Brain Tumor Classification Using CNN and Vision Transformer (ViT)

श्रद्धाबान लभते ज्ञानम्

UNIVERSITY OF ENGINEERING
&
MANAGEMENT, JAIPUR

Brain Tumor Classification Using CNN and Vision Transformer (ViT)

Submitted in the partial fulfillment of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

under

UNIVERSITY OF ENGINEERING & MANAGEMENT, JAIPUR

BY

HIMANISH CHOWDHURY

University Roll no: 12020002001032

University Registration no: 204202000200026

HIMANGSHU CHOWDHURY

University Roll no: 12020002001031

University Registration no: 204202000200025

ANJAN MAHAPATRA

University Roll no: 12020002001157

University Registration no: 204202000200108

UNDER THE GUIDANCE OF

PROF. Dr TAPAS SI

COMPUTER SCIENCE & ENGINEERING

UNIVERSITY OF ENGINEERING & MANAGEMENT, JAIPUR

Approval Certificate

This is to certify that the project report entitled "Brain Tumor Classification Using CNN and Vision Transformer (ViT)" submitted by **HIMANGSHU CHOWDHURY** (Roll:**12020002001031**) , **HIMANISH CHOWDHURY** (Roll:**12020002001032**) , **ANJAN MAHAPATRA (**Roll:**12020002001157**) in partial fulfillment of the requirements of the degree of **Bachelor of Technology** in **Computer Science & Engineering** from University **of Engineering and Management, Jaipur** was carried out in a systematic and procedural manner to the best of our knowledge. It is a bona fide work of the candidate and was carried out under our supervision and guidance during the academic session of 2011-2015.

Prof. Dr TAPAS SI
Project Guide, Assistant Professor (CSE)
UEM, JAIPUR

Prof. Mrinal Kanti Sarkar
HOD (CSE)
UEM, JAIPUR

Prof. A Mukherjee
Dean
UEM, JAIPUR

ACKNOWLEDGEMENT

We place on record and warmly acknowledge the continuous encouragement, invaluable supervision, timely suggestions and inspired guidance offered by Prof. Dr. Tapas Si Sir in bringing the report to a successful completion. We are grateful to my project partners who have patiently extended all sorts of help for accomplishing this undertaking. Finally we extend my gratitude to all that are directly or indirectly Involved in the successful completion of this project work.

HIMANISH CHOWDHURY

HIMANGSHU CHOWDHURY

ANJAN MAHAPATRA

ABSTRACT

Traditional computer vision approach of hand crafting features followed by developing machine learning (ML) models. Traditional computer vision involved an in-depth analysis of the input and output. The in-depth analysis revealed what mathematically representable features could be extracted from an image and coupled with an efficient algorithm to produce the desired result. This computer vision algorithms are being practiced in medical image analysis and are transfiguring the perception and interpretation of Imaging data. Among these algorithms, Vision Transformers (ViTs) are evolved as one of the most contemporary and dominant architectures that are being used in the field of computer vision. These are immensely utilized by a plenty of researchers to perform new as well as former experiments. Here, in this article we investigate the intersection of Vision Transformers and Medical images and proffered an overview of various ViTs based frameworks that are being used by different researchers in order to decipher the obstacles in Medical Computer Vision. We surveyed the application of Vision transformers in different areas of medical computer vision such as image-based disease classification, anatomical structure segmentation, registration, region-based lesion Detection, captioning, report generation, reconstruction using multiple medical imaging modalities that greatly assist in medical diagnosis and hence treatment process. Along with this, we also demystify several imaging modalities used in Medical Computer Vision. . Within this review, we investigated key challenges including the use of transformers in different learning paradigms, improving model efficiency, and coupling with other techniques. However, the ViT models were found to be reliably accurate in classification scenario, achieving accuracy in identifying tumor.

Table of Contents

List of Figures

List of Table

1. CHAPTER

INTRODUCTION

1.1 Machine learning:

Machine Learning. Reviewed by Jake Frankenfield. Updated Mar 6, 2018. Machine learning is the concept that a computer program can learn and adapt to new data without human interference. Machine learning is a field of artificial intelligence (AI) that keeps a computer's built-in algorithms current regardless of changes in the worldwide economy.

Machine Learning, as the name says, is all about machines learning automatically without being explicitly programmed or learning without any direct human intervention. This machine learning process starts with feeding them good quality data and then training the machines by building various machine learning models using the data and different algorithms. The choice of algorithms depends on what type of data we have and what kind of task we are trying to automate.

1.2 Image Classification :

In a society where digital images have become prominent in our daily lives, the volumes of data it has generated over the decades are massive. Computer Vision is a field of AI which uses a lot of data, mainly for image detection, recognition, and classification. Image Classification uses Machine Learning algorithms to analyze the presence of items in a picture and to categorize the picture.
This particular task forms the basis of Computer Vision and Image Recognition. Machines don't analyze a picture as a whole. They only analyze a picture through pixel patterns or vectors. They will then categorize and assign labels to the elements they detect and classify them depending on the different rules that have been set up when configuring the algorithm. Classifiers' task is to take an input (a photograph for example) and to output a class label (extract the image features and predict the category from them).

There are two types of Image Classification techniques:

- If you are facing various pictures, and you only want to know whether the object in them is a cat or not, the problem you will have to handle is a binary classification. You only need to label one class of items for all images, or not to label it. The binary classification model is in charge of computing the presence or the absence of the object.
- If there is more than one category, you handle a multiclass classification problem which implies the creation of multiple labels that will correspond to various objects. The machine will then predict which single object class is contained in the photographs or pictures, among a set of predefined labels.

These techniques both mainly rely on the way the reference images are labeled. The following parts of this article will give a more detailed presentation of the way image classification works.

How it work:-
Machines have a very specific way to analyse images. The various techniques try to imitate the functioning of the human brain and eyes to propose an optimized analysis performance. The algorithms will leverage some pixel patterns which are very similar to what the machine has already seen. A whole process is necessary to build up an image classifier.

The first step: creating a dataset for the machine to use as a reference
In the first place, Image Classification needs to have a reference dataset to work on. You can import a set of images from the API (Application Programming Interface) Keras via a code line. If you choose to use Python coding language, it could be a great solution for you. Once your dataset is installed, you might want to explore it for a few minutes, to discover the classes which have already been set.

Depending on how you want to use the images from your dataset, you might want to go through them and modify some of the original parameters:

• Making sure the machine can read the images and pictures with a few lines of code.
• All pictures and images should be the same size so that the computer can process all the images in a standardized format. That way, the machine will be able to go through the analysis more rapidly than having to analyze various pictures with different dimensions.
• The dataset should be augmented with more data, based on the resources there already are. Practicing data augmentation allows the machine to analyze many different versions of a dataset during the testing section. This specific step is a way to prevent overfitting, which refers to the risk for the machine to learn "by heart" the data seen during training sessions. The machine might ignore completely all unknown data and it might not be able to take into account new sets of pictures. Data augmentation can be done by changing the orientation of a picture, converting it to grayscale, rotating, translating, or blurring it. The more options you give to the machine, the higher the accuracy will be when analyzing the data. Pre-processing your database is important if you want to have a solid dataset to work on, the next step is to create and set up a model which will learn to classify images.

Second Step: creating a model to detect objects: focus on Convolutional Neural Network
To classify images into multiple categories, you need to configure a classifier: an algorithm able to support your request. The most popular and accurate model type used to categorize images is CNN, which stands for Convolutional Neural Network.
Discover how training data can make or break your AI projects, and how to implement the Data Centric AI philosophy in your ML projects.

Third Step: training and validation of your algorithm
Now that you have created a valid dataset and set up a model to be used as a classification tool, we have to train it and test it to see if it is precise enough to provide us with the correct information.

Training with Supervised Learning

Computer Vision and Image Recognition tasks are based on the actions of the human brain. So if we want the method to be accurate, we need to train it and support it with a human hand. Supervised learning refers to a training of the data with a set that you labeled yourself. In other words, you imported your own set of pictures and created the classes by yourself as well. Input and output data are given to the algorithm. For example, You select a picture (input) of a group of cats with a bird, and you only want to know if there is a bird (output) on the picture.

To train the algorithm, various methods can be used: vector-related ones, which are used to divide the classes with a linear boundary. Decision Trees are also very common methods. And of course, Neural Networks like CNN-based models can be used. The algorithm simply has to answer yes-no questions to identify and categorize the different objects in the picture.

Supervised learning is much simpler to use but it can be very time-consuming and it might not be able to classify big data. The dataset has to be checked thoroughly. When somebody identifies a category of item, he or she can label all the classes the way he or she wants to. This allows the creation of a wide enough dataset for training, but it can be challenging. Results are usually better in terms of accuracy.

1.3 Convolutional Neural Network (CNN):

CNNs, or convolutional neural networks, are extensively employed in image classification, detection, and recognition. A CNN is an image-training neural network architecture that draws inspiration from human neurons. It uses a variety of filters and convolution layers, which require precise pre-configuration, to process images.

Using these layers, CNN will generate an image feature map based on the represented pixels. After that, the entire element matrix will be processed. Comprehending CNN design: Convolutional layers and pooling layers, or buried layers, make up CNN's extremely particular design. It is possible to employ a different number of layers while processing an image. Three to about one hundred layers are typically accessible for computer vision analysis in a CNN model. Convolution layers are methods for applying filters to an input, such as an image. For example, one method may filter pixel patterns according to the picture's colours, while another method may filter recognized shapes. The algorithm will apply a pooling layer after employing a convolution layer. The information received by the preceding convolutional layer will be collected by this one. Its primary responsibility is to clean the region and gather information prior to applying a new filter. Applying the last layer is all that's required to synthesis the information produced by the earlier layers once all required filters and layers have been applied. At that point, the classification results are prepared for delivery. These complementary steps make CNN's the most popular and effective Classifier tool in Machine Learning. They currently are at the state of the art for Image Classification tasks, due to their accuracy in the results and their ability to deliver them very quickly.

Convolutional neural networks, often known as CNNs or ConvNets, are a class of artificial neural networks (ANNs) used in deep learning that are most frequently used to analyse visual data. Because of the shared-weight architecture of the convolution kernels or filters, which slide along input features and produce translation-equivariant replies known as feature maps, CNNs are sometimes referred to as Shift Invariant or Space Invariant Artificial Neural Networks (SIANN). Contrary to popular belief, most convolutional neural networks do not translate well because of the down sampling process they perform on the input. Applications for them include financial time series, natural language processing, recommender systems, medical image analysis, image and video recognition, image classification, image segmentation, and brain–computer interfaces.

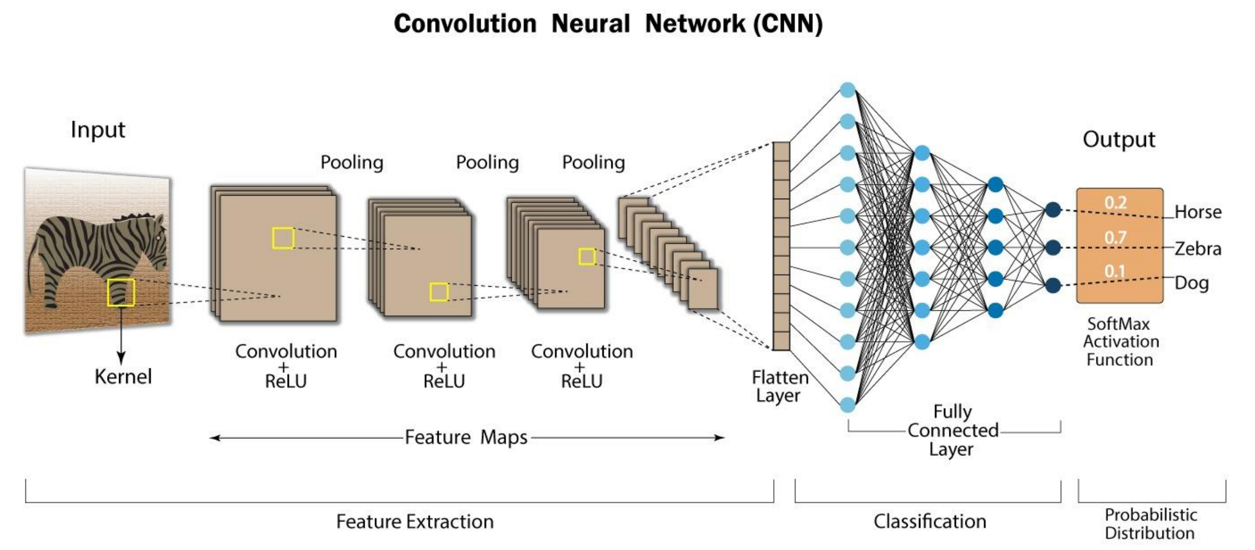


Figure 1:-CNN model

1.3.1 How do convolutional neural networks work?

A CNN can have multiple layers, each of which learns to detect the different features of an input image. A filter or kernel is applied to each image to produce an output that gets progressively better and more detailed after each layer. In the lower layers, the filters can start as simple features. At each successive layer, the filters increase in complexity to check and identify features that uniquely represent the input object. Thus, the output of each convolved image -- the partially recognized image after each layer -- becomes the input for the next layer. In the last layer, which is an FC layer, the CNN recognizes the image or the object it represents. With convolution, the input image goes through a set of these filters. As each filter activates certain features from the image, it does its work and passes on its output to the filter in the next layer. Each layer learns to identify different features and the operations end up being repeated for dozens, hundreds or even thousands of layers. Finally, all the image data progressing through the CNN's multiple layers allow the CNN to identify the entire object.

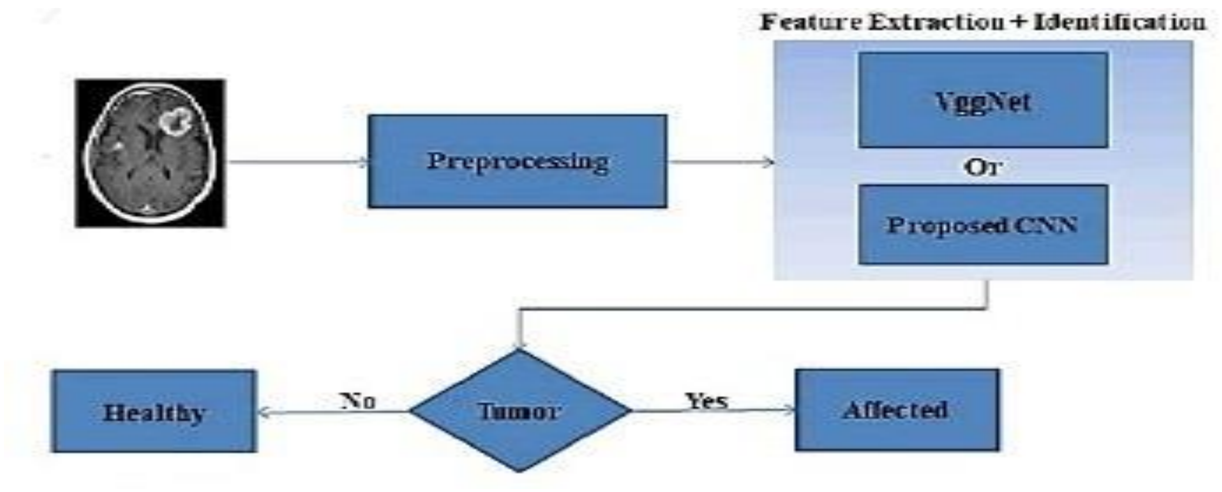


Figure 2:-CNN working process

1.3.2 Benefits of using CNNs for deep learning :

Deep learning is a subset of machine learning that uses neural networks with at least three layers. Compared to a network with just one layer, a network with multiple layers can deliver more accurate results. Both RNNs and CNNs are used in deep learning, depending on the application. For image recognition, image classification and computer vision (CV) applications, CNNs are particularly useful because they provide highly accurate results, especially when a lot of data is involved. The CNN also learns the object's features in successive iterations as the object data moves through the CNN's many layers. This direct (and deep) learning eliminates the need for manual feature extraction (feature engineering).CNNs can be retrained for new recognition tasks and built on preexisting networks. These advantages open up new opportunities to use CNNs for real-world applications without increasing computational complexities or costs. As seen earlier, CNNs are more computationally efficient than regular NNs since they use parameter sharing. The models are easy to deploy and can run on any device, including smartphones.

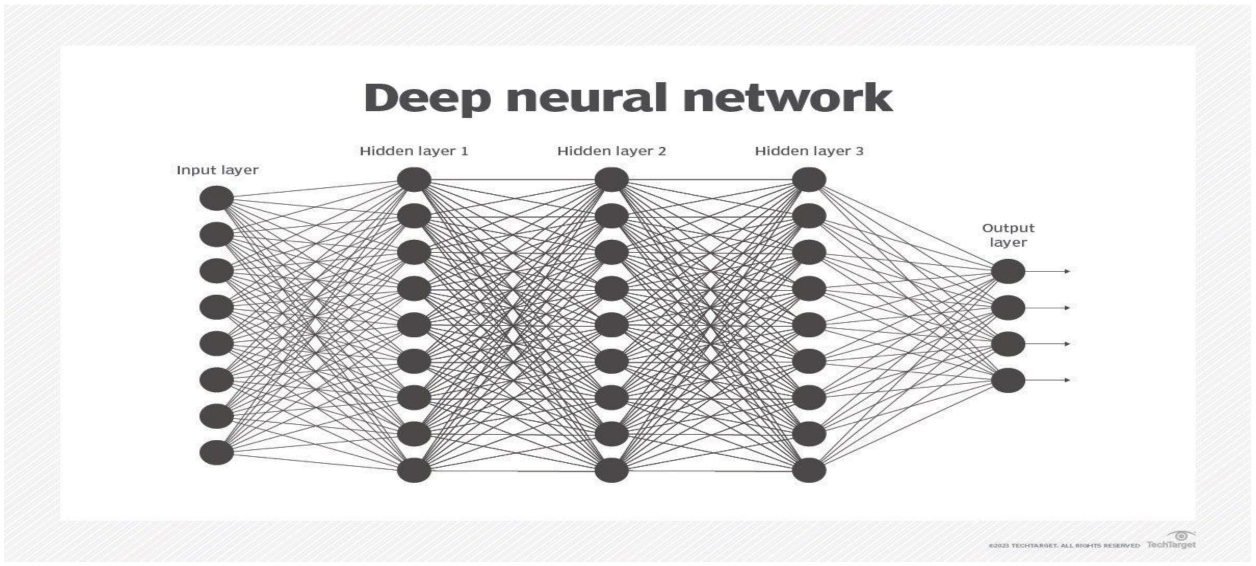


Figure 3:-Deep Neural Network

1.3.3 Applications of convolutional neural networks

Convolutional neural networks are already used in a variety of CV and image recognition applications. Unlike simple image recognition applications, CV enables computing systems to also extract meaningful information from visual inputs (e.g., digital images) and then take appropriate action based on this information.

The most common applications of CV and CNNs are used in fields such as the following:

- **Healthcare.** CNNs can examine thousands of visual reports to detect any anomalous conditions in patients, such as the presence of malignant cancer cells.
- **Automotive**. CNN technology is powering research into autonomous vehicles and self-driving cars.
- **Social media.** Social media platforms use CNNs to identify people in a user's photograph and help the user tag their friends.
- **Retail.** E-commerce platforms that incorporate visual search allow brands to recommend items that are likely to appeal to a shopper.

- **Facial recognition for law enforcement.** Generative adversarial networks (GANs) are used to produce new images that can then be used to train deep learning models for facial recognition
- **Audio processing for virtual assistants.** CNNs in virtual assistants learn and detect user-spoken keywords and process the input to guide their actions and respond to the user.

1.4 Vision Transformers(ViT)

Over the years, we have been using Computer vision (CV) and image processing techniques from artificial intelligence (AI) and pattern recognition to derive information from images, videos, and other visual inputs. Underlying methods successfully achieve this by manipulating digital images through computer algorithms.

Researchers found that regular models had limitations in some applications, which prompted advancements in traditional deep learning and deep neural networks. This brought about the popularity of transformer models. They have the ability known as “self-attention”. This provides them with an edge over other model architectures, and researchers have introduced it extensively in natural language processing and computer vision.

2. CHAPTER

2.1 Introduction of Vision Transformers?

In simple terms, vision transformers are types of transformers used for visual tasks such as in image processing. This entails that transformers are being used in many areas, including NLP, but ViT specifically focuses on processing image-related tasks. Recently, used majorly in Generative artificial intelligence and stable diffusion. ViT measures the relationships between input images in a technique called attention. It enhances some parts of the image and diminishes other parts while mimicking cognitive attention. The goal is to learn the important parts of the input. The instructions that provide context and constraints guide this approach.

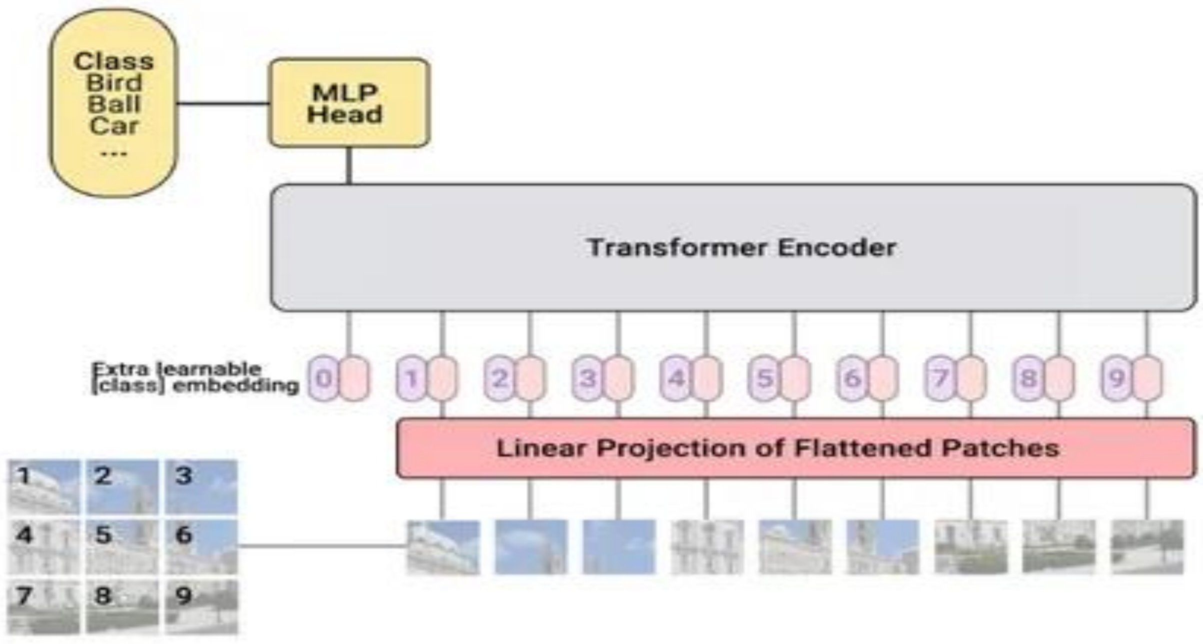

Class
Bird
Ball
Car
...
MLP Head
Transformer Encoder
Extra learnable [class] embedding
0
1
2
3
4
5
6
7
8
9
Linear Projection of Flattened Patches
1
2
3
4
5
6
7
8
9

Figure 4 :-ViT model

2.1.1 How Do Vision Transformers Work?

Vision Transformer applies the transformer to image classification tasks with a model architecture similar to a regular transformer. It adjusts itself to allow efficient handling of images, as other models will perform for natural language processing tasks. Key concepts of vision transformers include ‘attention’ and ‘multi-head attention’. Having an understanding of these concepts is very essential in how vision transformers work. Attention is a key mechanism unique to transformers and is the secrete to their strength. Let’s look at the transformer architecture and see how it works. The Masked Multi-Head Attention is a central mechanism of the Transformer similar to skip-joining as in ResNet50 architecture. This means that there is a shortcut connection or skipping of some layers of the network.

The multi-head attention calculates the attention weight of a Query token which could be the prompt of an image. Both the Key token and the Value associated with each Key are multiplied together. We can also say it calculates the relationship or attention weight between the Query and the Key and then multiplies the Value associated with each Key.

We can conclude that multi-head attention allows us to treat different parts of the input sequence differently. The model bests capture positional details since each head will separately attend to different input elements. This gives us a more robust representation.

2.1.2 Applications of Vision Transformers

Vision Transformers have revolutionized traditional Computer Vision tasks. Following are the areas of application of the vision transformers:

- Image Detection and Classification
- Video Deepfake Detection and Anomaly Detection
- Image segmentation and cluster analysis
- Autonomous Driving

2.2 Vision Transformers versus Convolutional Neural Networks :

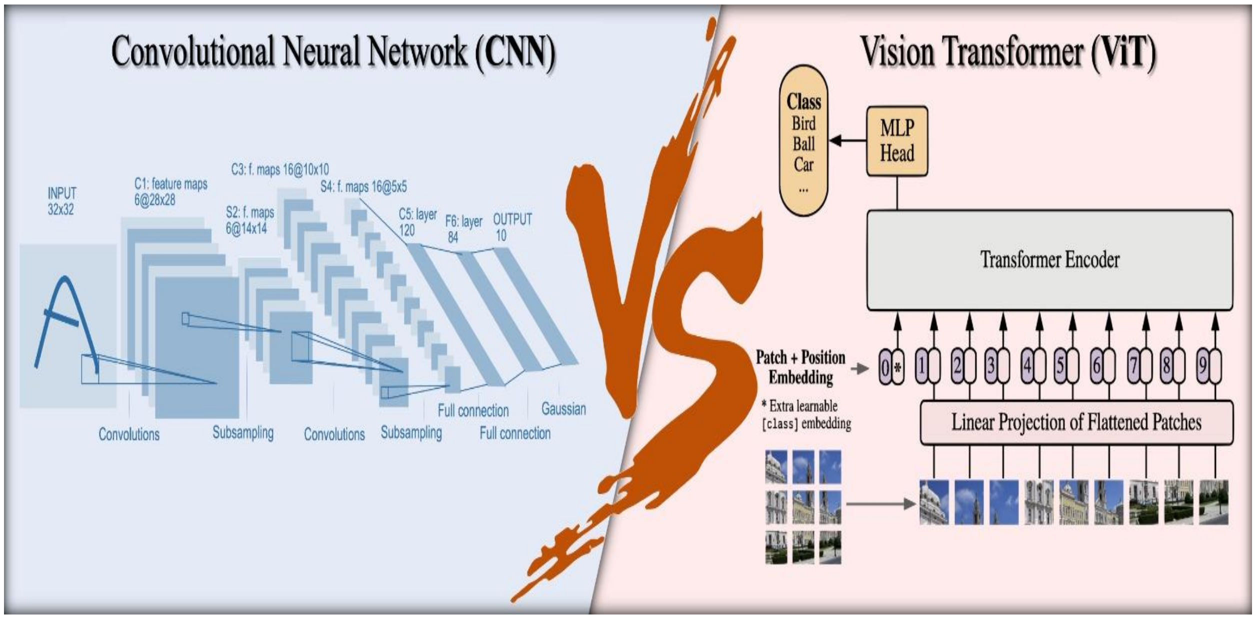


Figure 5 :-ViT Vs CNN

It is beneficial to also look at the comparison between the two as this can help understand transformers. The differences are many; besides, both have different architecture.

1. Major Building Blocks: Vision transformers are made up of three major components, including the optimizer and dataset-specific parameters valued to control the learning process and the network depth. Convolutional neural networks are less complex compared to optimization.
2. CNNs require and learn better based on data volume. The better the dataset, the better the accuracy. This is not exactly the same for Vision transformers, as they perform satisfactorily at comparatively fewer datasets.
3. CNNs tend to have inductive biases. Inductive bias or learning bias is the assumption the model makes when making predictions limiting it to fail in global relations or generalization. Vision Transformers does not have these biases making them work well generalized by the approach of their training method.

4. By their performance, Vision Transformers are more robust in dealing with input image distortions than CNNs.
5. Transformers work non-sequentially whereas CNNs are sequential in the data processing. CNN will take an image at a time or in batches while transformers can take all the images input at once.
6. A huge difference is the presence of an attention mechanism in transformers. The attention helps transformers work according to prompts or contexts while still using past information, but CNNs can only use learned knowledge without any contextual strength.

2.3 Vision Transformers for Dense Prediction :

Intel labs has certainly played a vital role in researching and presenting work on vision transformers in the context of making predictions on images. This is seen as a dense prediction. Dense prediction learns a mapping from a simple input image to a complex output. This might have to do with semantic segmentation or image depth estimation, etc. Depth estimation looks at the pixel of images, so it is very handy for computer vision used in object tracking, augmented reality, and autonomous cars.

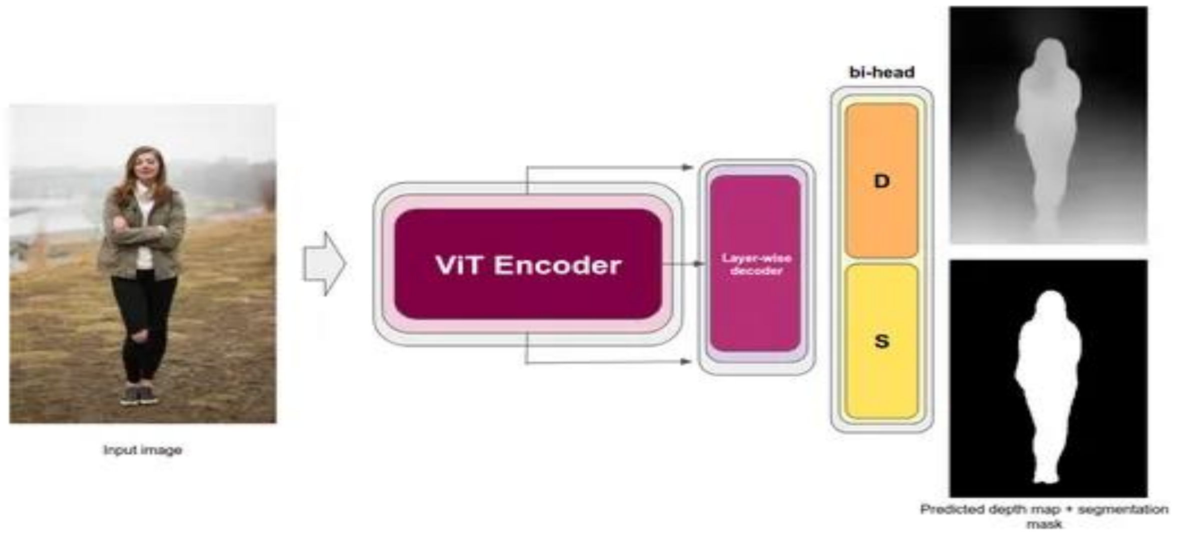


Figure 6 :-ViT Dense Prediction

2.4 Vision Transformer ViT Architecture :

Several vision transformer models have been proposed in the literature. The overall structure of the vision transformer architecture consists of the following steps:

1.Split an image into patches (fixed sizes)

2.Flatten the image patches

3.Create lower-dimensional linear embeddings from these flattened image patches

4. Include positional embeddings

5.Feed the sequence as an input to a state-of-the-art transformer encoder

6.Pre-train the ViT model with image labels, which is then fully supervised on a big dataset

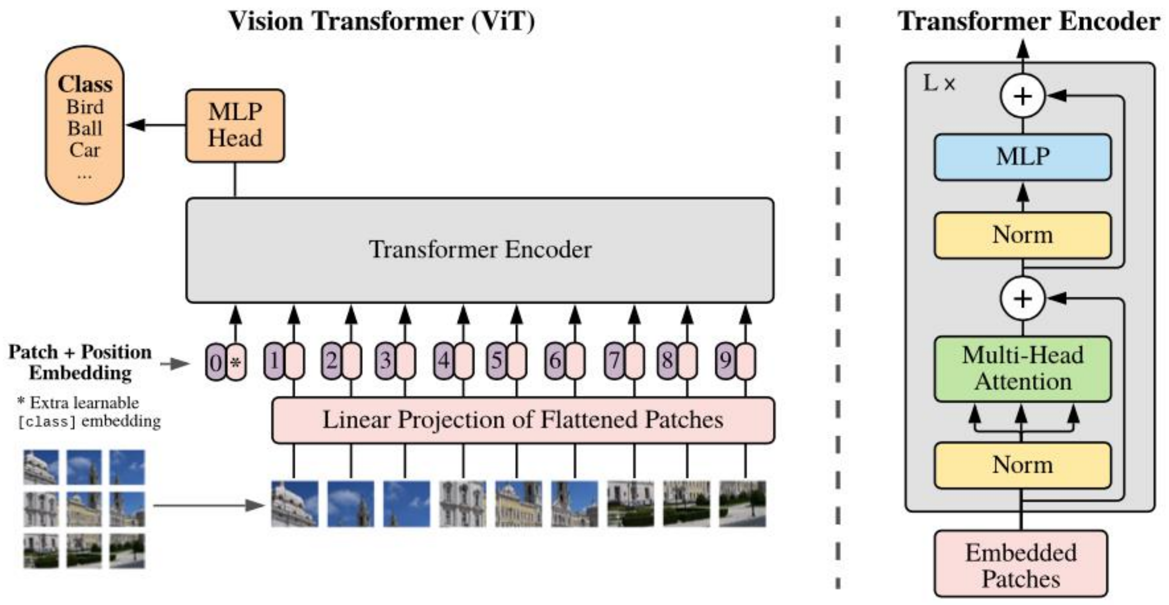


Figure 7 :-ViT Architecture

2.5 Real-World Vision Transformer (ViT) Use Cases and Applications :

Vision transformers have extensive applications in popular image recognition tasks such as object detection, segmentation, image classification, and action recognition. Moreover, ViTs are applied in generative modeling and multi-model tasks, including visual grounding, visual-question answering, and visual reasoning.

Video forecasting and activity recognition are all parts of video processing that require ViT. Moreover, image enhancement, colorization, and image super-resolution also use ViT models. Last but not least, ViTs have numerous applications in 3D analysis, such as segmentation and point classification.

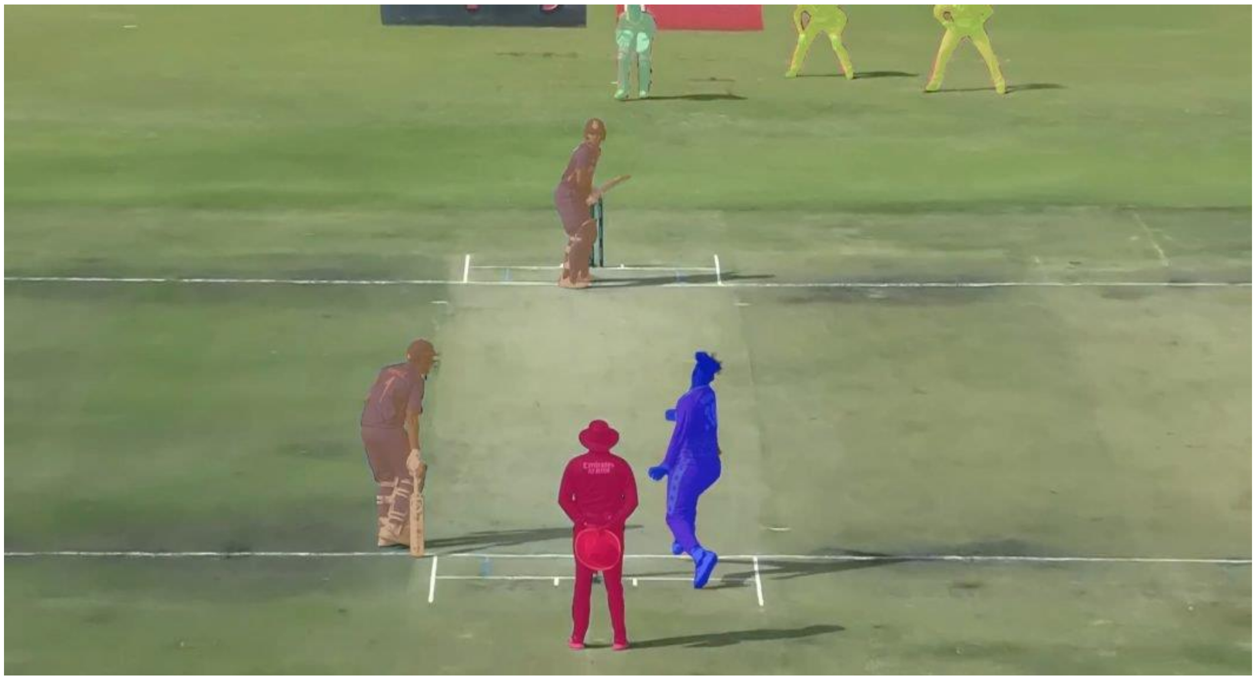


Figure 8 :-ViT Real-World Applications

The vision transformer model uses multi-head self-attention in Computer Vision without requiring image-specific biases. The model splits the images into a series of positional embedding patches, which are processed by the transformer encoder. It does so to understand the local and global features that the image possesses. Last but not least, the ViT has a higher precision rate on a large dataset with reduced training time.

2.6 Multi-layer Perceptron :

Multi-layer perception is also known as MLP. It is fully connected dense layers, which transform any input dimension to the desired dimension. A multi-layer perception is a neural network that has multiple layers. To create a neural network, we combine neurons together so that the outputs of some neurons are inputs of other neurons. A multi-layer perceptron has one input layer and for each input, there is one neuron (or node), it has one output layer with a single node for each output and it can have any number of hidden layers and each hidden layer can have any number of nodes.

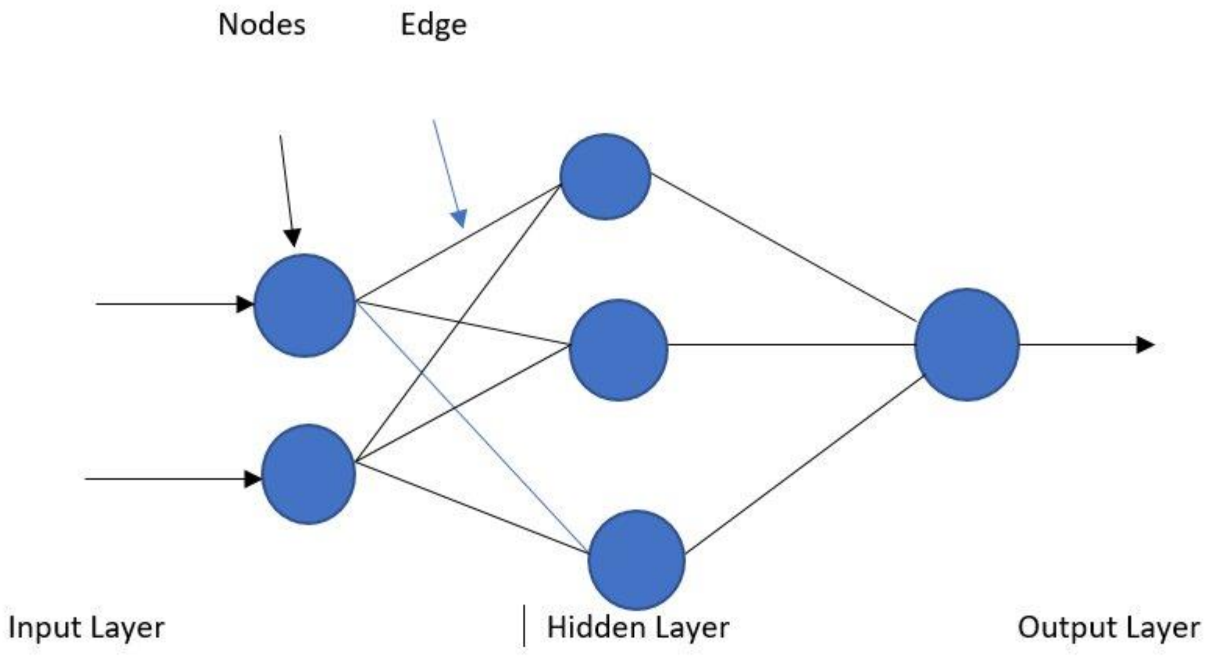


Figure 9 :- Multi-layer Perceptron

2.6.1 What is MultiLayer Perceptron Neural Network?

A multilayer perceptron (MLP) Neural network belongs to the feedforward neural network. It is an Artificial Neural Network in which all nodes are interconnected with nodes of different layers.

The word Perceptron was first defined by Frank Rosenblatt in his perceptron program. Perceptron is a basic unit of an artificial neural network that defines the artificial neuron in the neural network. It is a supervised learning algorithm that contains nodes' values, activation functions, inputs, and node weights to calculate the output.

The Multilayer Perceptron (MLP) Neural Network works only in the forward direction. All nodes are fully connected to the network. Each node passes its value to the coming node only in the forward direction. The MLP neural network uses a Backpropagation algorithm to increase the accuracy of the training model.

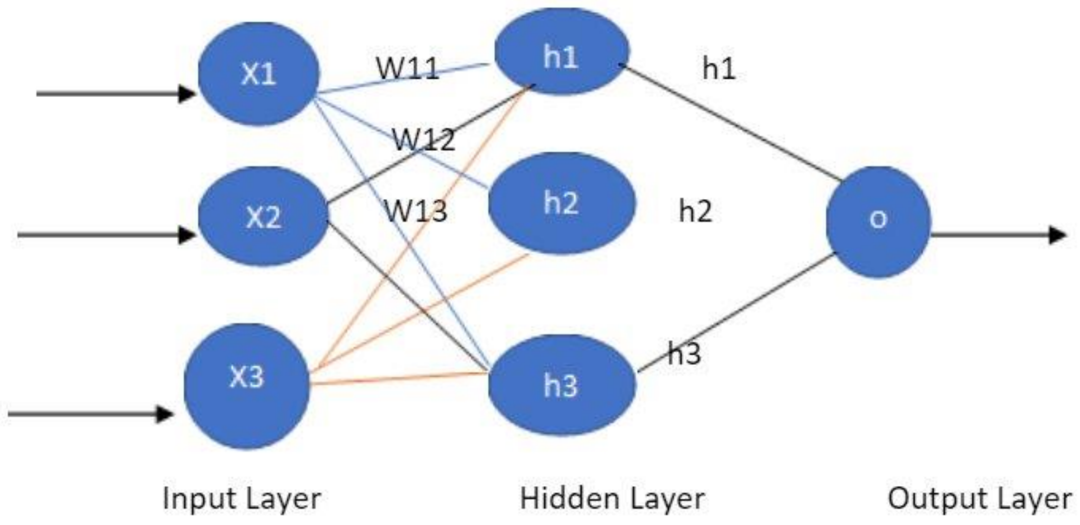


Figure 10 :- Multi-layer Perceptron Neural Network

.

2.7 Different types of Advance ViT model

2.7.1 DeepViT: Towards Deeper Vision Transformer

Vision transformers (ViTs) have been successfully applied in image classification tasks recently. In this paper, we show that, unlike convolution neural networks (CNNs)that can be improved by stacking more convolutional layers, the performance of ViTs saturate fast when scaled to be deeper. More specifically, we empirically observe that such scaling difficulty is caused by the attention collapse issue: as the transformer goes deeper, the attention maps gradually become similar and even much the same after certain layers. In other words, the feature maps tend to be identical in the top layers of deep ViT models. This fact demonstrates that in deeper layers of ViTs, the self-attention mechanism fails to learn effective concepts for representation learning and hinders the model from getting expected performance gain. Based on above observation, we propose a simple yet effective method, named Re-attention, to re-generate the attention maps to increase their diversity at different layers with negligible computation and memory cost. The pro-posed method makes it feasible to train deeper ViT models with consistent performance improvements via minor modification to existing ViT models.

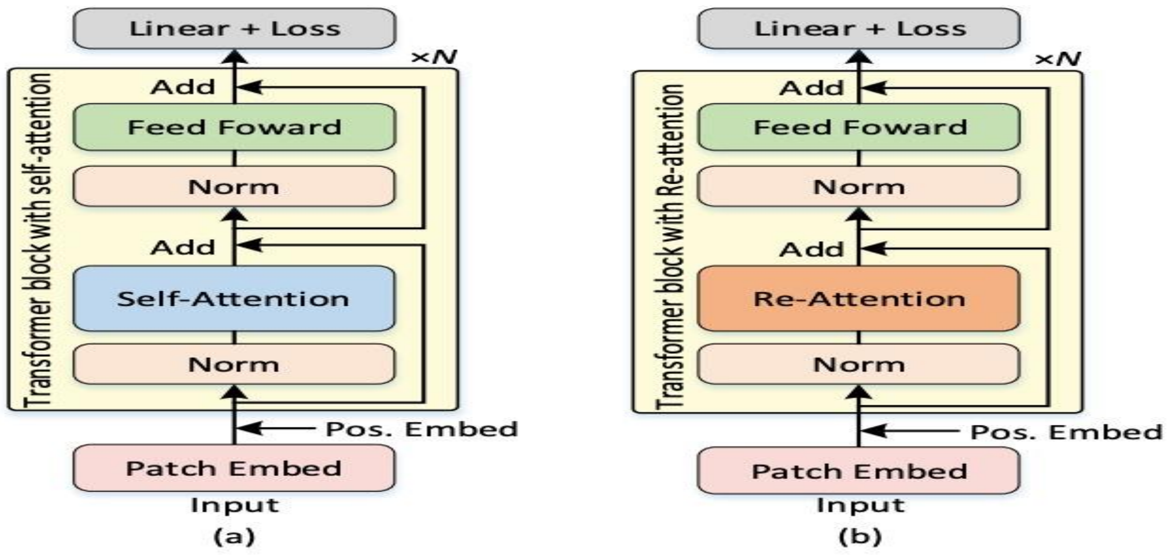


Figure 11 :- Comparison between the original ViT(a) and DeepViT(b) model

2.7.2 MobileViT:

A mobile-friendly Transformer-based model for image classification Light-weight convolutional neural networks (CNNs) are the de-facto for mobile vision tasks. Their spatial inductive biases allow them to learn representations with fewer parameters across different vision tasks. However, these networks are spatially local. To learn global representations, self-attention-based vision trans-formers (ViTs) have been adopted. Unlike CNNs, ViTs are heavy-weight. In this paper, we ask the following question: is it possible to combine the strengths of CNNs and ViTs to build a light-weight and low latency network for mobile vision tasks? Towards this end, we introduce MobileViT, a light-weight and general-purpose vision transformer for mobile devices. MobileViT presents a different perspective for the global processing of information with transformers, i.e., transformers as convolutions. Our results show that MobileViT significantly outperforms CNN- and ViT-based networks across different tasks and datasets. On the ImageNet-1k dataset, MobileViT achieves top-1 accuracy of 78.4% with about 6 million parameters, which is 3.2% and 6.2% more accurate than MobileNetv3 (CNN-based) and DeIT (ViT-based) for a similar number of parameters. On the MS-COCO object detection task, MobileViT is 5.7% more accurate than MobileNetv3 for a similar number of parameters.

Mobile ViT utilities

The MobileViT architecture is comprised of the following blocks:

- Strided 3x3 convolutions that process the input image.
- Mobile NetV2-style inverted residual blocks for down sampling the resolution of the intermediate feature maps.
- MobileViT blocks that combine the benefits of Transformers and convolutions. It is presented in the figure below -

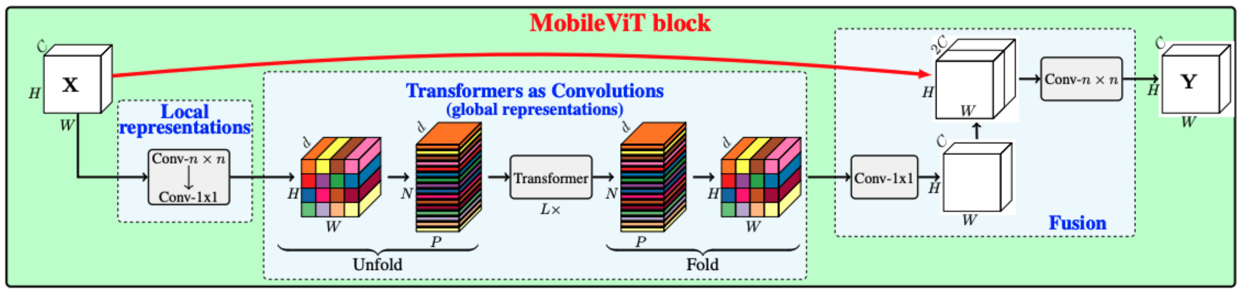


Figure 12 :- MobileViT model

2.7.3 Class-Attention in Image Transformers (CaiT) :

CaiT, or Class-Attention in Image Transformers, is a type of vision transformer with several design alterations upon the original ViT. First a new layer scaling approach called LayerScale is used, adding a learnable diagonal matrix on output of each residual block, initialized close to (but not at) 0, which improves the training dynamics. Secondly, class-attention layers are introduced to the architecture. This creates an architecture where the transformer layers involving self-attention between patches are explicitly separated from class-attention layers -- that are devoted to extract the content of the processed patches into a single vector so that it can be fed to a linear classifier.Transformers have been recently adapted for large scale image classification, achieving high scores shaking up the long supremacy of convolutional neural networks. However the optimization of image transformers has been little studied so far. In this work, we build and optimize deeper transformer networks for image classification. In particular, we investigate the interplay of architecture and optimization of such dedicated transformers. We make two transformers architecture changes that significantly improve the accuracy of deep transformers. This leads us to produce models whose performance does not saturate early with more depth, for instance we obtain 86.5% top-1 accuracy on Imagenet when training with no external data, we thus attain the current SOTA with less FLOPs and parameters. Moreover, our best model establishes the new state of the art on Imagenet with Reassessed labels and Imagenet-V2 / match frequency, in the setting with no additional training data. We share our code and models.

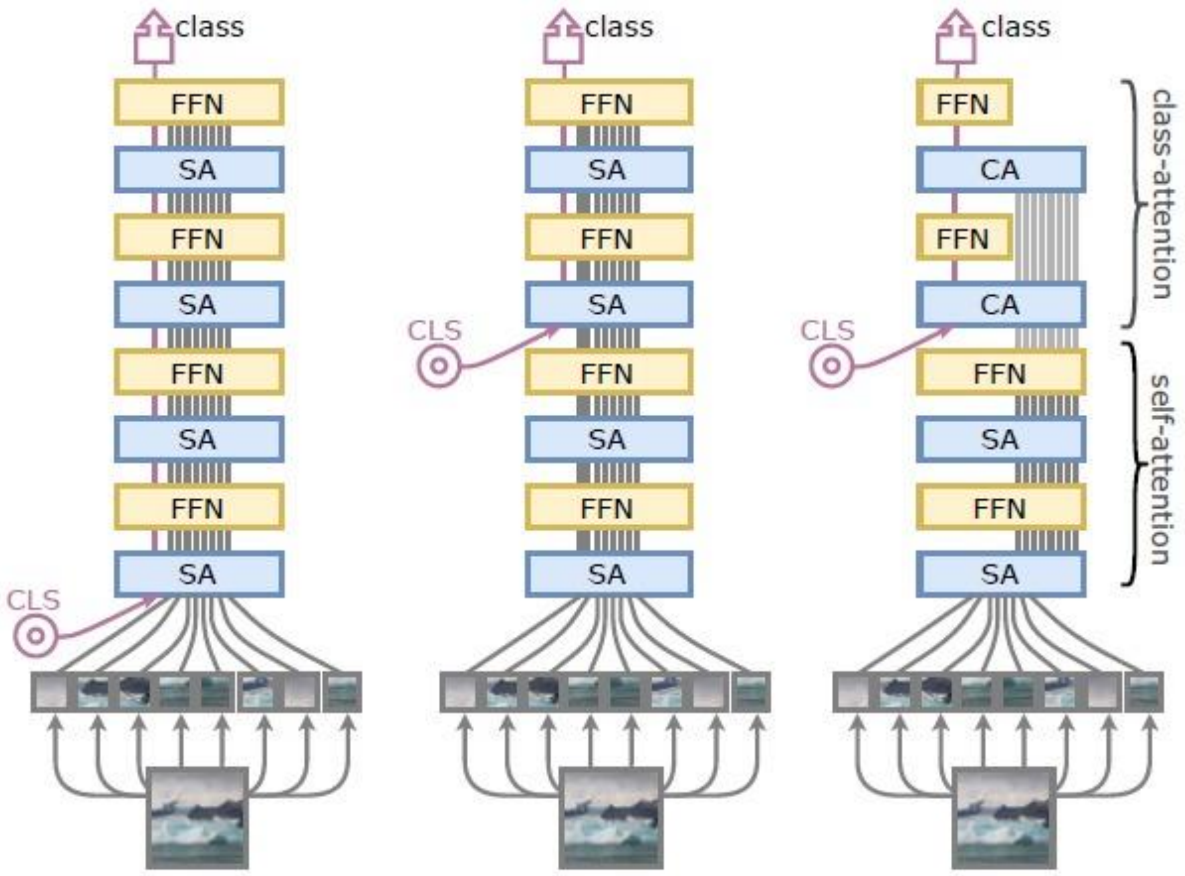


Figure 13 :- CaiT model

2.7.4 LeViT: a Vision Transformer in ConvNet's Clothing for Faster Inference :

We design a family of image classification architectures that optimize the trade-off between accuracy and efficiency in a high-speed regime. Our work exploits recent findings in attention-based architectures, which are competitive on highly parallel processing hardware. We revisit principles from the extensive literature on convolutional neural networks to apply them to transformers, in particular activation maps with decreasing resolutions. We also introduce the attention bias, a new way to integrate positional information in vision transformers. As a result, we propose LeVIT: a hybrid neural network for fast inference image classification. We consider different measures of efficiency on different hardware platforms, so as to best reflect a wide range of application scenarios. Our extensive experiments empirically validate our technical choices and show they are suitable to most architectures. Overall, LeViT significantly outperforms existing convnets and vision transformers with respect to the speed/accuracy tradeoff. For example, at 80% ImageNet top-1 accuracy, LeViT is 5 times faster than EfficientNet on CPU.

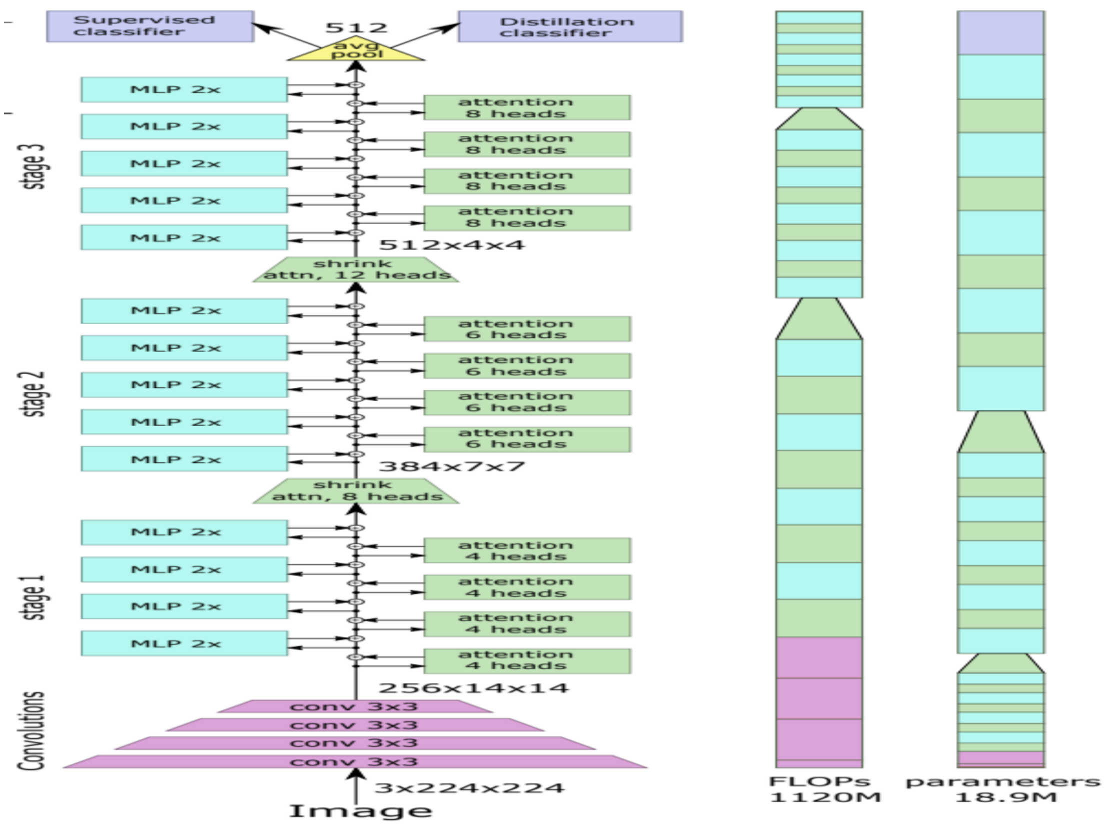
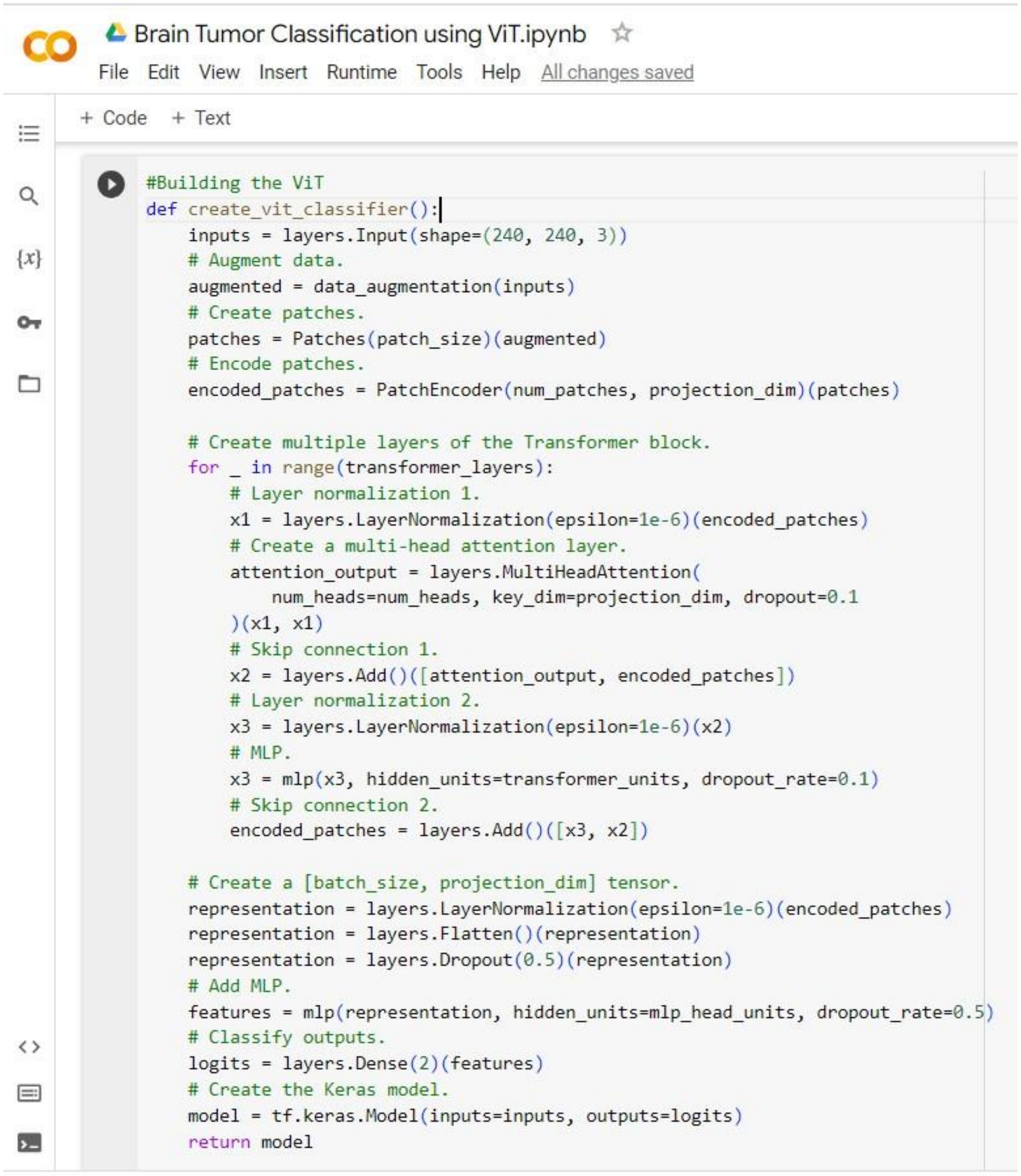


Figure 14 :- LeViT model

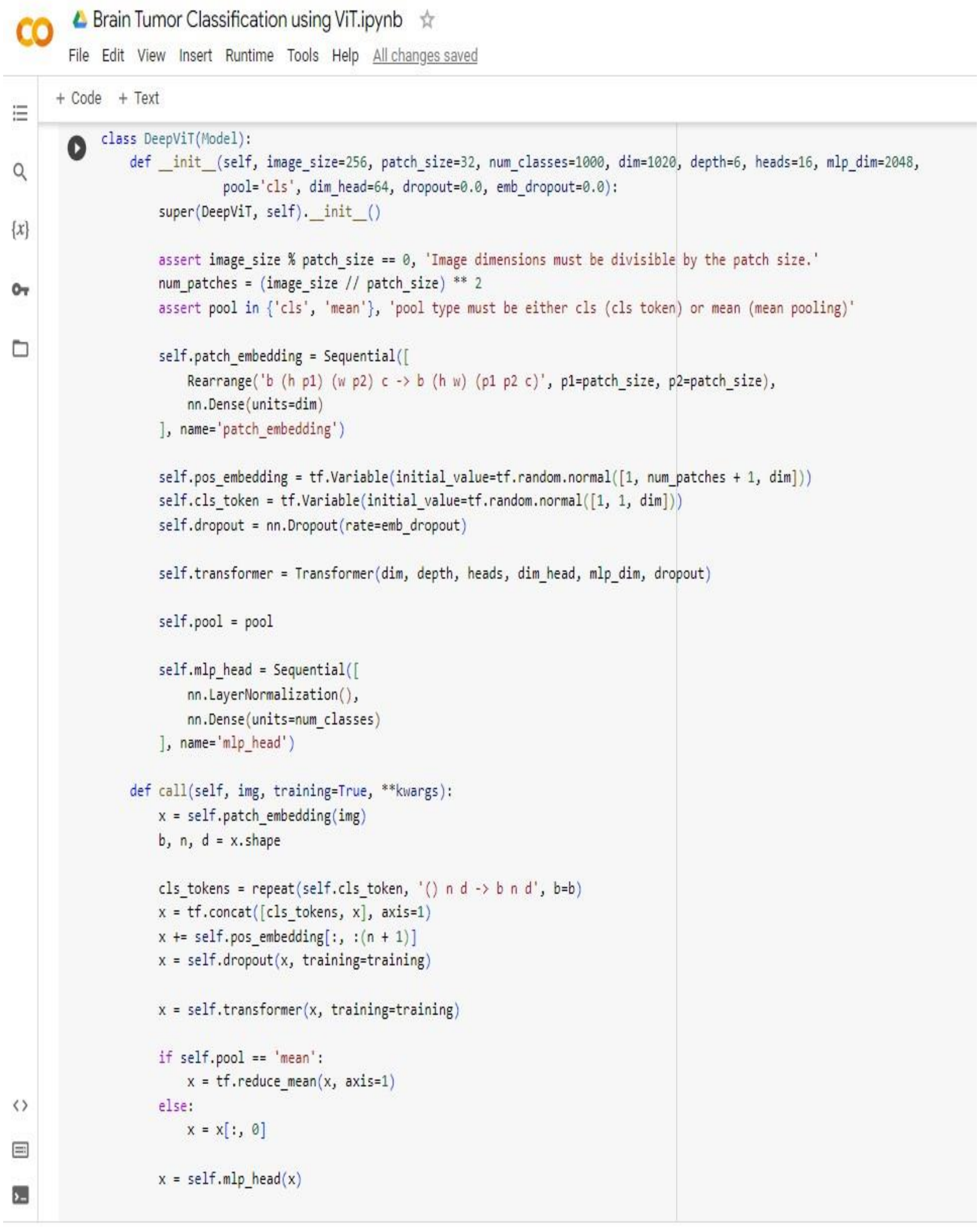
2.7.5 RegionViT : RegionViT consists of two tokenization processes that convert an image into regional (upper path) and local tokens (lower path). Each tokenization is a convolution with different patch sizes, the patch size of regional tokens is while is used for local tokens with dimensions projected to, which means that one regional token covers local tokens based on the spatial locality, leading to the window size of a local region to. At stage 1, two set of tokens are passed through the proposed regional-to-local transformer encoders. However, for the later stages, to balance the computational load and to have feature maps at different resolution, the approach uses a downsampling process to halve the spatial resolution while doubling the channel dimension like CNN on both regional and local tokens before going to the next stage. Finally, at the end of the network, it simply averages the remaining regional tokens as the final embedding for the classification while the detection uses all local tokens at each stage since it provides more fine-grained location information. By having the pyramid structure, the ViT can generate multi-scale features and hence it could be easily extended to more vision applications, e.g., object detection, rather than image classification only.

APPENDIX


Brain Tumor Classification using ViT.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

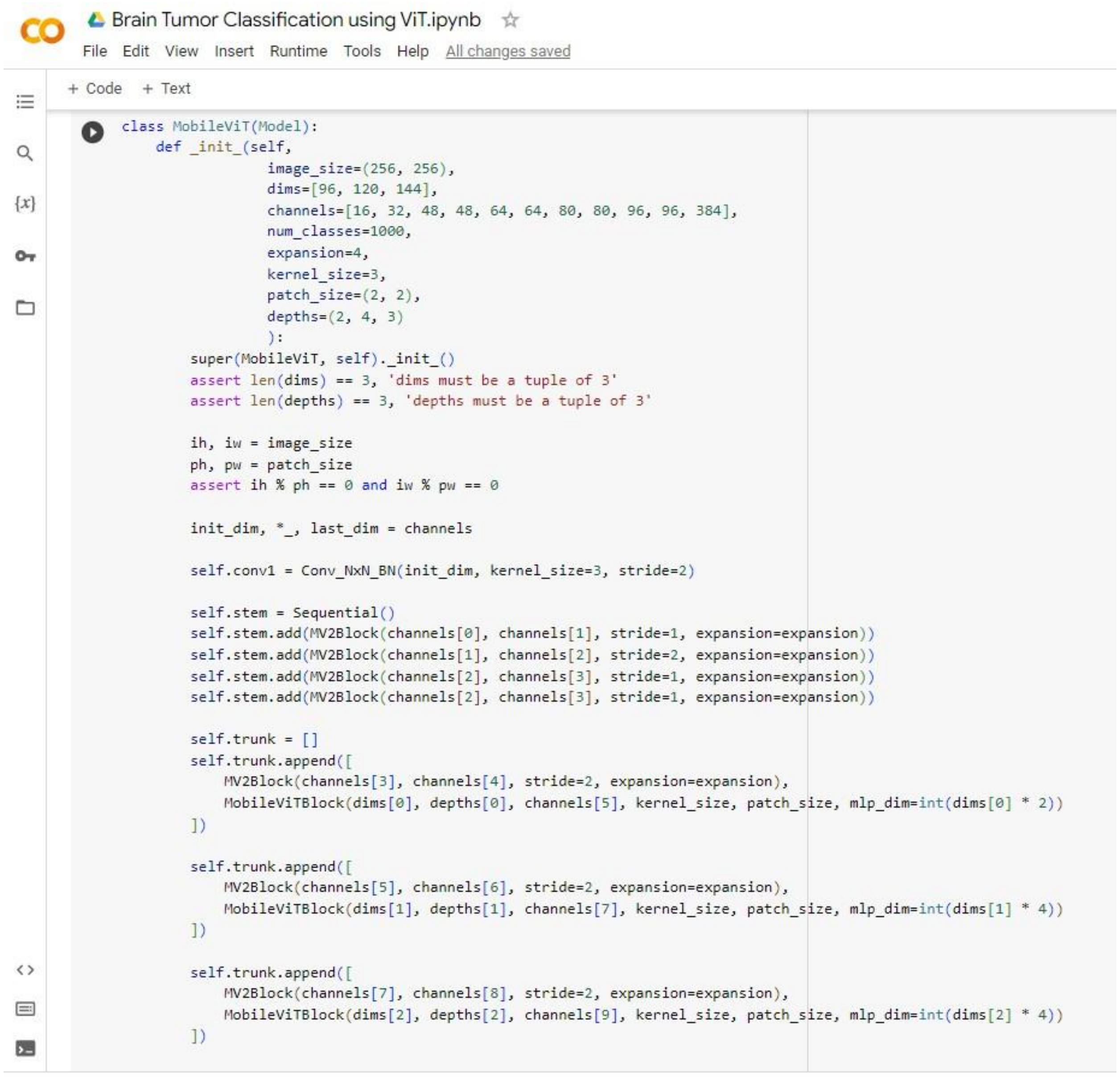
```python
#Building the ViT
def create_vit_classifier():
    inputs = layers.Input(shape=(240, 240, 3))
    # Augment data.
    augmented = data_augmentation(inputs)
    # Create patches.
    patches = Patches(patch_size)(augmented)
    # Encode patches.
    encoded_patches = PatchEncoder(num_patches, projection_dim)(patches)

    # Create multiple layers of the Transformer block.
    for _ in range(transformer_layers):
        # Layer normalization 1.
        x1 = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
        # Create a multi-head attention layer.
        attention_output = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=projection_dim, dropout=0.1
        )(x1, x1)
        # Skip connection 1.
        x2 = layers.Add()([attention_output, encoded_patches])
        # Layer normalization 2.
        x3 = layers.LayerNormalization(epsilon=1e-6)(x2)
        # MLP.
        x3 = mlp(x3, hidden_units=transformer_units, dropout_rate=0.1)
        # Skip connection 2.
        encoded_patches = layers.Add()([x3, x2])

    # Create a [batch_size, projection_dim] tensor.
    representation = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
    representation = layers.Flatten()(representation)
    representation = layers.Dropout(0.5)(representation)
    # Add MLP.
    features = mlp(representation, hidden_units=mlp_head_units, dropout_rate=0.5)
    # Classify outputs.
    logits = layers.Dense(2)(features)
    # Create the Keras model.
    model = tf.keras.Model(inputs=inputs, outputs=logits)
    return model
```

Figure 15 :- ViT model Code


Brain Tumor Classification using ViT.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

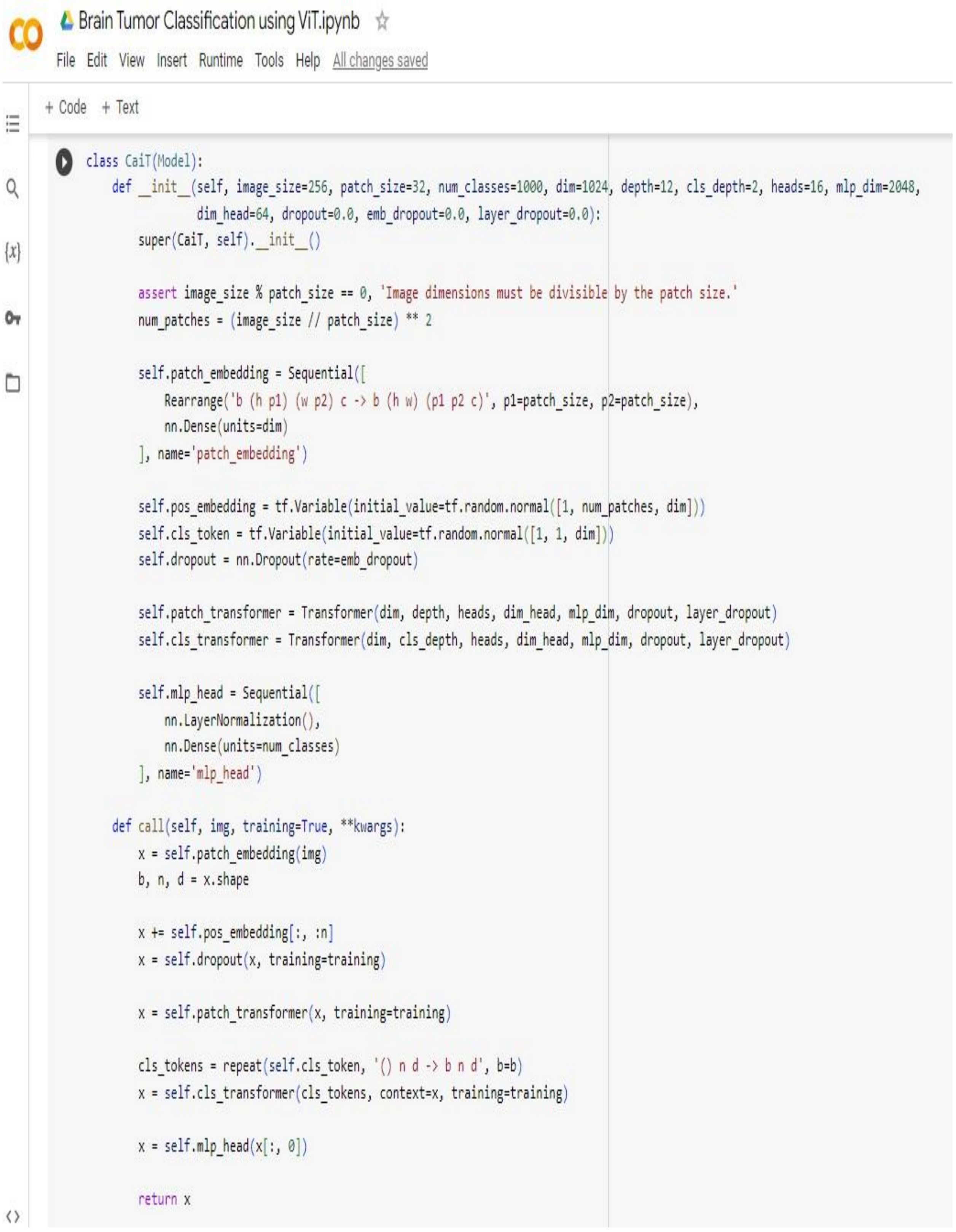
```
class DeepViT(Model):
    def __init__(self, image_size=256, patch_size=32, num_classes=1000, dim=1020, depth=6, heads=16, mlp_dim=2048,
                 pool='cls', dim_head=64, dropout=0.0, emb_dropout=0.0):
        super(DeepViT, self).__init__()

        assert image_size % patch_size == 0, 'Image dimensions must be divisible by the patch size.'
        num_patches = (image_size // patch_size) ** 2
        assert pool in {'cls', 'mean'}, 'pool type must be either cls (cls token) or mean (mean pooling)'

        self.patch_embedding = Sequential([
            Rearrange('b (h p1) (w p2) c -> b (h w) (p1 p2 c)', p1=patch_size, p2=patch_size),
            nn.Dense(units=dim)
        ], name='patch_embedding')

        self.pos_embedding = tf.Variable(initial_value=tf.random.normal([1, num_patches + 1, dim]))
        self.cls_token = tf.Variable(initial_value=tf.random.normal([1, 1, dim]))
        self.dropout = nn.Dropout(rate=emb_dropout)

        self.transformer = Transformer(dim, depth, heads, dim_head, mlp_dim, dropout)

        self.pool = pool

        self.mlp_head = Sequential([
            nn.LayerNormalization(),
            nn.Dense(units=num_classes)
        ], name='mlp_head')

    def call(self, img, training=True, **kwargs):
        x = self.patch_embedding(img)
        b, n, d = x.shape

        cls_tokens = repeat(self.cls_token, '() n d -> b n d', b=b)
        x = tf.concat([cls_tokens, x], axis=1)
        x += self.pos_embedding[:, :(n + 1)]
        x = self.dropout(x, training=training)

        x = self.transformer(x, training=training)

        if self.pool == 'mean':
            x = tf.reduce_mean(x, axis=1)
        else:
            x = x[:, 0]

        x = self.mlp_head(x)
```

Figure 16 :- DeepViT model Code


Brain Tumor Classification using ViT.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

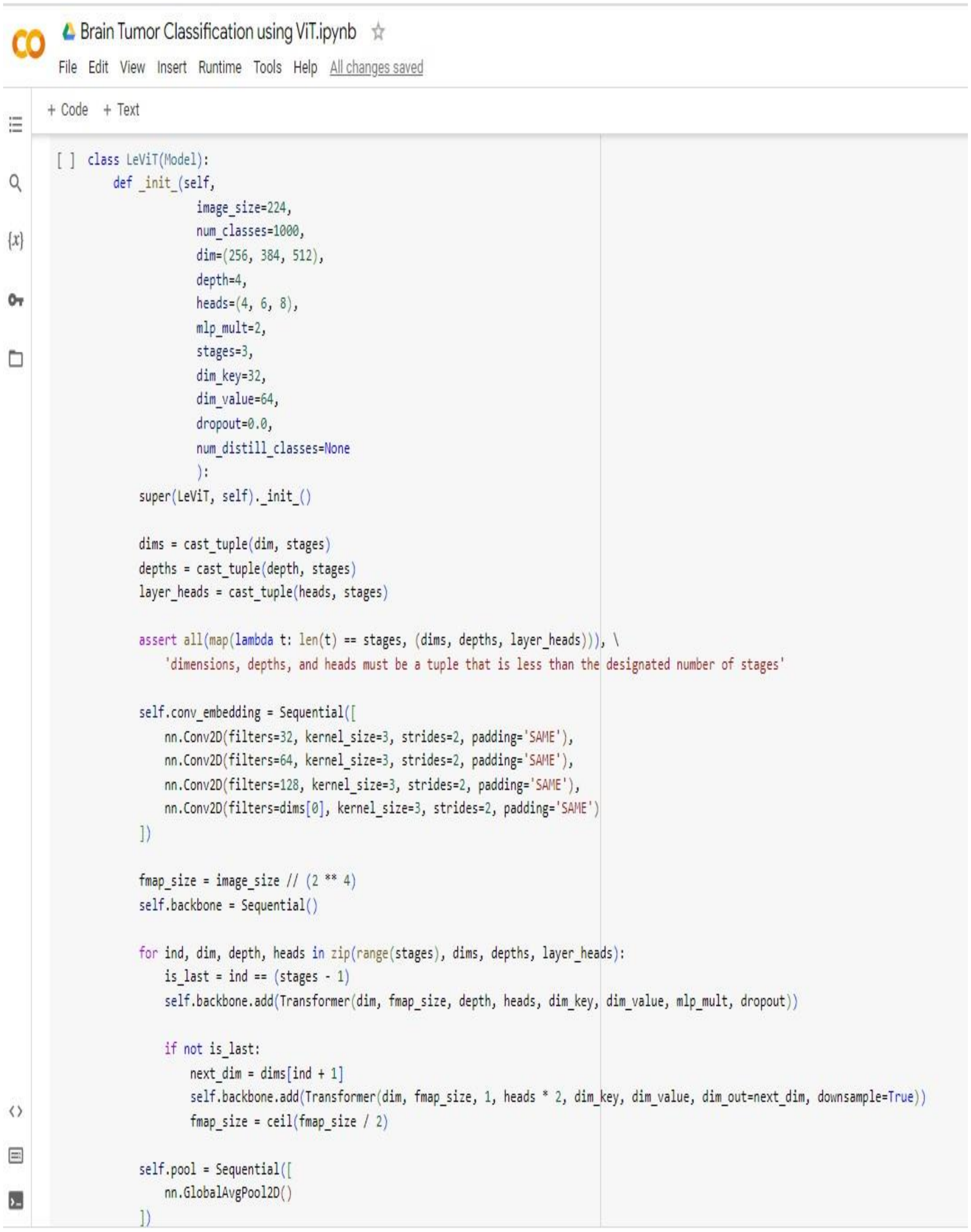
```
class MobileViT(Model):
    def _init_(self,
                image_size=(256, 256),
                dims=[96, 120, 144],
                channels=[16, 32, 48, 48, 64, 64, 80, 80, 96, 96, 384],
                num_classes=1000,
                expansion=4,
                kernel_size=3,
                patch_size=(2, 2),
                depths=(2, 4, 3)
                ):
        super(MobileViT, self)._init_()
        assert len(dims) == 3, 'dims must be a tuple of 3'
        assert len(depths) == 3, 'depths must be a tuple of 3'

        ih, iw = image_size
        ph, pw = patch_size
        assert ih % ph == 0 and iw % pw == 0

        init_dim, *_, last_dim = channels

        self.conv1 = Conv_NxN_BN(init_dim, kernel_size=3, stride=2)

        self.stem = Sequential()
        self.stem.add(MV2Block(channels[0], channels[1], stride=1, expansion=expansion))
        self.stem.add(MV2Block(channels[1], channels[2], stride=2, expansion=expansion))
        self.stem.add(MV2Block(channels[2], channels[3], stride=1, expansion=expansion))
        self.stem.add(MV2Block(channels[2], channels[3], stride=1, expansion=expansion))

        self.trunk = []
        self.trunk.append([
            MV2Block(channels[3], channels[4], stride=2, expansion=expansion),
            MobileViTBlock(dims[0], depths[0], channels[5], kernel_size, patch_size, mlp_dim=int(dims[0] * 2))
        ])

        self.trunk.append([
            MV2Block(channels[5], channels[6], stride=2, expansion=expansion),
            MobileViTBlock(dims[1], depths[1], channels[7], kernel_size, patch_size, mlp_dim=int(dims[1] * 4))
        ])

        self.trunk.append([
            MV2Block(channels[7], channels[8], stride=2, expansion=expansion),
            MobileViTBlock(dims[2], depths[2], channels[9], kernel_size, patch_size, mlp_dim=int(dims[2] * 4))
        ])
```

Figure 17 :- MobileViT model Code


Brain Tumor Classification using ViT.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

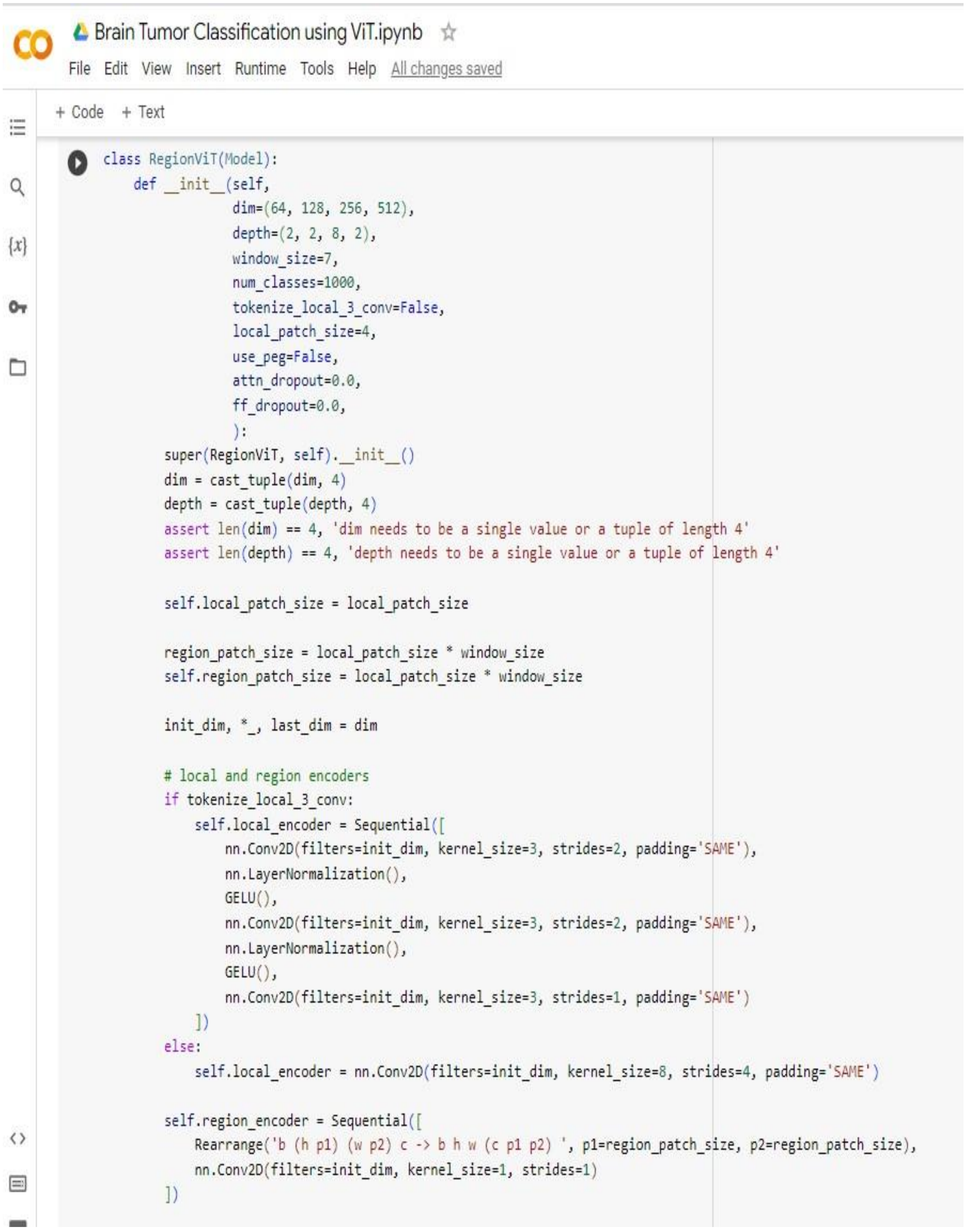
```python
class CaiT(Model):
    def __init__(self, image_size=256, patch_size=32, num_classes=1000, dim=1024, depth=12, cls_depth=2, heads=16, mlp_dim=2048,
                 dim_head=64, dropout=0.0, emb_dropout=0.0, layer_dropout=0.0):
        super(CaiT, self).__init__()

        assert image_size % patch_size == 0, 'Image dimensions must be divisible by the patch size.'
        num_patches = (image_size // patch_size) ** 2

        self.patch_embedding = Sequential([
            Rearrange('b (h p1) (w p2) c -> b (h w) (p1 p2 c)', p1=patch_size, p2=patch_size),
            nn.Dense(units=dim)
        ], name='patch_embedding')

        self.pos_embedding = tf.Variable(initial_value=tf.random.normal([1, num_patches, dim]))
        self.cls_token = tf.Variable(initial_value=tf.random.normal([1, 1, dim]))
        self.dropout = nn.Dropout(rate=emb_dropout)

        self.patch_transformer = Transformer(dim, depth, heads, dim_head, mlp_dim, dropout, layer_dropout)
        self.cls_transformer = Transformer(dim, cls_depth, heads, dim_head, mlp_dim, dropout, layer_dropout)

        self.mlp_head = Sequential([
            nn.LayerNormalization(),
            nn.Dense(units=num_classes)
        ], name='mlp_head')

    def call(self, img, training=True, **kwargs):
        x = self.patch_embedding(img)
        b, n, d = x.shape

        x += self.pos_embedding[:, :n]
        x = self.dropout(x, training=training)

        x = self.patch_transformer(x, training=training)

        cls_tokens = repeat(self.cls_token, '() n d -> b n d', b=b)
        x = self.cls_transformer(cls_tokens, context=x, training=training)

        x = self.mlp_head(x[:, 0])

        return x
```

Figure 18 :- CaiT model Code


Brain Tumor Classification using ViT.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

```
class LeViT(Model):
    def _init_(self,
               image_size=224,
               num_classes=1000,
               dim=(256, 384, 512),
               depth=4,
               heads=(4, 6, 8),
               mlp_mult=2,
               stages=3,
               dim_key=32,
               dim_value=64,
               dropout=0.0,
               num_distill_classes=None
               ):
        super(LeViT, self)._init_()

        dims = cast_tuple(dim, stages)
        depths = cast_tuple(depth, stages)
        layer_heads = cast_tuple(heads, stages)

        assert all(map(lambda t: len(t) == stages, (dims, depths, layer_heads))), \
            'dimensions, depths, and heads must be a tuple that is less than the designated number of stages'

        self.conv_embedding = Sequential([
            nn.Conv2D(filters=32, kernel_size=3, strides=2, padding='SAME'),
            nn.Conv2D(filters=64, kernel_size=3, strides=2, padding='SAME'),
            nn.Conv2D(filters=128, kernel_size=3, strides=2, padding='SAME'),
            nn.Conv2D(filters=dims[0], kernel_size=3, strides=2, padding='SAME')
        ])

        fmap_size = image_size // (2 ** 4)
        self.backbone = Sequential()

        for ind, dim, depth, heads in zip(range(stages), dims, depths, layer_heads):
            is_last = ind == (stages - 1)
            self.backbone.add(Transformer(dim, fmap_size, depth, heads, dim_key, dim_value, mlp_mult, dropout))

            if not is_last:
                next_dim = dims[ind + 1]
                self.backbone.add(Transformer(dim, fmap_size, 1, heads * 2, dim_key, dim_value, dim_out=next_dim, downsample=True))
                fmap_size = ceil(fmap_size / 2)

        self.pool = Sequential([
            nn.GlobalAvgPool2D()
        ])
```

Figure 19 :- LeViT model Code


Brain Tumor Classification using ViT.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

```
class RegionViT(Model):
    def __init__(self,
                 dim=(64, 128, 256, 512),
                 depth=(2, 2, 8, 2),
                 window_size=7,
                 num_classes=1000,
                 tokenize_local_3_conv=False,
                 local_patch_size=4,
                 use_peg=False,
                 attn_dropout=0.0,
                 ff_dropout=0.0,
                 ):
        super(RegionViT, self).__init__()
        dim = cast_tuple(dim, 4)
        depth = cast_tuple(depth, 4)
        assert len(dim) == 4, 'dim needs to be a single value or a tuple of length 4'
        assert len(depth) == 4, 'depth needs to be a single value or a tuple of length 4'

        self.local_patch_size = local_patch_size

        region_patch_size = local_patch_size * window_size
        self.region_patch_size = local_patch_size * window_size

        init_dim, *_, last_dim = dim

        # local and region encoders
        if tokenize_local_3_conv:
            self.local_encoder = Sequential([
                nn.Conv2D(filters=init_dim, kernel_size=3, strides=2, padding='SAME'),
                nn.LayerNormalization(),
                GELU(),
                nn.Conv2D(filters=init_dim, kernel_size=3, strides=2, padding='SAME'),
                nn.LayerNormalization(),
                GELU(),
                nn.Conv2D(filters=init_dim, kernel_size=3, strides=1, padding='SAME')
            ])
        else:
            self.local_encoder = nn.Conv2D(filters=init_dim, kernel_size=8, strides=4, padding='SAME')

        self.region_encoder = Sequential([
            Rearrange('b (h p1) (w p2) c -> b h w (c p1 p2) ', p1=region_patch_size, p2=region_patch_size),
            nn.Conv2D(filters=init_dim, kernel_size=1, strides=1)
        ])
```

Figure 20 :- RegionViT model Code

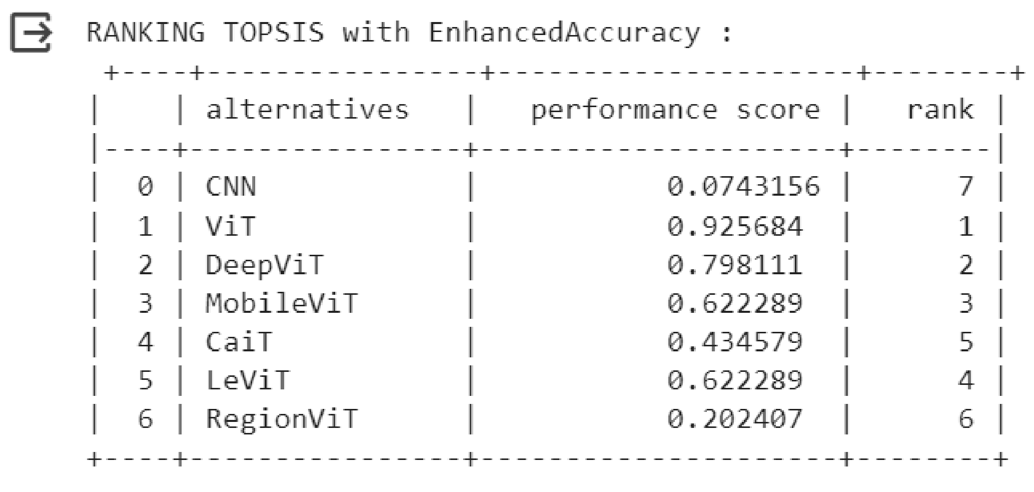
RESULTS & DISCUSSION

We take a brain tumor dataset and add CNN model, VIT model and 5 advanced VIT models then we calculate Accuracy, precision, recall, specificity, F1 etc. Here is the table-

Model Name	Accuracy	Recall	Specificity	Precision	F1	GM	FPR
CNN	0.80	0.80	0.78	0.60	0.71	0.9	0.22
ViT	0.9	0.89	0.91	0.85	0.87	0.9	0.18
DeepViT	0.88	0.89	0.88	0.80	0.84	0.88	0.12
MobileViT	0.86	0.84	0.88	0.80	0.82	0.86	0.12
CaitViT	0.84	0.83	0.85	0.75	0.79	0.86	0.22
LeViT	0.86	0.84	0.88	0.80	0.82	0.86	0.18
RegionVit	0.80	0.86	0.78	0.60	0.71	0.88	0.12

Table 1:- Model Result

Since there are multiple criteria they are giving different outputs from different models. So it is difficult to select the best model so we use the multi criteria decision making method as TOPSIS to find the best model.



```
RANKING TOPSIS with EnhancedAccuracy :
 +----+---------------+--------------------+--------+
|    | alternatives   |   performance score |   rank |
|----+---------------+--------------------+--------|
|  0 | CNN            |          0.0743156 |      7 |
|  1 | ViT            |          0.925684  |      1 |
|  2 | DeepViT        |          0.798111  |      2 |
|  3 | MobileViT      |          0.622289  |      3 |
|  4 | CaiT           |          0.434579  |      5 |
|  5 | LeViT          |          0.622289  |      4 |
|  6 | RegionViT      |          0.202407  |      6 |
+----+---------------+--------------------+--------+
```

Table 2:- Result Analysis

According to the table the best model is the VIT for the current problem in our project.

CONCLUSION

The ViT-based project presented in this study has demonstrated the effectiveness and versatility of Vision Transformer architectures in addressing the specific goals outlined. The implementation and experimentation phases have provided valuable insights into the capabilities of ViTs in handling complex visual data and performing tasks such as image classification or object detection. The results indicate that Vision Transformers exhibit promising performance in comparison to traditional convolutional neural networks (CNNs) for certain types of data and tasks. The self-attention mechanism employed in ViTs has proven to capture long-range dependencies in visual data, contributing to their success in image recognition tasks. However, it is crucial to acknowledge the project's limitations, such as the computational resources required for training large ViT models and potential challenges in fine-tuning for specific domains. Addressing these issues could enhance the practicality and applicability of Vision Transformers in real-world scenarios. Looking ahead, further research is warranted to explore optimization strategies for training Vision Transformers on diverse datasets and to investigate their adaptability to different domains. Additionally, understanding the interpretability of ViT models and addressing concerns related to model explainability will be pivotal for broader adoption in critical applications.

FUTURE WORK

In order to propel the development of vision transformers in medical domain, we propose various significant future directions. One direction is that, several papers in this study have done a comparative analysis of CNNs and ViT, thus, it can be inferred that transformer models have given better results in terms of efficiency, computational cost, and accuracy. Since these models have achieve such outcomes where researchers are talking about CNN’s being replaceable, vision transformers require more research and implementation as they are unraveling a path towards more resource-efficient models. The paper demonstrates the significance of using vision transformers on medical images. The future directions of research involve working with more heterogeneous data sets, and more extensive comparisons with other models to give validity to the proposed transformer models. Next, the studies, may they be related to any category or modality, are using different datasets, hence there cannot be a comparative analysis in terms of the proposed vision transformer models. Therefore, we should work on creating a benchmark dataset.

BIBLIOGRAPHY

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.

[2] Havaei M, Davy A, Warde-Farley D, et al. Brain tumor segmentation with deep neural networks. Med Image Anal. 2017;35:18–31.

[3] Agravat RR, Raval MS. A survey and analysis on automated glioma brain tumor segmentation and overall patient survival prediction. Arch Comput Methods Eng. 2021;28:4117–4152

[4] Liu Z, Shen L. Medical image analysis based on transformer: a review. arXiv preprint arXiv. 2022;2208:06643.

[5] Tensorflow ViT models : https://github.com/taki0112/vit-tensorflow/tree/main/vit_tensorflow

[6] He K, Gan C, Li Z, et al. Transformers in medical image analysis: a review. Intell Med. 2022;3(1):59–78.

[7] Shamshad F, Khan S, Zamir SW, et al. Transformers in medical imaging: a survey. arXiv preprint arXiv. 2022;2201:09873.

[8] Parvaiz A, Khalid MA, Zafar R, Ameer H, Ali M, Fraz MM. Vision transformers in medical computer vision–a contemplative retrospection. arXiv preprint arXiv. 2022; 2203:15269.

[9] Henry EU, Emebob O, Omonhinmin CA. Vision transformers in medical imaging: a review. arXiv preprint arXiv. 2022;2211:10043.

[10] Ghosh A, Thakur S. Review of brain tumor MRI image segmentation methods for BraTS challenge dataset. In: Proc. International Conference on Cloud Computing. Data Science and Engineering (Confluence); 2022:405–410

[11] Daquan Zhou , Bingyi Kang , Xiaojie Jin , Linjie Yang , Xiaochen Lian , Zihang Jiang , Qibin Hou , Jiashi Feng . DeepViT: Towards Deeper Vision Transformer ; a rXiv:2103.11886v4 [cs.CV] 19 Apr 2021

[12] Ben Graham, Alaaeldin El-Nouby, Hugo Touvron, Pierre Stock, Armand Joulin, Herve J ´ egou, Matthijs Douze. LeViT: a Vision Transformer in ConvNet's Clothing for Faster Inference ; 20 Aug 2021