Revision Session

June 5, 2025

Welcome to the Exciting World of Neural Networks!

This book will guide you through the essential concepts of activation functions and non-linearity, using the popular Google Colab environment and the Python programming language. We'll avoid overwhelming you with technical jargon at the start. Instead, let's begin with a powerful visual example:

1 Finding Meaning in the Chaos - An Intuitive Introduction

Imagine you are standing in front of Pablo Picasso's famous painting, "Guernica." What do you see? A chaotic scene filled with distorted figures, fragmented shapes, and monochrome colors, right?

- Some might focus on the suffering of the people depicted.
- Others might see the powerful symbolism of the wounded horse and bull.
- Still others might be struck by the stark contrast of black and white, or the use of sharp, angular lines.

The beauty of "Guernica" (and art in general) is that everyone perceives it differently. Each of us brings our own experiences and perspectives to the viewing process. Even the artist's perspective on the painting conveys so much about war.

This act of gaining different perspectives is at the heart of **convolutional neural networks (CNNs)**, which we'll explore in this book.

Now, let's consider two key aspects about that painting:

Summary

How would you describe it to someone who cannot see it? After a moment, you might say it depicts the "sufferings of the Spanish Civil War." That is a short summarization.

Straight Lines

Life isn't about straight lines only. It's about curves and the complex shapes in "Guernica" that contribute to its emotional power. Without such shapes, the art would be boring.

In the same way, activation functions and non-linearity add complexity and meaning to the way AI learns, and the models perform.

So, how does all of this relate to activation functions and CNNs? Let's delve deeper.

2 The Three Pillars: Convolution, Pooling, and Activation

Our exploration begins with three core concepts:

1. **Convolution:**

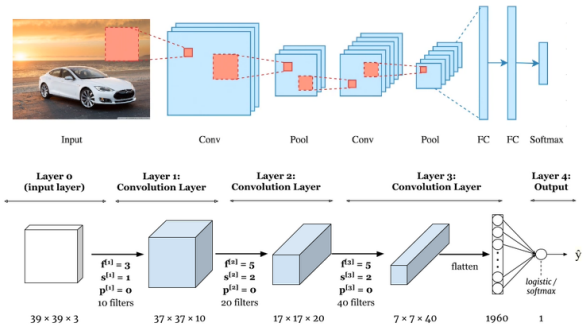

Figure 1: Layers

2. **Pooling:**
3. **Activation Function:**

2.1 Convolution: Seeing Different Perspectives

As we saw with "Guernica," gaining different perspectives is crucial. In CNNs, we achieve this through **filters** (also called **kernels**).

Imagine taking a small magnifying glass (our filter) and sliding it across the painting. At each position, the magnifying glass reveals a different aspect of the artwork, which could be edges, colors, or unique shapes, all of them contributing to what is known as *features*.

Similarly, convolutional layers use filters to extract different features from an image. These filters "convolve" (slide) across the image, performing calculations at each location to produce a **feature map**.

Filter Size: The size of the filter (e.g., 3x3) determines the area of the image it examines at each step.

Number of Filters: The number of filters determines the number of different perspectives the network learns to extract (e.g., one filter might detect edges, another might detect corners).

In essence, convolution is about building filters to extract specific perspectives from the same image.

2.2 Pooling: Summarizing the Information

After extracting features using convolution, we often want to summarize the information. This is where **pooling** comes in.

Think back to our "Guernica" example. Instead of describing every detail of the painting, we offered a one-line summary: "It shows the suffering of the war."

Pooling layers do something similar. They reduce the spatial size of the feature maps, retaining only the most important information.

Max Pooling: Selects the maximum value from each region. This helps highlight the strongest features.

Average Pooling: Calculates the average value from each region.

Pooling reduces the dimensionality of the data, making the network more efficient and robust to variations in the input.

Pooling layers give a single summary for each block, in relation to the original image.

2.3 Activation Functions: Introducing Non-Linearity

Now, let's talk about **activation functions**. These functions introduce non-linearity into the network. Without non-linearity, the network would only be able to learn linear relationships.

Activation functions add complexities and make each AI model unique, as we saw from the painting.

Some popular activation functions are:

- **ReLU (Rectified Linear Unit):** Returns the input if it's positive, otherwise returns zero.

$$f(x) = \max(0, x)$$

ReLU is commonly used in hidden layers of CNNs due to its simplicity and effectiveness in avoiding the "vanishing gradient" problem (which we'll discuss shortly).

- **Sigmoid:** Squashes the input to a range between 0 and 1.

$$f(x) = \frac{1}{1 + e^{-x}}$$

- **Tanh (Hyperbolic Tangent):** Squashes the input to a range between -1 and 1.

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

- **Softmax:** Converts a vector of numbers into a probability distribution. Often used in the output layer for multi-class classification problems.

$$\text{softmax}(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$

where K is the number of classes.

Vanishing Gradient Problem: The reason why we usually prefer ReLU over the rest is that ReLU does not have the issue of vanishing gradients. *Vanishing gradients happen when you keep updating your weights and then they get multiplied by small weight values, so they eventually stop contributing.*

3 Building a CNN with Keras - A Practical Example

Now, let's put these concepts into practice using the Keras library in Google Colab. We'll use the Fashion MNIST dataset, which contains grayscale images of clothing items.

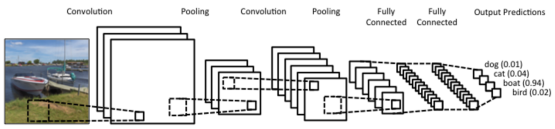

Figure 2: CNN

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
    Activation
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.utils import to_categorical

# 1. Load and Preprocess the Data
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data
    ()

# Reshape and normalize images
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255

```

```
# Convert labels to categorical
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# 2. Build the Model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')  # 10 classes for Fashion MNIST
])

# 3. Compile the Model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# 4. Train the Model
model.fit(train_images, train_labels, epochs=5, batch_size=64)

# 5. Evaluate the Model
loss, accuracy = model.evaluate(test_images, test_labels, verbose=0)
print('Test Accuracy: %.3f' % accuracy)
```

Explanation:

- **Sequential Model:** Creates a linear stack of layers.
- **Conv2D:** Adds a convolutional layer with 32 filters, a 3x3 kernel size, ReLU activation, and an input shape of (28, 28, 1) for the first layer.
- **MaxPooling2D:** Adds a max pooling layer with a 2x2 pool size.
- **Flatten:** Converts the 2D feature maps into a 1D vector.
- **Dense:** Adds a fully connected (dense) layer with 64 units and ReLU activation.
- **Output Layer:** Adds a dense layer with 10 units (one for each clothing class) and softmax activation.
- **Compile:** Configures the model for training, specifying the optimizer, loss function, and metrics.
- **Fit:** Trains the model using the training data.
- **Evaluate:** Evaluates the model on the test data.

A general "rule of thumb" when building Neural Networks is to gradually increase the number of filters, that is, in each layer.

4 Experimentation

This code provides a starting point. Experiment with different numbers of filters, kernel sizes, activation functions, and layer architectures to see how they affect the model's performance. Try to increase that percentage of efficiency by the end of the course.

5 Conclusion

By understanding the principles of convolution, pooling, and activation functions, you can build powerful CNNs for image recognition and other tasks. This is your door to the fascinating landscape of neural networks.