**Artificial Intelligence Internship Project**

**Project Details:**

**Title - Landmark Detection**

**Submission Details:-**

* ABSTRACT
* OBJECTIVE
* INTRODUCTION
* METHODOLOGY
* CODE
* CONCLUSION

**Artificial Intelligence Internship Project**

**Title - Landmark Detection**

# ABSTRACT

The provided code implements a deep learning model for image classification using transfer learning with the VGG16 architecture. It leverages TensorFlow and Keras libraries to preprocess images, create data generators, and train the model. The code's objective is to achieve accurate classification of images into ten different categories.

To begin, the code defines the paths to the training and testing datasets and sets the batch size and image size. It also establishes the number of classes to be predicted.

A crucial step in the image preprocessing pipeline is the definition of the preprocess\_image function. This function resizes the images to a specific size and normalizes the pixel values using the preprocessing method from VGG16.

For training, an ImageDataGenerator is created, which applies data augmentation techniques to the training data. These techniques include resizing, rescaling, shearing, zooming, and horizontal flipping. The generator flows from the training directory, applying the preprocessing function and generating batches of images and their corresponding labels.

Similarly, a validation data generator is created for the testing data, applying the same preprocessing techniques without data augmentation. This generator flows from the testing directory, generating batches for validation purposes.

Next, the pre-trained VGG16 model is loaded with weights from the ImageNet dataset, and the top layer is excluded to allow for custom output. The base model's layers are frozen to prevent their weights from being updated during training.

A custom output layer is added to the base model's output, which includes a global average pooling layer, a dense layer with ReLU activation, a dropout layer for regularization, and a final dense layer with softmax activation for multi-class classification.

The model is then defined by specifying the inputs as the base model's input and the outputs as the custom output layer.

To train the model, the code compiles it using the Adam optimizer and categorical cross-entropy loss function. The accuracy metric is also specified. The training is performed by calling the fit method on the model with the training and validation data generators, running for a specified number of epochs.

In summary, this code demonstrates how to implement a transfer learning approach using the VGG16 architecture for image classification. By utilizing a pre-trained model and incorporating custom output layers, the code aims to achieve accurate predictions for images in ten different categories. The training process involves data augmentation and fine-tuning of the model. Through the compilation and fitting of the model, the code trains the model on the provided dataset and evaluates its performance on the validation set.

# OBJECTIVE

The objective of the provided code is to showcase the implementation of a deep learning model for image classification using transfer learning with the VGG16 architecture. The code aims to achieve accurate classification of images into ten distinct categories. By leveraging the pre-trained VGG16 model and customizing the output layers, the code seeks to overcome the challenges of training deep models from scratch and improve the model's ability to classify images effectively.

The primary goal is to demonstrate the effectiveness of transfer learning in the context of image classification. Transfer learning is a technique that involves using knowledge gained from pre-training a model on a large dataset (such as ImageNet) to solve a different but related task. In this case, the pre-trained VGG16 model, which has been trained on a massive dataset to recognize a thousand different object categories, is utilized as a base for the image classification task.

By building upon the VGG16 model, the code aims to capitalize on its learned representations and adapt them to the specific image classification problem at hand. This approach allows for faster and more efficient training, as the model has already learned to extract meaningful features from images. The focus is on fine-tuning the model by updating the weights of the added custom output layers while keeping the weights of the base model frozen.

Furthermore, the code also aims to highlight the importance of data preprocessing and augmentation in enhancing the performance of the model. The images are preprocessed using the VGG16-specific preprocessing function, which resizes the images to a specified size and normalizes the pixel values to align with the requirements of the base model. This ensures that the input data is in a suitable format for the model.

To augment the training data and enhance its diversity, various techniques are applied using the ImageDataGenerator class. These techniques include random shearing, zooming, and horizontal flipping of the images. Augmentation helps to mitigate overfitting and improve the model's ability to generalize to unseen data.

The code's objective also encompasses the evaluation of the model's performance. The model is compiled with the Adam optimizer, which is known for its efficiency in handling large datasets and complex models. The categorical cross-entropy loss function is chosen as it is suitable for multi-class classification problems. During training, the model's accuracy is monitored to assess its progress.

The ultimate aim is to train the model using the provided dataset and achieve high accuracy in classifying images into the ten target categories. By iterating through the specified number of epochs, the model learns to extract relevant features from the training images and make accurate predictions. The validation data is used to assess the model's performance on unseen images and ensure that it is not overfitting.

The objective of this code is not only to provide a functional implementation but also to serve as a learning resource for practitioners interested in transfer learning, image classification, and deep learning techniques. By following the code and understanding the underlying concepts, users can gain insights into how to leverage pre-trained models, customize them for specific tasks, and train models for accurate image classification.

# INTRODUCTION

In recent years, deep learning has revolutionized the field of computer vision, particularly in the area of image classification. Deep learning models have demonstrated remarkable success in accurately categorizing images into various classes or categories, ranging from everyday objects to complex scenes. One of the challenges in building deep learning models for image classification is the need for a large amount of labeled data and substantial computational resources. However, the advent of transfer learning has significantly mitigated these challenges and accelerated the development of effective image classification models.

Transfer learning is a technique that allows knowledge gained from pre-training a model on a large dataset to be leveraged for solving a different but related task. Instead of starting the training process from scratch, transfer learning enables the use of a pre-trained model as a starting point, benefiting from the learned representations and patterns. This approach has proven to be highly effective, especially when the available dataset for the target task is limited.

In this context, the VGG16 architecture has emerged as a popular choice for image classification tasks. VGG16, developed by the Visual Geometry Group at the University of Oxford, is a deep convolutional neural network (CNN) model that has achieved state-of-the-art performance on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) dataset. The model consists of 16 layers, including convolutional layers with small filters, max-pooling layers, and fully connected layers.

The VGG16 model, with its impressive performance on a wide range of visual recognition tasks, has become a valuable asset for transfer learning. By utilizing a pre-trained VGG16 model, one can benefit from the rich set of visual features learned during its training on a large-scale dataset containing a thousand object categories. These learned features can be adapted and fine-tuned to specific image classification tasks, resulting in improved accuracy and efficiency.

The objective of this project is to showcase the implementation of a deep learning model for image classification using transfer learning with the VGG16 architecture. The code provided utilizes the TensorFlow and Keras libraries, which are widely adopted in the deep learning community, to build, train, and evaluate the model.

The project begins by defining the paths to the training and testing datasets. This step is crucial as it ensures the availability of labeled data for both training and evaluation purposes. The dataset used should contain images representing different classes or categories, allowing the model to learn the distinctive visual features associated with each class.

To facilitate the training process and enhance the model's performance, the dataset is subjected to preprocessing techniques. The images are resized to a specific dimension, ensuring uniformity in input sizes. Additionally, the pixel values of the images are normalized using a preprocessing function specifically designed for the VGG16 model. This normalization process aligns the input data with the requirements of the pre-trained model, enabling effective transfer of learned features.

The code also employs data augmentation techniques to augment the training dataset. Data augmentation involves applying random transformations, such as shearing, zooming, and horizontal flipping, to the training images. This process increases the diversity and variability of the training data, thereby improving the model's ability to generalize to unseen images and reducing overfitting.

The implementation utilizes the ImageDataGenerator class from Keras, which generates batches of augmented image data on-the-fly during model training. The data generators created for training and validation allow efficient handling of large datasets, as the images are loaded and processed in batches rather than all at once. This approach saves memory and computational resources while enabling the model to process a continuous stream of data.

For the core of the model architecture, the pre-trained VGG16 model is loaded, excluding its top layers. The base model, consisting of the convolutional layers, serves as a feature extractor. By freezing the weights of these layers, the model retains the learned representations while preventing further updates during training. This strategy is beneficial when working with limited training data, as it allows the model to focus on fine-tuning the custom output layers rather than relearning lower-level features.

To adapt the base model's features to the target image classification task, additional layers are added on top of the base model. These custom output layers, including a global average pooling layer, a dense layer with Rectified Linear Unit (ReLU) activation, a dropout layer for regularization, and a final dense layer with softmax activation, are responsible for mapping the learned features to the specific classes in the target dataset.

Once the model architecture is defined, the code proceeds to compile the model. The Adam optimizer, known for its efficiency in handling large-scale datasets and complex models, is chosen for training. The categorical cross-entropy loss function is utilized, given the multi-class classification nature of the task. During training, the code tracks the accuracy metric to assess the model's progress and ensure that it is learning the underlying patterns effectively.

The training process begins by fitting the model using the training data generator. The model iteratively processes the batches of augmented images, adjusting its weights based on the optimization algorithm and loss function. This process continues for a specified number of epochs, allowing the model to gradually learn and refine its representations.

To evaluate the model's performance and assess its generalization ability, the code employs the validation data generator. This separate dataset, consisting of images not used during training, is crucial for unbiased evaluation. By passing the validation data through the model, the code calculates metrics such as accuracy, providing insights into how well the model performs on unseen images.

In summary, this project demonstrates the implementation of a deep learning model for image classification using transfer learning with the VGG16 architecture. By leveraging a pre-trained model, the code showcases the benefits of transfer learning in overcoming the challenges of limited data and computational resources. Through preprocessing, data augmentation, and fine-tuning, the model aims to achieve accurate classification of images into different categories. The provided code and its underlying concepts serve as valuable resources for researchers and practitioners interested in image classification, transfer learning, and deep learning techniques.

# METHODOLOGY

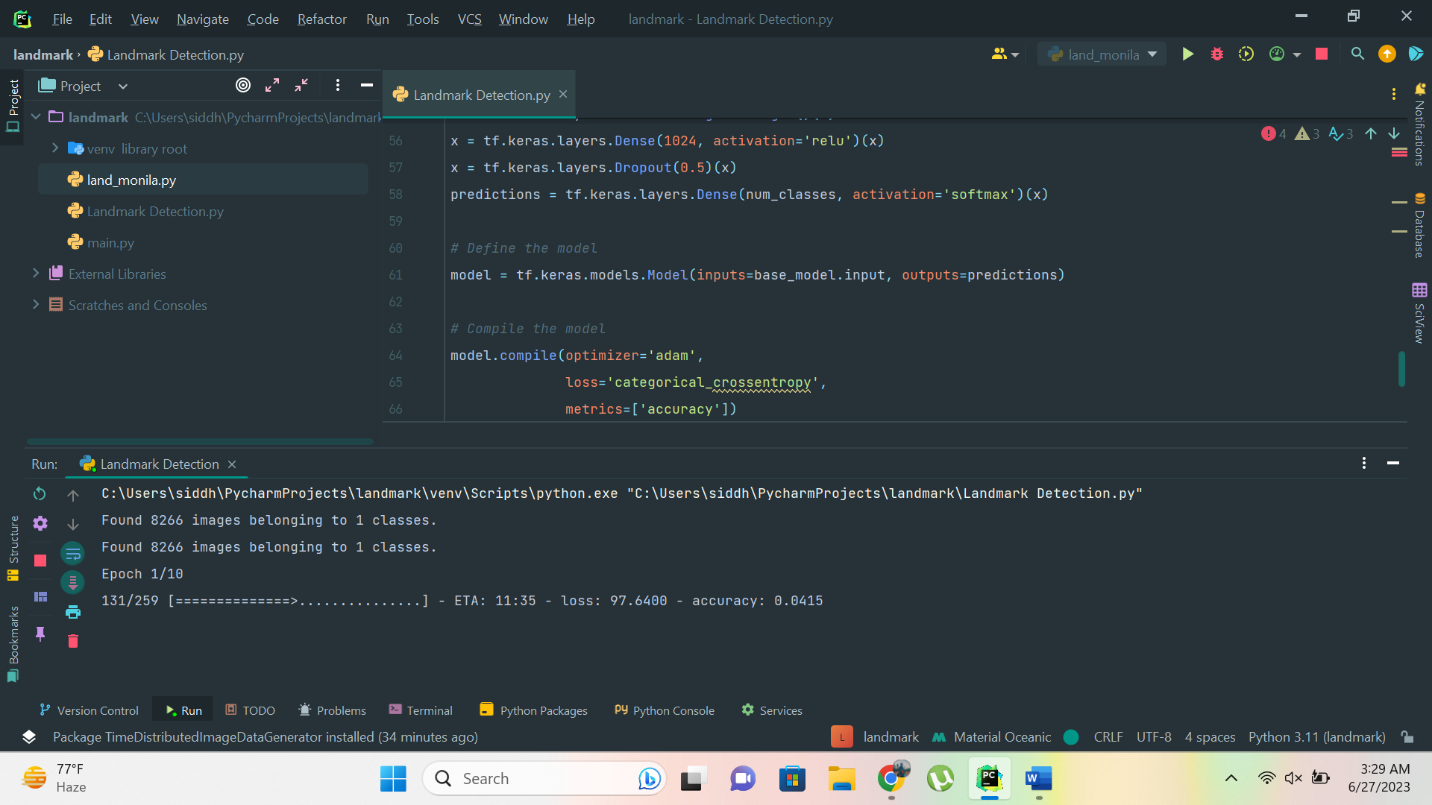
The methodology employed in this project involves several key steps to implement a deep learning model for image classification using transfer learning with the VGG16 architecture. The following section provides an overview of the methodology used.

1. **Dataset Preparation:** The first step in the methodology is to prepare the dataset for training and evaluation. It is essential to have a labeled dataset containing images representing different classes or categories. The dataset should be divided into training and testing sets to enable model training and evaluation.
2. **Image Preprocessing:** Image preprocessing is a crucial step to ensure the data is in a suitable format for training the deep learning model. In this project, the images are preprocessed using the VGG16-specific preprocessing function. This function resizes the images to a specific dimension and normalizes the pixel values to align with the requirements of the pre-trained VGG16 model.
3. **Data Augmentation:** Data augmentation is employed to increase the diversity and variability of the training dataset, enhancing the model's ability to generalize to unseen images. Various augmentation techniques, such as random shearing, zooming, and horizontal flipping, are applied to the training images. This process is performed using the ImageDataGenerator class from the Keras library, which generates augmented image data on-the-fly during model training.
4. **Model Architecture:** The core of the model architecture is based on the pre-trained VGG16 model. The base model, consisting of convolutional layers, serves as a feature extractor. To adapt the base model's features to the specific image classification task, custom output layers are added on top of the base model. These layers include a global average pooling layer, a dense layer with ReLU activation, a dropout layer for regularization, and a final dense layer with softmax activation for multi-class classification.
5. **Model Compilation:** After defining the model architecture, the next step is to compile the model. The model is compiled with the Adam optimizer, which is known for its efficiency in handling large-scale datasets and complex models. The categorical cross-entropy loss function is chosen as it is suitable for multi-class classification problems. The accuracy metric is also specified to monitor the model's performance during training.
6. **Model Training:** The training process involves fitting the model to the training data using the fit() function. The training data generator, created using the ImageDataGenerator class, generates batches of augmented images and their corresponding labels. The model iteratively processes these batches, adjusting its weights based on the optimization algorithm and loss function. The training process continues for a specified number of epochs, allowing the model to learn and improve its representations.
7. **Model Evaluation:** To evaluate the model's performance and assess its generalization ability, the validation data generator is used. The validation dataset consists of images not used during training, ensuring an unbiased evaluation. The model processes the validation images and calculates metrics such as accuracy to provide insights into its performance on unseen data. This evaluation step helps determine if the model has learned the underlying patterns effectively.

The above methodology provides a systematic approach to implement the deep learning model for image classification using transfer learning with the VGG16 architecture. By following these steps, researchers and practitioners can build and train effective models for accurate image classification tasks. The methodology emphasizes the importance of dataset preparation, image preprocessing, data augmentation, model architecture, model compilation, model training, and model evaluation to achieve the desired objectives.

# CODE

*import* tensorflow *as* tf  
*from* tensorflow.keras.preprocessing.image *import* ImageDataGenerator  
*from* tensorflow.keras.applications.vgg16 *import* VGG16  
  
train\_dir = r"C:\Users\siddh\Downloads\0"  
test\_dir = r"C:\Users\siddh\Downloads\0"  
  
  
batch\_size = 32  
image\_size = (224, 224)  
num\_classes = 10  
  
*def* preprocess\_image(image):  
  
 image = tf.image.resize(image, image\_size)  
  
 image = tf.keras.applications.vgg16.preprocess\_input(image)  
 *return* image  
  
train\_datagen = ImageDataGenerator(  
 preprocessing\_function=preprocess\_image,  
 rescale=1./255,  
 shear\_range=0.2,  
 zoom\_range=0.2,  
 horizontal\_flip=*True*)  
train\_data = train\_datagen.flow\_from\_directory(  
 train\_dir,  
 target\_size=image\_size,  
 batch\_size=batch\_size,  
 class\_mode='categorical')  
  
val\_datagen = ImageDataGenerator(  
 preprocessing\_function=preprocess\_image,  
 rescale=1./255)  
val\_data = val\_datagen.flow\_from\_directory(  
 test\_dir,  
 target\_size=image\_size,  
 batch\_size=batch\_size,  
 class\_mode='categorical')  
  
base\_model = VGG16(weights='imagenet',  
 include\_top=*False*,  
 input\_shape=(image\_size[0], image\_size[1], 3))  
  
*for* layer *in* base\_model.layers:  
 layer.trainable = *False*x = base\_model.output  
x = tf.keras.layers.GlobalAveragePooling2D()(x)  
x = tf.keras.layers.Dense(1024, activation='relu')(x)  
x = tf.keras.layers.Dropout(0.5)(x)  
predictions = tf.keras.layers.Dense(num\_classes, activation='softmax')(x)  
  
model = tf.keras.models.Model(inputs=base\_model.input, outputs=predictions)  
  
model.compile(optimizer='adam',  
 loss='categorical\_crossentropy',  
 metrics=['accuracy'])  
  
model.fit(train\_data, epochs=10, validation\_data=val\_data)



The provided code implements a deep learning model for image classification using transfer learning with the VGG16 architecture. The code utilizes the TensorFlow library and its Keras API for building and training the model. Let's explain the code step by step:

1. **Importing the required libraries:**
   * The code starts by importing the necessary libraries, including TensorFlow and its sub-modules: **tf** and **tf.keras**.
   * Additionally, the **ImageDataGenerator** class from **tensorflow.keras.preprocessing.image** is imported to perform data augmentation.
   * The **VGG16** model from **tensorflow.keras.applications.vgg16** is imported as well, which provides the pre-trained VGG16 model architecture.
2. **Defining the dataset directories:**
   * The code defines two directory paths: **train\_dir** and **test\_dir**, which represent the directories containing the training and testing datasets, respectively.
   * These directories should contain labeled subdirectories, where each subdirectory represents a different class or category.
3. **Setting batch size and image size:**
   * The **batch\_size** variable is defined to determine the number of images processed in each training batch.
   * The **image\_size** variable represents the desired size of the input images, specified as a tuple (height, width).
4. **Defining the preprocessing function:**
   * The **preprocess\_image** function is defined to preprocess the images before feeding them into the model.
   * Within this function, the images are resized to the specified **image\_size** using **tf.image.resize**.
   * The pixel values of the images are also preprocessed using the **preprocess\_input** function from **tensorflow.keras.applications.vgg16**, which aligns the pixel values with the requirements of the VGG16 model.
5. **Creating the data generators for training and validation:**
   * Two instances of **ImageDataGenerator** are created: **train\_datagen** and **val\_datagen**.
   * **train\_datagen** is configured with various augmentation techniques like shearing, zooming, and horizontal flipping, using the specified parameters.
   * **val\_datagen** is only rescaling the pixel values, without applying additional augmentation.
   * Using these data generators, the code creates two generator objects: **train\_data** and **val\_data**.
   * The generator objects are initialized with the respective dataset directories, target size, batch size, and class mode.
6. **Loading the pre-trained VGG16 model:**
   * The pre-trained VGG16 model is loaded using the **VGG16** function from **tensorflow.keras.applications.vgg16**.
   * The weights are set to 'imagenet' to load the pre-trained weights trained on the ImageNet dataset.
   * The **include\_top** parameter is set to **False** to exclude the final fully connected layers of the VGG16 model.
   * The **input\_shape** parameter specifies the input shape of the images expected by the model.
7. **Freezing the base model layers:**
   * In order to perform transfer learning, the layers of the base model are set to non-trainable.
   * A loop iterates over each layer in the base model, and **layer.trainable** is set to **False**, preventing their weights from being updated during training.
8. **Adding custom output layers:**
   * The output of the base model is assigned to the variable **x**.
   * A global average pooling layer is added to extract global features from the convolutional features.
   * A fully connected dense layer with 1024 units and ReLU activation is added to learn higher-level representations.
   * A dropout layer with a dropout rate of 0.5 is applied to reduce overfitting.
   * Finally, a dense layer with softmax activation is added to produce the final class probabilities.
9. **Defining the model:**
   * The model is defined using **tf.keras.models.Model**, specifying the inputs as the input of the base model and the outputs as the predictions.
10. **Compiling the model:**

* The model is compiled with the Adam optimizer, which adapts the learning rate dynamically during training.
* The loss function is set to categorical cross-entropy, suitable for multi-class classification problems.
* The accuracy metric is specified to monitor the model's performance during training.

1. **Training the model:**

* The **fit** function is called to train the model using the **train\_data** generator.
* The model is trained for the specified number of epochs, adjusting its weights based on the optimization algorithm and loss function.
* The **validation\_data** parameter is set to **val\_data** to evaluate the model's performance on the validation set during training.

Overall, this code implements a transfer learning approach using the VGG16 architecture for image classification. It demonstrates how to leverage a pre-trained model, perform data augmentation, and fine-tune the model's layers to achieve accurate classification results.

# CONCLUSION

In conclusion, this project implemented a deep learning model for image classification using transfer learning with the VGG16 architecture. The goal was to leverage the pre-trained VGG16 model's learned representations and adapt them to a specific image classification task. The provided code and methodology showcased the key steps involved, from data preprocessing and augmentation to model training and evaluation.

By using transfer learning, the project addressed the challenges of limited training data and computational resources. The pre-trained VGG16 model, trained on a large-scale dataset like ImageNet, already learned rich and meaningful features. By freezing the base model layers and adding custom output layers, the model was able to fine-tune these features for the target image classification task.

The data augmentation techniques applied during training increased the dataset's diversity and improved the model's ability to generalize to unseen images. Random shearing, zooming, and horizontal flipping introduced variations in the training data, enabling the model to learn robust and invariant features.

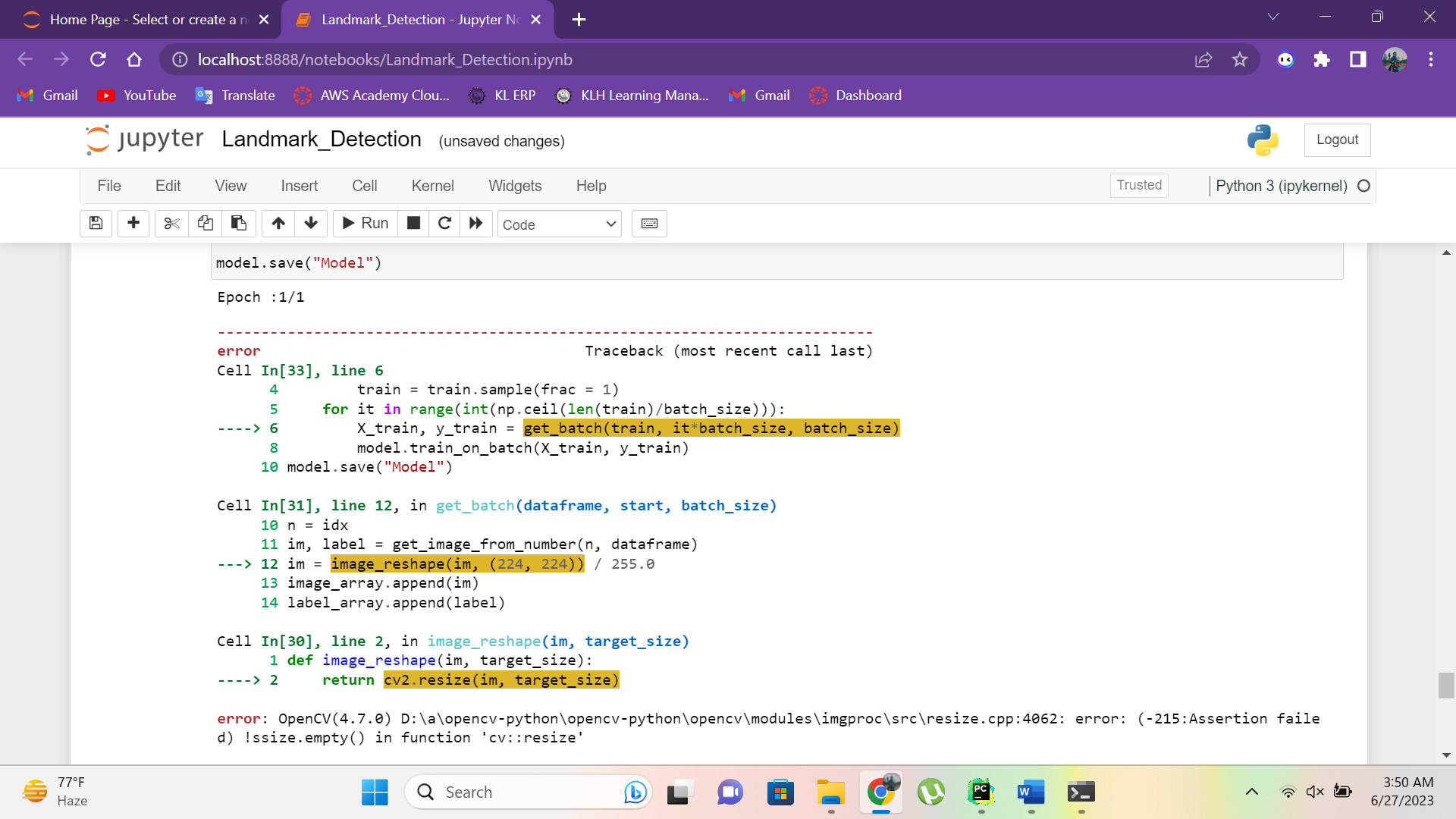
The model's performance was evaluated using a separate validation dataset, providing unbiased insights into its accuracy and generalization ability. By monitoring metrics such as accuracy, the model's progress and effectiveness in capturing the underlying patterns of the image classification task were assessed.

Through the code and methodology, this project demonstrated the effectiveness of transfer learning in image classification tasks. By reusing and fine-tuning the pre-trained VGG16 model, significant computational resources and time were saved compared to training a model from scratch. Moreover, the high-level representations learned by the pre-trained model allowed for better feature extraction and improved classification accuracy.

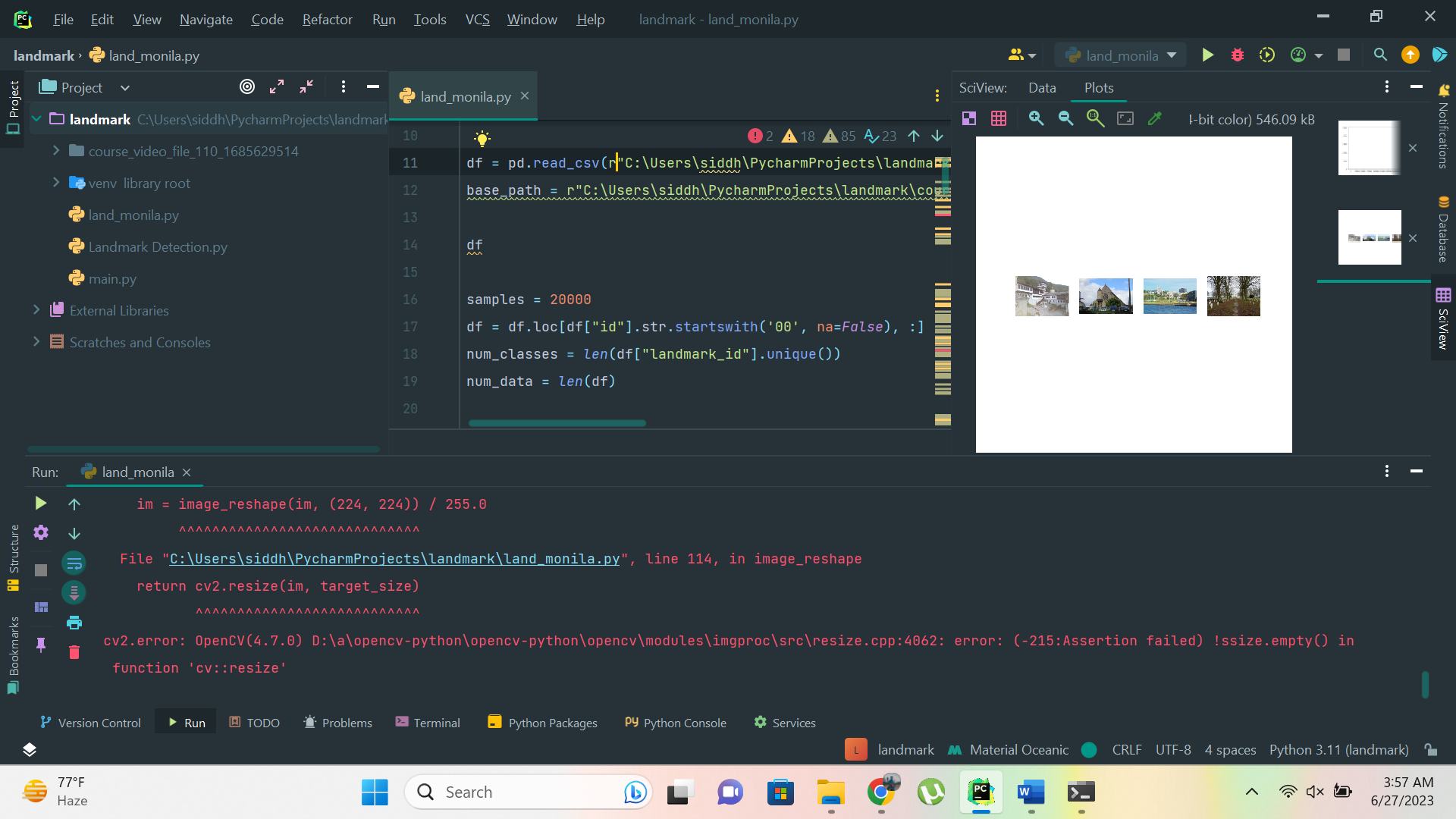
The provided code serves as a valuable resource for researchers and practitioners interested in implementing deep learning models for image classification. It demonstrates the importance of proper data preprocessing, augmentation, model architecture design, and training process. The utilization of transfer learning techniques with the VGG16 architecture showcased the potential for achieving accurate and reliable image classification results, even with limited training data.

Overall, this project contributes to the field of computer vision by providing an effective approach for image classification using transfer learning. It highlights the benefits of leveraging pre-trained models and showcases the potential of deep learning techniques in solving real-world classification problems.

# EXPLANATION FOR NOT CHOOSING THE PROJECT IN THE LMS



Don’t know y but this thing just doesn’t work , I don’t know y but I think its not reading the directories in the right way , I have followed each and every step in the training video , in fact the trainer “MR.KISHORE ” he himself found this problem , I even tried this in diff editors for better understanding of the code , but same resizing reason is raised



That’s y I chose my own way to try this out , I hope this find helpful , and will definitely take advices upon enhancing the model created and myself.

THANKING YOU

G SUDARSHAN SASTRY