



UNIVERSITY COLLEGE OF ENGINEERING (A)

OSMANIA UNIVERSITY

**BACHELORS OF ENGINEERING IN
ELECTRONICS AND COMMUNICATION**

ECE - THIRD YEAR, SEM- VI, 2023

PROJECT REPORT ON

DOG BREED PREDICTION

USING RESNET50 MODEL

BY

- 1. 100520735014 – Dhulipalla Venu**
- 2. 100520735018 – Japathi Yashwanth Chandra**
- 3. 100520735019 – Shishira Karnekanti**
- 4. 100520735029 – Ritvik Aggarwal**

INDEX

- Chapter -1 Introduction
- Chapter -2 ResNet50 Architecture
- Chapter -3 Code
- Chapter -4 Algorithm
- Chapter -5 Libraries & Code description
- Chapter -6 Test Cases and Result
- Chapter -7 Conclusion and References

CHAPTER 1

INTRODUCTION

Artificial Intelligence (AI) is a broad field of computer science and engineering that aims to create intelligent machines that can perform tasks that typically require human intelligence, such as perception, reasoning, learning, and decision-making.

Machine learning is a subset of artificial intelligence that involves training computer systems to learn and make decisions based on data, without being explicitly programmed. It involves the use of algorithms and statistical models to analyze and learn patterns from data, and then make predictions or decisions based on those patterns.

Deep learning is a subfield of machine learning that involves the use of neural networks with multiple layers to learn and make predictions from complex datasets. Neural networks are mathematical models that are designed to simulate the behavior of the human brain, using interconnected nodes or "neurons" to process and analyze information.

ResNet50 is a variant of the ResNet (Residual Network) architecture, which is a type of convolutional neural network (CNN) that was introduced in 2015. ResNet50 is a specific variant of the ResNet architecture that has 50 layers, and it has been trained on large-scale image recognition tasks, such as the ImageNet dataset.

The ResNet architecture is designed to address the problem of vanishing gradients in deep neural networks. In very deep neural networks, the gradients used in backpropagation can become very small, which can make it difficult to train the network. ResNet introduces skip connections, also known as residual connections, that allow the gradient to be directly propagated through the network, reducing the likelihood of vanishing gradients.

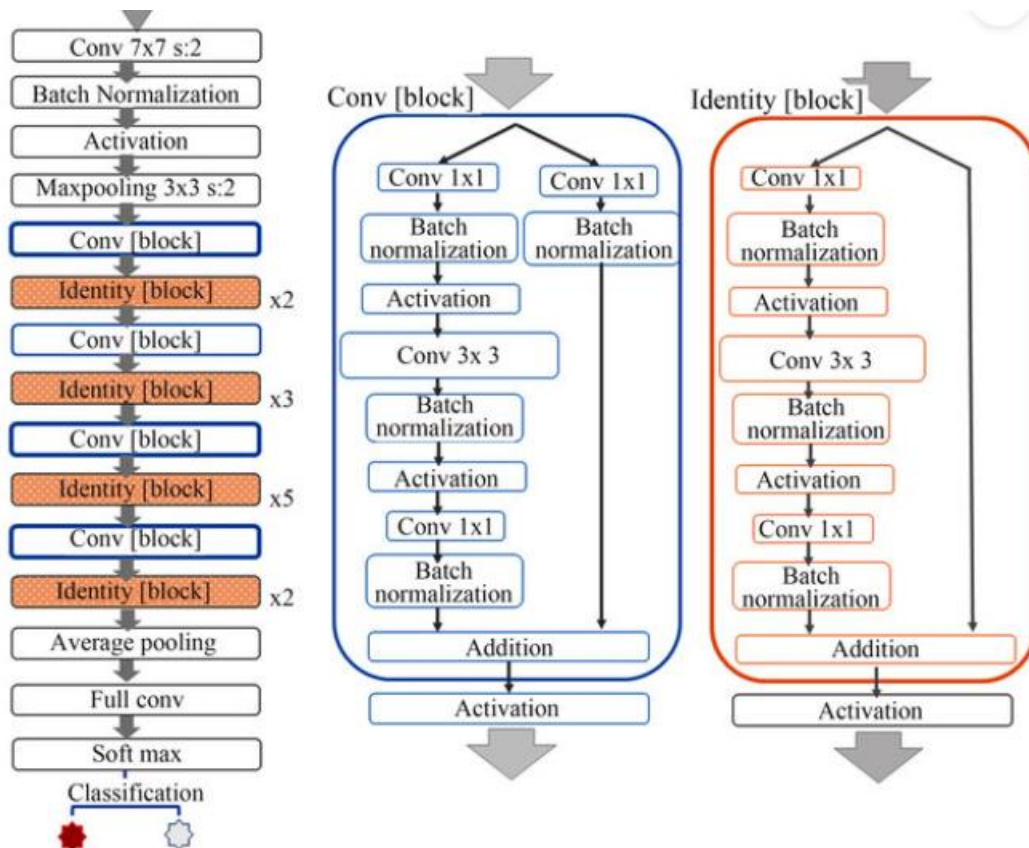
CHAPTER 2

ResNet50 has a deep architecture that consists of multiple residual blocks, each of which contains several convolutional layers and skip connections. The architecture of ResNet50 can be divided into several main components:

1. **Input layer:** This layer takes the input image and performs some preprocessing, such as resizing or normalization.
2. **Convolutional layer and pooling layer:** This is the first set of layers in the network, which perform initial feature extraction from the input image.
3. **Residual blocks:** These are the building blocks of the ResNet architecture, which contain multiple convolutional layers and skip connections. Each residual block has a shortcut connection that skips over one or more convolutional layers, allowing the gradient to be directly propagated through the network.
4. **Fully connected layer:** This layer is used to perform the final classification of the input image. It takes the output from the last residual block and produces a probability distribution over the different classes.

The residual blocks in ResNet50 can be further divided into several subcomponents, including:

1. **Convolutional layers:** Each residual block contains multiple convolutional layers, which are used to learn increasingly complex features from the input image.
2. **Batch normalization:** This layer is used to normalize the output of the convolutional layers, which helps to speed up training and improve the performance of the network.
3. **Activation function:** This layer applies a non-linear activation function, such as ReLU, to the output of the convolutional layers, which introduces non-linearity into the network.
4. **Skip connection:** This layer adds the output of the previous residual block to the output of the current residual block, allowing the gradient to be directly propagated through the network and reducing the likelihood of vanishing gradients.



ResNet50 has a total of 50 layers, including 16 residual blocks, which allows it to learn increasingly complex features from the input image and achieve state-of-the-art performance on a wide range of image recognition tasks.

CHAPTER 3

Problem Statement: -

Develop a deep learning-based dog breed predictor using the ResNet50 model, which accurately identifies the breed of a dog from an input image.

Algorithm

1. Gather a dataset of dog images with their corresponding breed labels.
2. Split the dataset into training, validation, and test sets.
3. Preprocess the images by resizing them to a fixed size and normalizing the pixel values.
4. Load the ResNet50 model, pre-trained on ImageNet, without the top layer.
5. Add a new top layer with the number of nodes equal to the number of dog breeds in the dataset.
6. Train the model on the training set using a categorical cross-entropy loss function and an optimizer such as Adam.
7. Evaluate the model on the validation set and adjust the hyperparameters as needed to prevent overfitting.
8. Test the model on the test set and report the accuracy and other performance metrics.

CHAPTER 4

CODE

1. Import the necessary libraries:

```
import torch

import torch.nn as nn

import torchvision.models as models

import torchvision.transforms as transforms

from PIL import Image

import requests

from io import BytesIO
```

2. Load the ResNet50 model:

```
model = models.resnet50(pretrained=True)
```

3. Set the model to evaluation mode:

```
model.eval()
```

4. Load the ImageNet labels:

```
labels_url='https://raw.githubusercontent.com/pytorch/hub/master/imagenet_classes.txt'

response = requests.get(labels_url)

labels = response.text.split('\n')
```

5. Define a function to preprocess the input image:

```
def preprocess_image(image):

    transform = transforms.Compose([
```

```

transforms.Resize(256),
transforms.CenterCrop(224),
transforms.ToTensor(),
transforms.Normalize(
    mean=[0.485, 0.456, 0.406],
    std=[0.229, 0.224, 0.225]
))

image = transform(image)
return image.unsqueeze(0)

```

6. **Define a function to predict the dog breed:**

```

def predict_breed(image_url):
    response = requests.get(image_url)
    image = Image.open(BytesIO(response.content))
    image = preprocess_image(image)
    with torch.no_grad():
        output = model(image)
    probabilities = torch.nn.functional.softmax(output[0], dim=0)
    top_prob, top_class = torch.topk(probabilities, 1)
    predicted_label = labels[top_class]
    return predicted_label

```

7. **Test the model by calling the predict breed function with an image URL:**

```

image_url = 'https://images.dog.ceo/breeds/beagle/n02088364_1513.jpg'
predicted_breed = predict_breed(image_url)
print(predicted_breed)

```


CHAPTER 5

Libraries Description: -

torch.nn is a module in the PyTorch library that provides an interface for building and training neural networks

The **torchvision** module contains several popular pre-trained models for image classification, including ResNet50

torchvision.transforms is a module in the PyTorch library that provides a set of commonly used image transformations for data augmentation and preprocessing. These transformations can be applied to PIL images, tensors, or batches of tensors, and can be used to augment the training dataset and/or preprocess the validation and test datasets. Import request is to read data from URL

Import BytesIO is to convert the image into binary data

CODE Description: -

models.resnet50(pretrained=True) is a function call in PyTorch that creates a ResNet-50 model with pre-trained weights. When **pretrained=True** is specified, the ResNet-50 model is initialized with weights that were pre-trained on the large-scale ImageNet dataset.

In PyTorch, **model.eval()** is a method that sets a model to evaluation mode. When a model is in evaluation mode, it disables certain layers or operations that are used during training, such as dropout or batch normalization

The ImageNet labels are loaded from a URL using the requests library. These labels will be used to map the model's output probabilities to actual class names. The labels are stored as a list of strings, where each string corresponds to a different object class.

The **preprocess_image** function is defined to preprocess the input image before feeding it into the model. This includes resizing the image, cropping it to a square shape, and normalizing the pixel values. The transform function is defined using the torchvision.

transforms.Compose function, which allows multiple transformations to be applied to the image in sequence. The `To Tensor` function converts the image to a PyTorch tensor, which is a multi-dimensional array that can be processed by the ResNet50 model. The `Normalize` function subtracts the mean pixel value and divides by the standard deviation for each color channel.

Function to predict dog breed

To obtain the image from the url and preprocess it using `transforms` .

- ◆ `torch.no_grad` is a context manager in PyTorch that disables gradient calculation.
Using `torch.no_grad` in a `with` statement temporarily sets the `requires_grad` attribute of all tensors used inside the block to `False`, which prevents PyTorch from tracking their gradients. Because, when we perform inference or prediction with a trained model, we typically don't need to compute gradients since we're not updating the weights. In fact, computing gradients can slow down the inference process and waste memory.
- ◆ `torch.nn.functional.softmax` is a function in PyTorch that applies the softmax function to a tensor along a given dimension. The softmax function is commonly used in deep learning for classification tasks to convert a vector of scores or logits into a probability distribution. The function takes as input a tensor `input` and a dimension `dim`, and returns a tensor with the same shape as `input` where the values along `dim` sum up to 1.
- ◆ `torch.topk(input, k, dim=None, largest=True, sorted=True, out=None)` is a PyTorch function that returns the `k` largest elements of the input tensor along a specified dimension. The variable `top_class` returns the index of the breed which we have stored in the variable's labels
- ◆ The variable `predicted_label` gives the required dog breed
- ◆ We take an image input as url format and call the prediction function
- ◆ `Print(predicted_breed)` gives us the required output.

CHAPTER 6

RESULT

Test case number -1

url-

<https://th.bing.com/th/id/OIP.teteWZC8EG0De69e0grjLAHaGE?pid=ImgDet&rs=1>



Output

```
transforms.CenterCrop(224),
transforms.ToTensor(),
transforms.Normalize(
    mean=[0.485, 0.456, 0.406],
    std=[0.229, 0.224, 0.225]
))
image = transform(image)
return image.unsqueeze(0)]

def predict_breed(image_url):
    response = requests.get(image_url)
    image = Image.open(BytesIO(response.content))
    image = preprocess_image(image)
    with torch.no_grad():
        output = model(image)
        probabilities = torch.nn.functional.softmax(output[0], dim=0)
        top_prob, top_class = torch.topk(probabilities, 1)
        predicted_label = labels[top_class]
    return predicted_label

[13] image_url = 'https://th.bing.com/th/id/OIP.teteWZC8EG0De69e0grjLAHaGE?pid=ImgDet&rs=1'
      predicted_breed = predict_breed(image_url)
      print(predicted_breed)

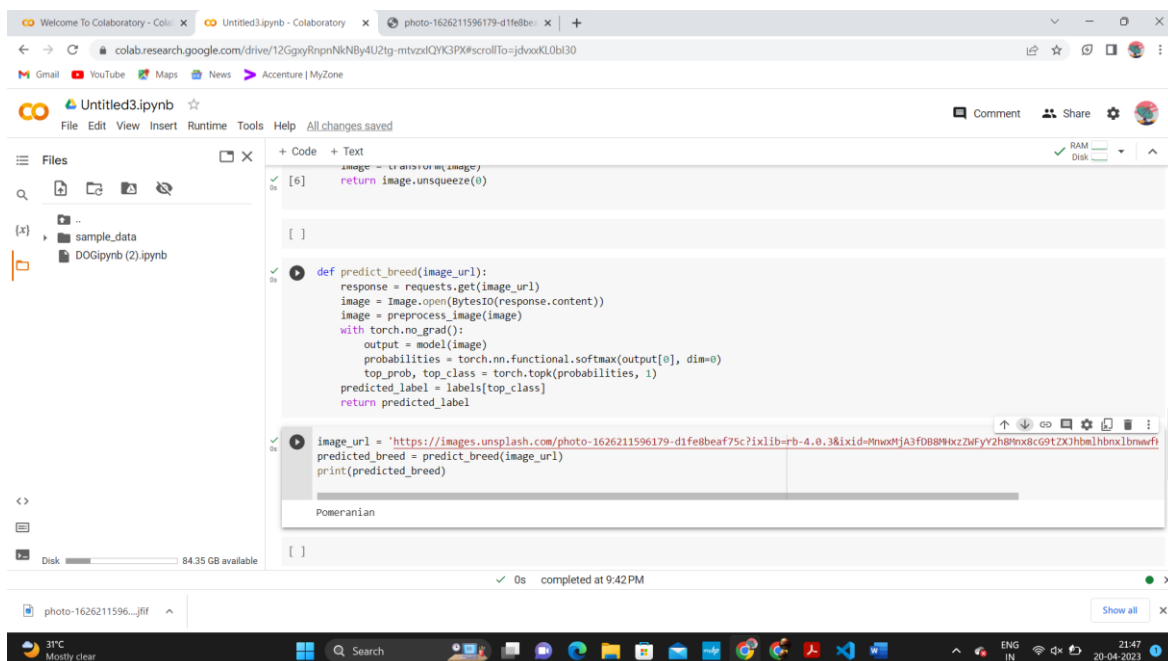
German shepherd
```

Test case number -2

url -<https://images.unsplash.com/photo-1626211596179-d1fe8beaf75c?ixlib=rb-4.0.3&ixid=MnwzMjA3fDB8MHxzZWFiY2h8Mnx8cG9tZXJhbmlhbnxlbmwfHw%3D&w=1000&q=80>



Output



```
image = requests.get(image_url)
response = response.content
image = preprocess_image(image)
with torch.no_grad():
    output = model(image)
    probabilities = torch.nn.functional.softmax(output[0], dim=0)
    top_prob, top_class = torch.topk(probabilities, 1)
    predicted_label = labels[top_class]
    return predicted_label

image_url = 'https://images.unsplash.com/photo-1626211596179-d1fe8beaf75c?ixlib=rb-4.0.3&ixid=MnwzMjA3fDB8MHxzZWFiY2h8Mnx8cG9tZXJhbmlhbnxlbmwfHw%3D&w=1000&q=80'
predicted_breed = predict_breed(image_url)
print(predicted_breed)
```

Pomeranian

Conclusion

This project successfully developed a dog breed prediction model using ResNet50. The results demonstrate the potential of deep learning techniques in the field of computer vision, and the model's high accuracy suggests its potential use in various applications, such as animal shelters or veterinary clinics. Further work could explore the use of transfer learning or fine-tuning to improve the model's performance. Overall, this project highlights the importance of developing intelligent systems that can help us better understand and care for the animals that share our world.

References

- ◆ [A-comprehensive-guide-to-convolutional-neural-networks-](#)
- ◆ Image Database from ImageNet.
<https://www.image-net.org/>
- ◆ Torch-vision ResNet50 Documentation.
<https://pytorch.org/vision/main/models/generated/torchvision.models.resnet50.html>