Understanding Convolutional Neural Networks (CNNs) with Keras

Name: Maryum Asif

Student ID: 23088820

GitHub: <https://github.com/ma24aes/cnn_assignment_machine_learning.git>

1. Introduction

Convolutional Neural Networks (CNNs) are a type of deep learning model mainly used for working with image data. Unlike traditional methods that require manual feature extraction, CNNs automatically learn patterns and features directly from the raw image. They are widely used in applications such as facial recognition, autonomous driving, medical image analysis, and object detection.

The design of CNNs is inspired by how the human brain processes visual information. The human visual system breaks down images in parts and understands features step by step. Similarly, CNNs use layers to gradually extract more complex features from an image.

In this document, we will explain how CNNs work, using simple terms. We'll also look at an example of building a CNN using the Keras library to classify handwritten digits using the MNIST dataset. We'll cover the structure of a CNN, how the model is trained, how it's evaluated, and the strengths and weaknesses of this technology. This documentation is designed to help beginners understand not only how to implement CNNs but also how and why they work.

1. CNN Architecture Explained

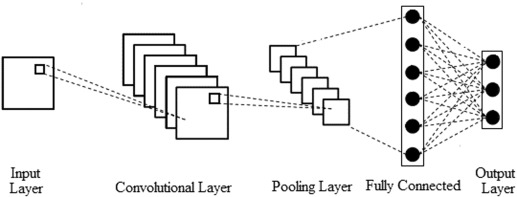
CNNs are made up of several layers. Each layer plays a role in helping the model understand the input image. The main types of layers are:

**2.1 Convolutional Layer**

This is the most important part of a CNN. It uses filters (also called kernels) that move across the image to highlight specific features like edges, corners, and textures. Each filter creates a new version of the image called a "feature map."

Each filter looks at a small region of the image at a time. This small region is called a receptive field. The filter slides over the image (this movement is called a stride), computing a dot product between the filter values and the image pixel values. This dot product is then stored in the feature map.

Using multiple filters allows the model to learn different types of features. For instance, one filter may detect vertical edges, another might detect horizontal lines, and another might detect corners.

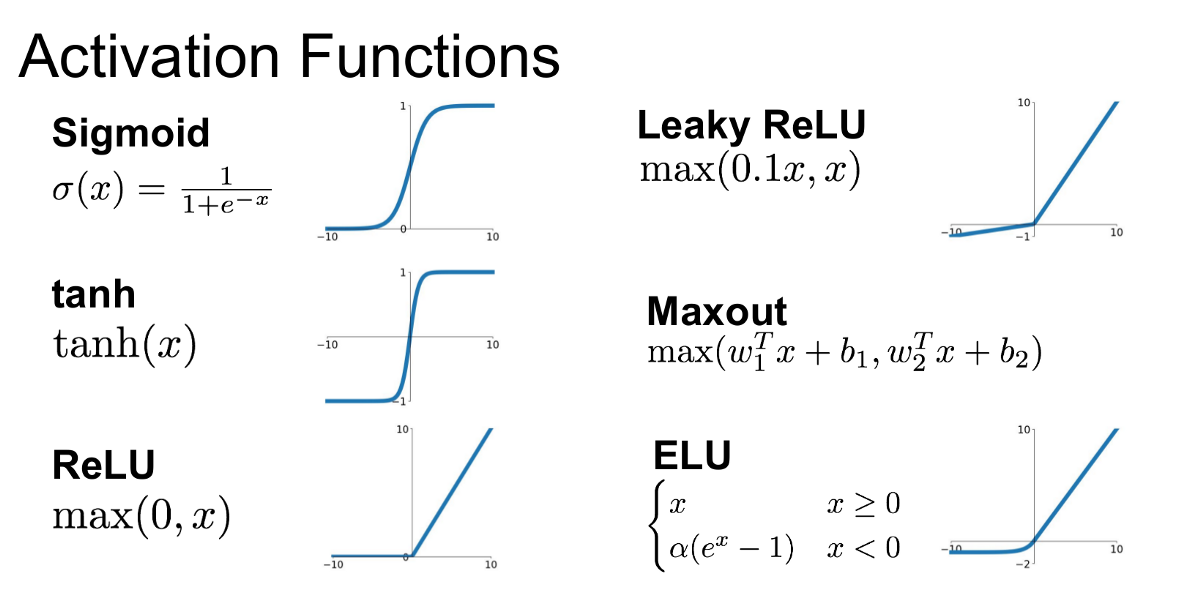


**2.2 Activation Function (ReLU)**

After convolution, we apply an activation function like ReLU (Rectified Linear Unit). This helps the model learn non-linear relationships in the data. Without this function, the entire network would behave like a simple linear function, which cannot solve complex problems.

ReLU simply changes all negative numbers in the feature map to 0, while keeping the positive ones the same. This makes it easier for the network to learn complex patterns and helps with faster training.

Other activation functions like sigmoid and tanh can also be used, but ReLU is preferred because it is faster and often performs better.

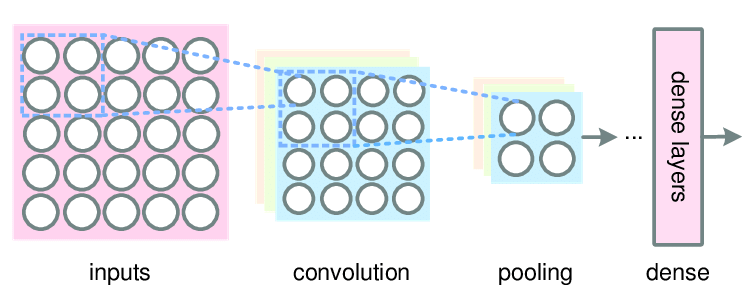


**2.3 Pooling Layer**

Pooling layers reduce the size of the feature maps. This lowers the number of calculations the model needs to make and helps prevent overfitting. It also helps make the model more robust to small changes in the input image.

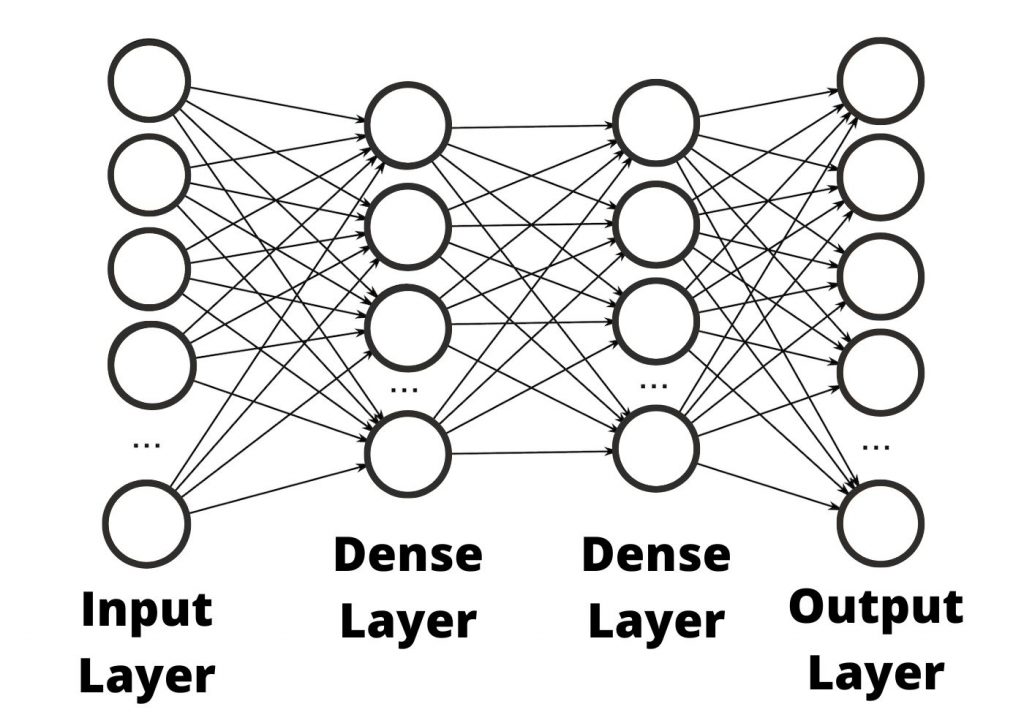
The most common pooling method is Max Pooling. It takes the maximum value in a small region (like 2x2) of the feature map, keeping the most important information and discarding the rest.

Other pooling methods include average pooling (which takes the average value) and global pooling (which compresses the entire feature map to one number).



**2.4 Fully Connected Layer (Dense Layer)**

After several convolution and pooling layers, the feature maps are flattened into a single long vector. This vector is then passed through fully connected layers, which make the final prediction. These layers are similar to traditional neural networks.



In a classification problem like digit recognition, the final dense layer will have as many neurons as there are classes. For MNIST, this means 10 neurons, one for each digit from 0 to 9. The output layer uses the softmax function to convert values to probabilities.

**2.5 Dropout and Batch Normalization**

* **Dropout** is used to prevent overfitting. It randomly turns off some neurons during training so that the model does not rely too much on any one feature. This forces the network to learn more robust features that are useful in general.
* **Batch Normalization** helps speed up training and makes the model more stable by normalizing the input to each layer. It reduces internal covariate shift and helps in faster convergence.

**2.6 Summary of CNN Workflow**

1. Start with an input image of size 28x28 pixels.
2. Apply a convolutional layer with 32 filters of size 3x3, resulting in 32 feature maps of 28x28.
3. Use a Max Pooling layer to reduce the size to 14x14x32.
4. Add another convolution and pooling layer to reduce the size further to 7x7x64.
5. Flatten the result and pass it through Dense layers to classify the image.
6. Case Study: MNIST Digit Classification with CNN using Keras
7. Dataset

**3.1 Dataset Overview**

MNIST is a well-known dataset that contains 70,000 grayscale images of handwritten digits (0-9). Each image is 28x28 pixels. The dataset is divided into 60,000 training images and 10,000 test images.

This dataset is often used to test basic image classification models. It is simple, small in size, and easy to visualize, making it ideal for beginners.

**3.2 Data Preprocessing**

Before training the model, we need to prepare the data:

# load the MNIST image dataset

(x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.\

mnist.load\_data()

train\_images = x\_train

train\_labels = y\_train

test\_images = x\_test

test\_labels = y\_test

# Normalize the images.

train\_images = (train\_images / 255) - 0.5

test\_images = (test\_images / 255) - 0.5

# Reshape the images

train\_images = np.expand\_dims(train\_images, axis=3)

test\_images = np.expand\_dims(test\_images, axis=3)

print(train\_images.shape)

print(test\_images.shape)

**3.3 CNN Model in Keras**

We build a simple CNN using the Keras Sequential API:

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

num\_filters = 8

filter\_size = 3

pool\_size = 2

model = Sequential([

Conv2D(num\_filters, filter\_size, input\_shape=(28, 28, 1)),

MaxPooling2D(pool\_size=pool\_size),

Flatten(),

Dense(10, activation='softmax'),

])

This model consists of:

* Two convolution layers
* Two pooling layers
* Two dropout layers
* One flattening layer
* Two dense layers

**3.4 Training the Model**

model.compile(

'adam',

loss='categorical\_crossentropy',

metrics=['accuracy'],

)

model.fit(

train\_images,

to\_categorical(train\_labels),

epochs=25,

validation\_data=(test\_images, to\_categorical(test\_labels)),

)

We use the Adam optimizer, which adjusts the learning rate automatically. The model is trained for 10 passes (epochs) over the data, using 128 images in each batch.

Training with a validation split allows us to monitor the model’s performance on unseen data while training, helping to detect overfitting early.

**3.5 Model Evaluation**

score = mnist\_model.evaluate(test\_images, test\_labels, verbose=0)

print("Test loss:", score[0])

print("Test accuracy:", score[1])

The model achieves more than 98% accuracy on the test data. This shows that the model is very good at recognizing handwritten digits.

**3.6 Visualization and Analysis**

Using visualization tools like Matplotlib or TensorBoard, we can:

* Plot training and validation accuracy/loss over epochs
* Display example predictions and misclassifications
* Visualize the feature maps to understand what the CNN is learning at each layer
* Generate a confusion matrix to analyze classification performance

A number in a square

AI-generated content may be incorrect.

A number with numbers on it

AI-generated content may be incorrect.

1. Model Discussion

**4.1 Model Performance**

Our CNN model does a great job with MNIST. Even though the dataset is simple, the process we used can be applied to more difficult tasks like recognizing animals, vehicles, or people in larger and coloured images.

CNNs have consistently outperformed traditional machine learning algorithms like SVMs and decision trees for image-related tasks due to their ability to extract spatial hierarchies.

A graph of a training loss

AI-generated content may be incorrect.

**4.2 Benefits of CNNs**

* Automatic Feature Learning: No need to manually define what features to look for.
* Efficient Computation: Shared weights in convolution layers reduce the number of parameters.
* Excellent for Images: CNNs are designed to understand spatial hierarchies in image data.
* Translation Invariance: Because of pooling and convolution, CNNs are less sensitive to position changes in the image.

**4.3 Limitations**

* High Resource Usage: CNNs need strong hardware like GPUs.
* Lots of Data Needed: They perform best with large datasets.
* Hard to Explain: CNNs are like black boxes—it’s not easy to know why they made a certain decision.
* Overfitting Risk: If not carefully designed, CNNs may be overfit on small datasets.

**4.4 Ethical Considerations**

* Bias: If the training data is biased, the model will also be biased.
* Privacy: CNNs are used in surveillance; we must ensure they are used responsibly.
* Transparency: In critical areas like healthcare, decisions made by CNNs should be understandable.

1. Conclusion

CNNs are a powerful tool in deep learning, especially when working with image data. In this tutorial, we explained how CNNs work in a simple way and built a real CNN using Keras to classify digits.

The skills you learned here can be used to build models for:

* Detecting traffic signs
* Finding tumours in X-ray or MRI images
* Recognizing faces or objects in security systems
* Filtering inappropriate content on social media

As technology grows, CNNs are becoming even better with innovations like ResNet and attention mechanisms. Learning CNNs is a great first step in becoming a deep learning expert.

1. References

* LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE.
* Chollet, F. (2018). Deep Learning with Python. Manning Publications.
* TensorFlow Documentation: <https://www.tensorflow.org/tutorials/images/cnn>
* Keras API Reference: <https://keras.io/api/>
* Wikipedia: Convolutional Neural Networks - <https://en.wikipedia.org/wiki/Convolutional_neural_network>
* Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
* Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556.
* Zeiler, M. D., & Fergus, R. (2014). Visualizing and Understanding Convolutional Networks. ECCV.