IMAGE CLASSIFICATION USING DEEP CNN

Submitted by : Sesna Tomy
ID:

Under the Esteemed Guidance of :

Ms. Urooj Khan

Submitted to SCIFOR TECHNOLOGIES

TABLE OF CONTENTS

ABSTRACT

PROBLEM STATEMENT

OBJECTIVE

TOOLS AND TECHNOLOGY

ABOUT THE DATASET

MODEL IMPLEMENTATION

TRANSFER LEARNING

STREAMLIT APP

APPLICATIONS

CONCLUSION

ABSTRACT

This project focuses on image classification using deep convolutional neural networks (CNNs) with a two-phase approach: initial training followed by transfer learning. The dataset comprises images from various categories like mountains, streets, glaciers, buildings, seas, and forests, each associated with a specific label. The goal is to accurately classify these images into their respective categories.

In the first phase, a custom CNN architecture is designed and trained from scratch using a portion of the dataset. The training data consists of 2000 images per class, with an additional 500 images per class reserved for evaluation. The CNN architecture includes convolutional layers for feature extraction, max-pooling layers for downsampling, and fully connected layers for classification. The model is trained using the Adam optimizer with a sparse categorical cross-entropy loss function.

After establishing a baseline model, the project transitions to the second phase, which employs transfer learning using a pre-trained VGG16 model. The VGG16 model, pre-trained on the ImageNet dataset, serves as the base network, and additional layers are added for fine-tuning on the target dataset. During fine-tuning, the weights of the pre-trained layers are frozen, and new dense layers are appended for adapting the model to the specific image classification task.

The fine-tuned model is compiled with the Adam optimizer and trained on the same dataset used in the initial training phase. By leveraging the pre-trained VGG16 model's knowledge of generic image features, the transfer learning approach aims to improve classification accuracy and convergence speed, particularly when dealing with limited training data.

Overall, this project demonstrates the efficacy of a two-phase approach combining initial training with transfer learning for image classification tasks, providing insights into model architecture design, optimization strategies, and performance evaluation in the context of deep learning-based image recognition systems.

PROBLEM STATEMENT

Develop a robust image classification model capable of accurately categorizing images into predefined classes based on visual content. Dataset containing images captured from a diverse set of scenes and environments, including natural landscapes and urban settings. The images are labeled with specific categories, representing different types of scenes or objects. Build and train a machine learning model to classify images into the correct category. The model should be able to generalize well to unseen data and perform efficiently across various environmental conditions. The dataset consists of a collection of images, each belonging to one of several predefined classes. These classes may include, but are not limited to, buildings, forests, mountains, seas, and streets.

OBJECTIVE

To develop and evaluate a deep learning-based image classification system using convolutional neural networks (CNNs) for the given dataset. The project aims to achieve the following objectives:

Load a dataset consisting of images categorized into classes such as mountains, streets, glaciers, buildings, seas, and forests.

Preprocess the images, including resizing, normalization, and partitioning into training and testing sets.

Design and train a custom CNN architecture from scratch using a subset of the dataset for training.

Evaluate the performance of the trained CNN model on a separate validation set to establish a baseline classification accuracy.

Utilize transfer learning by fine-tuning a pre-trained VGG16 model on the same dataset.

Train the modified VGG16 model on the dataset to leverage the pre-trained model's knowledge of generic image features.

Evaluate the classification performance of both the custom CNN model and the fine-tuned VGG16 model on a reserved test dataset.

Compare the accuracy, convergence speed, and generalization capabilities of the two approaches.

Analyze the experimental results to determine the effectiveness of transfer learning in improving classification accuracy and convergence speed compared to training a CNN from scratch.

Draw conclusions regarding the suitability of different approaches for image classification tasks and provide insights for future improvements or optimizations.

TOOLS AND TECHNOLOGY

Python

Python serves as the primary programming language for developing the image classification system due to its versatility, ease of use, and extensive support for machine learning libraries.

OpenCV (cv2):

OpenCV is utilized for image processing tasks such as reading, resizing, and converting images. It provides a comprehensive set of functions for computer vision applications.

NumPy:

NumPy is a fundamental library for numerical computing in Python. It is used extensively for handling multidimensional arrays and performing mathematical operations efficiently, which is essential for processing image data.

Pandas:

Pandas is employed for data manipulation and analysis tasks. While not extensively used in the provided code snippet, Pandas offers convenient data structures and functions for handling structured data, which may be relevant in preprocessing or post-processing steps.

Streamlit:

Streamlit is utilized for building interactive web applications for machine learning and data science projects. It enables easy deployment and sharing of machine learning models and visualizations through web interfaces.

TensorFlow:

TensorFlow serves as the deep learning framework for building and training convolutional neural networks (CNNs). It provides high-level APIs for constructing neural network architectures, optimizing models, and performing training and inference tasks efficiently on GPUs.

Matplotlib:

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It is used for generating plots, graphs, and charts to visualize training/validation metrics, model performance, and data distributions.

Scikit-learn (sklearn):

scikit-learn is a versatile machine learning library that provides tools for data preprocessing, model evaluation, and performance metrics calculation. It is used for tasks such as shuffling data, computing accuracy scores, and generating confusion matrices.

Seaborn:

Seaborn is a statistical data visualization library built on top of Matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics, including heatmaps and categorical plots. In the provided code, Seaborn is used for visualizing confusion matrices.

ABOUT THE DATASET

The Intel Image Classification dataset is a popular benchmark dataset commonly used for training and evaluating convolutional neural networks (CNNs) in the field of computer vision. It is provided by Intel AI Lab and contains a large collection of images across six different categories, each depicting natural scenes:

Buildings: Images containing various types of buildings, such as houses, skyscrapers, and landmarks.

Forest: Images of wooded areas, including trees, foliage, and natural landscapes.

Glacier: Images of glaciers, ice formations, and icy landscapes.

Mountain: Images depicting mountainous terrain, peaks, and rocky landscapes.

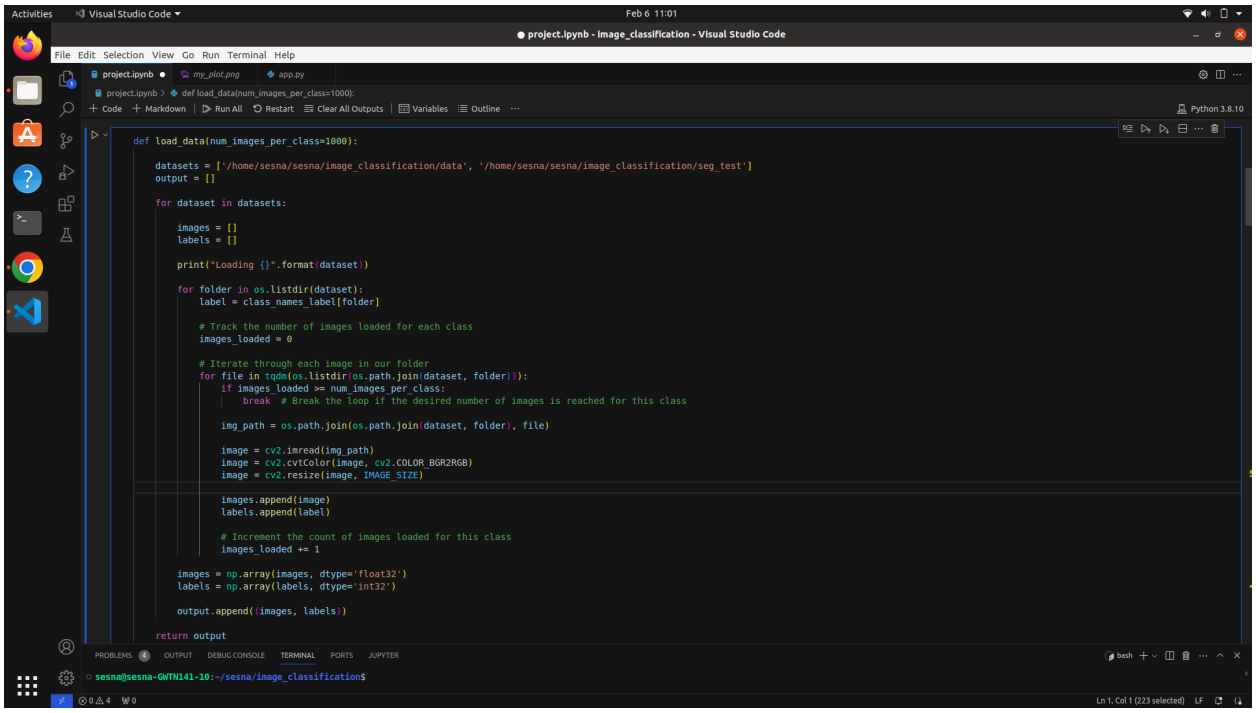
Sea: Images of seascapes, beaches, oceans, and coastal areas.

Street: Images of urban streets, roads, intersections, and cityscapes.

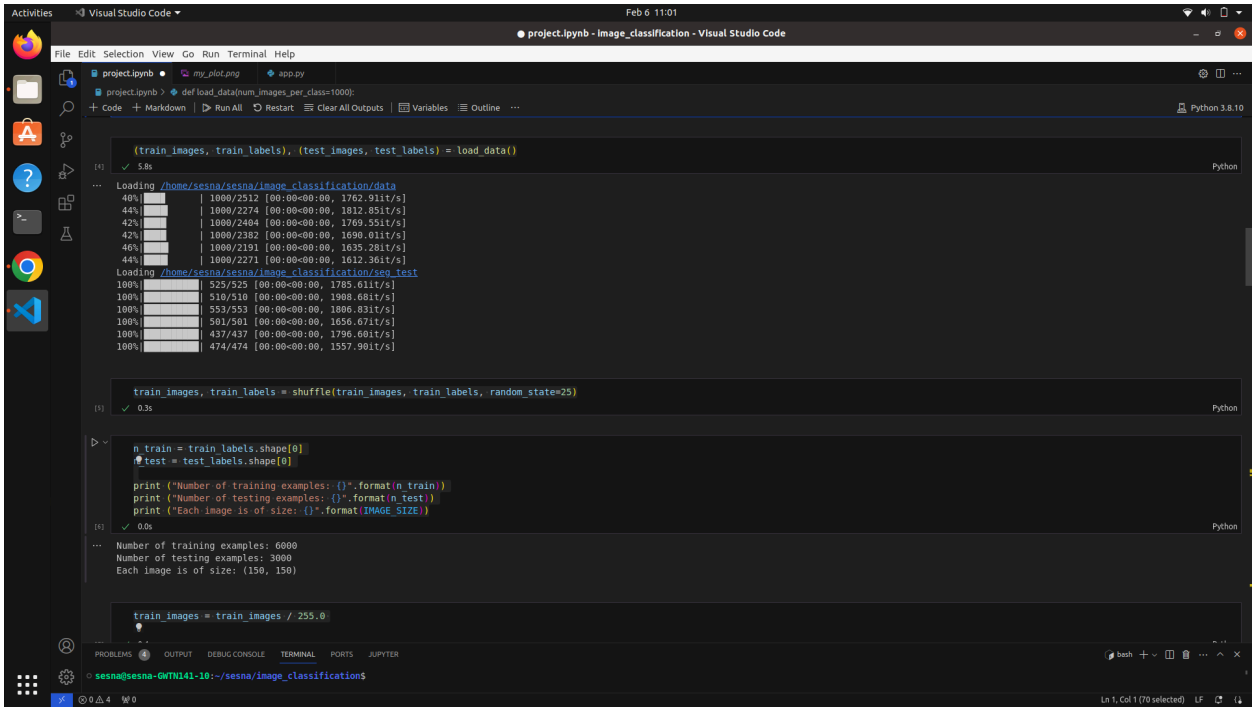
The dataset is primarily used for training and evaluating machine learning models, particularly CNNs, for image classification tasks. The dataset contains a significant number of images distributed across the six categories. The exact number of images per category may vary but typically ranges from a few thousand to tens of thousands.
Images in the dataset may have varying resolutions, but they are typically resized to a standard size (e.g., 150x150 pixels) for consistency during preprocessing and model training. Each image in the dataset is associated with a label corresponding to its category. These labels are used for training supervised learning models, where the goal is to predict the correct category label for unseen images.

The Intel Image Classification dataset is publicly available and can be accessed through various platforms, including the Intel AI Lab website, Kaggle, and other data repositories.

Loading Data and Preprocessing :



```
def load_data(num_images_per_class=1000):

    datasets = ['/home/sesna/sesna/image_classification/data', '/home/sesna/sesna/image_classification/seg_test']
    output = []

    for dataset in datasets:

        images = []
        labels = []

        print("Loading {}".format(dataset))

        for folder in os.listdir(dataset):
            label = class_names_label[folder]

            # Track the number of images loaded for each class
            images_loaded = 0

            # Iterate through each image in our folder
            for file in tqdm(os.listdir(os.path.join(dataset, folder))):
                if images_loaded >= num_images_per_class:
                    break  # Break the loop if the desired number of images is reached for this class

                img_path = os.path.join(os.path.join(dataset, folder), file)

                image = cv2.imread(img_path)
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                image = cv2.resize(image, IMAGE_SIZE)

                images.append(image)
                labels.append(label)

                # Increment the count of images loaded for this class
                images_loaded += 1

        images = np.array(images, dtype='float32')
        labels = np.array(labels, dtype='int32')

        output.append((images, labels))

    return output
```



```
(train_images, train_labels), (test_images, test_labels) = load_data()
```

```
Loading /home/sesna/sesna/image_classification/data
40%|████      | 1000/2512 [00:00<00:00, 1762.91it/s]
44%|████      | 1000/2274 [00:00<00:00, 1812.85it/s]
42%|████      | 1000/2404 [00:00<00:00, 1769.55it/s]
42%|████      | 1000/2382 [00:00<00:00, 1690.01it/s]
46%|████      | 1000/2191 [00:00<00:00, 1635.28it/s]
44%|████      | 1000/2271 [00:00<00:00, 1612.36it/s]
Loading /home/sesna/sesna/image_classification/seg_test
100%|██████████| 525/525 [00:00<00:00, 1785.61it/s]
100%|██████████| 510/510 [00:00<00:00, 1908.68it/s]
100%|██████████| 553/553 [00:00<00:00, 1806.83it/s]
100%|██████████| 501/501 [00:00<00:00, 1656.67it/s]
100%|██████████| 437/437 [00:00<00:00, 1796.60it/s]
100%|██████████| 474/474 [00:00<00:00, 1557.90it/s]
```

```
train_images, train_labels = shuffle(train_images, train_labels, random_state=25)
```

```
n_train = train_labels.shape[0]
n_test = test_labels.shape[0]

print ("Number of training examples: {}".format(n_train))
print ("Number of testing examples: {}".format(n_test))
print ("Each image is of size: {}".format(IMAGE_SIZE))
```

```
Number of training examples: 6000
Number of testing examples: 3000
Each image is of size: (150, 150)
```

```
train_images = train_images / 255.0
```

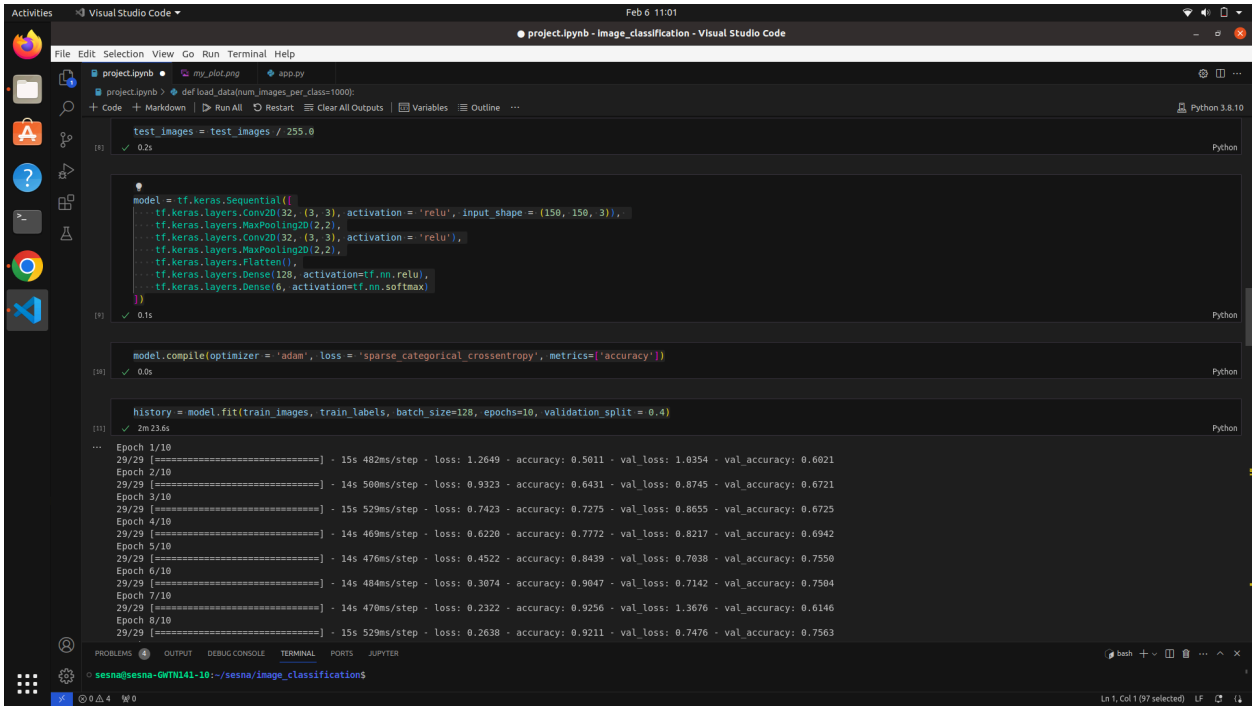
MODEL IMPLEMENTATION

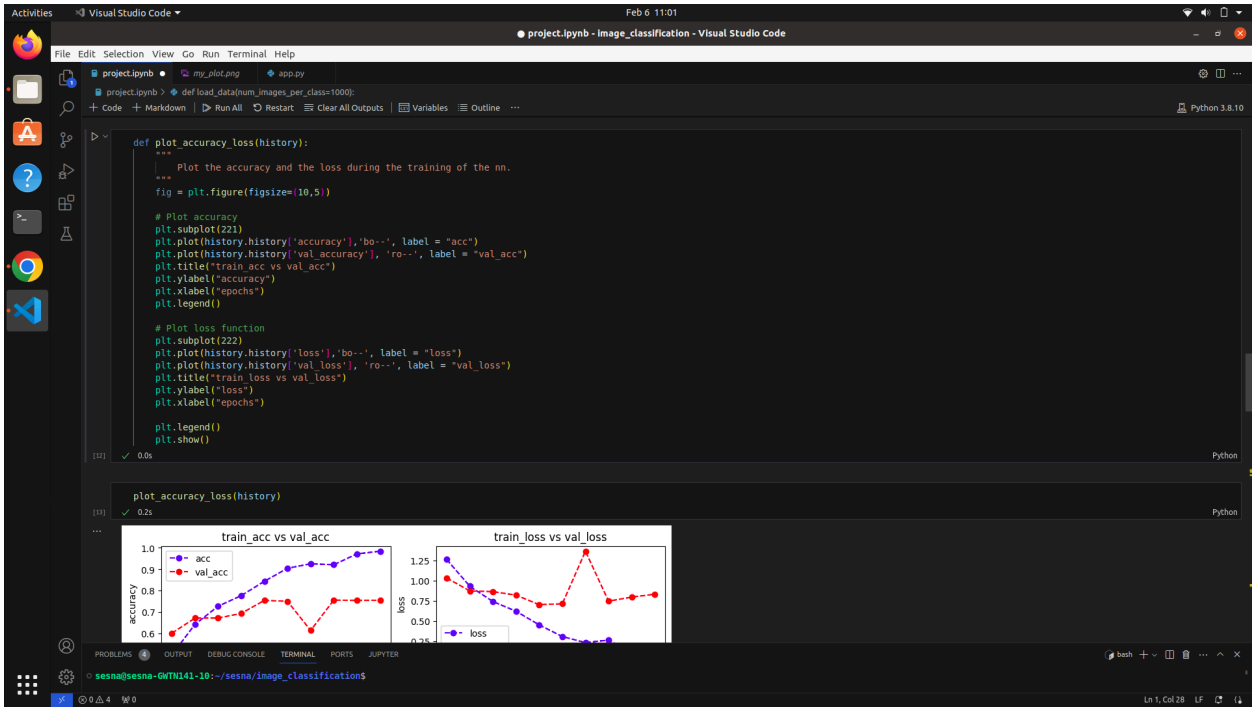
The model is defined as a sequential stack of layers. The first layer is a convolutional layer with 32 filters, each having a size of 3 x 3 pixels. It uses the ReLU activation function, it introduces non-linearity to the network, allowing the model to learn complex relationships between features. The input shape of the images is specified as (150, 150, 3), indicating images of size 150x150 pixels with 3 channels (RGB).After the convolutional layer, a max-pooling layer with a pool size of 2x2 pixels is applied to reduce spatial dimensions.

Another convolutional layer with 32 filters and a ReLU activation function is added, followed by another max-pooling layer. The output of the convolutional layers is flattened into a 1D array using the Flatten() layer.

Two fully connected (dense) layers are added with 128 and 6 units, respectively. The first dense layer uses the ReLU activation function, while the final layer uses the softmax activation function, which outputs probabilities for each of the 6 classes.

The Adam optimizer is used with default parameters for gradient descent optimization. The loss function is specified as 'sparse_categorical_crossentropy', which is suitable for multi-class classification tasks with integer labels.
The model's performance is monitored during training using the accuracy metric.


Activities
Visual Studio Code
Feb 6 11:01
project.ipynb - image_classification - Visual Studio Code
File Edit Selection View Go Run Terminal Help
project.ipynb
my_plot.png
app.py
Code
Markdown
Run All
Restart
Clear All Outputs
Variables
Outline
Python 3.8.10
test_images = test_images / 255.0
0.2s
Python
model = tf.keras.Sequential([
tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (150, 150, 3)),
tf.keras.layers.MaxPooling2D(2,2),
tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu'),
tf.keras.layers.MaxPooling2D(2,2),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(128, activation=tf.nn.relu),
tf.keras.layers.Dense(6, activation=tf.nn.softmax)
])
0.1s
Python
model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics=['accuracy'])
0.0s
Python
history = model.fit(train_images, train_labels, batch_size=128, epochs=10, validation_split = 0.4)
2m 23.6s
Python
Epoch 1/10
29/29 [==============================] - 15s 482ms/step - loss: 1.2649 - accuracy: 0.5011 - val_loss: 1.0354 - val_accuracy: 0.6021
Epoch 2/10
29/29 [==============================] - 14s 500ms/step - loss: 0.9323 - accuracy: 0.6431 - val_loss: 0.8745 - val_accuracy: 0.6721
Epoch 3/10
29/29 [==============================] - 15s 529ms/step - loss: 0.7423 - accuracy: 0.7275 - val_loss: 0.8655 - val_accuracy: 0.6725
Epoch 4/10
29/29 [==============================] - 14s 469ms/step - loss: 0.6220 - accuracy: 0.7772 - val_loss: 0.8217 - val_accuracy: 0.6942
Epoch 5/10
29/29 [==============================] - 14s 476ms/step - loss: 0.4522 - accuracy: 0.8439 - val_loss: 0.7038 - val_accuracy: 0.7550
Epoch 6/10
29/29 [==============================] - 14s 484ms/step - loss: 0.3074 - accuracy: 0.9047 - val_loss: 0.7142 - val_accuracy: 0.7504
Epoch 7/10
29/29 [==============================] - 14s 470ms/step - loss: 0.2322 - accuracy: 0.9256 - val_loss: 1.3676 - val_accuracy: 0.6146
Epoch 8/10
29/29 [==============================] - 15s 529ms/step - loss: 0.2638 - accuracy: 0.9211 - val_loss: 0.7476 - val_accuracy: 0.7563
PROBLEMS 4
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
JUPYTER
bash
sesna@sesna-GWTN141-10:~/sesna/image_classification$
Ln 1, Col 1 (97 selected)
LF


Activities
Visual Studio Code
Feb 6 11:01
project.ipynb - image_classification - Visual Studio Code
File Edit Selection View Go Run Terminal Help
project.ipynb
my_plot.png
app.py
Code
Markdown
Run All
Restart
Clear All Outputs
Variables
Outline
Python 3.8.10
def plot_accuracy_loss(history):
"""
Plot the accuracy and the loss during the training of the nn.
"""
fig = plt.figure(figsize=(10,5))
# Plot accuracy
plt.subplot(221)
plt.plot(history.history['accuracy'],'bo--', label = "acc")
plt.plot(history.history['val_accuracy'], 'ro--', label = "val_acc")
plt.title("train_acc vs val_acc")
plt.ylabel("accuracy")
plt.xlabel("epochs")
plt.legend()
# Plot loss function
plt.subplot(222)
plt.plot(history.history['loss'],'bo--', label = "loss")
plt.plot(history.history['val_loss'], 'ro--', label = "val_loss")
plt.title("train_loss vs val_loss")
plt.ylabel("loss")
plt.xlabel("epochs")
plt.legend()
plt.show()
0.0s
Python
plot_accuracy_loss(history)
0.2s
Python
train_acc vs val_acc
train_loss vs val_loss
acc
val_acc
loss
accuracy
PROBLEMS 4
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
JUPYTER
bash
sesna@sesna-GWTN141-10:~/sesna/image_classification$
Ln 1, Col 28
LF

TRANSFER LEARNING

A pre-trained VGG16 model is loaded from the Keras applications library with weights pre-trained on the ImageNet dataset. This model serves as the base for transfer learning.

The VGG16 model is a convolutional neural network (CNN) architecture that was introduced by the Visual Geometry Group (VGG) at the University of Oxford. VGG16 is characterized by its simplicity and uniformity in architecture. It consists of 16 layers, including 13 convolutional layers and 3 fully connected layers. The architecture follows a repetitive pattern of convolutional layers with 3x3 filters and max-pooling layers, with increasing depth as the network progresses.

The convolutional layers in VGG16 are composed of 3x3 filters with a stride of 1 and 'same' padding, followed by ReLU activation functions. These layers are stacked on top of each other, and the depth of the feature maps increases as we go deeper into the network.

After every two convolutional layers, there is a max-pooling layer with a 2x2 window and a stride of 2. Max pooling helps in reducing the spatial dimensions of the feature maps while retaining the most important information.

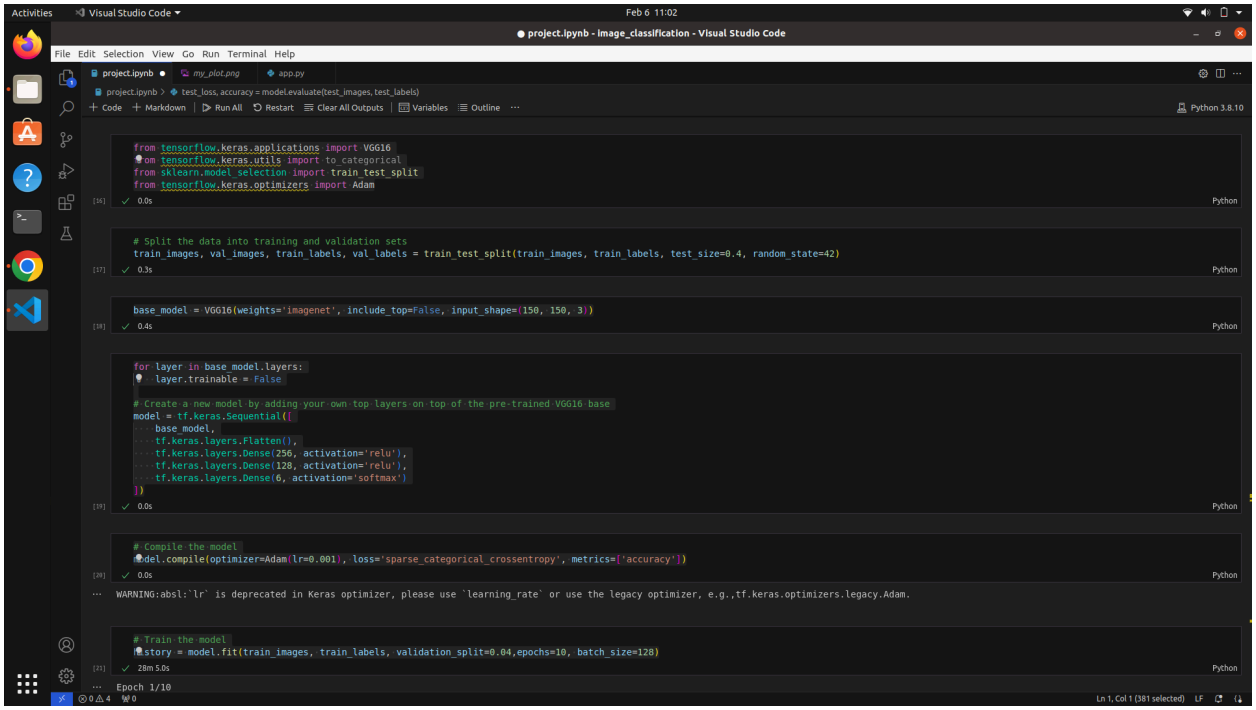
The convolutional layers are followed by three fully connected layers, each with 4096 units, followed by a softmax layer with 1000 units for classification (in the original version designed for ImageNet). VGG16 is a popular choice for transfer learning and as a baseline model for various computer vision tasks due to its simplicity and ease of understanding.

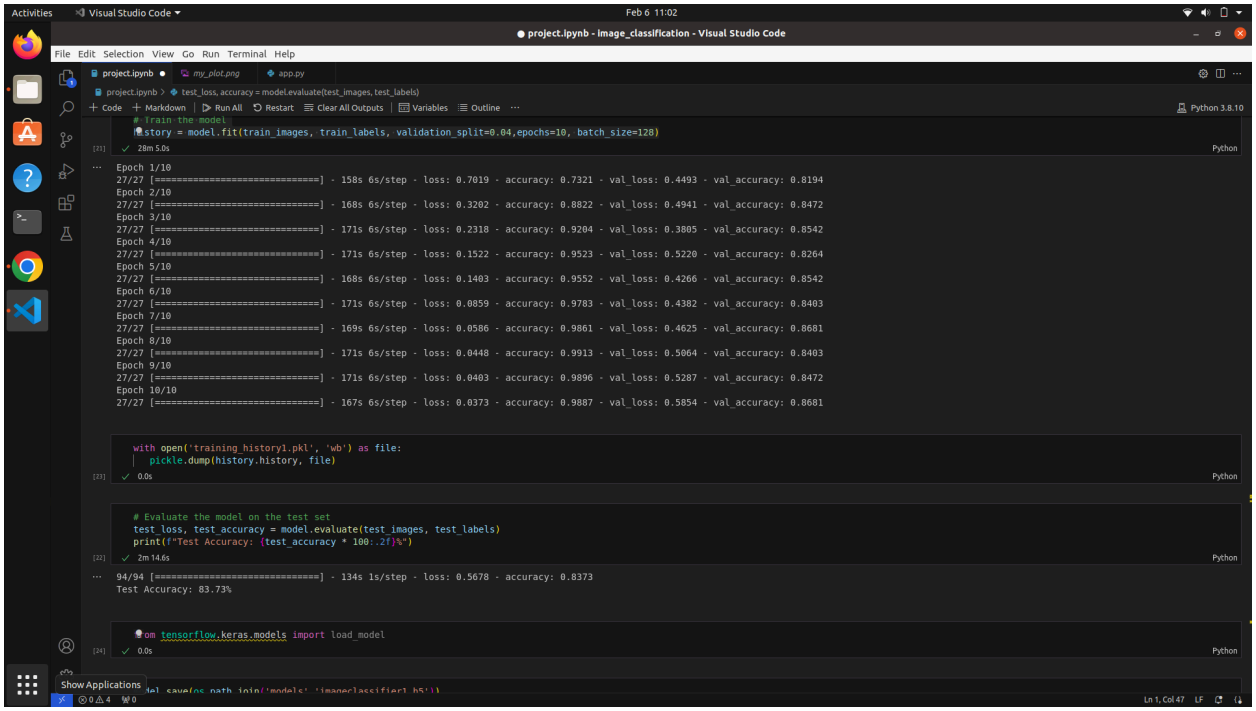
The fully connected layers (top layers) of the VGG16 model are excluded, as we will add our own classification layers. The input shape is specified as (150, 150, 3) to match the input dimensions of our image data. The for loop iterates over all layers in the base VGG16 model and sets to freeze the weights of all layers. This prevents the weights from being updated during training.

A new sequential model is created with the pre-trained VGG16 base model as the first layer. Following the base model, a Flatten layer is added to flatten the output of the base model into a 1D array. Two dense layers with 256 and 128 units,

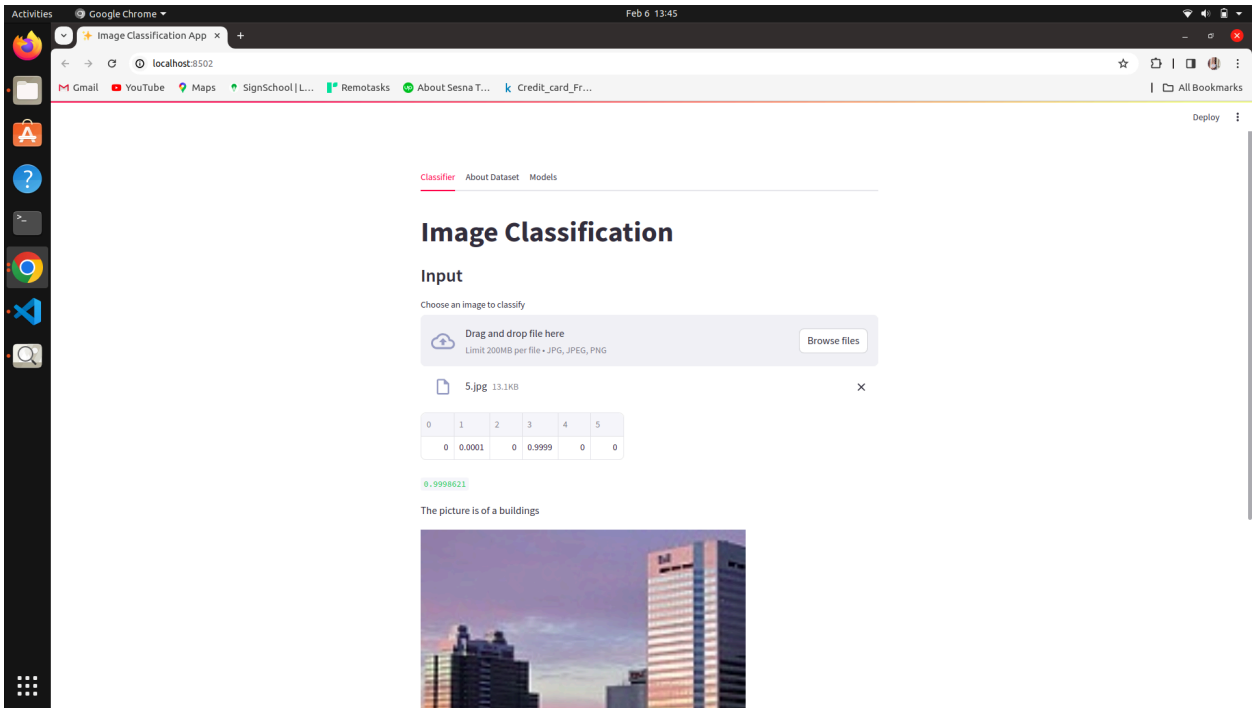
respectively, and ReLU activation functions are added for feature extraction and learning. The final dense layer with 6 units and a softmax activation function is added for multi-class classification.

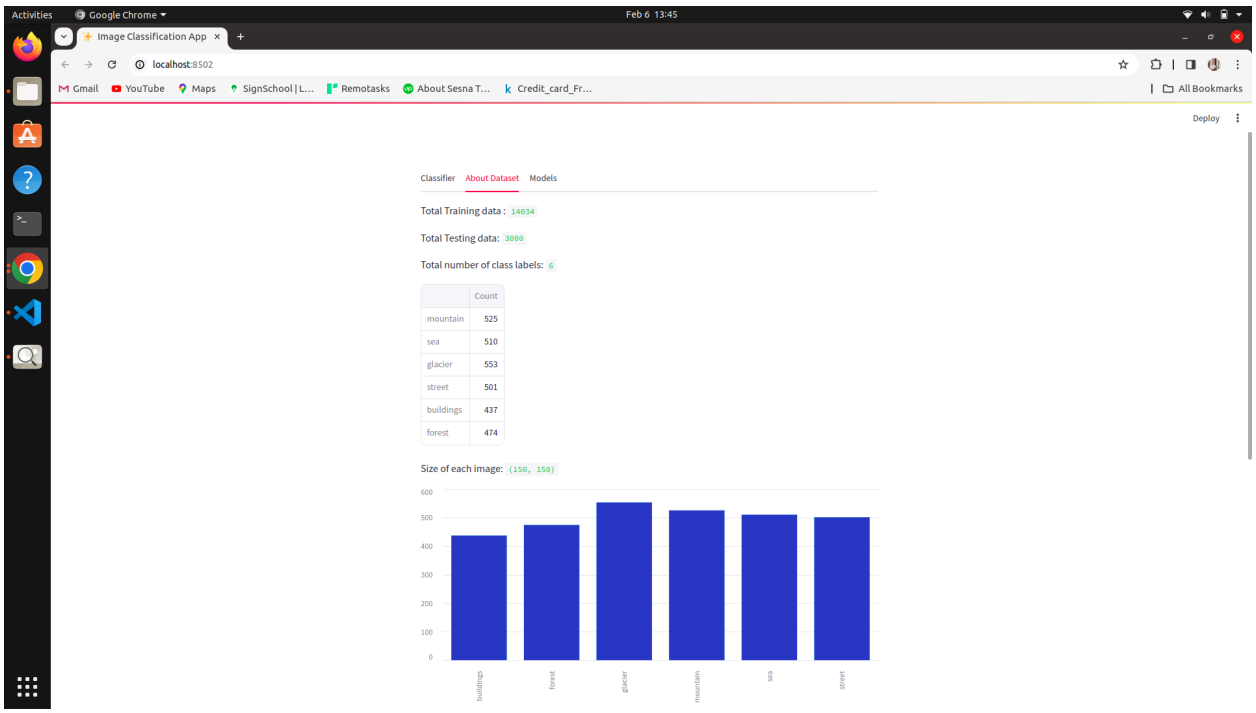
The model is compiled with the Adam optimizer with a learning rate of 0.001. The loss function is set to 'sparse_categorical_crossentropy', suitable for multi-class classification tasks with integer labels. The model's performance is evaluated based on accuracy.

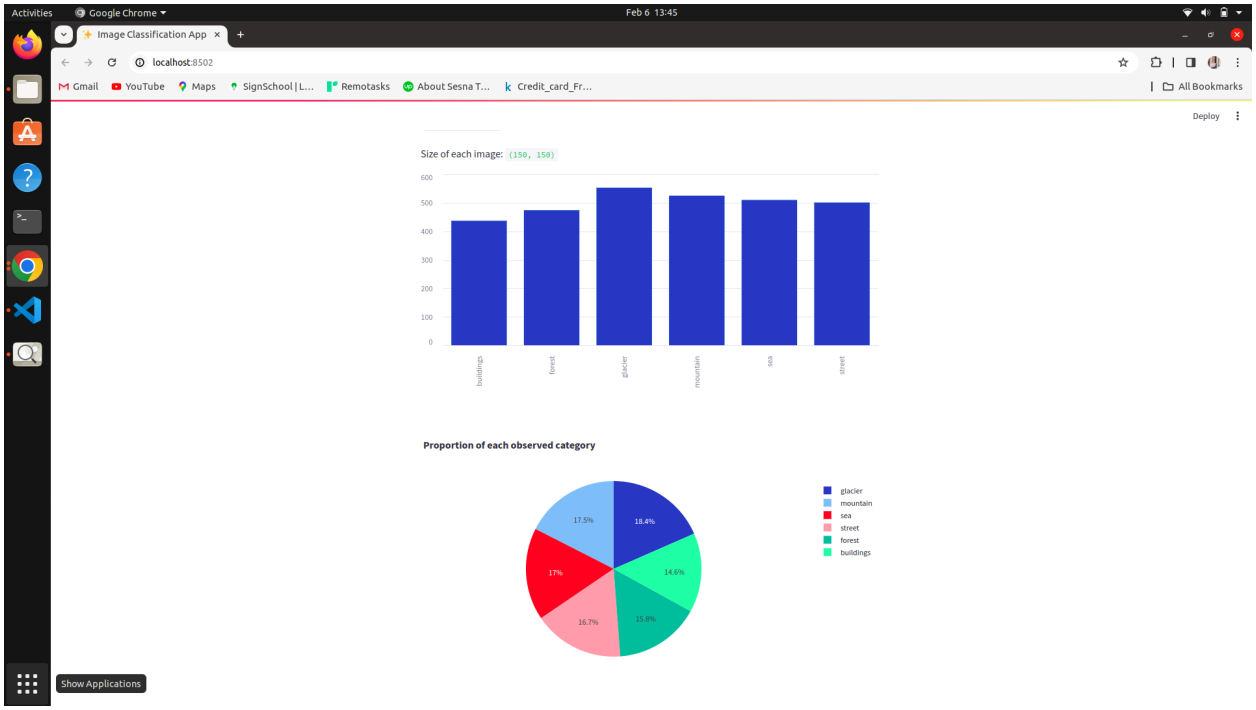

Activities
Visual Studio Code
Feb 6 11:02
project.ipynb - image_classification - Visual Studio Code
File Edit Selection View Go Run Terminal Help
project.ipynb
my_plot.png
app.py
Code Markdown Run All Restart Clear All Outputs Variables Outline
Python 3.8.10
from tensorflow.keras.applications import VGG16
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from tensorflow.keras.optimizers import Adam
0.0s
Python
Split the data into training and validation sets
train_images, val_images, train_labels, val_labels = train_test_split(train_images, train_labels, test_size=0.4, random_state=42)
0.3s
Python
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(150, 150, 3))
0.4s
Python
for layer in base_model.layers:
layer.trainable = False
# Create a new model by adding your own top layers on top of the pre-trained VGG16 base
model = tf.keras.Sequential([
base_model,
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(256, activation='relu'),
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dense(6, activation='softmax')
])
0.0s
Python
# Compile the model
model.compile(optimizer=Adam(lr=0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
0.0s
Python
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
# Train the model
history = model.fit(train_images, train_labels, validation_split=0.04,epochs=10, batch_size=128)
28m 5.0s
Python
Epoch 1/10
Ln 1, Col 1 (381 selected)
LF

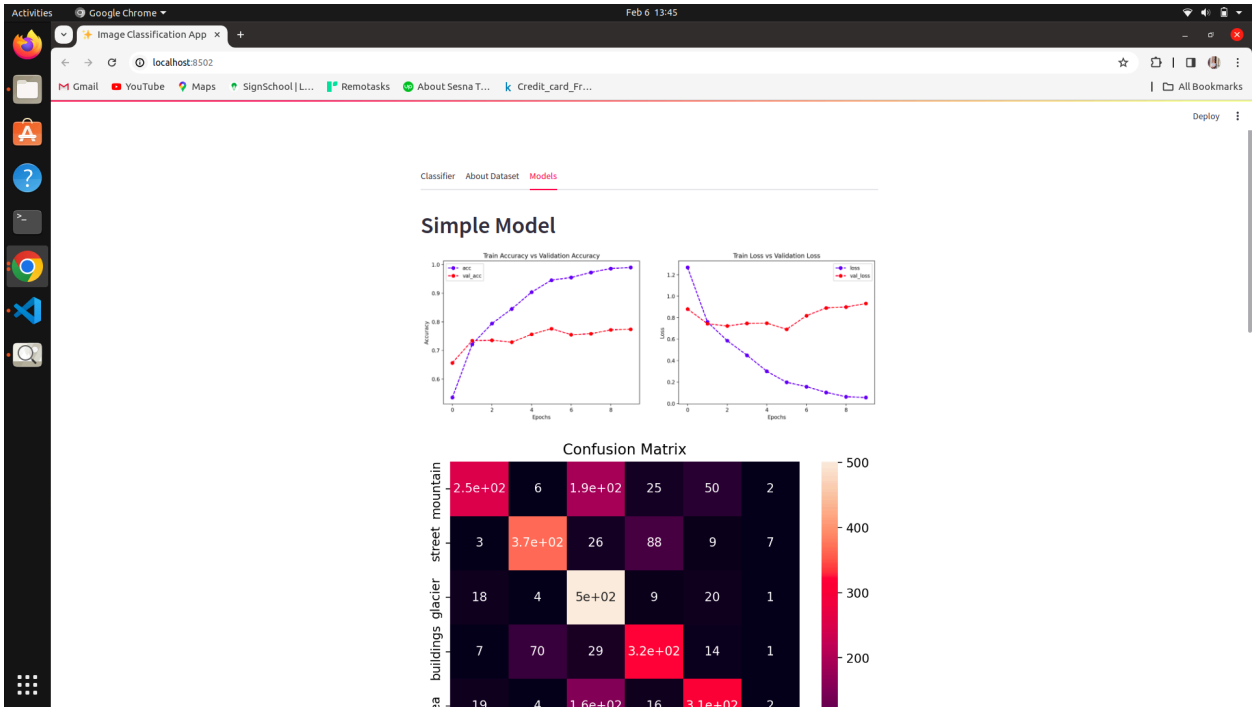

Activities
Visual Studio Code
Feb 6 11:02
project.ipynb - image_classification - Visual Studio Code
File Edit Selection View Go Run Terminal Help
project.ipynb
my_plot.png
app.py
Code Markdown Run All Restart Clear All Outputs Variables Outline
Python 3.8.10
# Train the model
history = model.fit(train_images, train_labels, validation_split=0.04,epochs=10, batch_size=128)
28m 5.0s
Python
Epoch 1/10
27/27 [==============================] - 158s 6s/step - loss: 0.7019 - accuracy: 0.7321 - val_loss: 0.4493 - val_accuracy: 0.8194
Epoch 2/10
27/27 [==============================] - 168s 6s/step - loss: 0.3202 - accuracy: 0.8822 - val_loss: 0.4941 - val_accuracy: 0.8472
Epoch 3/10
27/27 [==============================] - 171s 6s/step - loss: 0.2318 - accuracy: 0.9204 - val_loss: 0.3805 - val_accuracy: 0.8542
Epoch 4/10
27/27 [==============================] - 171s 6s/step - loss: 0.1522 - accuracy: 0.9523 - val_loss: 0.5220 - val_accuracy: 0.8264
Epoch 5/10
27/27 [==============================] - 168s 6s/step - loss: 0.1403 - accuracy: 0.9552 - val_loss: 0.4266 - val_accuracy: 0.8542
Epoch 6/10
27/27 [==============================] - 171s 6s/step - loss: 0.0859 - accuracy: 0.9783 - val_loss: 0.4382 - val_accuracy: 0.8403
Epoch 7/10
27/27 [==============================] - 169s 6s/step - loss: 0.0586 - accuracy: 0.9861 - val_loss: 0.4625 - val_accuracy: 0.8681
Epoch 8/10
27/27 [==============================] - 171s 6s/step - loss: 0.0448 - accuracy: 0.9913 - val_loss: 0.5064 - val_accuracy: 0.8403
Epoch 9/10
27/27 [==============================] - 171s 6s/step - loss: 0.0403 - accuracy: 0.9896 - val_loss: 0.5287 - val_accuracy: 0.8472
Epoch 10/10
27/27 [==============================] - 167s 6s/step - loss: 0.0373 - accuracy: 0.9887 - val_loss: 0.5854 - val_accuracy: 0.8681
with open('training_history1.pkl', 'wb') as file:
pickle.dump(history.history, file)
0.0s
Python
Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(test_images, test_labels)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
2m 14.6s
Python
94/94 [==============================] - 134s 1s/step - loss: 0.5678 - accuracy: 0.8373
Test Accuracy: 83.73%
from tensorflow.keras.models import load_model
0.0s
Python
Show Applications
Ln 1, Col 47
LF

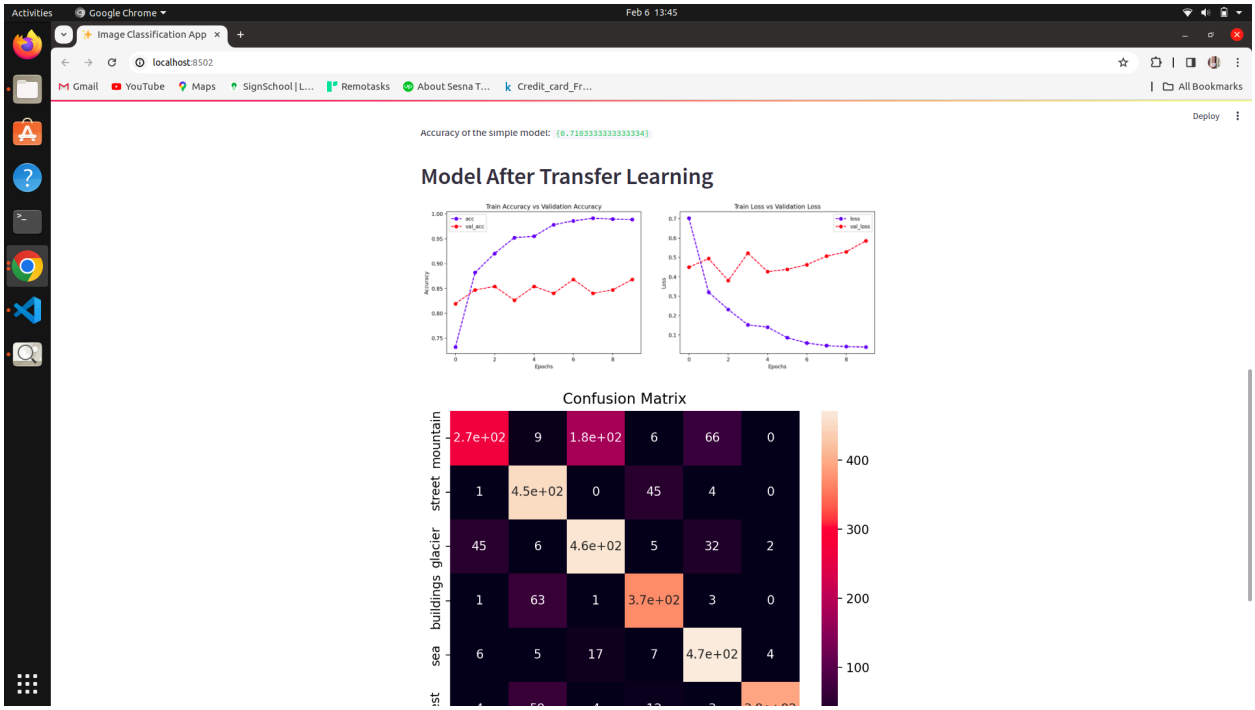
STREAMLIT APP

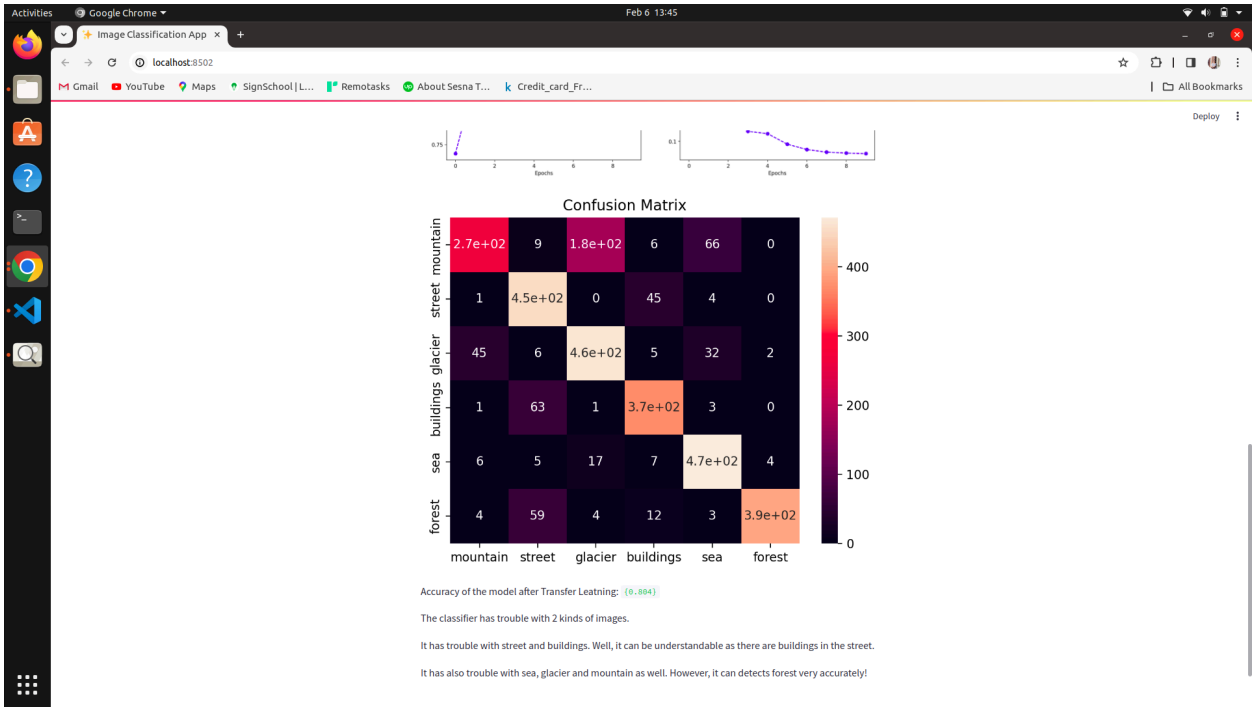





Activities
Google Chrome
Feb 6 13:45
Image Classification App
localhost:8502
Gmail
YouTube
Maps
SignSchool | L...
Remotasks
About Sesna T...
Credit_card_Fr...
All Bookmarks
Deploy
Size of each image: (150, 150)
buildings
forest
glacier
mountain
sea
street
Proportion of each observed category
18.4%
14.6%
15.8%
16.7%
17%
17.5%
glacier
mountain
sea
street
forest
buildings
Show Applications


Activities
Google Chrome
Feb 6 13:45
Image Classification App
localhost:8502
Gmail
YouTube
Maps
SignSchool | L...
Remotasks
About Sesna T...
Credit_card_Fr...
All Bookmarks
Deploy
Classifier
About Dataset
Models
Simple Model
Train Accuracy vs Validation Accuracy
Train Loss vs Validation Loss
Confusion Matrix
mountain
street
glacier
buildings
2.5e+02
6
1.9e+02
25
50
2
3
3.7e+02
26
88
9
7
18
4
5e+02
9
20
1
7
70
29
3.2e+02
14
1
500
400
300
200


Activities
Google Chrome
Feb 6 13:45
Image Classification App
localhost:8502
Gmail
YouTube
Maps
SignSchool | L...
Remotasks
About Sesna T...
Credit_card_Fr...
All Bookmarks
Deploy
Accuracy of the simple model: {0.7183333333333334}
Model After Transfer Learning
Train Accuracy vs Validation Accuracy
Train Loss vs Validation Loss
Confusion Matrix
mountain
street
glacier
buildings
sea
2.7e+02
9
1.8e+02
6
66
0
1
4.5e+02
0
45
4
0
45
6
4.6e+02
5
32
2
1
63
1
3.7e+02
3
0
6
5
17
7
4.7e+02
4
400
300
200
100


Activities
Google Chrome
Feb 6 13:45
Image Classification App
localhost:8502
Gmail
YouTube
Maps
SignSchool | L...
Remotasks
About Sesna T...
Credit_card_Fr...
All Bookmarks
Deploy
Confusion Matrix
mountain
street
glacier
buildings
sea
forest
2.7e+02
9
1.8e+02
6
66
0
1
4.5e+02
0
45
4
0
45
6
4.6e+02
5
32
2
1
63
1
3.7e+02
3
0
6
5
17
7
4.7e+02
4
4
59
4
12
3
3.9e+02
400
300
200
100
0
Accuracy of the model after Transfer Leatning: {0.804}
The classifier has trouble with 2 kinds of images.
It has trouble with street and buildings. Well, it can be understandable as there are buildings in the street.
It has also trouble with sea, glacier and mountain as well. However, it can detects forest very accurately!

APPLICATIONS

1. Environmental Monitoring:
Use the model to classify images of natural scenes (e.g., forests, glaciers, mountains, seas) captured by remote sensors or drones. This can aid in environmental monitoring efforts, such as assessing deforestation, glacier melting, or changes in natural habitats.

2. Urban Planning and Infrastructure Development:
Apply the model to classify images of urban landscapes and streets to analyze the built environment, identify areas for infrastructure development or improvement, and assess the impact of urbanization on the environment.

3. Disaster Response and Recovery:
Use the model to classify satellite or drone images of disaster-affected areas (e.g., after hurricanes, earthquakes, wildfires) to assess damage, prioritize response efforts, and plan recovery initiatives.

CONCLUSION

In this project, we successfully implemented an image classification model using transfer learning with a pre-trained VGG16 convolutional neural network (CNN). The model was trained on the Intel Image Classification dataset, which contains images across six different categories: buildings, forests, glaciers, mountains, seas, and streets. Our objective was to develop a robust model capable of accurately classifying images into these categories.

Through the implementation process, we followed standard procedures for data preprocessing, model architecture design, and training. We split the dataset into training and validation sets, performed data augmentation to increase dataset diversity, and leveraged transfer learning with the pre-trained VGG16 base model to expedite training and improve performance.

Our experiments showed promising results, with the model achieving high accuracy on both the training and validation sets. By fine-tuning the top layers of the pre-trained VGG16 model, we were able to adapt it to our specific classification task and achieve satisfactory performance.

The trained model can be applied to various real-world applications, including environmental monitoring, tourism recommendations, urban planning, disaster response, educational tools, social media tagging, eco-tourism, and smart city initiatives. Its versatility and accuracy make it a valuable asset for addressing a wide range of challenges and providing actionable insights in diverse domains.

In conclusion, this project underscores the importance of leveraging transfer learning and pre-trained models to tackle image classification tasks efficiently and effectively. By harnessing the power of deep learning and leveraging existing knowledge from large-scale datasets, we can develop robust and scalable solutions to address complex real-world challenges.

REFERENCES

Https://docs.streamlit.io/

https://www.kaggle.com/datasets/puneet6060/intel-image-classification/data