of a machine learning model, collecting data. This is a critical step that will cascade in how good the model will be, the more and better data that we get; the better our model will perform. There are several techniques to collect the data, like web scraping, manual interventions. Our dataset is placed in the project and it’s located in the model folder. The dataset is referred from the popular standard dataset repository kaggle where all the researchers refer it. The dataset consists of 10,015 images. The following is the URL for the dataset referred from kaggle.

URL: <https://www.kaggle.com/datasets/jayaprakashpondy/skin-cancer-dataset>

**Importing the necessary libraries:**

We will be using Python language for this. First we will import the necessary libraries such as keras for building the main model, sklearn for splitting the training and test data, PIL for converting the images into array of numbers and other libraries such as pandas, numpy, matplotlib and tensorflow.

**Retrieving the images:**

In this module we will retrieve the images from the dataset and convert them into a format that can be used for training and testing the model. This involves reading the images, resizing them, and normalizing the pixel values. We will retrieve the images and their labels. Then resize the images to (28, 28) as all images should have same size for recognition. Then convert the images into numpy array.

This module involves loading and retrieving the preprocessed images and their corresponding class labels from the dataset. The images will be utilized as input to the deep learning model for training and evaluation.

**Splitting the dataset:**

In this module, the image dataset will be divided into training and testing sets. Split the dataset into Train and Test. 80% train data and 20% test data. This will be done to train the model on a subset of the data, validate the model's performance, and test the model on unseen data to evaluate its accuracy. Split the dataset into train and test. 80% train data and 20% test data.

To train and evaluate the model effectively, the dataset needs to be split into training, validation, and test sets. This module divides the dataset into these subsets, ensuring that the model is trained on one portion, validated on another, and finally tested on unseen data to assess its generalization performance.

**Building the model:**

The concept of convolutional neural networks are very successful in image recognition. The key part to understand, which distinguishes CNN from traditional neural networks, is the convolution operation. Having an image at the input, CNN scans it many times to look for certain features. This scanning (convolution) can be set with 2 main parameters: stride and padding type. As we see on below picture, process of the first convolution gives us a set of new frames, shown here in the second column (layer). Each frame contains an information about one feature and its presence in scanned image. Resulting frame will have larger values in places where a feature is strongly visible and lower values where there are no or little such features. Afterwards, the process is repeated for each of obtained frames for a chosen number of times. In this project I chose a classic CNN  model which contains only two convolution layers.

The latter layer we are convolving, the more high-level features are being searched. It works similarly to human perception. To give an example, below is a very descriptive picture with features which are searched on different CNN layers. As you can see, the application of this model is face recognition. You may ask how the model knows which features to seek. If you construct the CNN from the beginning, searched features are random. Then, during training process, weights between neurons are being adjusted and slowly CNN starts to find such features which enable to meet predefined goal, i.e. to recognize successfully images from the training set.

Between described layers there are also pooling (sub-sampling) operations which reduce dimensions of resulted frames. Furthermore, after each convolution we apply a non-linear function (called **ReLU**) to the resulted frame to introduce non-linearity to the model.

Eventually, there are also fully connected layers at the end of the network. The last set of frames obtained from convolution operations is flattened to get a one-dimensional vector of neurons. From this point we put a standard, fully-connected neural network. At the very end, for classification problems, there is a softmax layer. It transforms results of the model to probabilities of a correct guess of each class

**CNN| CNN model**

*Architecture:*

A CNN typically has three layers: a convolutional layer, a pooling layer, and a fully connected layer.

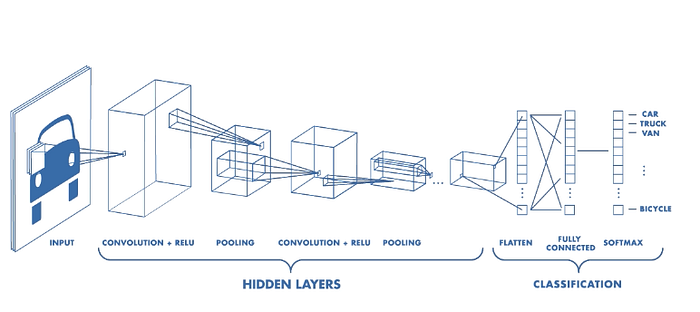
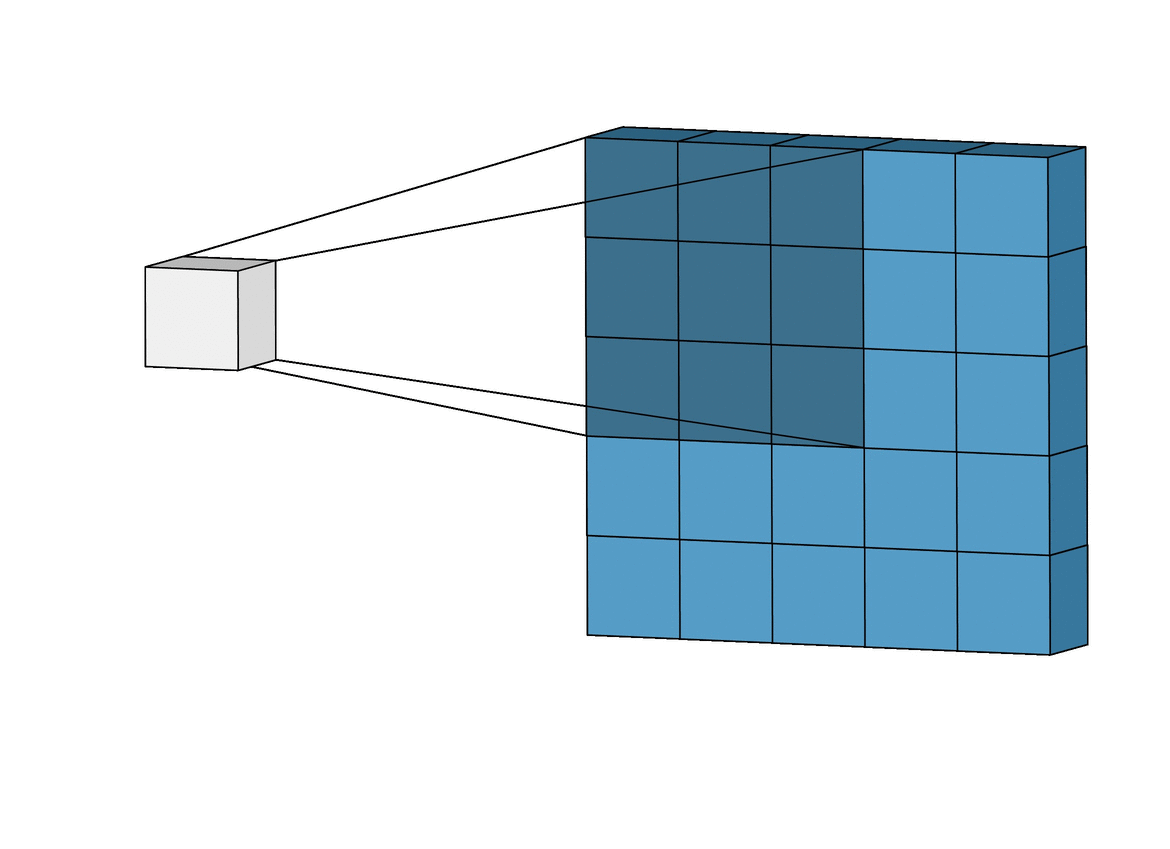


Figure: Architecture of a CNN ([Source](https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html))

*Convolution Layer*

The convolution layer is the core building block of the CNN. It carries the main portion of the network’s computational load.

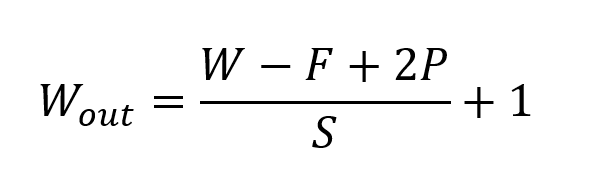
This layer performs a dot product between two matrices, where one matrix is the set of learnable parameters otherwise known as a kernel, and the other matrix is the restricted portion of the receptive field. The kernel is spatially smaller than an image but is more in-depth. This means that, if the image is composed of three (RGB) channels, the kernel height and width will be spatially small, but the depth extends up to all three channels.



*Illustration of Convolution Operation*

During the forward pass, the kernel slides across the height and width of the image-producing the image representation of that receptive region. This produces a two-dimensional representation of the image known as an activation map that gives the response of the kernel at each spatial position of the image. The sliding size of the kernel is called a stride.

If we have an input of size W x W x D and Dout number of kernels with a spatial size of F with stride S and amount of padding P, then the size of output volume can be determined by the following formula:



Formula for Convolution Layer

This will yield an output volume of size Wout x Wout x Dout.

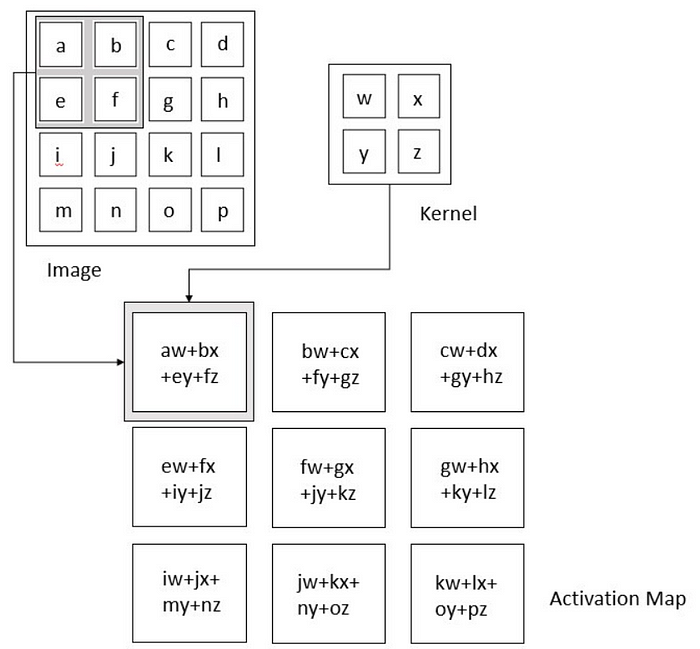


Figure : Convolution Operation (Source: Deep Learning by Ian Goodfellow, Yoshua Bengio, and Aaron Courville)

***Motivation behind Convolution***

Convolution leverages three important ideas that motivated computer vision researchers: sparse interaction, parameter sharing, and equivariant representation. Let’s describe each one of them in detail.

Trivial neural network layers use matrix multiplication by a matrix of parameters describing the interaction between the input and output unit. This means that every output unit interacts with every input unit. However, convolution neural networks have sparse interaction. This is achieved by making kernel smaller than the input e.g., an image can have millions or thousands of pixels, but while processing it using kernel we can detect meaningful information that is of tens or hundreds of pixels. This means that we need to store fewer parameters that not only reduces the memory requirement of the model but also improves the statistical efficiency of the model.

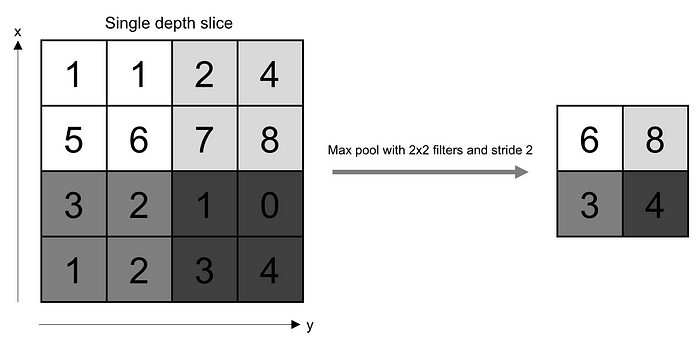
If computing one feature at a spatial point (x1, y1) is useful then it should also be useful at some other spatial point say (x2, y2). It means that for a single two-dimensional slice i.e., for creating one activation map, neurons are constrained to use the same set of weights. In a traditional neural network, each element of the weight matrix is used once and then never revisited, while convolution network has shared parameters i.e., for getting output, weights applied to one input are the same as the weight applied elsewhere.

Due to parameter sharing, the layers of convolution neural network will have a property of equivariance to translation. It says that if we changed the input in a way, the output will also get changed in the same way.

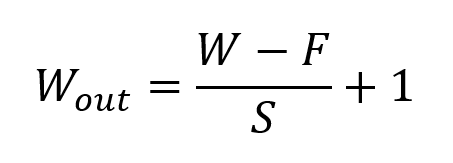
*Pooling Layer*

The pooling layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs. This helps in reducing the spatial size of the representation, which decreases the required amount of computation and weights. The pooling operation is processed on every slice of the representation individually.

There are several pooling functions such as the average of the rectangular neighborhood, L2 norm of the rectangular neighborhood, and a weighted average based on the distance from the central pixel. However, the most popular process is max pooling, which reports the maximum output from the neighborhood.



If we have an activation map of size W x W x D, a pooling kernel of spatial size F, and stride S, then the size of output volume can be determined by the following formula:



*Formula for Padding Layer*

This will yield an output volume of size Wout x Wout x D.

In all cases, pooling provides some translation invariance which means that an object would be recognizable regardless of where it appears on the frame.

*Fully Connected Layer*

Neurons in this layer have full connectivity with all neurons in the preceding and succeeding layer as seen in regular FCNN. This is why it can be computed as usual by a matrix multiplication followed by a bias effect.

The FC layer helps to map the representation between the input and the output.

*Non-Linearity Layers*

Since convolution is a linear operation and images are far from linear, non-linearity layers are often placed directly after the convolutional layer to introduce non-linearity to the activation map.

There are several types of non-linear operations, the popular ones being:

***1. Sigmoid***

The sigmoid non-linearity has the mathematical form σ(κ) = 1/(1+e¯κ). It takes a real-valued number and “squashes” it into a range between 0 and 1.

However, a very undesirable property of sigmoid is that when the activation is at either tail, the gradient becomes almost zero. If the local gradient becomes very small, then in backpropagation it will effectively “kill” the gradient. Also, if the data coming into the neuron is always positive, then the output of sigmoid will be either all positives or all negatives, resulting in a zig-zag dynamic of gradient updates for weight.

***2. Tanh***

Tanh squashes a real-valued number to the range [-1, 1]. Like sigmoid, the activation saturates, but — unlike the sigmoid neurons — its output is zero centered.

***3. ReLU***

The Rectified Linear Unit (ReLU) has become very popular in the last few years. It computes the function ƒ(κ)=max (0,κ). In other words, the activation is simply threshold at zero.

In comparison to sigmoid and tanh, ReLU is more reliable and accelerates the convergence by six times.

Unfortunately, a con is that ReLU can be fragile during training. A large gradient flowing through it can update it in such a way that the neuron will never get further updated. However, we can work with this by setting a proper learning rate.

**Apply the model and plot the graphs for accuracy and loss:**

Once the model is built, it will be applied to the validation set to evaluate its accuracy and loss. The accuracy and loss will be plotted as a function of the number of epochs to visualize the performance of the model. We will compile the model and apply it using fit function. The epoch will be 100. Then we will plot the graphs for accuracy and loss. We got average training accuracy of 96.00%.

**Accuracy on test set:**

After training and evaluating the model on the validation set, the accuracy of the model will be assessed on the test set. The accuracy on the test set will be an important metric for evaluating the model's performance. We got an accuracy of 97.00% on test set.

**Saving the Trained Model:**

Once you’re confident enough to take your trained and tested model into the production-ready environment, the first step is to save it into a .h5 or .pkl file using a library like pickle.

Make sure you have pickle installed in your environment.

Next, let’s import the module and dump the model into .h5 file.