FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

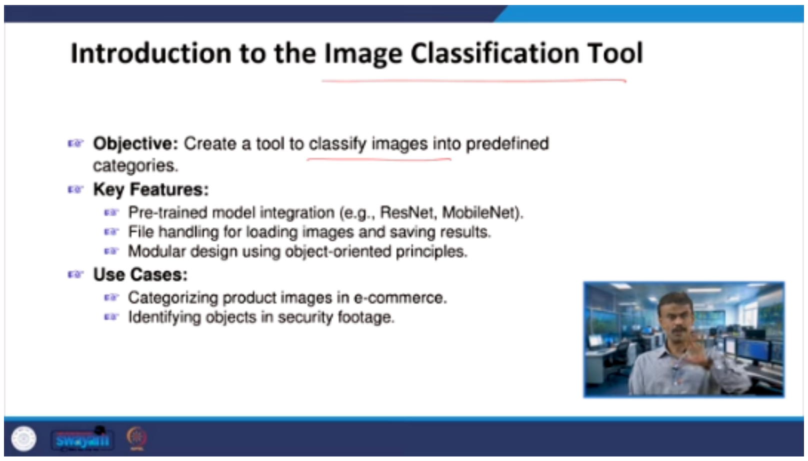
Lecture60

Lecture 60: Case Study - Image Classification Tool Python

So, welcome to lecture number 60, case studies in object oriented programming. So, we had seen two case study in C++ and two case studies in Java, right. So, in this lecture, what I can do, I will talk about Python, right. So, Python, I have chosen the problem of image classification, right, the digital image classification, because the whole world is behind the AI now, right. So, we will see some of the examples of

the neural networks right. So, how to use these neural nets using python right. So, I take python over here. So, the main objective of this case study is to create a tool to classify the images right into predefined categories right. Suppose if you are given a image.

So, let us say the image contains cats and dogs right. So, you have several images right it contains one image cat, another image dog, right, cat and so on. So, you have to do the classification. So, this is cat, this is dog.



Introduction to the Image Classification Tool

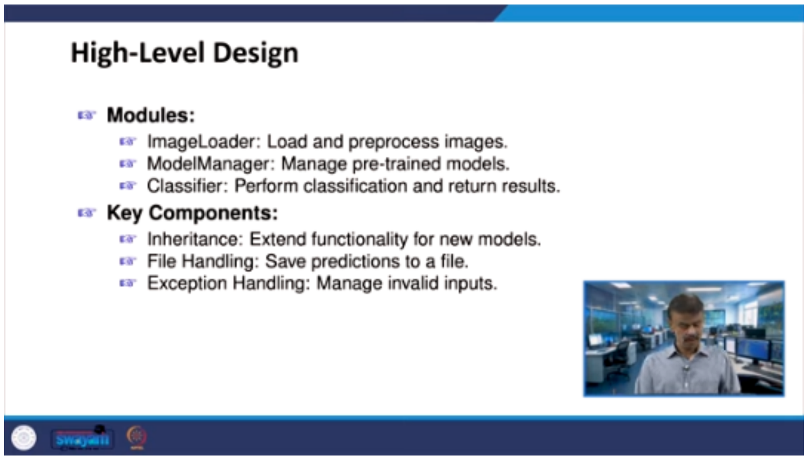
- **Objective:** Create a tool to classify images into predefined categories.
- **Key Features:**
  - Pre-trained model integration (e.g., ResNet, MobileNet).
  - File handling for loading images and saving results.
  - Modular design using object-oriented principles.
- **Use Cases:**
  - Categorizing product images in e-commerce.
  - Identifying objects in security footage.

So, that is called the classification, all right. So, the predefined categories. So, you call it as a supervised classification problem, right. So, we have a label, right. So, that we are going to train.

So, you are going to talk about the train. We call it as a pre-trained model integration. So, for this we use ResNet and MobileNet. Some of you might have studied the neural nets or deep learning algorithms. The ResNet, MobileNet.

So, these are all the standard terminology in the deep learning or even the neural network terminologies. Then file handling. We are going to handle the file where we have to load the images and then we have to save the results as well. So, this involves future extraction etcetera. So, therefore, we have to save the results.

Here we are going to have the modular design using the object oriented principles. The use cases are categorizing the product images in e-commerce and identifying the objects in security footage. So, the modules and key components in high level design. So, we are going to have image loader. So, which will load and pre-process the images and when you want to manage the pre-trained model, we have the model manager and in order to perform the classification and store the results or return the results, we have the classifier, all right.



High-Level Design

- **Modules:**
  - ImageLoader: Load and preprocess images.
  - ModelManager: Manage pre-trained models.
  - Classifier: Perform classification and return results.
- **Key Components:**
  - Inheritance: Extend functionality for new models.
  - File Handling: Save predictions to a file.
  - Exception Handling: Manage invalid inputs.

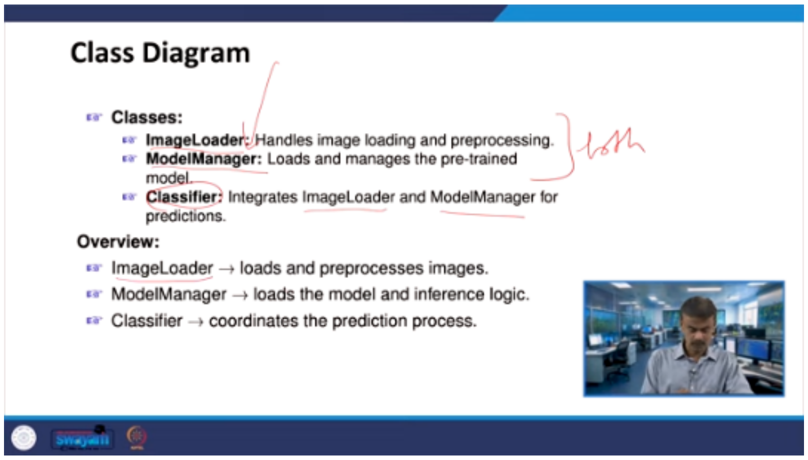
So, the classifier is the standard terminology in machine learning or deep learning. Right. And object oriented point of view, we are going to use inheritance. So this extends functionality for new model. Then we are going to have file handling.

Right. So whatever the predictions that you are having, so that will be saved in a file. Right. So prediction is another key word in machine learning or deep learning. Right.

So whatever the predictions that has come as a output, we have to save it as a file. right or save them as a file exception handling right. So, this will help you to manage invalid input right. So, this will manage the invalid input. So, as usual.

So, the class diagram the classes that we are going to use image loader. So, this will handle the images for loading and pre-processing. Then we are going to have model manager. So, for the pre-trained data or pre-trained model right. So, for labeling and managing we use model manager and the classifiers right.

The classifiers integrate image loader and model manager. So, this will integrate classifier integrate both that means, image loader and model manager right for the final predictions. So, image loader you have right. So, that will load and pre-process the image this overview. And model manager loads and manages the predefined model.


Class Diagram

- **Classes:**
  - **ImageLoader:** Handles image loading and preprocessing.
  - **ModelManager:** Loads and manages the pre-trained model.
  - **Classifier:** Integrates ImageLoader and ModelManager for predictions.

both

Overview:
- ImageLoader → loads and preprocesses images.
- ModelManager → loads the model and inference logic.
- Classifier → coordinates the prediction process.

So, here you call it as loading the model and inference logic, right? And the classifier coordinates the prediction process. So, this is the overview. So, in general, you call this a class diagram. So, let us see this Python code, right?

So, you have a class called ImageLoader. So, under this, you have __init__. This we had seen before, right? The __init__ method. Right. So, this is initializing the class with the image path.

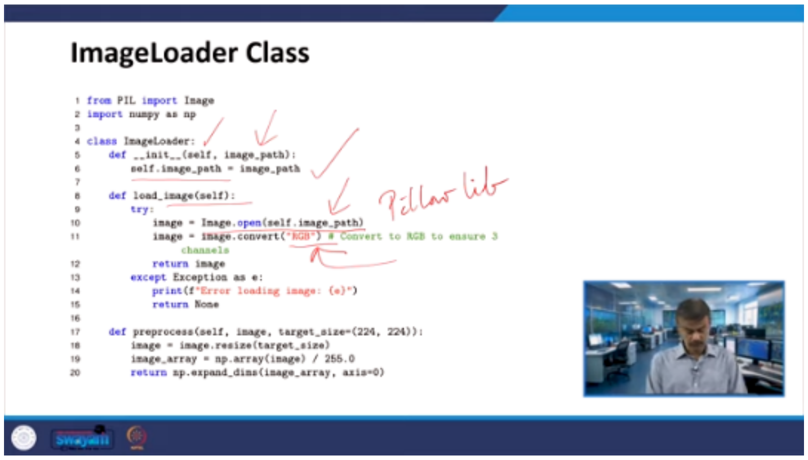
Right. So, that is provided by the user. Right. That is provided by the user. And self.image_path.

This is storing the path of the image file. Right. This is storing the path of the image file. So this is what we have seen up to line number six. Now.

If you go to line number 8, define load underscore image of self, right? Load underscore image. So, this method attempts to load the image, right? So, this method attempts to load the image from the given file path, all right? From the given file path, right?

So, that means it is opening the image using, the image is invoking open, right? So, here you have self which is invoking image underscore path. So you call this as a pillow library all right. So this is pillow library this is pillow library line number 10 and image is invoking convert RGB that means it is converting right it is converting the image to RGB format converting the image to RGB format. So we are using the convert

function right. So, convert you are pausing and in fact, you have under code you have RGB. So, this ensure the image as three color channels right. So, this is ensuring the image as three color channels the color channels are nothing, but red green and blue right. The color changes are nothing, but the red green and blue and you have return image.



ImageLoader Class

```
1  from PIL import Image
2  import numpy as np
3
4  class ImageLoader:
5      def __init__(self, image_path):
6          self.image_path = image_path
7
8      def load_image(self):
9          try:
10             image = Image.open(self.image_path)
11             image = image.convert("RGB") # Convert to RGB to ensure 3
                   channels
12             return image
13         except Exception as e:
14             print(f"Error loading image: {e}")
15             return None
16
17     def preprocess(self, image, target_size=(224, 224)):
18         image = image.resize(target_size)
19         image_array = np.array(image) / 255.0
20         return np.expand_dims(image_array, axis=0)
```

So, I will come to that right. If there is an error occurs right. So, because we are talking about the path and file etc., right because you are going to load the image from the given file that is what we had seen for the case of a load image, right.

Suppose as usual if you are not able to open the file or the file does not exist that means if you are having any exception, right.

So, this message will be printed it is like a e dot what, right the message will be printed the error message will be printed. So now if there is no error it is returning a image all right. It can be able to open the file and it is converting to RGB then it is able to return the file otherwise it will return none right. So this will return none ok. So this is what we had seen up to line number 15.

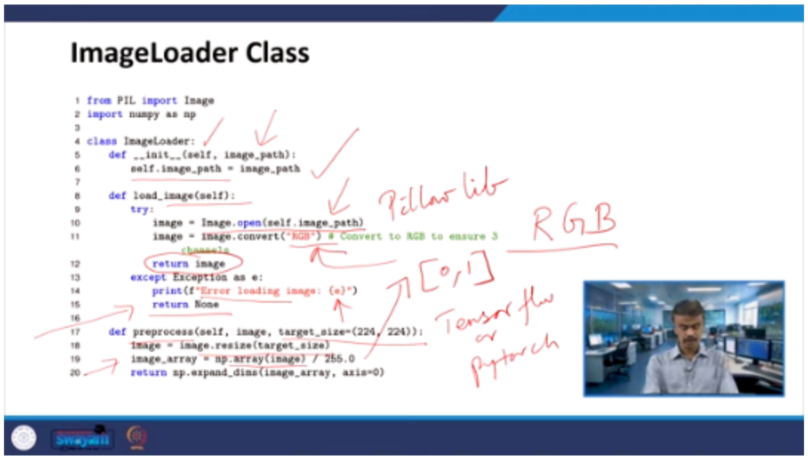
So next we have the pre-process method, right? The pre-process method, right? So under pre-process, Right. So, this prepares the image for use in machine learning models, right. So, this pre-process method, right, prepares the image for use in machine learning models, right.

So, mainly we use it for deep learning models, right. So, the frameworks like TensorFlow or PyTorch, right? TensorFlow or PyTorch, right. So here, what it is doing, all right, is resizing the image to 224 by 224, right? Whatever be the input, right. So let us say our target size is 224 by 224, right? The target size is 224 by 224. So here, you have line number 18, you have resize, right.

So whatever be the input, all right. By line number 18, right. So, this will be resized. Your target size is 224 by 224. So, that will be whatever be the image that you are giving, that will be resized to 224 by 224, right. And line number 19, so you have NumPy np, right.

So, which is converting, right, you have an array of image, that means every pixel, the gray level value you are dividing by 255, the meaning is this will go into the range 0 to 1 closed interval 0 to 1 all right. So, this is a very common in the deep learning models all right. So, this is I mean the gray level value correct. So, which will be convert m cross n image assume that m cross n image all right.

So, each array elements right that will be divided by 255. So, that means assume that you have 0 to 255 right. So, that means we know that the range will be now image underscore array range will be now 0 to 1. So, this is very common for deep learning algorithms and then finally, you have the numpy which is invoking expand underscore dimensions right DIMS.



ImageLoader Class

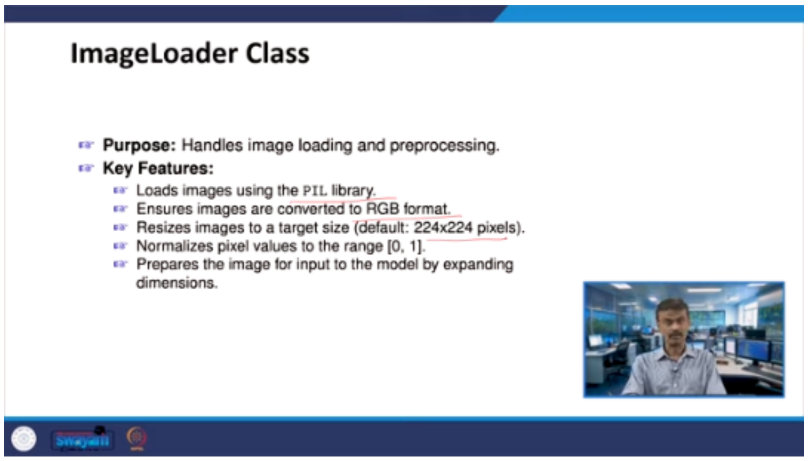
```
1  from PIL import Image
2  import numpy as np
3
4  class ImageLoader:
5      def __init__(self, image_path):
6          self.image_path = image_path
7
8      def load_image(self):
9          try:
10             image = Image.open(self.image_path)
11             image = image.convert("RGB") # Convert to RGB to ensure 3
                   channels
12             return image
13         except Exception as e:
14             print(f"Error loading image: {e}")
15             return None
16
17     def preprocess(self, image, target_size=(224, 224)):
18         image = image.resize(target_size)
19         image_array = np.array(image) / 255.0
20         return np.expand_dims(image_array, axis=0)
```

So, you have image array and axis is equal to 0 that means, this adds a new dimension right for batch size this adds the new dimensions for batch size, making it compatible with models, expecting your input shapes like, right, batch size, batch underscore size, right, what else, height, width, right, width and channels. all right and then it is returning numpy array that image array is numpy array right. So, this is what exactly it is nothing, but the image loader this class is considered as all right or it is called as the image loader class I hope it is very clear. So, whatever image first it is converting into RGB all right and here you have the pre process stage

right whatever be the image you have in hand will be resized to 224 by 224 and then every pixels you are converting into 0 to 1 right by dividing by normalizing right line number 19 is nothing but the normalization right and it is returning this particular case. So, image array with axis is equal to 0 right I hope it is very clear. So, the image loader class that we had seen in the last line. So, this is handling image loading and pre-processing

All right. So, key features are loading the image using pillow library and it is ensuring the image is converted into RGB format and we have resized. All right. So, we resize to 224 by 224 pixels. So, this is mainly for the mobile net.

All right. So, why you are converting? You may ask the question. All right. So, this is a mobile net features the mobile net.



ImageLoader Class

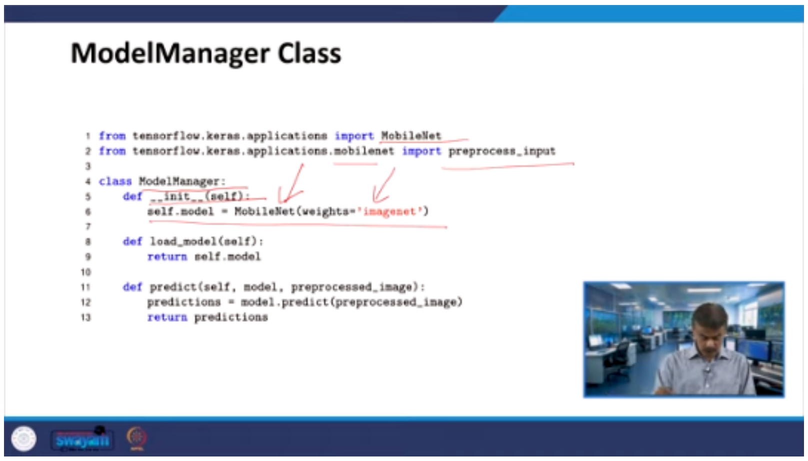
- **Purpose:** Handles image loading and preprocessing.
- **Key Features:**
  - Loads images using the `PIL` library.
  - Ensures images are converted to RGB format.
  - Resizes images to a target size (default: 224x224 pixels).
  - Normalizes pixel values to the range [0, 1].
  - Prepares the image for input to the model by expanding dimensions.

We are converting any size into 224 by 224. All right. In the deep learning MobileNet. And it is normalizing pixel values from 0 to 1, right? The range is 0 to 1, as we had seen, all right.

And then finally, it is preparing the image for input to the model by expanding the dimensions. So, these are all the key features we had seen in the last slide. And next, we will talk about the model manager, right. So, we use TensorFlow Keras applications, which imports MobileNet, right? So, the 224 by 224, right.

So, the MobileNet I talked about, this one, right. So, from TensorFlow Keras applications, MobileNet, we import preprocess_input, right. So, here you have the class ModelManager, right. So, the init method over here in line number 5. So, this is initializing the ModelManager, right.

So this is initializing the model manager class and loads the pre-trained mobile net and loads the pre-trained mobile net model right and loads the pre-trained mobile net model with image net weights right. So here you have the weights. So weights is image net right. So the meaning of image net here right. So this is nothing but the model is pre-trained right.



ModelManager Class

```python
1  from tensorflow.keras.applications import MobileNet
2  from tensorflow.keras.applications.mobilenet import preprocess_input
3
4  class ModelManager:
5      def __init__(self):
6          self.model = MobileNet(weights='imagenet')
7
8      def load_model(self):
9          return self.model
10
11     def predict(self, model, preprocessed_image):
12         predictions = model.predict(preprocessed_image)
13         return predictions
```

where on the ImageNet dataset, right, the model is pre-trained on the ImageNet dataset and then this will be stored here, right. So, once it is done, that is stored in self.model attribute, right, that is stored in self.model attribute. And line number 8, you have load underscore model, right. So, the load underscore model, right, so this returns MobileNet model stored in self dot model right mobile net model stored in self dot model right that is the purpose and this method is also allowing to access the model instance right if needed elsewhere in the program right.

So, these are all the purpose right. So, this method is allowing to access the model instance if needed elsewhere in the program, right? So, this is up to line number 9. So, here you have predict method, right?

The main idea of predict method is to make predictions using mobile net model, all right? So, here you have the model that you are explicitly passing all right and then you have preprocessed underscore image. So, which we know it is nothing but a numpy array of the preprocessed image right. So, array m cross n assume that is m cross n array right and then you have let us say process right. So the process which is calling the predict, right?

The models predict, right? Where this is the input image. The model is invoking predict, all right? Where you are pausing the pre-process the image, all right? So that will be stored in the predictions, right?

In the left-hand side, you have predictions. And then you are returning the predictions, all right? You are returning the predictions, right? When you are returning these predictions, so typically you have the probability distribution right over image net right.

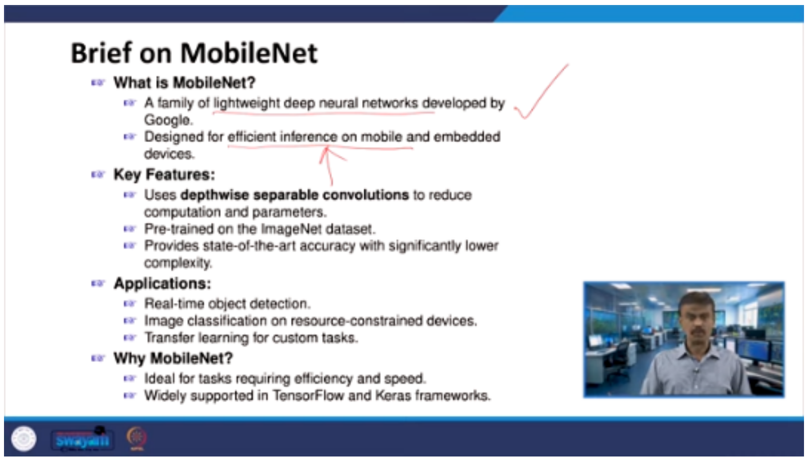
So, this will have, let us say, 1000 classes, right? So, this returns predictions, right? So, it returns the predictions. So, the meaning of this is the probability distribution, right, over ImageNet's 1000 classes, fine.

So, this is what the model manager class is doing, right? So, the purpose of the manager class. So, this is managing the pre-trained MobileNet model and predictions, which is what we had seen, right? So, the key features were loading the MobileNet model pre-trained on the ImageNet dataset, right? So, you have to load the MobileNet model, correct.

And then, this is handling the pre-processing of inputs using MobileNet-specific requirements. And it is performing inference to generate predictions for a given input, all right. So, this is what we had seen for the case of the model manager class. So, I talked about MobileNet. Now, you may ask the question: what do you mean by MobileNet?

Right. So, those who already know deep learning algorithms, you know that this is a family of lightweight deep neural networks, right? So, this was developed by Google, right? The MobileNet that I talked about, right? So, in the case of a deep neural network, this is called the family of lightweight deep neural networks, right? And then this was developed by Google, and why it was developed is mainly designed for

efficient inference on mobile and embedded devices, right? So, here I am talking about efficient inference on mobile and embedded devices. So, for this particular purpose, MobileNet has been designed. So, the main key features are that MobileNet uses depth-wise separable convolutions. So, you go into the deep learning algorithm, convolutional neural networks, right?



Brief on MobileNet

- **What is MobileNet?**
  - A family of lightweight deep neural networks developed by Google.
  - Designed for efficient inference on mobile and embedded devices.
- **Key Features:**
  - Uses **depthwise separable convolutions** to reduce computation and parameters.
  - Pre-trained on the ImageNet dataset.
  - Provides state-of-the-art accuracy with significantly lower complexity.
- **Applications:**
  - Real-time object detection.
  - Image classification on resource-constrained devices.
  - Transfer learning for custom tasks.
- **Why MobileNet?**
  - Ideal for tasks requiring efficiency and speed.
  - Widely supported in TensorFlow and Keras frameworks.

So, just go through that. I am just giving an idea, right? So, where is it being used? So, in fact, we have used several built-in libraries. So, the key features of MobileNet, which is nothing but depth-wise separable convolutions, reduce the computations and parameters. Right.

And another key feature is that it is a pre-trained model on the ImageNet dataset, and it is also providing state-of-the-art accuracy—the literature accuracy—with significantly lower complexity, right. So, many are using it; there are deep learning researchers. So, you can see, I mean, MobileNet is one of the famous architectures, right. So, what are all the applications? Real-time object detection, suppose in a video, right? I want to detect a few objects. Right.

I can use MobileNet and image classification algorithms. Right. And transfer learning. Right. Transfer learning for custom tasks.

Right. So, I have a pre-trained model. Right. Assume that it is finding out the cow image. Right.

It is finding out the cow image and I have the pre-trained model. Correct. So now I will use it. For example, I have been given the first case I had been given. Right.

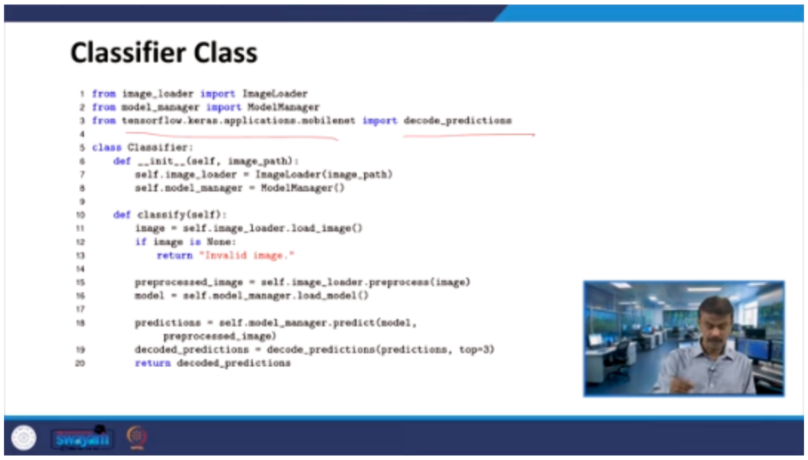
So it will detect the object. Right. Whether the cow present in the image or not. right or any other classification algorithm. So, the last one the cow example that I said it is a transfer learning.

So, already I had a pre-trained model. So, I can use it for the transfer learning, right. So, the main idea of using mobile net is. So, this is ideal for tasks requiring efficiency and this is also Slightly, right?

It is fast, right? Or more fast or faster than the other nets, right? So this is ideal for task requiring efficiency and speed. And this is widely supported in tensor flow and Keras, right? So in fact, we have imported the tensor flow and Keras, right?

So when I want to use the mobile net, if a first line, if you look, you have tensor flow and Keras. So now you have classifier class. right. So, here you have here importing image loader from image underscore loader model manager and tensor flow Keras and applications mobile net all right. So, these import decode underscore predictions.

So, now as usual you have a init method all right. So, the purpose of this is initializing the classifiers. right, it is a classifier class, right. So, this is initializing the classifier class, classifier class by creating instances of image loader and model manager, right. The instances are image loader and model manager, right.



Classifier Class

```python
from image_loader import ImageLoader
from model_manager import ModelManager
from tensorflow.keras.applications.mobilenet import decode_predictions

class Classifier:
    def __init__(self, image_path):
        self.image_loader = ImageLoader(image_path)
        self.model_manager = ModelManager()

    def classify(self):
        image = self.image_loader.load_image()
        if image is None:
            return "Invalid image."

        preprocessed_image = self.image_loader.preprocess(image)
        model = self.model_manager.load_model()

        predictions = self.model_manager.predict(model,
             preprocessed_image)
        decoded_predictions = decode_predictions(predictions, top=3)
        return decoded_predictions
```

So, what are all the inputs? Here you have image_underscore_path; you are passing image_underscore_path, right. See, this is a path to the image to be classified. So, this path you are giving. So, like a file image_underscore_path, and then you have self.image_underscore_loader, right? On the right-hand side, you have image_loader where you are passing this path image_underscore_path, right.

So, this is an instance: self.image_underscore_loader. This is an instance. So, this is the instance of the path image_loader. Image_loader is a class. For loading and pre-processing the specified image, correct? So, for loading and pre-processing the specified image. And then you have model_underscore_manager: self.model_underscore_manager.

So, this is an instance of the model_manager, right? Model_manager class for managing the MobileNet model, for managing the MobileNet model, all right? So, next one is you have classify method. So, the classify method you are loading the image. You are loading the image.

If there is no image like the file does not exist or you are not able to load the image. It is printing the invalid image. Now, we know the load image what it will do? It will convert into RGB. It is converting the image into RGB.

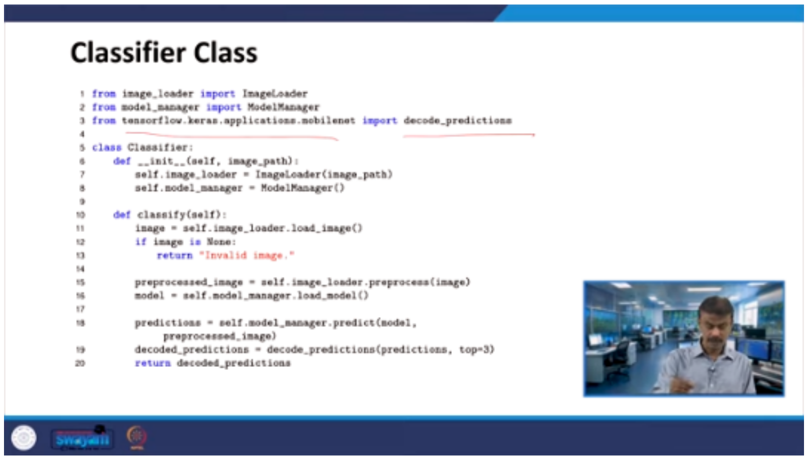
So, here you have image loader. right, which is invoking load image. So, that means, it will open and converts into RGB, right, it is converting into RGB. So, if you are not able to open the image, it will return invalid image. So, up to line number 13, it is very clear.

So, line number 19, you have the self, right, image underscore loader. So that is pre-process the image, right. So pre-process the image. What is the meaning of pre-process the image? It will resize.

We have seen 224 by 224 and normalized the image using the pre-process method, right. So this pre-process method, which we have already seen. So resize and do the normalization, right. So this is available in the image loader class. Right, or the image loader method.

Correct. Image loader method. This is for pre-processed images. Then you have load model. Right.

Line number 16, you have load model. So the load model fetches the MobileNet. Right. It is fetching the MobileNet. Or I can say it is fetching the MobileNet model.



Classifier Class

```
1  from image_loader import ImageLoader
2  from model_manager import ModelManager
3  from tensorflow.keras.applications.mobilenet import decode_predictions
4
5  class Classifier:
6      def __init__(self, image_path):
7          self.image_loader = ImageLoader(image_path)
8          self.model_manager = ModelManager()
9
10     def classify(self):
11         image = self.image_loader.load_image()
12         if image is None:
13             return "Invalid image."
14
15         preprocessed_image = self.image_loader.preprocess(image)
16         model = self.model_manager.load_model()
17
18         predictions = self.model_manager.predict(model,
               preprocessed_image)
19         decoded_predictions = decode_predictions(predictions, top=3)
20         return decoded_predictions
```

Right. With the help of this load underscore model method of model manager. model underscore manager, okay. So, so that will be stored in model, that will be stored in model. So, now you have to make the predictions, all right.

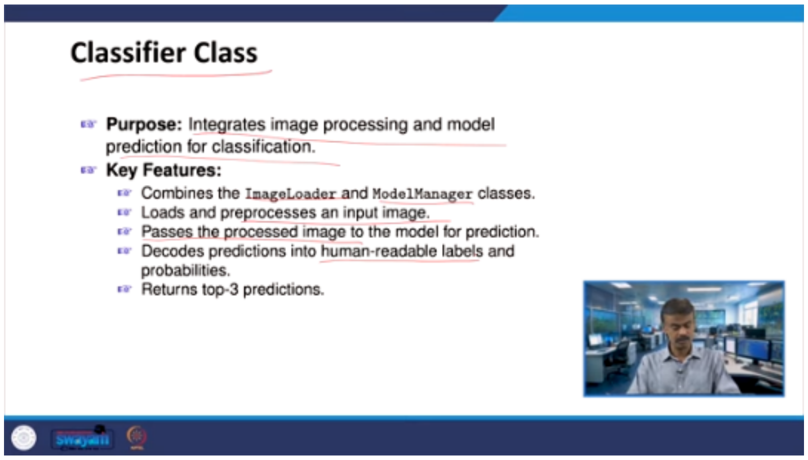
So, for that we use predict, right, you have to make the predictions. So, for that you use the predict method and here you are pausing model and the pre-processed image, all right. So, now here your predict method So, you are passing the pre-processed image right and then it will do the predictions right and then it is stored in the predictions left hand side. So, what do you mean by predictions?

So, it is nothing but the array of size 1000 right. So, this is what I said array of size 1000. So, which contains the probability scores right of the distribution over the image net class and the next one is line number 19 this is what about line number 18 and line number 19 you have decode predictions right so it is converting the raw predictions into a raw predictions into a right we have we had thousand the raw predictions into the top three predicted class right the top three predicted classes top three predicted classes top 3 predicted classes with confidence score with confidence score alright.

So, that will be assigned to decoded predictions, and that decoded underscore predictions will be returned, OK. So, this is what the classifier class will do. So, the classifier class, right? So, this integrates image processing and model prediction for classification, right? So, mainly the computer vision problem or image processing.

So, mainly this is useful for the classification model prediction for classification. So, the key features. So, we combine the image loader and model manager, right, with the help of the classifier class, and it is loading and pre-processing the input image, right. It passes the processed image to the model for prediction, and the classifier class decodes predictions into human-readable labels, right.

So, that is what I talked about, right? The top three, right? The human-readable labels. In fact, you had a thousand, right? You had a thousand with the probability distribution. So, out of this, it will return the top three predictions. So, these are all the key features we had seen in the last slide. So, here we are seeing file handling for saving predictions.



Classifier Class

- **Purpose:** Integrates image processing and model prediction for classification.
- **Key Features:**
  - Combines the `ImageLoader` and `ModelManager` classes.
  - Loads and preprocesses an input image.
  - Passes the processed image to the model for prediction.
  - Decodes predictions into human-readable labels and probabilities.
  - Returns top-3 predictions.

Here you have class file handler and this is called the decorator, right? The name of the decorator is static method, correct? So this decorator indicates, right? So you are going to define the save prediction method, right? So the save prediction method, which is a static method and the static method can be called without creating an instance of the class, right?

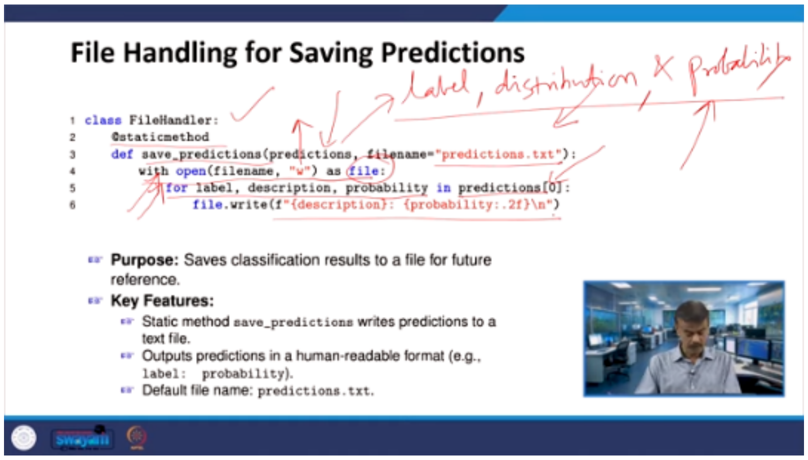
Without creating an instance of the class. the static methods can be called in Python. And here you have defining save underscore predictions, right? And then you are having predictions, you are passing predictions. And here you have the file name predictions.txt, right?

The file name is predictions.txt, right? So that means this particular line, I mean to say line number three, that is defining the save underscore predictions method, right? And the two arguments, the predictions, right? So the predictions is nothing but the list of predictions where every prediction or each prediction is a tuple containing a label, right? This contains label distribution, label distribution and probability, right?

So this predictions contain label distribution and probability, right? and then file name. So, the name of the file name is all right. So, the predictions right. So, the file name to save the predictions is predictions dot text right.

So, whatever the predictions that will be saved into predictions dot txt right. So, this is up to line number 3. So, now saving predictions to a file right. So, here you have the with statement right the with statement. So, this is ensuring that the file is closed properly right the file is closed properly even if an exception occurs right even if there is any exception right.

So, the file is closed properly and the next line you have for label comma description comma probability that is what I said in predictions of 0. So, that means this particular line iterates over the first element of right the 0th index or the first element of predictions list which is assumed to be a list of tuples right. The tuples are label, description and the probability right. So, this is what line number 5 and finally, line number 6 you have file dot write right f description probability. right.



File Handling for Saving Predictions

```
1 class FileHandler:
2     @staticmethod
3     def save_predictions(predictions, filename="predictions.txt"):
4         with open(filename, "w") as file:
5             for label, description, probability in predictions[0]:
6                 file.write(f"{description}: {probability:.2f}\n")
```

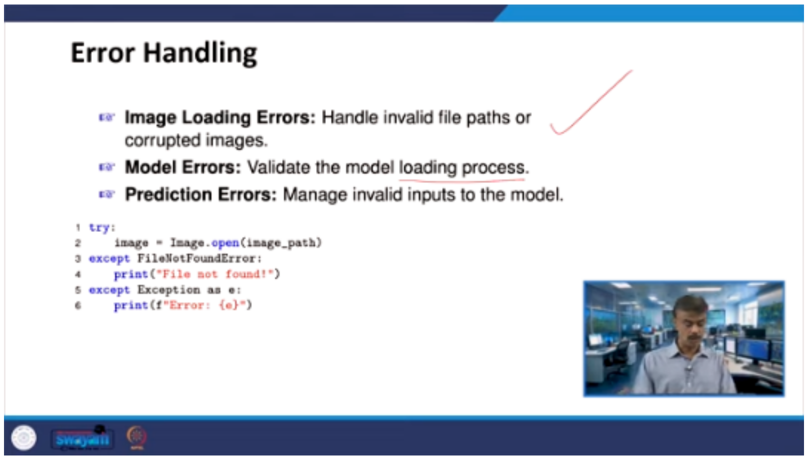
- **Purpose:** Saves classification results to a file for future reference.
- **Key Features:**
  - Static method `save_predictions` writes predictions to a text file.
  - Outputs predictions in a human-readable format (e.g., `label:  probability`).
  - Default file name: `predictions.txt`.

So, this is the line writes a formatted string to the file, right. So, the formatted string to the file you have f to f etcetera, all right. So, the formatted string will be returned. So, you have the description and the probability. So, this will be returned.

So, this is your file handling for saving prediction. So, that means the purpose is It will save the classification results. So, whatever the results that you have got. So, that will be saved to a file for the future reference.

So, here the key features are you have a static method right the static method. The static method is a save prediction. So, this is writing the prediction to a text file right and then output predictions in a human readable format. For example, you have the label probability etcetera right the label. probability all right and the default file name is predictions dot txt right.

So, this is all about your file handling for saving predictions. So, the next one is error handling right suppose if you have a invalid file name or invalid file path or the file name does not exist or you have a corrupted image all right. So, you get the image loading errors right this is the image loading errors model errors the validation right. But during the loading process, right, so you have to validate the model loading process. You call it as a model error and a prediction error.



Error Handling

- **Image Loading Errors:** Handle invalid file paths or corrupted images.
- **Model Errors:** Validate the model loading process.
- **Prediction Errors:** Manage invalid inputs to the model.

```python
try:
    image = Image.open(image_path)
except FileNotFoundError:
    print("File not found!")
except Exception as e:
    print(f"Error: {e}")
```

So, it is managing the invalid inputs to the model, right. So, like a try-through catch in Java or in C++ here, right. So, try, you are trying to open the image path, right, and then you are trying to store it as an image, right. So, if you are not able to open all right. So, if there is any exception all right.

So, these are all the cases the image loading error model error or prediction error or in particular right. So, this because you are trying to open this all right. As usual what we have studied in Java suppose you have all right any of the errors all right then the exception will be thrown. So, the exception will be file not found error. So, that will be printing file not found.

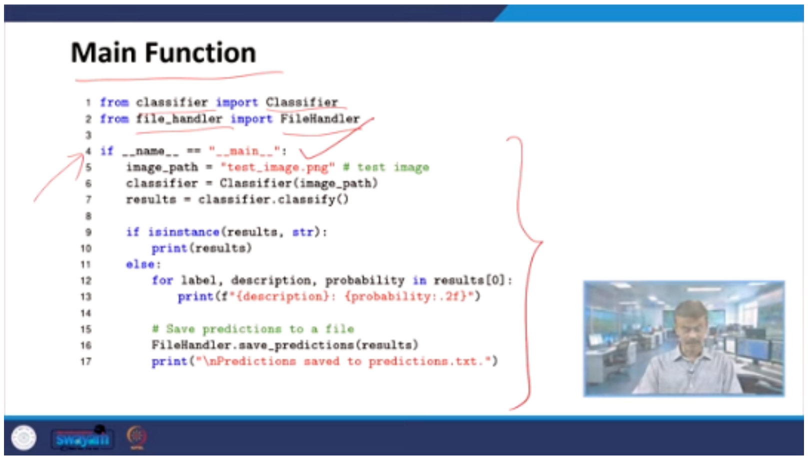
And the exception will be as the message and this particular will be printed error with the message will be printed. It is exactly like what we studied what. right in C++ e dot what it is equivalent to that. So, this is the error handling exception. So, with all now we are going to main function.

So, we have seen so many right classes. So, with the help of those classes we will go to the main function. So, here we import classifier with the help of small c classifier. And file handler with the help of file underscore handler. And line number 4 you have name is equal to main.

So the meaning is this particular line is ensuring that the following code only runs if the script is executed directly. Right. So not when it is imported as a module. So you got to make sure that is what this particular line. Line number 4.

Right. So now here you have image_underscore_path. Right. The image_underscore_path is test_underscore_image.png. Right.

And then the classifier is passing this image_underscore_path. Right. So that means this is creating an instance of the classifier class. Right. Passing image_underscore_path.



Main Function

```
1  from classifier import Classifier
2  from file_handler import FileHandler
3
4  if __name__ == "__main__":
5      image_path = "test_image.png" # test image
6      classifier = Classifier(image_path)
7      results = classifier.classify()
8
9      if isinstance(results, str):
10         print(results)
11     else:
12         for label, description, probability in results[0]:
13             print(f"{description}: {probability:.2f}")
14
15         # Save predictions to a file
16         FileHandler.save_predictions(results)
17         print("\nPredictions saved to predictions.txt.")
```

You are passing image_underscore_path as an argument. Right. So on the left-hand side, you have classifier. And then results. Right.

The classifier is invoking the classify method. Right. So, this classify method is on the classifier correct object to get the results, the classification results. So, up to line number 8 is over. So, now here you have an if statement: if isinstance of results, str, that means if the result is a string, right? If it is,

So, then in that case, it is printing the string directly, right. Otherwise, if the result is not a string, right, then it is assuming the three-tuple. The three tuples are label, description, and probability, right. So, then it iterates over the first tuple, right. So, you will have a tuple, right.

So, let us say I put some number, right, the label. You have the label. I put L. And some description, the probability points to something like that, right? So that means the first tuple iterates over the first tuple in the list, and it is printing the description and the probability in a formatted string, right? So that is what exactly is happening: printf. The print description and probability as a string, right.

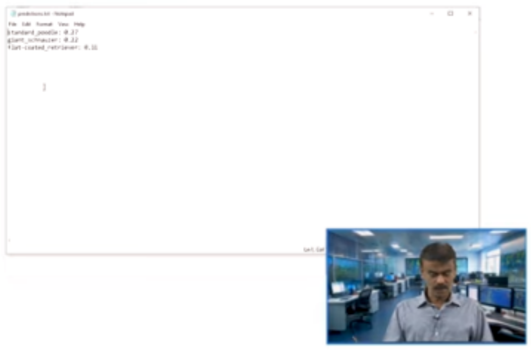
So now we are going to save predictions to a file, right? So the file handler is invoking save_predictions, right? So, it is saving the results, right? So, in fact, it is saving the results in predictions.txt. Right, it is saving the results in predictions.txt.

So, that is what you are printing. So, print 'predictions saved to predictions.txt'. So, this is about your main program or main function, right? So, this is what we had seen, right? So, in fact, this is useful for the main function, which is useful for the classification point of view. It is accepting an image path as input and initializing the classifier class to process the image.

And generate the predictions, all right? So, you have, let us say, thousands of distributions, all right? And then it is taking the top three predictions with probabilities—thousands of probability distributions—and you have the top three predictions of probability. The file handler is saving the predictions to a text file, which is nothing but predictions.txt, right? So, with all the classes and the main function, if I run the code, I have to get the output like this, okay? So, maybe what we can do is we will test this in Python.

So, I run the code all right. So, you can see the output and finally, it is saying that right it is the predictions are saved in predictions dot txt all right. So, I will give the complete code and you can run. So, the standard you can see this all right the all the numerical values and predictions dot txt.

So, what we can do we can open this predictions dot txt and you can see all right the output all right. So, this is slightly a harder case study because I just want to introduce the AAML concept all right. So, those who are already familiar this particular case study will be useful or otherwise those who have started fresh so this will be like a spark right so from here you will get an interest and then you try to run the similar code in python right similar code in deep neural network using python right so this is what exactly we got similar to this right and then it was saved in predictions dot txt right so what we can do the future enhancement so we can use the multiple image formats So, any image formats you have to do that and we can also integrate this with the custom trained models and the graphical user interface for user friendly interactions.



```
standard_poodle: 0.27
giant_schnauzer: 0.22
flat-coated_retriever: 0.11
```

So, here we had seen the output provide the CMD the command prompt. So, what you can do you can try the graphical user interface because we have already studied and there is a concept called batch processing. So, this batch processing we can handle the multiple images. and also the advanced error logging and reporting so these also we can think of right so these are all the future enhancements if those who are interested in this particular area or the case study like this so in this case study particularly the image classification tool so the main idea is to we have developed a modular python based tool for the image classification and then we use the pre-trained mobile net model right.

So, we have talked about the MobileNet model, 224 by 224, and the key components we discussed. I talked about the image loader, right? The model manager, the classifications—right? Like the top 3 classifications out of the 1000-probability distribution. We have also— Right, we also covered the file handling system because, all right. So, you have to save the prediction, and also you have to take the input file, right? As an image, all right. And then, finally, we run the main function. So, the features are—this can also handle invalid image paths and formats because of the error handling exception, all right. And it saves the predictions in a human-readable format. So, let us say the top—I mean, in fact, the 1000-probability distribution.

So, from there, we could predict the top 3, right? With the labels and the probabilities. So, with this, I am concluding the case study. The fifth case study is using Python, and in fact, in the complete course, right. So, we have covered classes and objects in object-oriented programming, and we also talked about inheritance, right. You also studied polymorphism, and now you know what we mean by constructors and destructors. We have seen examples of overloading and overriding.

Also, I covered encapsulation and abstraction. So, we have done error handling exceptions. In fact, all five case studies—I mean, we worked on error handling exceptions, file handling. This, too, we have done for all the case studies, and we have separately studied. You know what we mean by templates in C++ and generics in Java, right?

And I also covered the Standard Template Library. We have also seen design patterns, right? And then finally, we talked about advanced topics in multithreading, network programming, and graphical user interfaces, right? So, apart from all these, We have also worked on five different case studies using C++, Java, and Python.

I hope you have enjoyed the course. Thank you very much.