Aim –

Design and implement CNN for image classification.

a) Select a suitable image classification dataset (medical engineering, agricultural, etc.).

b) Optimized with different hyper-parameters including learning rate, filter size, no. oflayers,optimizers, dropouts, etc.

Objective –

Design and implement a Convolutional Neural Network (CNN) for image classification on a selected image classification dataset, such as medical engineering, agricultural to optimize with different hyper-parameters including learning rate, filter size, no. of layers, optimizers, dropouts ,etc.

Software Requirements –

● Python (3.x recommended)

● Jupyter Notebook or any Python IDE or Google Colab

Hardware Requirements –

A machine with sufficient RAM and processing power for model training(8GBRAMrecommended)

Dataset –

Inbuilt dataset –

MNIST https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz

Libraries or Modules Used –

• Keras (for building and training neural network models)

• NumPy (for numerical operations)

• Matplotlib (for plotting images)

• TensorFlow (deep learning framework)

• Adam (optimizer for training)

Theory –

Convolutional Neural Network (CNN) Convolutional Neural Networks (CNN) are a powerful type of deep learning model specifically designed for processing and analyzing visual data, such as images and videos. The have revolutionized the field of Computer Vision, enabling remarkable advancements in tasks like Image Recognition, Object Detection, and Image Segmentation. To grasp the essence of Convolutional Neural Networks (CNNs), it is essential to have a solid understanding of the basics of Deep Learning and acquaint yourself with the terminology and principles of neural

networks. If you're new to this, don't fret! I have previously covered these fundamentals in my blog posts, serving as primers to help you lay a strong foundation.

Basic Architecture

The architecture of Convolutional Neural Networks is meticulously designed to extract meaningful features from complex visual data. This is achieved through the use of specialized layers within the network architecture.

It comprises of three fundamental layer types:

1. Convolutional Layers

2. Pooling Layers

3. Fully-Connected Layers

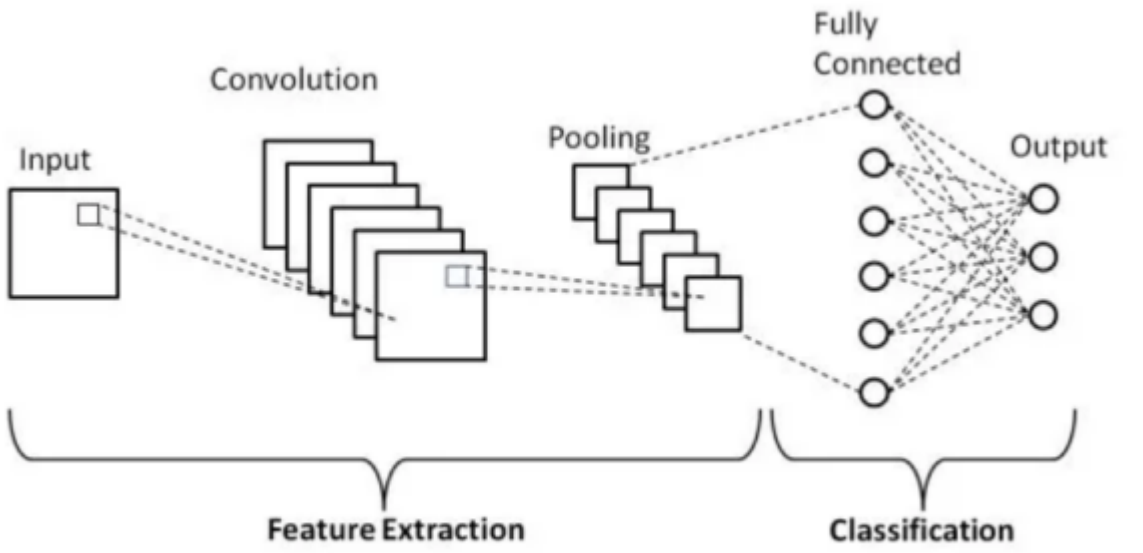


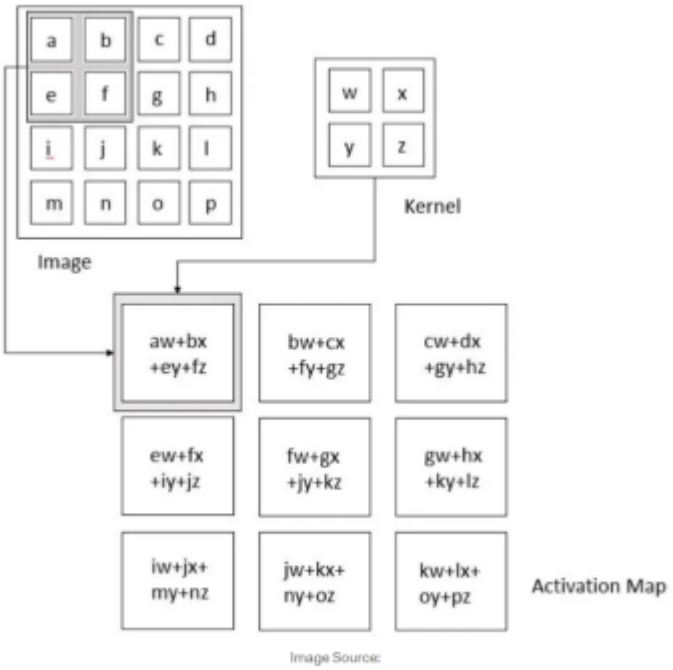
Image Source: Research Gate

Now, let's delve into each of these layers in detail to gain a deeper understandingof theirroleand significance in Convolutional Neural Networks (CNN).

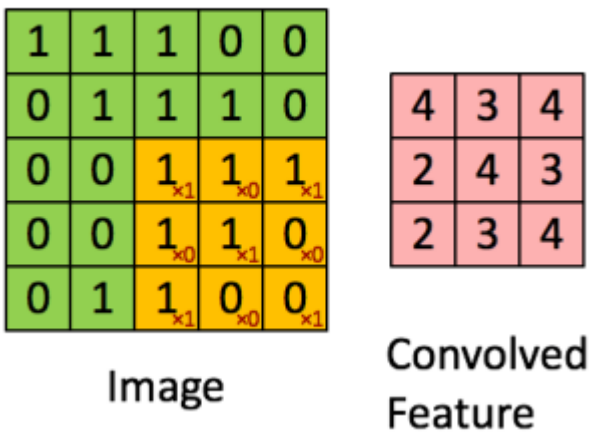
The Convolution Layer:

The convolutional layer serves as the fundamental building block within a Convolutional Neural Network (CNN), playing a central role in performing the majority of computations. It relies on several key components, including input data, filters, and feature maps. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data. It is a tensor operation (dot product) where two tensors serveas input, and a resulting tensor is generated as the output. This layer employs a tile-like filtering approach on an input tensor using a small window known as a kernel. The kernel specifies the specific characteristics that the convolution operation seeks to filter, generating a significantresponse when it detects the desired features. To

explore further details about various kernels and their functionalities, refer here. The convolutional layer computes a dot product between the filter value and the image pixel values, and the matrix formed by sliding the filter over the image is called the Convolved Feature ,Activation Map, or Feature Map.



Each element from one tensor (image pixel) is multiplied by the corresponding element (the element in the same position) of the second tensor(kernel value), and then all the values are summed to get the result.



The Pooling Layer:

Pooling layers, also referred to as down

sampling, serve to reduce the dimensionality of the input, thereby decreasing the number of parameters. Similar to convolutional layers, pooling operations involve traversing a filter across the input. However, unlike convolutional layers, the pooling filter does not possess weights. Instead, the filter applies an aggregation function to the values within its receptive field, generating the output array.

Two primary types of pooling are commonly employed:

Max Pooling:

It selects the pixel with the maximum value to send to the output array.

Average pooling:

It calculates the average value within the receptive field to send to the output array.

Fully-Connected Layer:

The Fully Connected Layer i.e dense layer aims to provide global connectivity between all neurons in the layer. Unlike convolutional and pooling layers, which operate on local spatial regions, the fully connected layer connects every neuron to every neuron in the previous and subsequent layers.

Algorithm –

1**. Load and Prepare Dataset:** Load the chosen image classification dataset (e.g., medical, agricultural) either from a local directory or using Tensor-Flow Datasets. Split the dataset into training and validation sets. Preprocess the images (resize, normalization) and preprocess labels (if necessary).

2. Define CNN Architecture: Design the CNN architecture using TensorFlow's Keras API. Construct the model with convolutional layers, pooling layers, fully connected layers, and appropriate activation functions. Specify input shape, number of classes, and layer configurations.

3. Compile the Model: Compile the model by specifying the optimizer, loss function, and evaluation metrics. Choose an appropriate optimizer (e.g., Adam) and a suitable loss function (e.g., sparse categorical cross entropy for multi-class classification).

4. Hyper-parameter Tuning and Training: Set hyper-parameters like learning rate, batch size, and number of epochs. Train the CNN model on the training set using the fit function. Utilize techniques like early stopping to prevent overfitting and monitor validation loss/accuracy.

5. Evaluate and Test: Evaluate the trained model's performance on the validation set to assess its accuracy and generalization .Once satisfied, use the model to predict and evaluate its accuracy on the test set to assess its real-world performance.

Application –

1. Medical Imaging: Disease Detection: Identifying tumors , lesions, or anomalies in MRI, X-ray, or CT scans for conditions like cancer, fractures, or internal organ abnormalities. Diagnosis Assistance: Analyzing retinal scans for diabetic retinopathy or identifying skin conditions through dermatology images.

2. Agriculture:

Crop Disease Identification: Classifying plant images to detect diseases, nutrient deficiencies ,or pest infestations in crops using systems like Plant Village.

Weed Detection: Identifying and distinguishing weeds from crops to enable targeted herbicide application.

3. Autonomous Vehicles: Object Recognition: Recognizing pedestrians, traffic signs, and other vehicles for safe navigation and decision-making in self-driving cars. Lane Detection: Identifying lane markings for autonomous vehicle guidance.

4. Retail and E-commerce:

Product Classification: Categorizing products for inventory management and cataloguing in e-commerce platforms.

Visual Search: Enabling visual search capabilities to find similar products using image scaptured by users.

5. **Security and Surveillance:**

Facial Recognition: Verifying identities or identifying individuals in security systems or surveillance footage.

Anomaly Detection: Spotting suspicious activities or objects in surveillance images or videos

Code –

```
import tensorflow as tf

from tensorflow.keras import datasets, layers, models

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from sklearn.model_selection import train_test_split import matplotlib.pyplot as plt

# Load and preprocess the MNIST dataset

(train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data() train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
 # Add channel dimension to the images

train_images = train_images.reshape((60000, 28, 28, 1)) test_images = test_images.reshape((10000, 28, 28, 1))

 # Split the dataset into training and validation sets

 train_images, val_images, train_labels, val_labels = train_test_split( train_images, train_labels, test_size=0.1, random_state=42 )

 # Data augmentation for training images

 datagen = ImageDataGenerator(rotation_range=10, zoom_range=0.1, width_shift_range=0.1,height_shift_range=0.1) datagen.fit(train_images)

 # Create a CNN model with hyperparameter tuning and regularization

 model = models.Sequential() model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1))) model.add(layers.MaxPooling2D((2, 2))) model.add(layers.Conv2D(64, (3, 3), activation='relu')) model.add(layers.MaxPooling2D((2, 2))) model.add(layers.Conv2D(128, (3, 3), activation='relu')) model.add(layers.Flatten()) model.add(layers.Dropout(0.5)) model.add(layers.Dense(128, activation='relu')) model.add(layers.Dense(10, activation='softmax'))

# Compile the model

 model.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy',metrics=['accuracy'])

# Train the model with data augmentation

history = model.fit(datagen.flow(train_images, train_labels, batch_size=64),

epochs=20, validation_data=(val_images, val_labels))

# Evaluate the model on the test set

 test_loss, test_acc = model.evaluate(test_images, test_labels) print(f"Test Accuracy: {test_acc}")

# Plot training history

 plt.plot(history.history['accuracy'], label='Train Accuracy')

 plt.plot(history.history['val_accuracy'], label='Validation Accuracy') plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

 plt.show()
```

Conclusion -