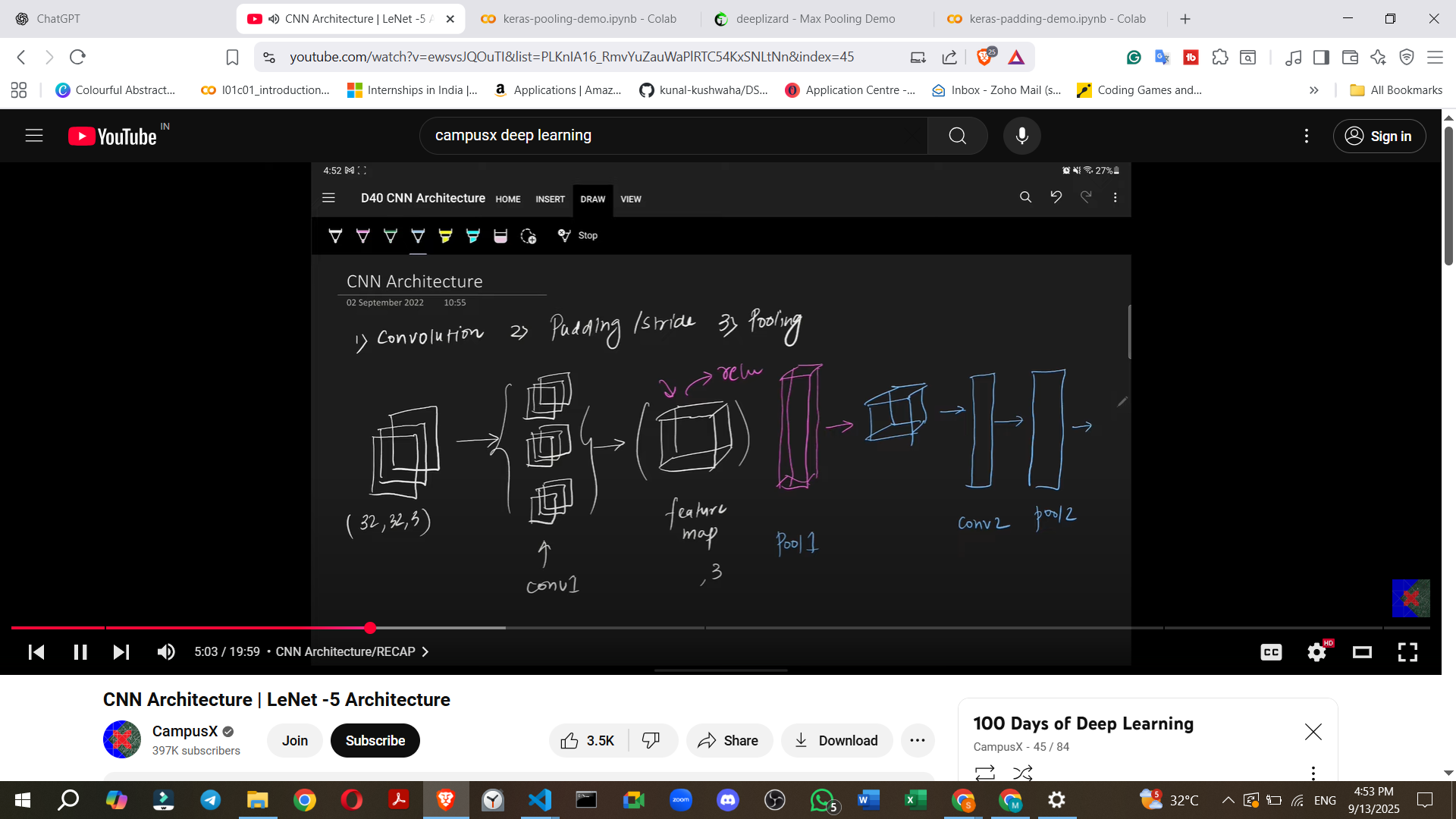
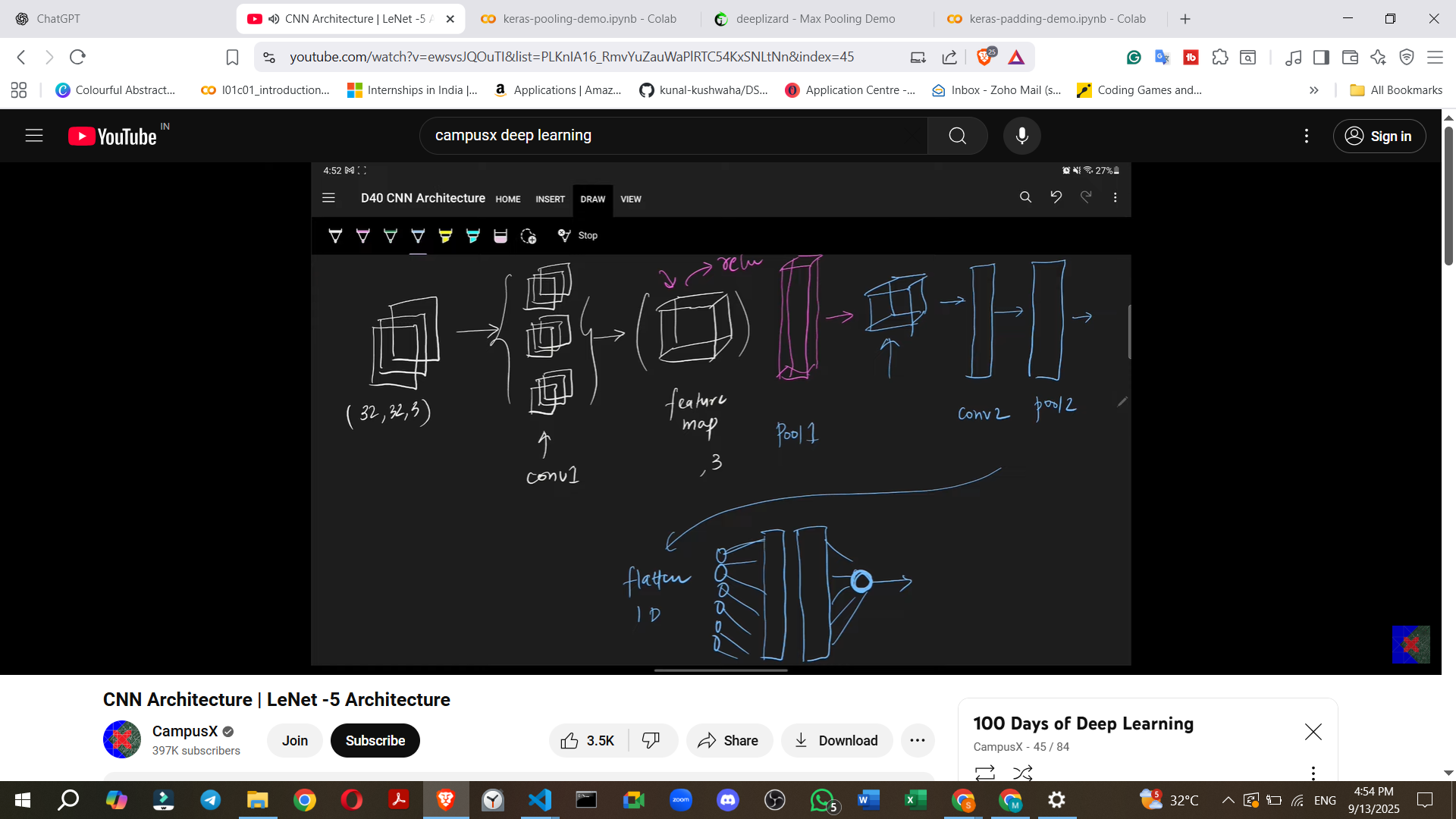
### **CNN Architecture & LeNet-5**

This document provides a detailed overview of Convolutional Neural Network (CNN) architecture, focusing on its fundamental components and a foundational model, LeNet-5, as discussed in the provided sources.

#### **1. Introduction to CNN Architecture**

* **What is a CNN?**
  + A Convolutional Neural Network (CNN) is a type of neural network used primarily for image processing and classification tasks.
  + It is built upon a combination of three core ideas: **convolution operation**, **padding and stride**, and **pooling**.
  + Understanding how to integrate these concepts is crucial for creating a CNN architecture.

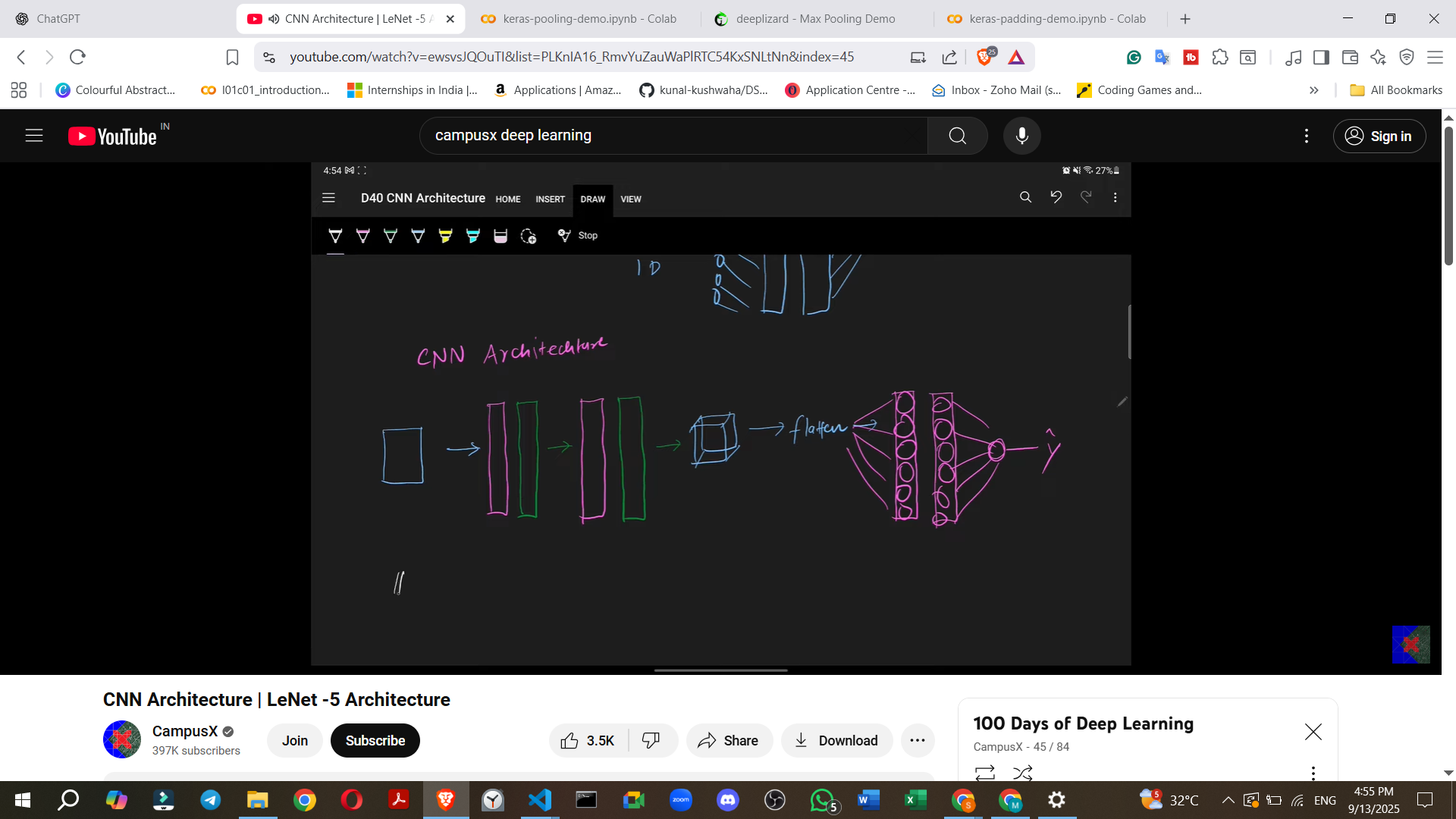


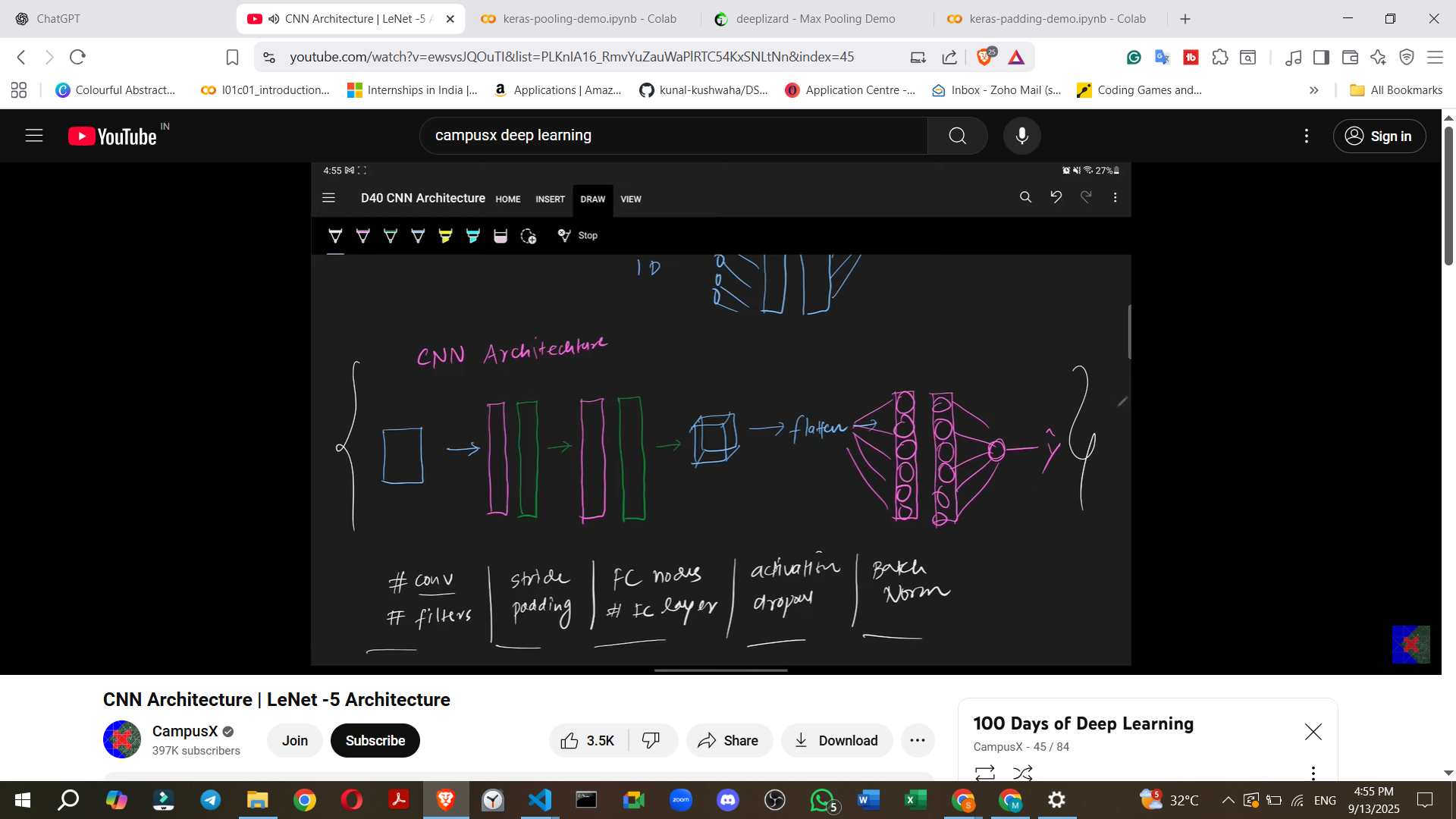


#### **2. Basic CNN Architecture: A Sequential Overview**

How does a typical, basic CNN look, layer by layer?

1. **Input Layer**:  
   * **Purpose**: Receives the raw image data.
   * **Example**: Often an RGB image, e.g., **32x32 pixels with 3 channels** (for Red, Green, Blue).
2. **Convolutional Layer (Conv Layer)**:  
   * **Purpose**: Applies filters (also called kernels) to the input to extract features.
   * **Process**: Filters slide across the input, performing convolution operations.
   * **Output**: Generates a **feature map** (a volume of numbers). The number of feature maps depends on the number of filters used.
   * **Key Question**: How many filters should be used in a convolutional layer?
3. **Non-linear Activation Function**:  
   * **Purpose**: Introduces non-linearity into the model, allowing it to learn more complex patterns.
   * **Placement**: Typically applied immediately after a convolutional layer.
   * **Example**: **ReLU (Rectified Linear Unit)** is a common choice in modern CNNs.
4. **Pooling Layer**:  
   * **Purpose**: Reduces the spatial dimensions (width and height) of the feature maps, thus reducing the number of parameters and computational cost, and providing some invariance to small shifts and distortions.
   * **Placement**: Usually follows a convolutional layer and activation function.
   * **Example**: **Max Pooling** or **Average Pooling**.
   * **Flexibility**: You can apply **multiple combinations of convolutional and pooling layers** to progressively extract higher-level features.
5. **Flattening Layer**:  
   * **Purpose**: Converts the multi-dimensional output from the convolutional/pooling layers (a 3D tensor) into a **one-dimensional vector**.
   * **Necessity**: This prepares the data to be fed into fully connected layers.
6. **Fully Connected Layers (FC Layers)**:  
   * **Purpose**: These are standard neural network layers where every neuron in one layer is connected to every neuron in the next layer. They perform high-level reasoning based on the features extracted by earlier layers.
   * **Placement**: Applied after the flattening layer.
   * **Flexibility**: You can use **multiple fully connected layers** depending on the problem.
   * **Key Question**: How many nodes should each fully connected layer have?
7. **Output Layer**:  
   * **Purpose**: Produces the final prediction of the network.
   * **Structure**:
     + For **binary classification**, typically a single node with a sigmoid activation function.
     + For **multi-class classification**, multiple nodes (one for each class) often with a **Softmax** activation function.
   * **Example**: For digit recognition (0-9), there would be 10 output nodes.

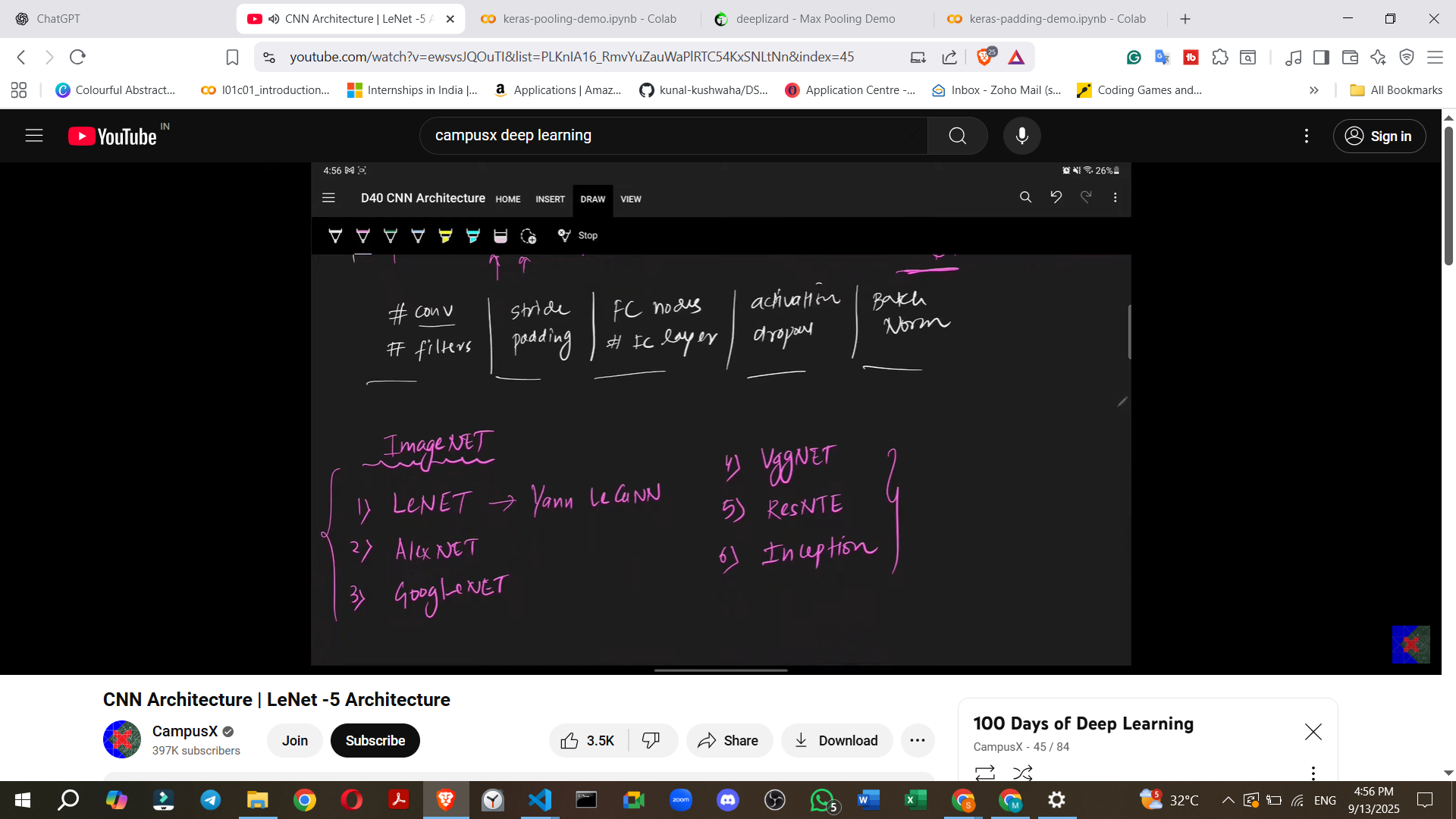




#### **3. Variations in CNN Architectures**

Different CNN architectures are created by varying several parameters of this basic structure:

* **Number of Convolutional Layers**: How many Conv + Pooling blocks are used?
* **Number of Filters**: The count of filters in each convolutional layer.
* **Stride Value**: The step size of the filter across the input.
* **Padding**: Whether padding is used (to preserve spatial dimensions) or not.
* **Nodes in Fully Connected Layers**: The number of neurons in the FC layers.
* **Number of Fully Connected Layers**: How many FC layers are present.
* **Activation Function**: The choice of non-linear function (e.g., ReLU, Tanh).
* **Regularization Techniques**: Use of techniques like **Dropout** or **Batch Normalization**.
* **Examples of other famous CNN architectures** (to be studied further): AlexNet, GoogLeNet, VGGNet, ResNet, Inception Module.



#### **4. LeNet-5 Architecture: A Foundational Example**

**LeNet-5** is a crucial historical CNN architecture, often considered the first.

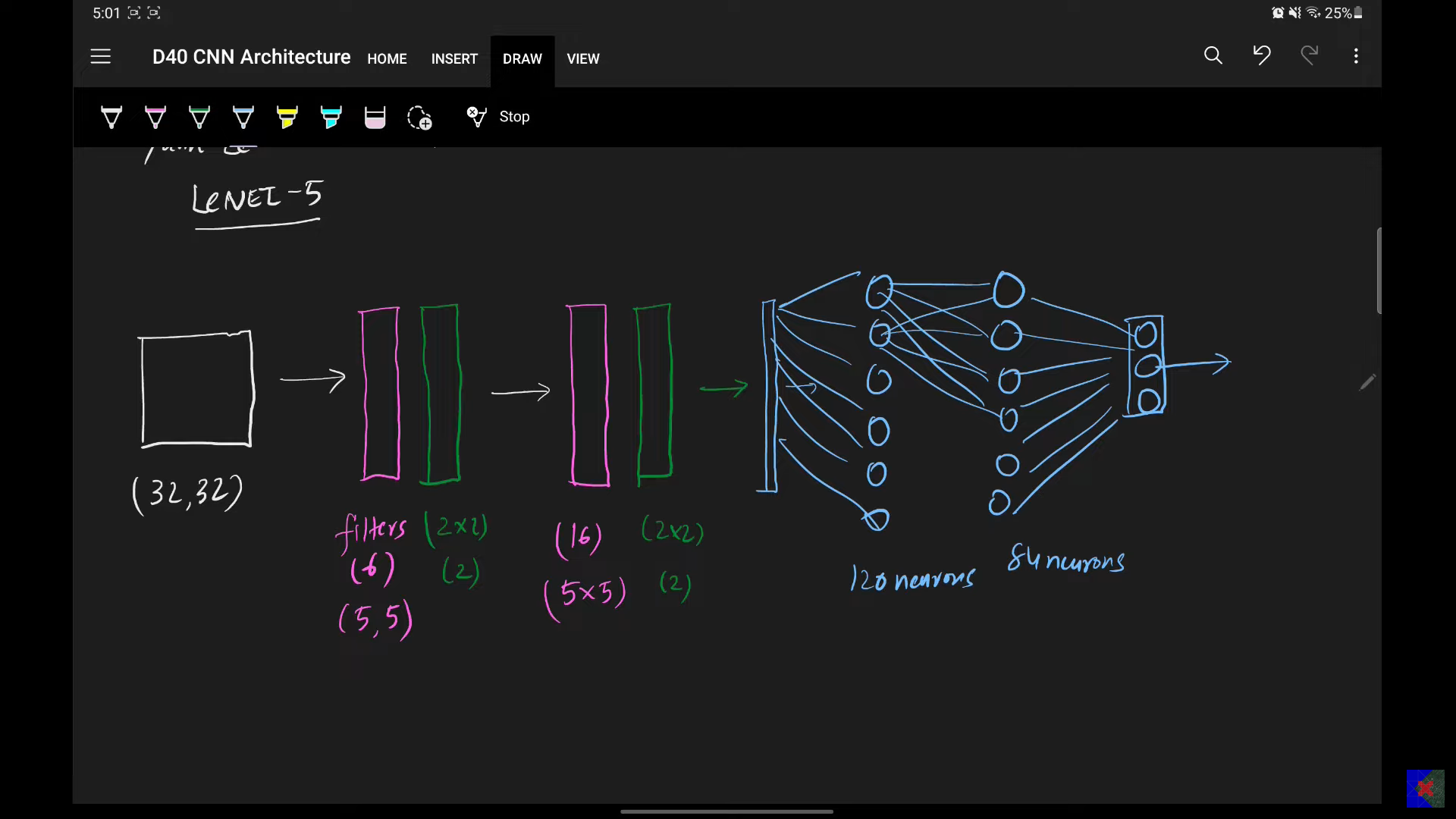
* **Creator**: Developed by **Yann LeCun** (regarded as the "father of CNN") in **1998**.
* **Purpose**: Used for a project with the **US Navy's postal service to recognise machine-written PIN codes** (digit recognition).
* **Name Origin**: It is called LeNet-5 because it comprises **five layers** that are trainable (excluding the input layer).
* **Key Observation**: In CNNs, as you go deeper into the network, the number of filters generally tends to increase.

Let's break down the LeNet-5 architecture sequentially:

**Input to LeNet-5:**

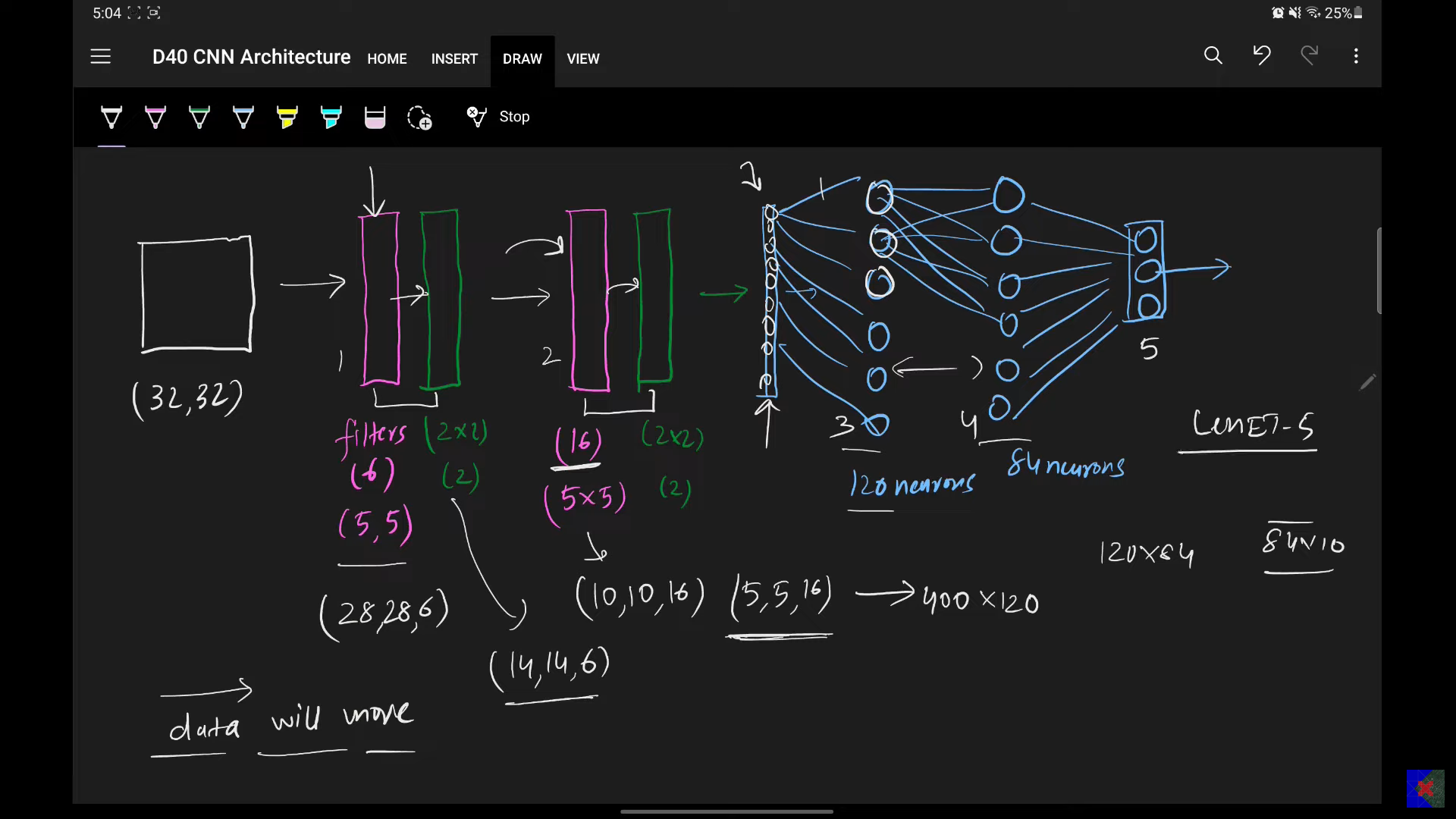
* **Input Image**: LeNet-5 expects a **32x32 pixel grayscale image**.

**Layer-by-Layer Architecture:**

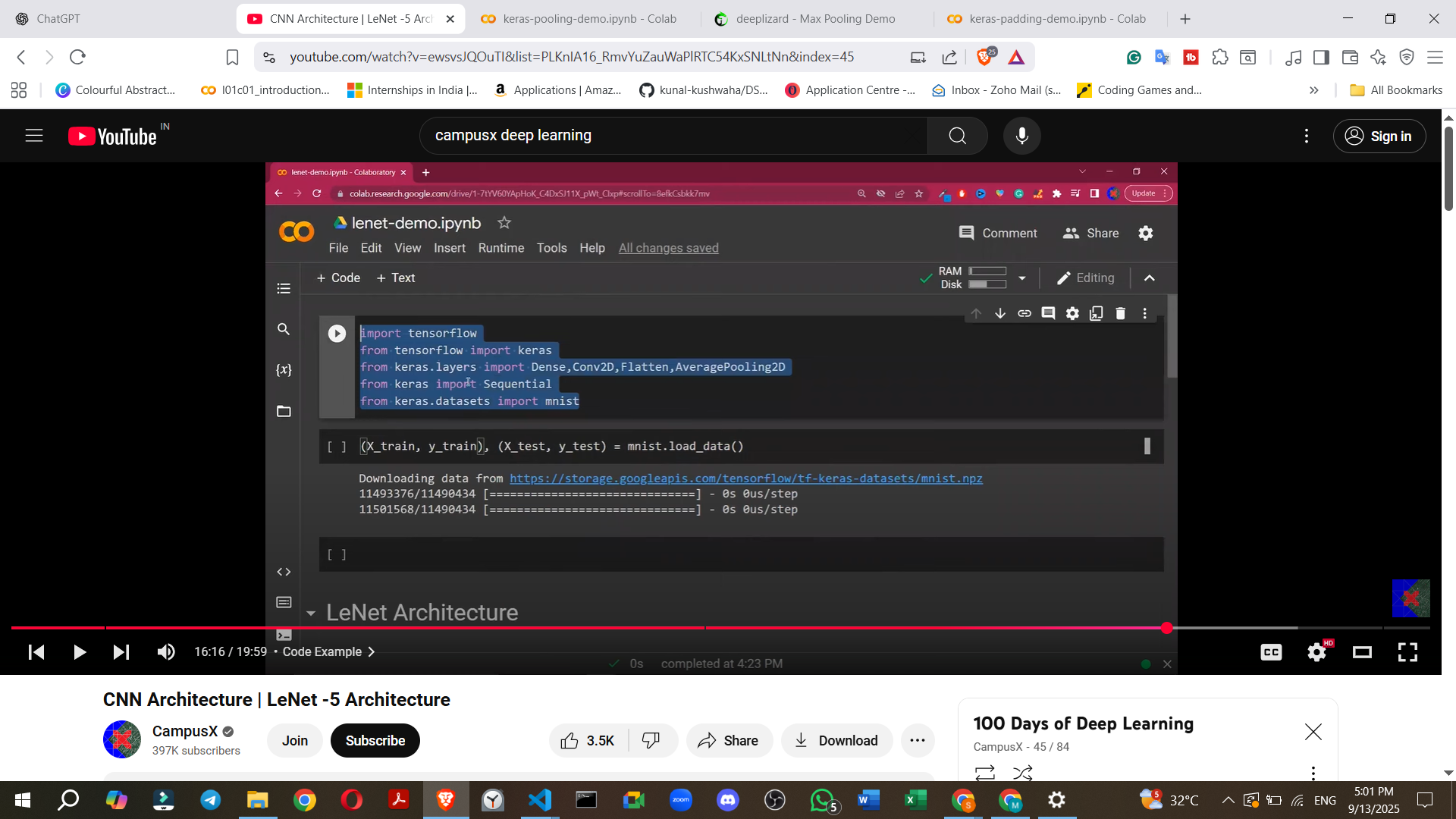
****

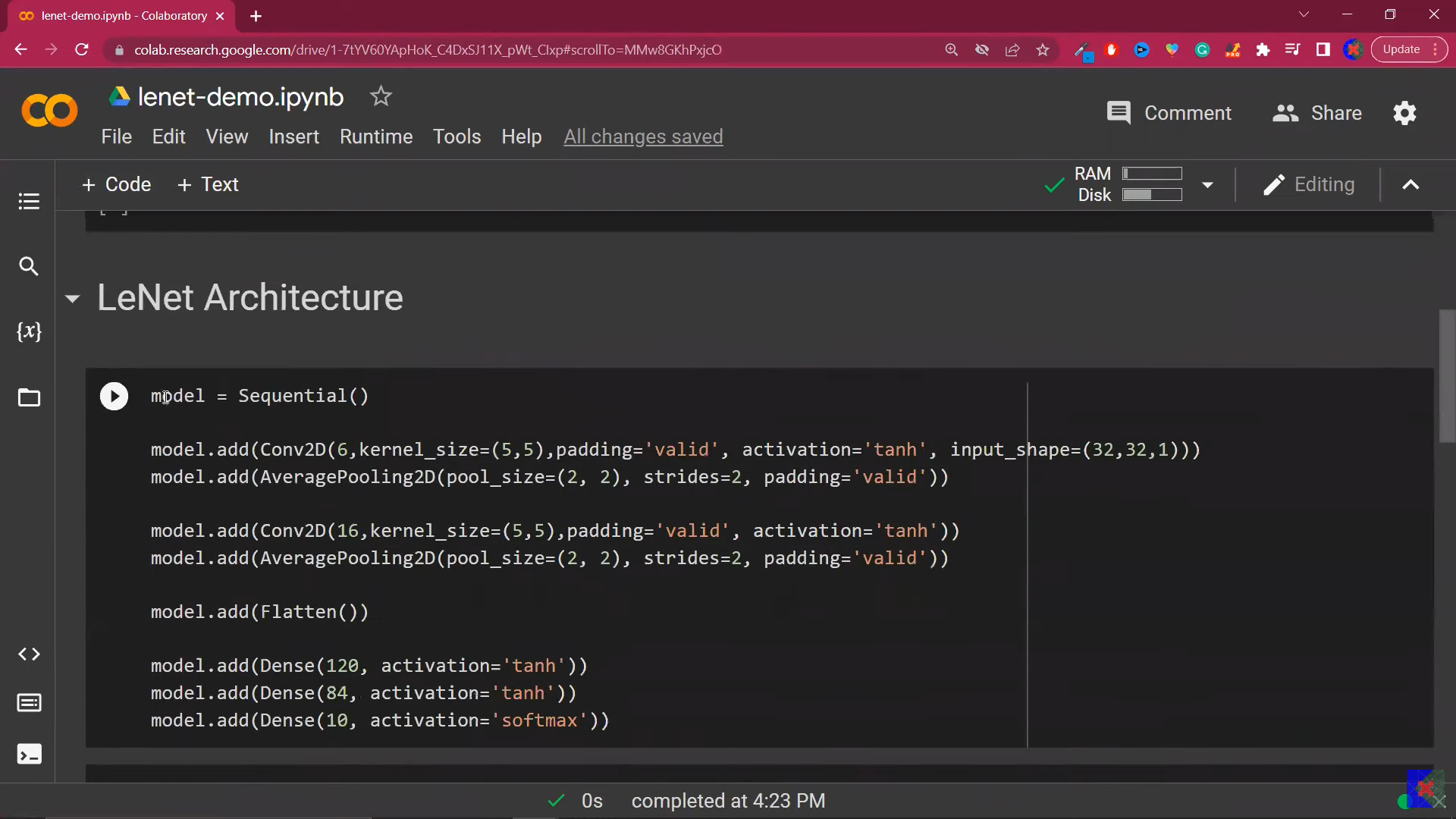
1. **Layer C1 (Convolutional Layer)**:  
   * **Filters**: 6 filters.
   * **Kernel Size**: 5x5.
   * **Stride**: 1.
   * **Padding**: None.
   * **Activation Function**: **Tanh** (tangent hyperbolic).
   * **Output Tensor Size**: 28x28x6 (derived from 32x32 input, 5x5 filters, stride 1, no padding).
2. **Layer S2 (Average Pooling Layer)**:  
   * **Type**: **Average Pooling** (not Max Pooling, which is more common today).
   * **Pool Size**: 2x2.
   * **Stride**: 2.
   * **Output Tensor Size**: 14x14x6 (28x28 input with 2x2 pooling, stride 2).
3. **Layer C3 (Convolutional Layer)**:  
   * **Filters**: 16 filters.
   * **Kernel Size**: 5x5.
   * **Stride**: 1.
   * **Padding**: None.
   * **Activation Function**: **Tanh**.
   * **Output Tensor Size**: 10x10x16 (14x14 input with 5x5 filters, stride 1, no padding).
4. **Layer S4 (Average Pooling Layer)**:  
   * **Type**: **Average Pooling**.
   * **Pool Size**: 2x2.
   * **Stride**: 2.
   * **Output Tensor Size**: 5x5x16 (10x10 input with 2x2 pooling, stride 2).
5. **Flattening Layer**:  
   * **Purpose**: Converts the 5x5x16 tensor into a 1D vector.
   * **Output Size**: 400 (5 \* 5 \* 16).
6. **Layer F5 (Fully Connected Layer)**:  
   * **Input**: 400 neurons.
   * **Output (Neurons)**: 120 neurons.
   * **Activation Function**: **Tanh**.
7. **Layer F6 (Fully Connected Layer)**:  
   * **Input**: 120 neurons.
   * **Output (Neurons)**: 84 neurons.
   * **Activation Function**: **Tanh**.
8. **Output Layer**:  
   * **Input**: 84 neurons.
   * **Output (Neurons)**: 10 neurons (representing digits 0-9).
   * **Activation Function**: **Softmax** (for multi-class classification).

* **Total Trainable Parameters**: LeNet-5 has approximately **60,000 trainable parameters**.
* **Why Tanh and not ReLU?**: In 1998, when LeCun developed LeNet-5, **Tanh was considered the most effective activation function**. ReLU, now widely used, was not the standard at that time.



#### **5. Implementing LeNet-5 in Keras**







* The LeNet-5 architecture can be implemented using the **Keras** deep learning library.
* The implementation involves defining a sequential model and adding layers such as Conv2D, AveragePooling2D, Flatten, and Dense.
* Specific parameters like filters, kernel\_size, pool\_size, strides, and activation functions are set for each layer according to the LeNet-5 specification.
* The model can then be compiled and trained on a dataset.

#### **6. Future Learning (What's Next?)**

* The difference between CNNs and Artificial Neural Networks (ANNs) will be discussed.
* Creating custom CNN architectures for projects will be explored.
* Further discussions will cover more advanced CNN architectures like AlexNet, ResNet, VGGNet, GoogLeNet, and Inception Modules.