# **What is Transfer Learning? Transfer Learning in Keras | Fine Tuning Vs Feature Extraction**

## 

**1. Introduction to Deep Learning and its Challenges**

* Deep Learning models are **data-hungry**, requiring a large amount of labelled data for training (e.g., 10,000 images).
* Collecting such data can involve scraping images, but **labelling each image is difficult, requires manual labour, and is costly** for companies.
* Training Deep Learning models, especially on large datasets, **takes a significant amount of time**.
* Due to these problems (data requirements and training time), people generally do not prefer to train their own Deep Learning models from scratch.

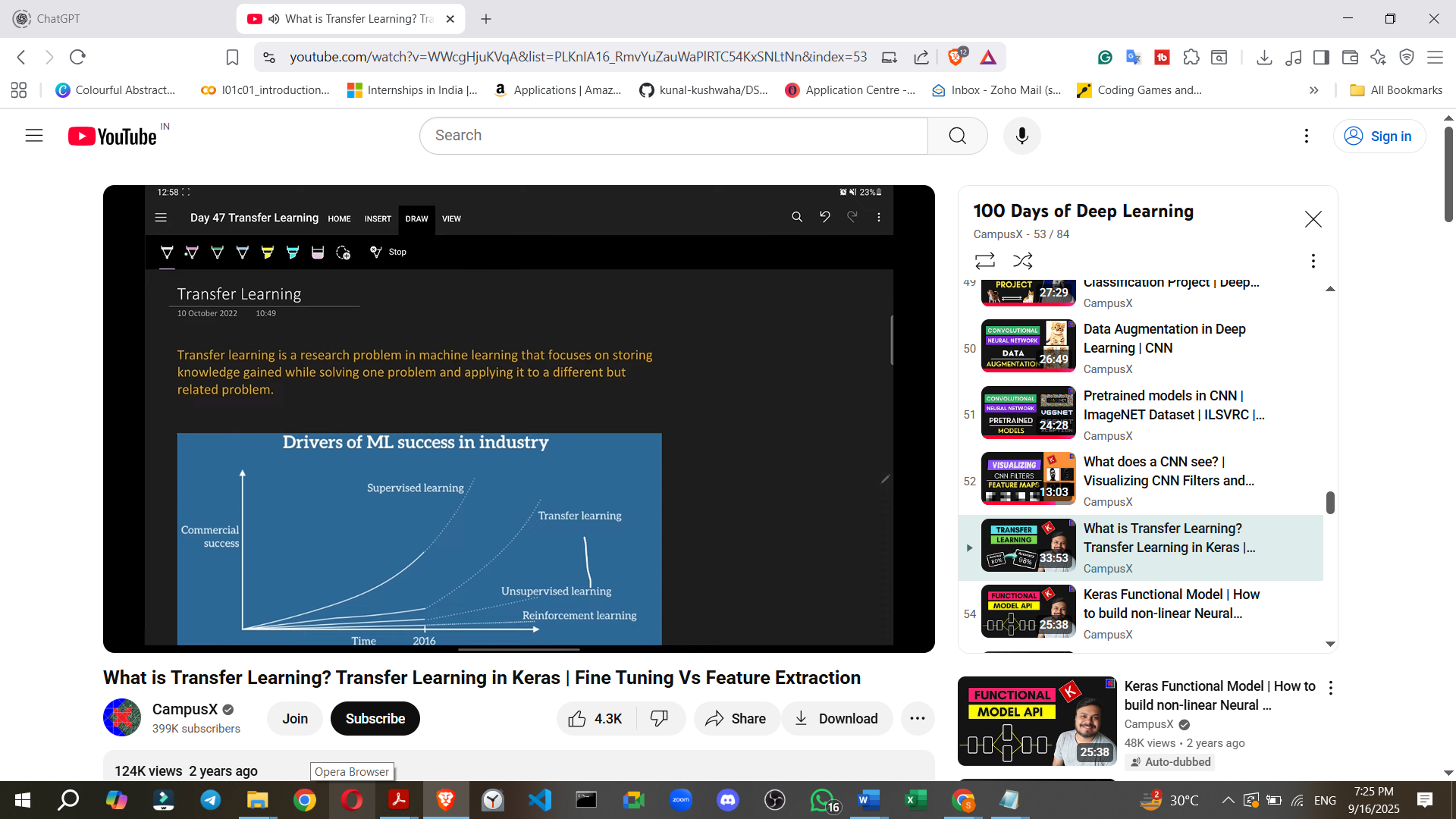
**2. Pre-trained Models as a Solution**

* The **solution to the challenges of training custom models is using pre-trained models**.
* The source refers to a previous video that discussed ImageNet, a very large dataset of daily objects and animals, containing around **1.4 million images across 1000 categories** (e.g., different dog breeds).
* An annual competition called ILSVRC (ImageNet Large Scale Visual Recognition Challenge) was organised on the ImageNet dataset.
* Models like VGG, ResNet, and InceptionNet emerged from this competition, demonstrating high performance after being trained on ImageNet.
* **Pre-trained models are CNN models trained on a different, large dataset** (like ImageNet). They have "learned" from this previous large dataset and can be used in other projects.

**3. The Problem with Pre-trained Models and the Emergence of Transfer Learning**

* A potential problem with pre-trained models is that the specific classes required for a new project might **not be present in the 1000 classes** the model was originally trained on.
* For example, if a project requires classifying images as "phone" or "tablet," and these specific classes are not among ImageNet's 1000 classes, directly using the pre-trained model might not work.
* **Transfer Learning is introduced as the solution to this problem**.

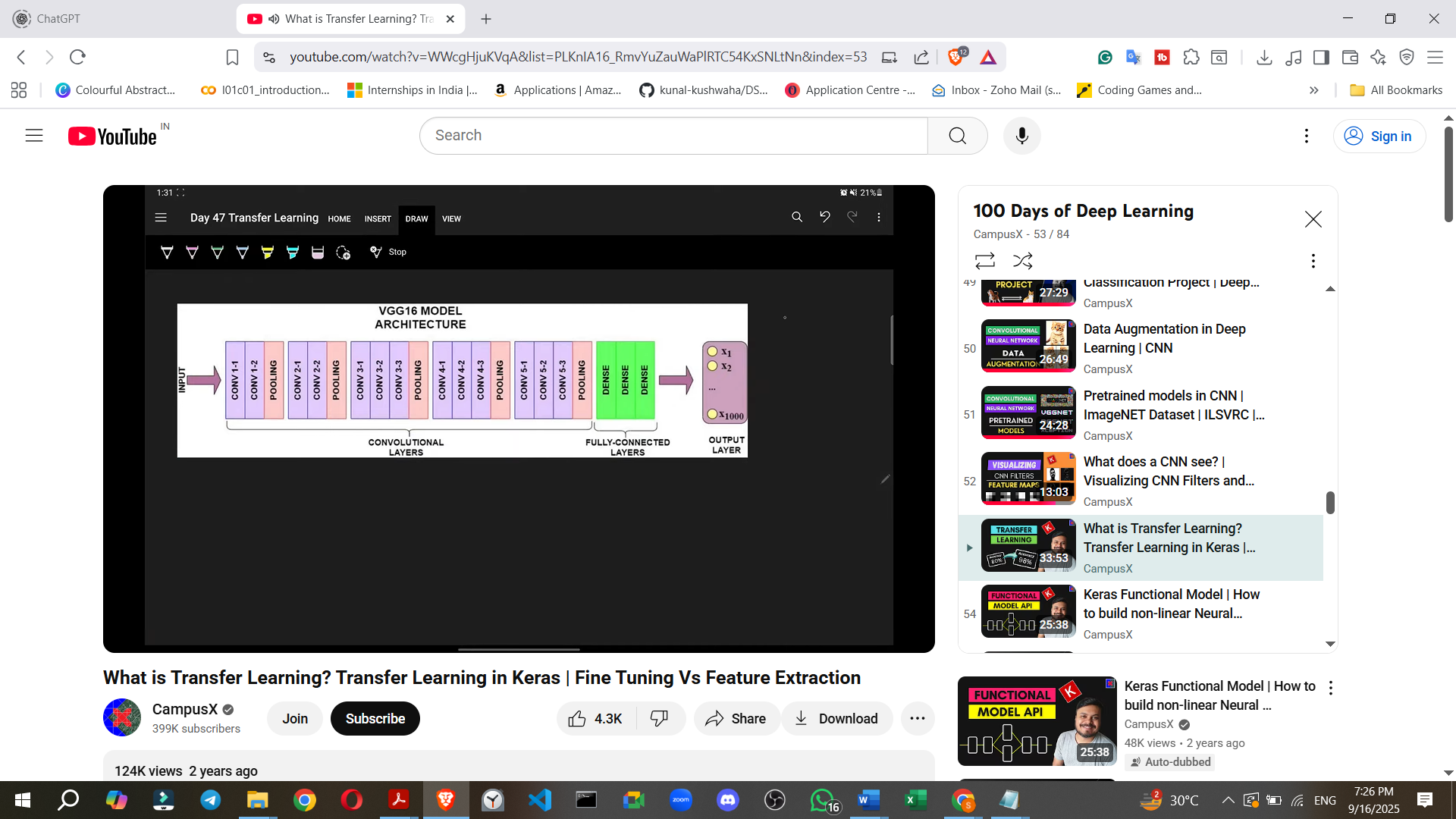
**4. What is Transfer Learning?**

****

* **Transfer Learning is a machine learning technique that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem**.
* It leverages the benefits of pre-trained models by using a CNN trained on another dataset (which saves cost and time on data collection and training).
* Transfer Learning has become a significant area in machine learning since 2015-2016.
* Andrew Ng stated that, besides supervised learning, **Transfer Learning will be the next big thing to advance machine learning in the industry**, even surpassing unsupervised and reinforcement learning in potential impact.
* Transfer Learning is inspired by real-life situations:
  + Learning to ride a bicycle makes learning to ride a motorcycle easier because the brain transfers balancing knowledge.
  + Knowing how to play a musical instrument like the violin makes learning a related instrument like the guitar easier due to an understanding of musical notes.
* In essence, Transfer Learning is the computer science integration of this idea: **applying knowledge from one domain to another**.

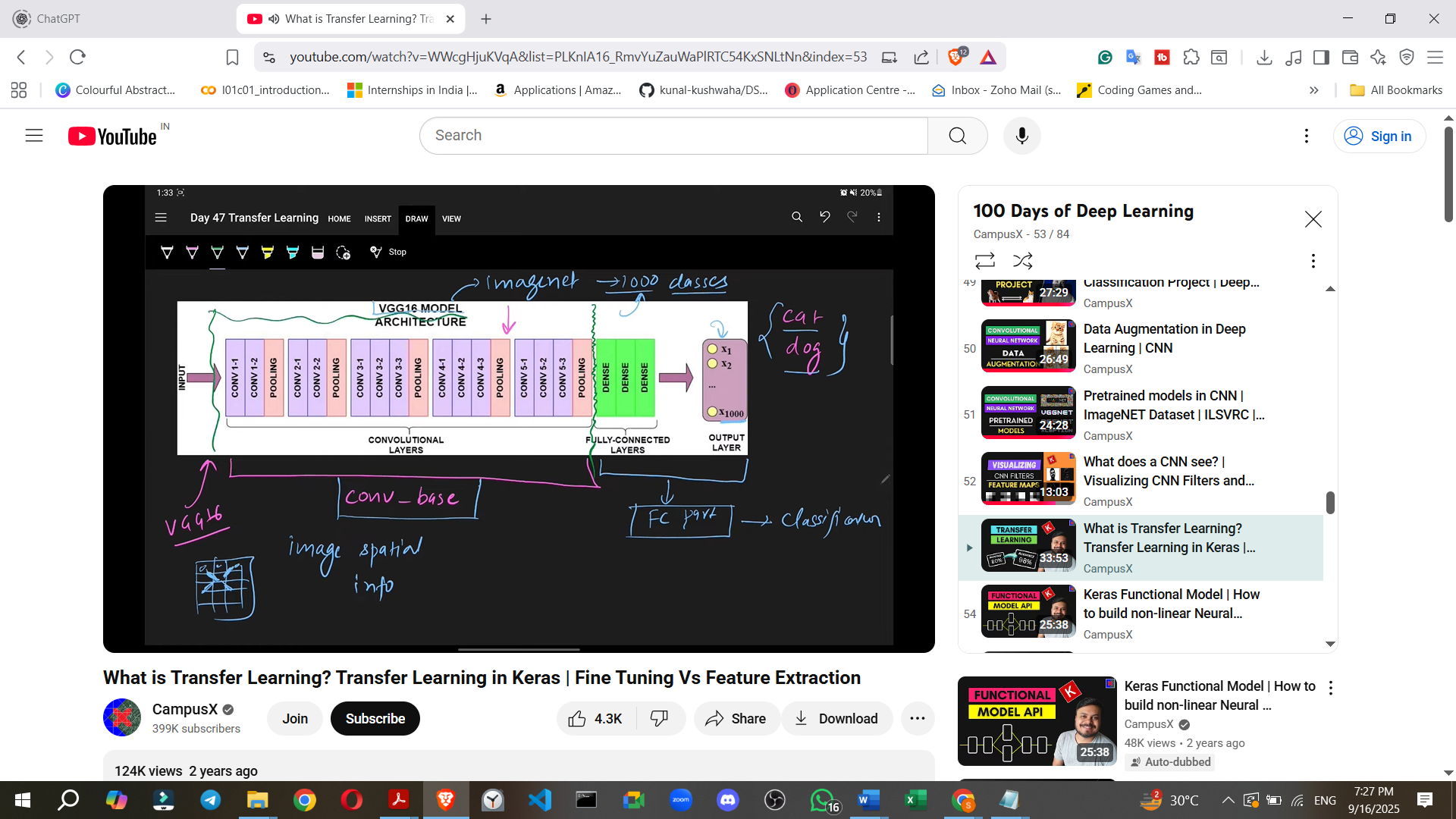
**5. How Transfer Learning Works (Under the Hood)**

* Let's consider an example of cat-dog classification using the pre-trained VGG16 model.

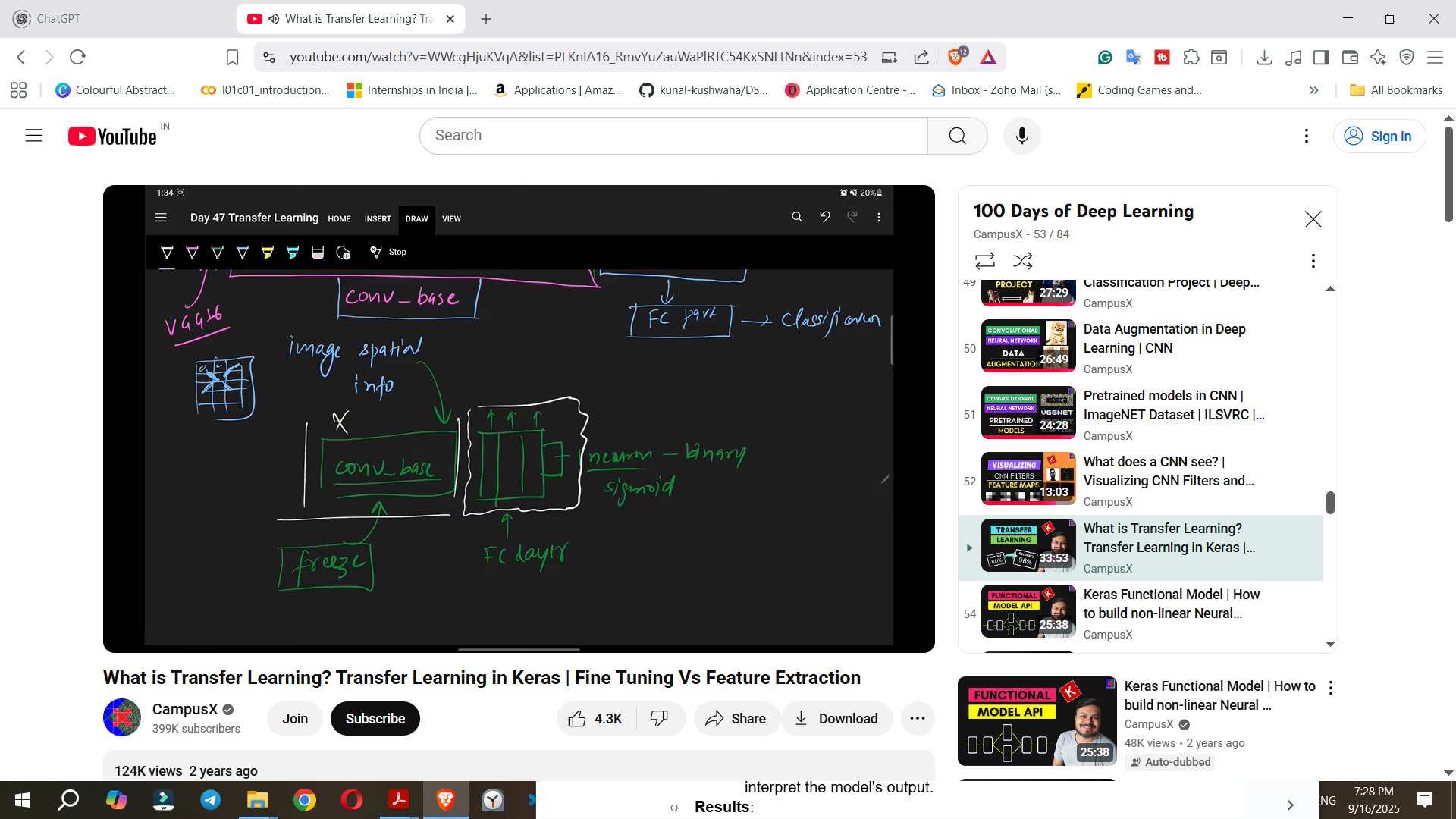


* A VGG16 model typically has **two main parts**:
  1. The **Convolutional Base (or Convolutional Layers)**: Responsible for extracting spatial information and features from an image (e.g., relationships between pixels).
  2. The **Fully Connected (FC) Layers (or Dense Layers)**: Responsible for classification based on the features extracted by the convolutional base.

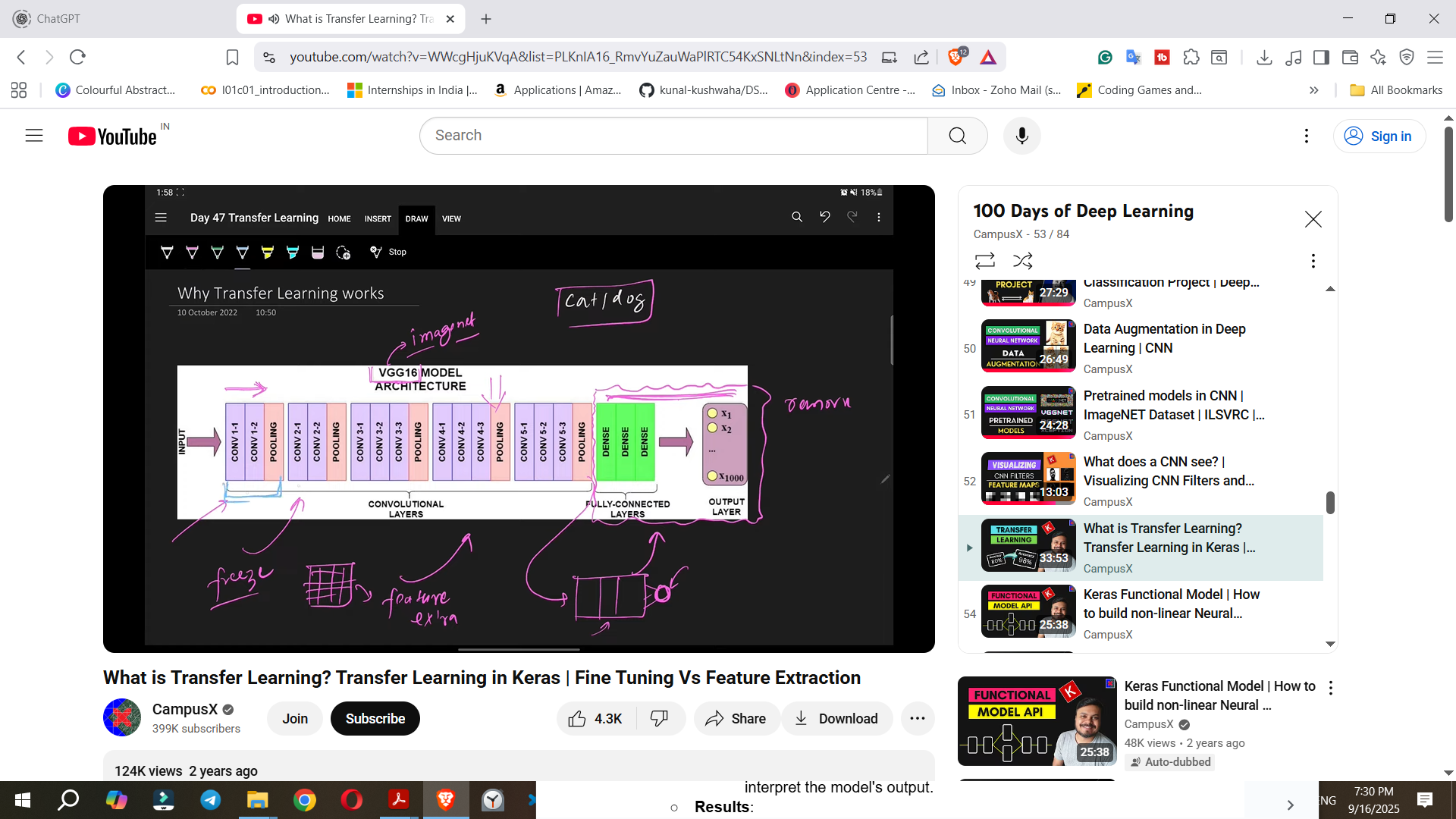




* VGG16 is trained on ImageNet for 1000 classes, so its FC layers are designed for multi-class classification.
* To apply Transfer Learning for a new task (e.g., binary cat-dog classification, where cat and dog might not be specific output classes in the VGG16's 1000 original classes):
  1. **Break the model**: The FC layers of the pre-trained model are removed.
  2. **Keep the Convolutional Base**: This part is retained.
  3. **Add Custom Dense/FC Layers**: New dense layers are added, tailored to the specific problem (e.g., one output neuron with sigmoid for binary classification).
  4. **Freeze the Convolutional Base**: The weights of the convolutional base are frozen, meaning they are not updated during the training process on the new dataset. This preserves the learned feature extraction knowledge.



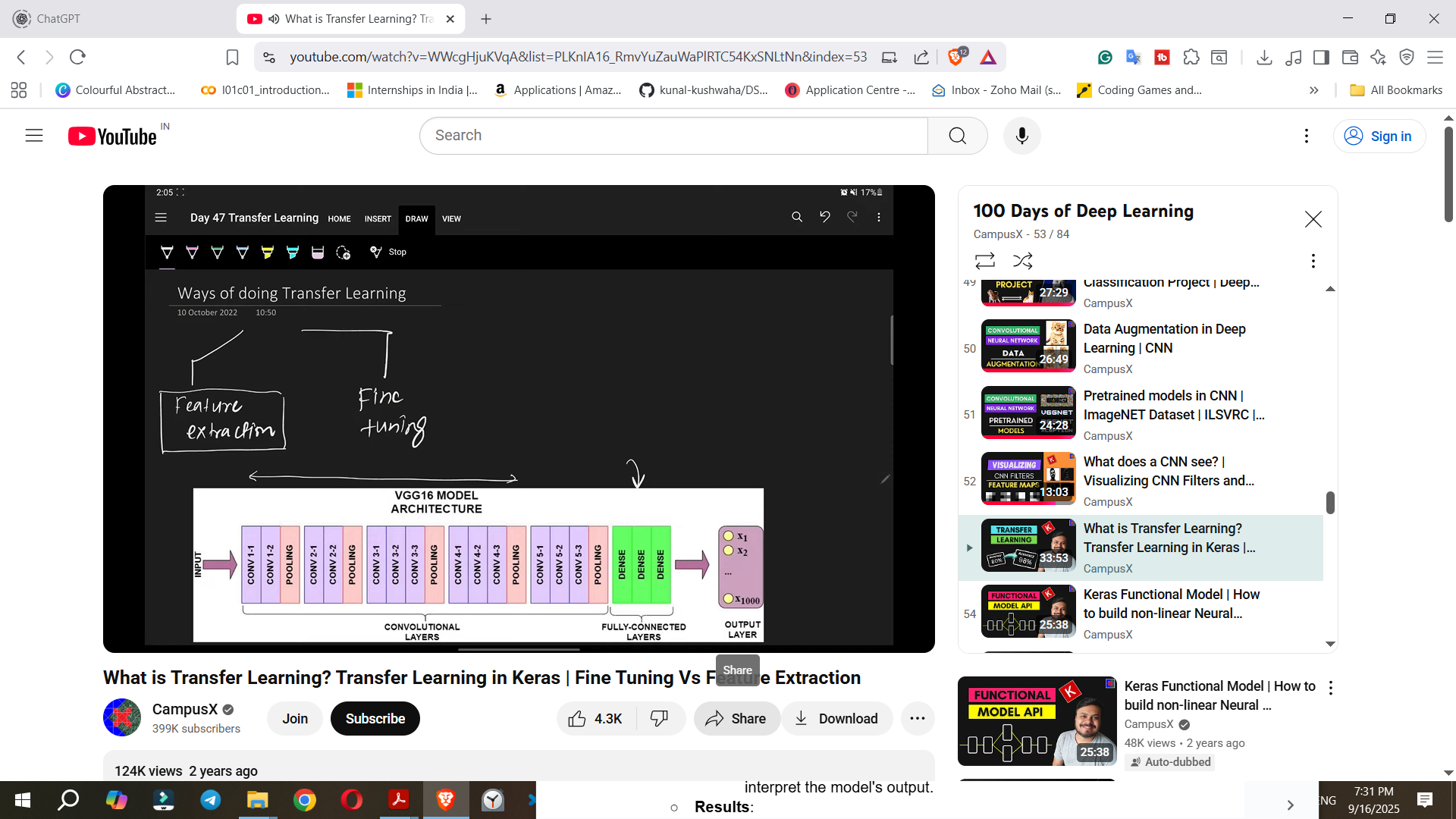
* 1. **Train the New Model**: The model is then trained on the new, smaller dataset, only updating the weights of the newly added FC layers.

****

**6. Why Transfer Learning Works**

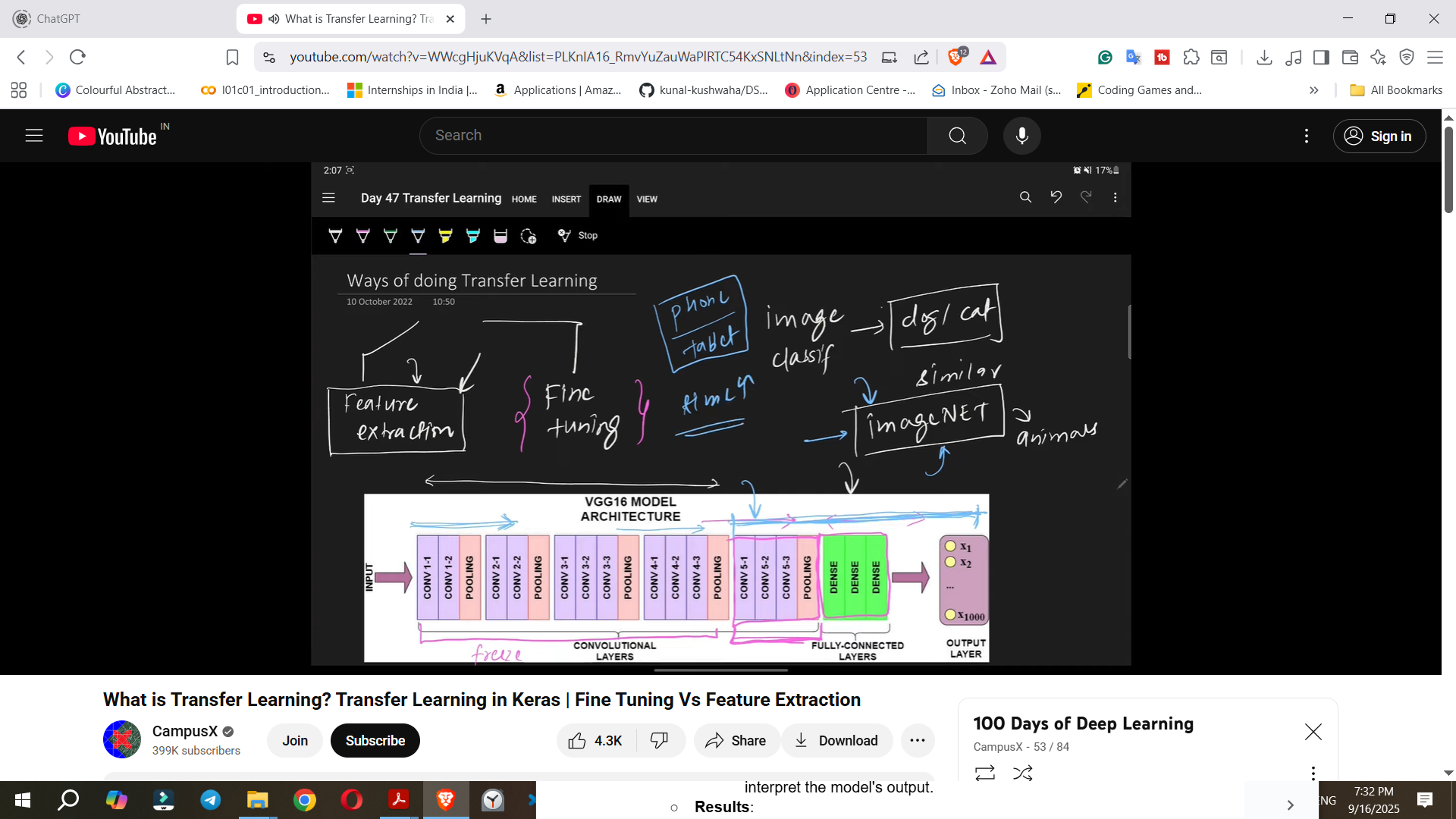
* The **early layers of a CNN extract primitive, general features** like edges and basic patterns.
* As the layers progress, they extract **more complex patterns and features** like shapes.
* The ImageNet dataset (1000 classes) covers a wide range of real-world objects, and their primitive features are generally similar or common.
* Therefore, the **knowledge of extracting these general, primitive features from the convolutional base does not need to be re-learned**.
* The philosophy is **"Do not reinvent the wheel"**: the wheel (primitive feature extraction) has already been built, so use it to build a car (specific classification).
* The convolutional base has already learned these primitive and general features during its initial training, so that work does not need to be repeated.

**7. Approaches to Transfer Learning**

****

There are two main approaches to Transfer Learning:

* **a) Feature Extraction**
  + **Definition**: This is exactly what was described above: **replacing the last dense layers with custom ones, freezing the convolutional base, and training the model on the new data**.
  + **When to Use**: It is generally applied when the labels of the new image classification task are **similar to the data on which the model was originally trained**.
  + **Example**: For dog/cat classification, feature extraction is suitable because ImageNet already contains training on animals and their features.
* **b) Fine-Tuning**

****

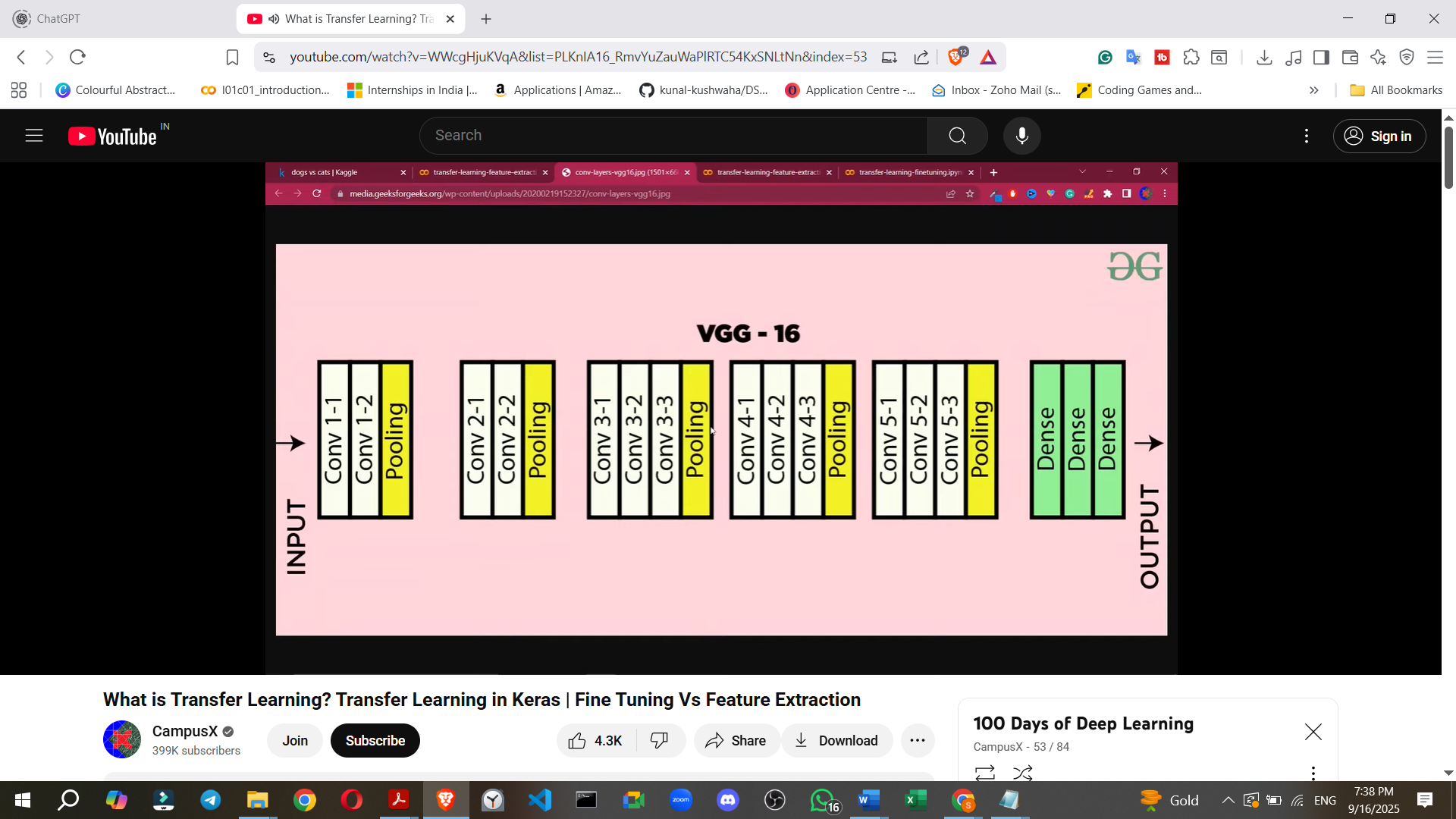
* + **Definition**: In fine-tuning, you **train the last few convolutional layers (unfreeze them) in addition to the newly added dense layers**, while keeping the initial convolutional layers frozen.
  + **When to Use**: This approach is used when the new problem's classification levels are **very different from the dataset the model was pre-trained on**.
  + **Example**: For phone vs. tablet classification, if ImageNet does not contain similar data, the later convolutional layers' knowledge might not be directly applicable. Unfreezing and retraining them allows the model to adapt its feature extraction for the new, distinct features.
  + **Consequence**: Fine-tuning generally **takes more time** because more layers are being trained.

**8. Practical Implementation in Keras (Cat/Dog Classification Example)**

The source demonstrates these techniques using Keras for a cat-dog classification task.

<https://www.kaggle.com/datasets/salader/dogsvscats>

* **Dataset**: A cat and dog image dataset is imported from Kaggle, resulting in train and test folders, each containing cats and dogs subfolders.
* **Common Setup Steps**:  
  1. Importing VGG16 from keras.applications.
  2. Creating a VGG16 object (conv\_base) with weights='imagenet' (to load pre-trained weights) and include\_top=False (to exclude the VGG16's original classification layers). The input shape is set (e.g., 150x150 pixels with 3 channels).
* **a) Feature Extraction Implementation**
  1. **Create a New Sequential Model**: An empty Sequential model is created.
  2. **Add the Convolutional Base**: The conv\_base is added to this new model.
  3. **Flatten**: A Flatten layer is added to convert the 2D output of the convolutional base into a 1D vector.
  4. **Add Custom Dense Layers**: Custom Dense layers are added, followed by an output Dense layer with one neuron and sigmoid activation for binary classification.
  5. **Freeze the Convolutional Base**: conv\_base.trainable = False is set. This means its weights will not be updated during training. The summary shows that only the newly added dense layers have trainable parameters.



* 1. **Data Generators**: ImageDataGenerator is used to load images from the train and test folders, automatically inferring labels. Images are resized and class\_mode='binary' is specified.
  2. **Pixel Value Normalization**: Pixel values (0-255) are rescaled to 0-1 using a preprocess\_input function (which VGG16 expects, or a simple division by 255) to speed up training.
  3. **Compile and Fit Model**: The model is compiled with optimizer='adam', loss='binary\_crossentropy', and metrics=['accuracy']. It is then trained using model.fit() for a specified number of epochs (e.g., 10).
  4. **Results (without Data Augmentation)**: Achieved around **90% accuracy** on validation data, a significant improvement from 81% accuracy achieved with a custom CNN from a previous video. However, **overfitting was observed**, with training accuracy reaching 98% and validation accuracy not increasing further, indicating a gap between training and validation performance.
* **b) Feature Extraction with Data Augmentation (to reduce Overfitting)**
  1. **Data Augmentation**: To reduce overfitting, ImageDataGenerator is configured with various transformations (e.g., rotation, width/height shift, shear, zoom, horizontal flip) for the training data. Only rescaling is applied to the test data.
  2. **Compile and Fit Model**: The model is compiled and trained as before, using model.fit\_generator due to data augmentation.
  3. **Results**: **Test accuracy improved to 91.4%**, and the gap between training and validation accuracy and loss curves significantly reduced, indicating less overfitting. The source suggests further improvements could be made using techniques like dropout, batch normalisation, or different weights.
* **c) Fine-Tuning Implementation**
  1. **Enable Training for Convolutional Base**: Unlike feature extraction, conv\_base.trainable = True is set.
  2. **Selective Unfreezing**: A loop is used to iterate through the layers of the conv\_base. **Only the layers from a certain block onwards (e.g., 'block5\_onwards') are unfrozen** (layer.trainable = True), while earlier layers remain frozen (layer.trainable = False).
  3. **Add Custom Dense Layers**: The same custom dense layers are added as in feature extraction.
  4. **Data Generators and Preprocessing**: The same data generators and pixel normalization are used.
  5. **Compile and Fit Model**: The model is compiled using optimizer=RMSprop with a **very low learning rate**, which is a common practice for fine-tuning.
  6. **Results**: Achieved **95.2% accuracy** on validation data. Training accuracy reached 99.8%. While still showing some overfitting, this is a significant improvement from the custom CNN's 81% and feature extraction's ~91%. The source suggests that data augmentation could further improve results in this fine-tuning scenario.

**9. Conclusion**

* Feature Extraction and Fine-Tuning are two powerful techniques for applying Transfer Learning in Keras.
* The source encourages users to run the provided notebooks, document their learning, and apply these techniques to other datasets for enhanced understanding.