Vision Vault

A MINI-PROJECT REPORT

18CSC305J - ARTIFICIAL INTELLIGENCE

Submitted by

Hem Swaroop Sai [Reg No: RA2111003010692]

T.M.S.S.ADITYA [Reg No: RA2111003010736]

Hemanth Devarenti [Reg No: RA2111003010726]

Under the Guidance of

Dr. Deepa.R

Associate Professor,Department of Computing Technologies

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Chengalpattu District

MAY 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this B. Tech project report titled "Vision Vault" is the bonafide work of Hem Swaroop Sai [Reg No: RA2111003010692], T.M.S.S.ADITYA [Reg No: RA2111003010736] and Hemanth Devarenti [Reg No: RA2111003010726] who carried out the project work under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form part of any other thesis or dissertation based on which a degree or award was conferred on an earlier occasion for this or any other candidate.

SIGNATURE

Faculty Name: Dr. Deepa.R

Designation: Assistant Professor

Department: Computing Technologies

ABSTRACT

While the Transformer architecture has solidified its position as the go-to for natural language processing tasks, its adaptation to computer vision has been somewhat limited. However, the emergence of the Vision Transformer (ViT) has catalysed a paradigm shift in this regard. Unlike conventional approaches in computer vision, where attention mechanisms are either integrated with convolutional networks or selectively replace certain components while maintaining the overall convolutional structure, ViT represents a departure by achieving remarkable performance in image recognition while demanding significantly fewer computational resources during training. ViT's efficacy stems from its innovative utilization of self-attention mechanisms, which empower the model to efficiently capture long-range dependencies and contextual information within images. This ability not only enhances the model's understanding of visual content but also improves its generalization capabilities across diverse datasets. Moreover, the attention-based approach adopted by ViT contributes to its interpretability, providing insights into the key features that drive its predictions. Such interpretability is crucial in scenarios where understanding the decision-making process of the model is paramount, facilitating trust and enabling more informed decision-making.A standout advantage of ViT lies in its reduced computational overhead, making it an attractive option for training on large-scale datasets and deployment in resource-constrained environments. This efficiency is further amplified by ViT's parallelizable architecture, which capitalizes on modern hardware accelerators to expedite both inference and training times. Consequently, ViT not only offers superior performance in image recognition tasks but also presents a more accessible and scalable solution, bridging the gap between advanced computer vision models and practical real-world applications.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

VIT Vision Transformer

NLP - Natural language processing

DETR - Detection Transformer

CNN - Convolutional Neural Network

UniVLM - Unified Vision-Language Pre-training

DL - Deep Learning

AI – Artificial intelligence

CHAPTER 1

INTRODUCTION

The adoption of self-attention-based architectures, notably Transformers, as the preferred model in natural language processing (NLP), has prompted exploration into their application for image-related tasks such as detection and classification. Drawing inspiration from the remarkable scaling achievements of Transformers in NLP, researchers have ventured into directly applying a standard Transformer architecture to images with minimal alterations.

While the Transformer architecture has become the de-facto standard for natural language processing tasks, its applications to computer vision remain limited. In vision, attention is either applied in conjunction with convolutional networks or used to replace certain components of convolutional networks while keeping their overall structure in place. We show that this reliance on CNNs is not necessary and a pure transformer applied directly to sequences of image patches can perform very well on image classification tasks. When trained on large amounts of data and transferred to multiple mid-sized or small image recognition benchmarks (ImageNet, CIFAR-100, VTAB, etc.), Vision Transformer (ViT) attains excellent results compared to state-of-the-art convolutional networks while requiring substantially fewer computational resources to train.

As researchers delve deeper into refining and optimising Transformer-based approaches for image processing applications, we can anticipate significant strides in the development of AIdriven solutions for image detection and classification tasks, unlocking new capabilities and improving performance across various domains and industries.

PROBLEM STATEMENT

Our vision is to create an efficient and automated system that not only classifies images but also facilitates seamless management and retrieval based on their visual content. This transformative approach, inspired by the success of self-attention-based architectures like Transformers, promises to revolutionize how we interact with and organize visual data, empowering users to navigate and utilise image collections with unprecedented ease and accuracy

CHAPTER 2

LITERATURE SURVEY

Transformers have made significant strides in image recognition, leveraging their effectiveness in capturing long-range dependencies and contextual information from natural language processing (NLP). Notable architectures include the Vision Transformer (ViT) for classification tasks, the Detection Transformer (DETR) for object detection, and the Unified Vision-Language Pre-training (UniVLM) for multimodal tasks.

Efforts have been made to scale Transformers to large datasets and high resolutions, as seen in the BigGAN-Transformer and Swin Transformer, focusing on efficiency and performance improvements. Transfer learning with Transformers, such as the FineTuned Vision Transformer (ViT-B/32), has shown promising results in reducing parameters and achieving competitive performance.

Despite their success, challenges remain in handling spatial information and maintaining computational efficiency. Future directions include exploring hybrid architectures and improved attention mechanisms to address these challenges. Overall, Transformers offers a versatile and powerful approach to image recognition, with ongoing research expected to drive further advancements in computer vision.

CHAPTER 3

System Architecture and Design

Fig-3.1

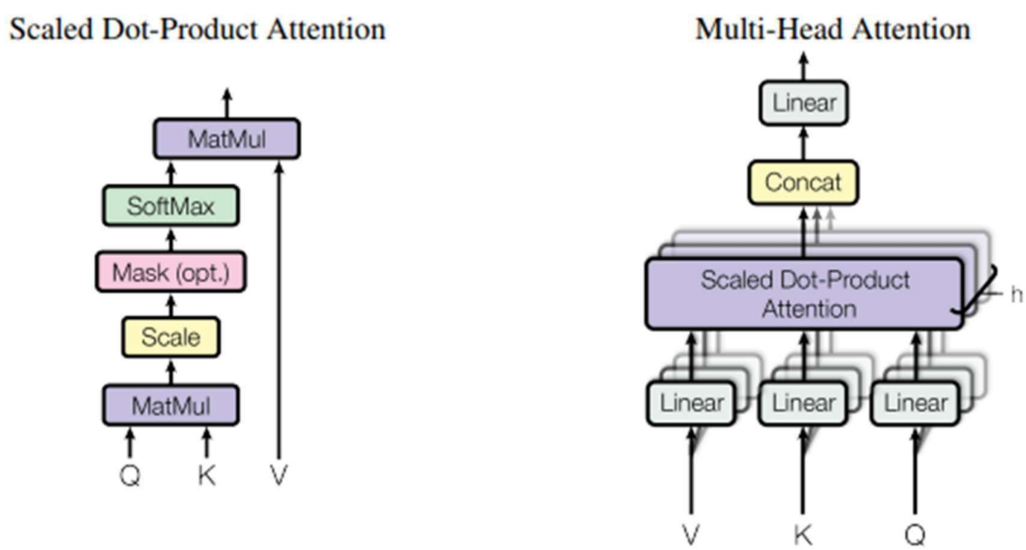
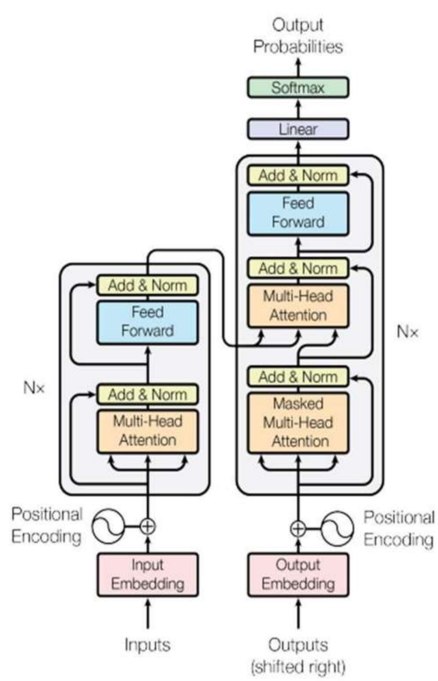


Fig-3.2

Architecture



DESCRIPTION OF COMPONENTS

The text in the diagram refers to two main parts: scaled dot-product attention and multi head attention.

Scaled Dot-Product Attention: This is the fundamental building block of multi-head attention. It calculates a score for each element (word) in the input sequence, indicating how relevant it is to the current word being processed. Mathematically, it multiplies a query vector (WQ) with a key vector (WK) from each element in the sequence, then divides by the square root of the dimension of the key vector and applies a SoftMax function. The SoftMax function makes these scores sum to 1, which allows them to be interpreted as probabilities. Finally, these scores are multiplied by a value vector (WV) from each element, and the resulting vectors are summed to create the output vector.

Multi-Head Attention: This mechanism uses multiple scaled dot product attention layers in parallel, allowing the model to focus on different aspects of the input sequence. Each attention head learns its own set of weights to attend to different parts of the input. The outputs from each attention head are then concatenated and fed to a linear layer. Here is a simplified breakdown of the multi-head attention process:

Input Processing: The input text is embedded into numerical vectors (Q, K, V).

Scaled Dot-Product Attention: Multiple attention heads perform independent attention calculations using the scaled dot-product attention mechanism.

Concatenation: The outputs from each attention head are combined.

Linear Layer: The combined outputs are transformed by a final linear layer.

CHAPTER 4

CODING AND TESTING

Fig-4.1

Fetching CIFAR 10 dataset

```python
"""
A series of helper functions used throughout the course.

If a function gets defined once and could be used over and over, it'll go in here.
"""
import torch
import matplotlib.pyplot as plt
import numpy as np

from torch import nn
import os
import zipfile
from pathlib import Path
import requests
import os



# Plot linear data or training and test and predictions (optional)
def plot_predictions(
    train_data, train_labels, test_data, test_labels, predictions=None
):
    """
  Plots linear training data and test data and compares predictions.
  """
    plt.figure(figsize=(10, 7))

    # Plot training data in blue
    plt.scatter(train_data, train_labels, c="b", s=4, label="Training data")

    # Plot test data in green
    plt.scatter(test_data, test_labels, c="g", s=4, label="Testing data")

    if predictions is not None:
        # Plot the predictions in red (predictions were made on the test data)
        plt.scatter(test_data, predictions, c="r", s=4, label="Predictions")

    # Show the legend
    plt.legend(prop={"size": 14})
```

```python
# Plot loss curves of a model
def plot_loss_curves(results):
    """Plots training curves of a results dictionary.

    Args:
        results (dict): dictionary containing list of values, e.g.
            {"train_loss": [...],
             "train_acc": [...],
             "test_loss": [...],
             "test_acc": [...]}
    """
    loss = results["train_loss"]
    test_loss = results["test_loss"]

    accuracy = results["train_acc"]
    test_accuracy = results["test_acc"]

    epochs = range(len(results["train_loss"]))

    plt.figure(figsize=(15, 7))

    # Plot loss
    plt.subplot(1, 2, 1)
    plt.plot(epochs, loss, label="train_loss")
    plt.plot(epochs, test_loss, label="test_loss")
    plt.title("Loss")
    plt.xlabel("Epochs")
    plt.legend()

    # Plot accuracy
    plt.subplot(1, 2, 2)
    plt.plot(epochs, accuracy, label="train_accuracy")
    plt.plot(epochs, test_accuracy, label="test_accuracy")
    plt.title("Accuracy")
    plt.xlabel("Epochs")
    plt.legend()
```

```python
# Download CIFAR-10 dataset
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # normalize images
])

train_set = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
test_set = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)

# CIFAR-10 class names
class_names = [
    'airplane', 'automobile', 'bird', 'cat', 'deer',
    'dog', 'frog', 'horse', 'ship', 'truck'
]

# Create directories for train and test data
train_dir = './data/train'
test_dir = './data/test'

if not os.path.exists(train_dir):
    os.makedirs(train_dir)

if not os.path.exists(test_dir):
    os.makedirs(test_dir)

# Organize train set into folders
for i, (img, label) in enumerate(train_set):
    class_name = class_names[label]
    class_dir = os.path.join(train_dir, class_name)
    if not os.path.exists(class_dir):
        os.makedirs(class_dir)
    img_path = os.path.join(class_dir, f'image_{i}.png')
    torchvision.utils.save_image(img, img_path)

# Organize test set into folders
for i, (img, label) in enumerate(test_set):
    class_name = class_names[label]
    class_dir = os.path.join(test_dir, class_name)
    if not os.path.exists(class_dir):
        os.makedirs(class_dir)
    img_path = os.path.join(class_dir, f'image_{i}.png')
    torchvision.utils.save_image(img, img_path)
```

Fig 4.2

Importing going_moduler from pytorch

```python
# Continue with regular imports
import matplotlib.pyplot as plt
import torch
import torchvision

from torch import nn
from torchvision import transforms

# Try to get torchinfo, install it if it doesn't work
try:
    from torchinfo import summary
except:
    print("[INFO] Couldn't find torchinfo... installing it.")
    !pip install -q torchinfo
    from torchinfo import summary

# Try to import the going_modular directory, download it from GitHub if it doesn't work
try:
    from going_modular.going_modular import data_setup, engine
except:
    # Get the going_modular scripts
    print("[INFO] Couldn't find going_modular scripts... downloading them from GitHub.")
    !git clone https://github.com/mrdbourke/pytorch-deep-learning
    !mv pytorch-deep-learning/going_modular .
    !rm -rf pytorch-deep-learning
    from going_modular.going_modular import data_setup, engine
```

Python

Fig 4.3

Summary of the model

```
============================================================================================================================================
Layer (type (var_name))                                      Input Shape          Output Shape         Param #              Trainable
============================================================================================================================================
VisionTransformer (VisionTransformer)                        [32, 3, 224, 224]    [32, 10]             768                  Partial
├─Conv2d (conv_proj)                                         [32, 3, 224, 224]    [32, 768, 14, 14]    (590,592)            False
├─Encoder (encoder)                                          [32, 197, 768]       [32, 197, 768]       151,296              False
│    └─Dropout (dropout)                                     [32, 197, 768]       [32, 197, 768]       --                   --
│    └─Sequential (layers)                                   [32, 197, 768]       [32, 197, 768]       --                   False
│    │    └─EncoderBlock (encoder_layer_0)                   [32, 197, 768]       [32, 197, 768]       (7,087,872)          False
│    │    └─EncoderBlock (encoder_layer_1)                   [32, 197, 768]       [32, 197, 768]       (7,087,872)          False
│    │    └─EncoderBlock (encoder_layer_2)                   [32, 197, 768]       [32, 197, 768]       (7,087,872)          False
│    │    └─EncoderBlock (encoder_layer_3)                   [32, 197, 768]       [32, 197, 768]       (7,087,872)          False
│    │    └─EncoderBlock (encoder_layer_4)                   [32, 197, 768]       [32, 197, 768]       (7,087,872)          False
│    │    └─EncoderBlock (encoder_layer_5)                   [32, 197, 768]       [32, 197, 768]       (7,087,872)          False
│    │    └─EncoderBlock (encoder_layer_6)                   [32, 197, 768]       [32, 197, 768]       (7,087,872)          False
│    │    └─EncoderBlock (encoder_layer_7)                   [32, 197, 768]       [32, 197, 768]       (7,087,872)          False
│    │    └─EncoderBlock (encoder_layer_8)                   [32, 197, 768]       [32, 197, 768]       (7,087,872)          False
│    │    └─EncoderBlock (encoder_layer_9)                   [32, 197, 768]       [32, 197, 768]       (7,087,872)          False
│    │    └─EncoderBlock (encoder_layer_10)                  [32, 197, 768]       [32, 197, 768]       (7,087,872)          False
│    │    └─EncoderBlock (encoder_layer_11)                  [32, 197, 768]       [32, 197, 768]       (7,087,872)          False
│    └─LayerNorm (ln)                                        [32, 197, 768]       [32, 197, 768]       (1,536)              False
├─Linear (heads)                                             [32, 768]            [32, 10]             7,690                True
============================================================================================================================================
Total params: 85,806,346
Trainable params: 7,690
...
Input size (MB): 19.27
Forward/backward pass size (MB): 3330.74
Params size (MB): 229.22
Estimated Total Size (MB): 3579.23
============================================================================================================================================
```

Fig 4.4

Training data

```
Epoch: 1 | train_loss: 0.6807 | train_acc: 0.7694 | test_loss: 0.6000 | test_acc: 0.7924
Epoch: 2 | train_loss: 0.5592 | train_acc: 0.8073 | test_loss: 0.5824 | test_acc: 0.7995
Epoch: 3 | train_loss: 0.5323 | train_acc: 0.8165 | test_loss: 0.5618 | test_acc: 0.8069
Epoch: 4 | train_loss: 0.5165 | train_acc: 0.8211 | test_loss: 0.5626 | test_acc: 0.8058
Epoch: 5 | train_loss: 0.5069 | train_acc: 0.8243 | test_loss: 0.5589 | test_acc: 0.8118
Epoch: 6 | train_loss: 0.5002 | train_acc: 0.8277 | test_loss: 0.5687 | test_acc: 0.8075
Epoch: 7 | train_loss: 0.4949 | train_acc: 0.8294 | test_loss: 0.5699 | test_acc: 0.8029
Epoch: 8 | train_loss: 0.4912 | train_acc: 0.8298 | test_loss: 0.5657 | test_acc: 0.8076
Epoch: 9 | train_loss: 0.4869 | train_acc: 0.8322 | test_loss: 0.5688 | test_acc: 0.8079
Epoch: 10 | train_loss: 0.4836 | train_acc: 0.8344 | test_loss: 0.5690 | test_acc: 0.8055
Epoch: 11 | train_loss: 0.4827 | train_acc: 0.8322 | test_loss: 0.5654 | test_acc: 0.8071
Epoch: 12 | train_loss: 0.4813 | train_acc: 0.8350 | test_loss: 0.5702 | test_acc: 0.8058
Epoch: 13 | train_loss: 0.4796 | train_acc: 0.8343 | test_loss: 0.5760 | test_acc: 0.8051
Epoch: 14 | train_loss: 0.4775 | train_acc: 0.8339 | test_loss: 0.5764 | test_acc: 0.8019
Epoch: 15 | train_loss: 0.4748 | train_acc: 0.8358 | test_loss: 0.5806 | test_acc: 0.8059
Epoch: 16 | train_loss: 0.4755 | train_acc: 0.8350 | test_loss: 0.5719 | test_acc: 0.8075
Epoch: 17 | train_loss: 0.4756 | train_acc: 0.8355 | test_loss: 0.5781 | test_acc: 0.8068
Epoch: 18 | train_loss: 0.4738 | train_acc: 0.8364 | test_loss: 0.5760 | test_acc: 0.8053
Epoch: 19 | train_loss: 0.4727 | train_acc: 0.8373 | test_loss: 0.5782 | test_acc: 0.8078
Epoch: 20 | train_loss: 0.4734 | train_acc: 0.8370 | test_loss: 0.5779 | test_acc: 0.8054
Epoch: 21 | train_loss: 0.4728 | train_acc: 0.8362 | test_loss: 0.5791 | test_acc: 0.8064
Epoch: 22 | train_loss: 0.4711 | train_acc: 0.8368 | test_loss: 0.5827 | test_acc: 0.8076
Epoch: 23 | train_loss: 0.4708 | train_acc: 0.8380 | test_loss: 0.5873 | test_acc: 0.8042
Epoch: 24 | train_loss: 0.4716 | train_acc: 0.8364 | test_loss: 0.5916 | test_acc: 0.8024
Epoch: 25 | train_loss: 0.4699 | train_acc: 0.8373 | test_loss: 0.5994 | test_acc: 0.8038
...
Epoch: 72 | train_loss: 0.4661 | train_acc: 0.8396 | test_loss: 0.6040 | test_acc: 0.8045
Epoch: 73 | train_loss: 0.4649 | train_acc: 0.8404 | test_loss: 0.6075 | test_acc: 0.8021
Epoch: 74 | train_loss: 0.4657 | train_acc: 0.8395 | test_loss: 0.6219 | test_acc: 0.7982
Epoch: 75 | train_loss: 0.4650 | train_acc: 0.8400 | test_loss: 0.6124 | test_acc: 0.8014
```

CHAPTER 5

RESULTS AND ACCURACY

Fig-5.1
Performance matrix

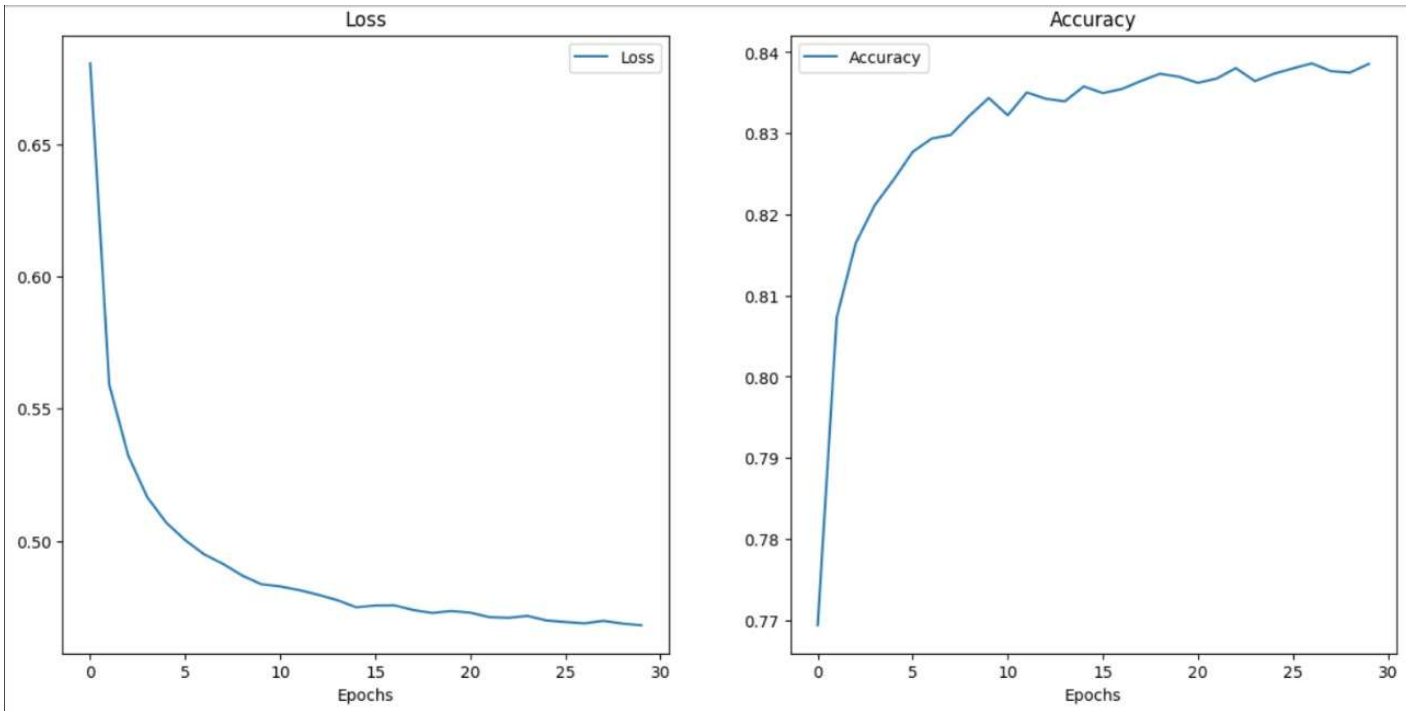
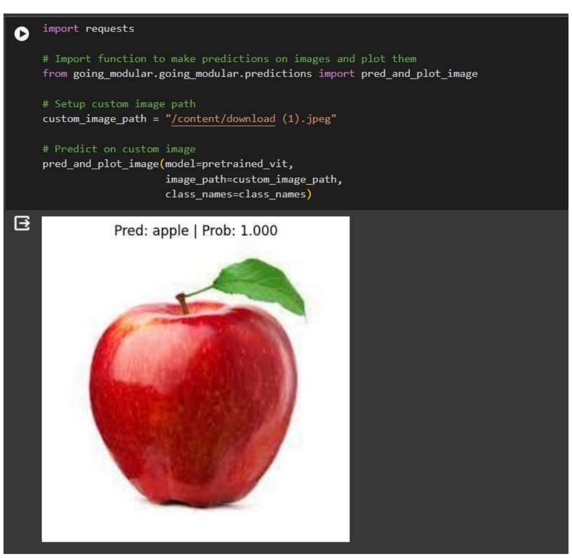


Fig 5.2
Prediction of image



```python
import requests

# Import function to make predictions on images and plot them
from going_modular.going_modular.predictions import pred_and_plot_image

# Setup custom image path
custom_image_path = "/content/download (5).jpeg"

# Predict on custom image
pred_and_plot_image(model=pretrained_vit,
                    image_path=custom_image_path,
                    class_names=class_names)
```

Pred: motorcycle | Prob: 0.691

```python
import requests

# Import function to make predictions on images and plot them
from going_modular.going_modular.predictions import pred_and_plot_image

# Setup custom image path
custom_image_path = ['/content/Gemini_Generated_Image (11).jpeg',
                     '/content/Gemini_Generated_Image (12).jpeg',
                     '/content/Gemini_Generated_Image (13).jpeg',
                     '/content/Gemini_Generated_Image (14).jpeg',
                     '/content/Gemini_Generated_Image (15).jpeg',
                     '/content/Gemini_Generated_Image (16).jpeg',
                     '/content/Gemini_Generated_Image (17).jpeg',
                     '/content/Gemini_Generated_Image (18).jpeg'
                     ]

for image in custom_image_path:
    # Make a prediction on custom image
    pred_and_plot_image(model=pretrained_vit,
                        image_path=image,
                        class_names=class_names)
```

Python

Pred: deer | Prob: 0.997

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENT

This project is the success of Vision Transformers (ViT) in image recognition. By creating an image as a sequence of patches and processing them with a standard NLP Transformer, ViT achieves state-of-the-art performance on image classification tasks, even without introducing computer visionspecific biases into the architecture. Additionally, ViT boasts a relatively low pre-training cost compared to other methods.

Future Enhancements:

Task Expansion: While successful in classification, applying ViT to other computer vision tasks like object detection and segmentation is crucial. The promising results here, along with those from Carson et al, suggest a fruitful path forward.

Pre-training Methods: Exploring advanced self-supervised pre-training methods holds significant potential. While initial experiments show improvement, the gap between self-supervised and large-scale supervised pre-training remains significant. Bridging this gap could further enhance ViT's capabilities.

Model Scaling: Further scaling of ViT, similar to how NLP Transformers have benefitted, is likely to lead to even better performance on various computer vision tasks. This research lays a strong foundation for further exploration in this direction.

In essence, ViT presents a powerful and adaptable approach to computer vision with significant room for future improvement. By addressing the areas mentioned above, researchers can unlock even greater potential from this innovative architecture.

REFERENCES

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. arXiv preprint arXiv:1607.06450, 2016.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. CoRR, abs/1409.0473, 2014.

[3] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc V. Le. Massive exploration of neural machine translation architectures. CoRR, abs/1703.03906, 2017.

[4] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. arXiv preprint arXiv:1601.06733, 2016

[5] Alexei Baevski and Michael Auli. Adaptive input representations for neural language modeling. In ICLR, 2019.

[6] I. Bello, B. Zoph, Q. Le, A. Vaswani, and J. Shlens. Attention augmented convolutional networks. In ICCV, 2019.

[7] Lucas Beyer, Olivier J. Henaff, Alexander Kolesnikov, Xiaohua Zhai, and A ´ aron van den Oord. Are ¨ we done with imagenet? arXiv, 2020.

[8] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. arXiv,

[9] 2020. Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In ECCV, 2020.

[10] Mark Chen, Alec Radford, Rewon Child, Jeff Wu, and Heewoo Jun. Generative pretraining from pixels. In ICML, 2020a.

[11] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for largescale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR), pages 770-778, 2016.

[13] François Chollet et al. Keras. https://keras.io, 2015.

[14] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. International Journal of Computer Vision, 115(3):211-252, 2015.

[15] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR), pages 28182826, 2016.