



**NORTH SOUTH UNIVERSITY**  
**DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING**

---

# **“A Comparative Study On Visual Question Answering”**

**Project Report**  
**Spring 2022**  
**CSE499A: Senior Design Project I**  
**Section: 07**  
**Submitted By: Group: 09**

<b>Group Members</b>	<b>ID</b>
<b>Ashfaq Uddin Ahmed</b>	<b>1911848042</b>
<b>S M Gazzali Arafat Nishan</b>	<b>1831513642</b>
<b>Tasfia Tabassum</b>	<b>1821391042</b>

**Submitted To Mohammad Ashrafuzzaman Khan (AZK)**  
**Date of submission: 27<sup>th</sup> April 2022**

# ACKNOWLEDGEMENT

First and foremost, we want to thank the Almighty for providing us with the strength to carry out our duties and finish the report. As part of the Bachelor of Science (BSc) curriculum, the 499A Senior Design project program is beneficial in bridging the gap between academic knowledge and real-world experience. This report was created to provide everyone with hands-on experience while also theoretical knowledge. We also want to thank all of the professors who have helped us complete the project with perfect comprehension by giving us technical expertise and spiritual support.

We must express our gratitude to our respected faculty member, **Dr. Mohammad Ashrafuzzaman Khan**, for his undivided attention and assistance in achieving this goal. Also, we owe a debt of gratitude to North South University's ECE department for offering us a course like CSE 499A, which allowed us to devote ourselves entirely to this project and see it through to completion.

We appreciate our friends and family's moral support in helping us carve out this endeavor and will continue to do so.

# ABSTRACT

VQA (Visual Inquiry Answering) is a relatively new activity that requires algorithms to reason about the visual content of a picture in order to respond to a natural language question. VQA requires an algorithm to respond to text-based inquiries about photos. Because many open-ended replies comprise either a few words or a closed set of options that may be supplied in a multiple-choice format, VQA adapts itself to automated review. To put it simply, the Visual Question Answering (VQA) job combines data processing issues with visual and linguistic processing challenges in order to answer basic 'common sense' questions regarding given images. VQA has piqued the interest of deep learning, computer vision, and natural language processing researchers. We have tried to examine the current state of VQA in terms of problem formulation, existing datasets, evaluation metrics, and algorithms. We reviewed existing algorithms for VQA. (CNN, RNN). To figure out where VQA its image understanding stands. Our proposed model was VIT, Vision Transformer. We applied multiple VIT approaches and found good Accuracies. The highest accuracy among these models is as high as 97%. It recognizes an image that seems identical to the original, VQA is supposed to do the same, but it must offer a different answer to the same query. VQA can aid in the development of user trust in machines.

# Index

---

<b>Introduction</b>	4
<b>Literature Review</b>	5
<b>Methodology</b>	12
<b>Results</b>	20
<b>Conclusion</b>	33
<b>References</b>	34

---

# CHAPTER 1: INTRODUCTION

## Introduction

Computer vision is a branch of artificial intelligence (AI) that allows computers and systems to extract meaningful information from digital photos, videos, and other visual inputs — and then act or make recommendations based on that knowledge. If artificial intelligence allows computers to think, computer vision will enable them to see, watch, and comprehend. Computer vision functions similarly to human vision, except that humans have a head start. Human vision benefits from lifetimes of context to train how to discern objects apart, how far away they are, if they are moving, and if something is wrong with a picture. Computer vision trains computers to execute these duties. Still, it must do so in a much shorter time, using cameras, data, and algorithms rather than retinas, optic nerves, and the visual cortex. Because a system trained to inspect items or monitor a manufacturing asset can examine thousands of products or processes per minute, detecting imperceptible faults or anomalies can swiftly outperform human capabilities.

Language and vision challenges such as picture captioning and visual question answering (VQA) have gained prominence in recent years as the computer vision research community has shifted its focus away from "bucketed" recognition and toward tackling multi-modal problems. In the VQA dataset, for example, this effect has been detected in picture captioning and visual question answering.

VQA is a relatively new problem in computer vision and natural language processing that has caught the interest of academics in deep learning, computer vision, and natural language processing. To answer text-based photo inquiries, VQA needs an algorithm. VQA adapts to automated review since many open-ended responses are either a few words or a closed set of possibilities that may be provided in a multiple-choice style.

---

## CHAPTER 2: LITERATURE REVIEW

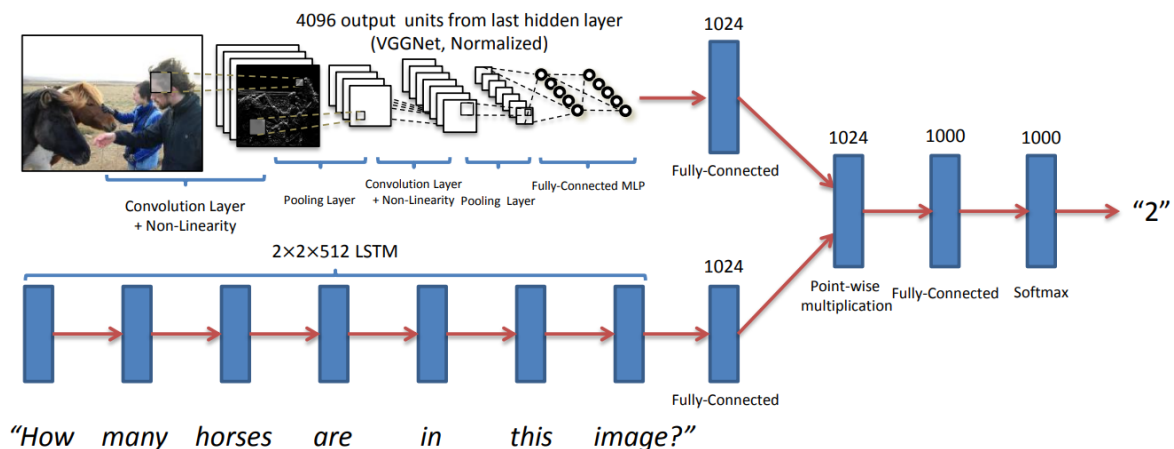
### EXISTING LITERATURE EXPLANATION

#### Paper - 1: VQA: Visual Question Answering

In this research, the suggested task entails open-ended, free-form questions and human answers. Their purpose was to broaden the range of information and reasoning skills required to deliver the right responses. This is crucial for success in this more demanding and unrestricted work. They used a VQA dataset that was two orders of magnitude larger than what had been used in prior VQA experiments. Other related actions are tied to the VQA work they submitted. In similar work, they recommended pairing an LSTM for the question with a CNN for the image to get a response. At each time step in their model, the CNN image features are used to condition the LSTM question representation, and the final LSTM hidden state is used to decode the response phrase sequentially. In contrast, the model proposed in this paper looks into "late fusion," in which the LSTM question representation and CNN image features are computed separately, fused via element-wise multiplication, and then passed through fully connected layers to generate a softmax distribution over output answer classes[5].

In this research, they gathered questions and responses in Chinese, which were then translated into English by humans for COCO pictures. Finally, they were able to use COCO captions to automatically produce four different sorts of questions (item, count, color, and location). They also used multiple baselines and unique methodologies to investigate the complexity of the VQA dataset for MS COCO pictures. On the VQA train+val, They practiced. All human accuracies are on test-standard, machine accuracies are on test-dev, and results incorporating human captions are on test-dev unless otherwise noted. They went on to construct a two-channel vision (picture) + language (question) model that ends with a softmax over K potential outputs. To get a

1024-dim embedding for the query, an LSTM with one hidden layer is utilized. The embedding derived from the LSTM is a concatenation of the LSTM's hidden layer's last cell state and last hidden state representations (each 512-dim)[5].



**Figure 1.** Vanilla VQA using VGGNet+LSTM.

Their best model (deeper LSTM Q + norm I) performed better. The questions are encoded using a two-layer LSTM, while the visuals are encoded using VGGNet's final hidden layer. After that, the picture characteristics are '2' normalized. To get distribution across replies, both the question and picture characteristics are translated to a common space and fused using element-wise multiplication, which is then sent via a fully connected layer followed by a softmax layer. To generate a unique embedding, the picture and question embeddings are joined. They simply concatenated the BoW Q and I embeddings for the BoW Q + I approach. The picture embedding is first changed to 1024-dim by a fully-connected layer + tanh nonlinearity to match the LSTM embedding of the question in LSTM Q + I and deeper LSTM Q + norm I approaches. Their best model (deeper LSTM Q + norm I, chosen using VQA test-dev accuracies on VQA test standard) has an accuracy of 58.16 percent (open-ended) / 63.09 percent (closed-ended) (multiple-choice). We can observe that their model outperforms both the vision-alone and language-alone baselines substantially[5].

## **Paper - 2: Making the V in VQA Matter: Elevating the Role of Image Understanding in Visual Question Answering**

In this research, For the objective of Visual Question Answering (VQA), they suggested negating these linguistic priors by making vision (the V in VQA) significant! Specifically, they attempted to balance the popular VQA dataset by gathering complimentary photos, so that each question in their balanced dataset was connected with a pair of comparable images that resulted in two possible responses to the question. By design, their dataset is more balanced than the original VQA dataset, with roughly twice the number of image-question pairs. They also used their balanced dataset to test a variety of state-of-the-art VQA models. On their balanced dataset, all models perform much worse, indicating that these models have learned to exploit linguistic priors. This discovery provides the first empirical proof for what appears to be a qualitative feeling among practitioners.

Several recent papers have suggested ways for creating 'explanations' for deep learning models' predictions, which are often 'black-box' and uninterpretable. provides a natural language (sentence) description for picture categories 'Visual explanations,' such as geographical maps superimposed on photographs, were provided to indicate the places that the model focused on when making predictions. We offer a third explanation modality in this paper: counter-examples, which are cases that the model feels are similar to but not identical to the category predicted by the model. Their hypothesis is that just training a model to properly answer questions on our balanced dataset would drive the machine to focus more on the visual signal, as the linguistic signal has been degraded. They combined the dataset using VQA modeling. It incorporates a CNN embedding of the image, an LSTM embedding of the question, a pointwise multiplication to combine the two embeddings, and then a multi-layer perceptron classifier to predict a probability distribution. This is a new attention-based VQA model that predicts a response by 'co-attending' to both the picture and the question. It uses the co-attention process to model the question and, as a result, the picture in a hierarchical way: at the word, phrase, and complete question levels.[3].



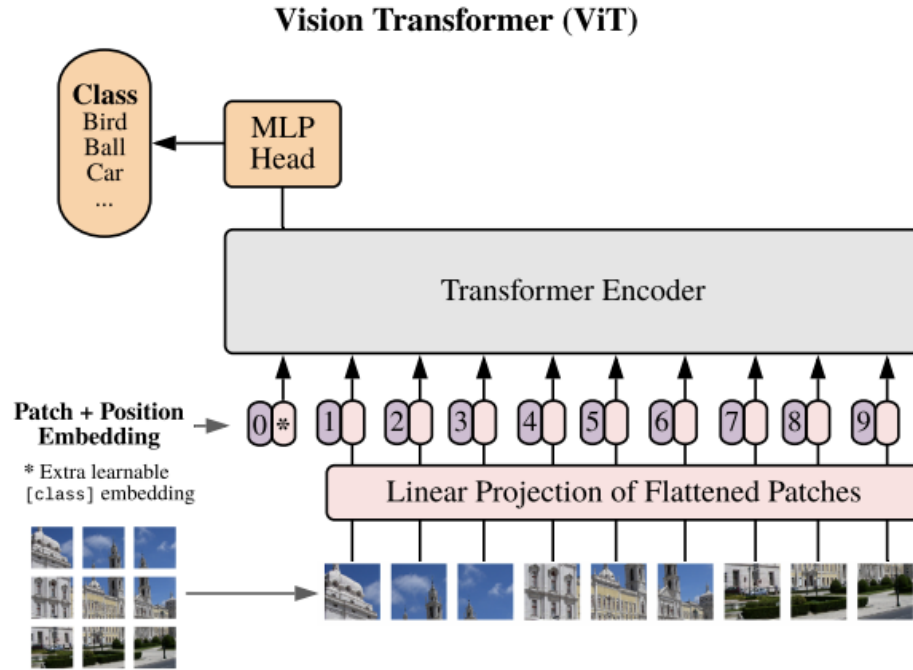
They also evaluated the accuracy of Multimodal Compact Bilinear Pooling (MCB) [6] across response categories. First, they found that the accuracy for the answer type "yes/no" is significantly lower than that of the Hierarchical Co-attention (HieCoAtt) models. This study also revealed that linguistic priors existing in the UU to UB transition decline considerably (10.8% for MCB and 12.4% for HieCoAtt). Unbalanced VQA datasets result in comparable accuracies for all state-of-the-art VQA models especially in the "yes/no" and "number" answer-type questions), making drastically different models almost indistinguishable in terms of their accuracies for these answer-types.

### **Paper - 3: AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE**

In this research, They intended to illustrate that CNNs are not required for image classification tasks and that a pure transformer applied directly to sequences of picture patches may perform extremely well. When compared to state-of-the-art convolutional networks, Vision Transformer (ViT) achieves great outcomes while requiring significantly fewer computing resources to train. In natural language processing, self-attention-based architectures, particularly Transformers, have become the model of choice (NLP). Pre-training on a huge text corpus and then fine-tuning on a smaller task-specific dataset is the most common method.

On numerous image recognition benchmarks, ViT outperformed the competition. The top model, in particular, achieves 88.55 percent accuracy on ImageNet, 90.72 percent on ImageNet-Real, 94.55 percent on CIFAR-100, and 77.63 percent on the VTAB suite of 19 tasks. Vaswani et al. (2017) developed transformers for machine translation, and they have subsequently become the gold standard in many NLP applications. Models based on big Transformers are frequently pre-trained on huge corpora before being fine-tuned for the job at hand: The pre-training job for BERT is a denoising self-supervised activity, whereas the pre-training task for the GPT line of study is language modeling. In a naive application of self-attention to pictures, each pixel would have to pay attention to every other pixel. This does not scale to practical input sizes because of the quadratic cost in the number of pixels. As a result, numerous approximations have been explored in the past to use Transformers in the context of image processing[7].

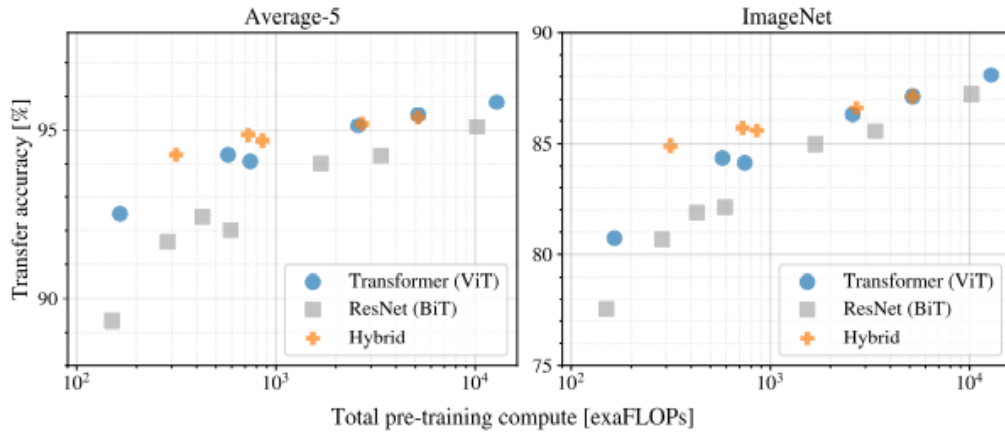
Their article joins a growing list of studies that look at picture identification at sizes beyond the typical ImageNet dataset. Using extra data sources provides for cutting-edge outcomes on a traditional benchmark. They mainly concentrated on these two latter datasets in their research as well, but instead of using ResNet-based models, they utilized Transformers instead.



**Figure 2:** The idea of ViT.

They simply took an image and split it into fixed-size patches, linearly embedded each one, added position embeddings, and fed the resultant vector sequence to a typical Transformer encoder. They used the conventional technique of adding an extra learnable "classification token" to the sequence in order to accomplish classification. Vaswani was the inspiration for the Transformer encoder artwork. The Transformer encoder is made up of multiple layers of multi-headed self-attention MSA and MLP blocks that alternate. Before each block, Layernorm (LN) is applied, and residual connections are applied after each block. We found that Vision Transformer has much less image-specific inductive bias than CNNs. In CNN, locality, two-dimensional neighborhood structure, and translation equivariance are baked into each layer throughout the whole model. In ViT, only MLP layers are local and translationally equivariant, while the self-attention layers are global. The input sequence could be formed from feature maps of a CNN instead of raw picture patches. Patch embedding projection is applied to patches extracted from a CNN feature map in this hybrid model[7].

They also explored ResNet, Vision Transformer (ViT), and the hybrid's representation learning capabilities. They also compared their biggest models – the ViT-H/14 and ViT-L/16 – to published state-of-the-art CNNs. First, they pre-trained ViT models using ImageNet, ImageNet-21k, and JFT300M datasets of increasing size. They aimed to adjust three basic regularization parameters – weight decay, dropout, and label smoothing – to improve performance on smaller datasets[7].



**Figure 3:** Performance versus cost for different architectures ResNets, Vision Transformers, and hybrids.

In their research, Vision Transformers outperformed ResNets with the same computational budget. For lower model sizes, hybrids outperformed pure Transformers, but the difference fades for bigger models. On ImageNet, BiT CNNs outperform ViT, but the difference vanishes for larger models. ViT overtook the larger datasets. Self-attention allowed ViT to integrate information throughout the full dataset. even in the lowest levels of the picture, They looked into how far they could go. This functionality is utilized by the network. They calculated, in particular, the average distance across which information is transmitted in picture space integrated, depending on the weights of attention[7].

---

## CHAPTER 3: METHODOLOGY

### 3.1 Introduction

This chapter provides a chronological outline of the various elements of the work. It primarily covers the work's ideas, methodology, and step-by-step procedure. In this chapter, we will also discuss the motivation we thought for implementing VQA using deep learning and computer vision. We will also discuss in this chapter why we have chosen the VQA field apart from all other fields to work in it.

### 3.2 Motivation for our project

Computer vision is an area of artificial intelligence, as we all know. The goal of computer vision is to teach a computer how to "understand" a scene or image's qualities. Computer vision is a technology that is gaining traction, and it is critical that everyone involved in technology grasp the potential it offers as well as its current limits. Traditional computer vision is concerned with image and video processing, with the goal of reliably extracting information from pictures and videos. The human vision system, which is our most complex sense, is the driving force behind the development of computer vision. As a result, we examine real-world applications ranging from factory inspection systems to autonomous vehicles, from license plate recognition to robot interaction with the environment, and from face recognition to augmented reality. Visual Question Answering (VQA) is a relatively new problem in computer vision and natural language processing that has piqued the interest of deep learning, computer vision, and natural language processing communities. VQA requires an algorithm to respond to text-based inquiries regarding photos. Additional datasets have been provided since the initial VQA dataset was released in 2014, and several techniques have been developed.

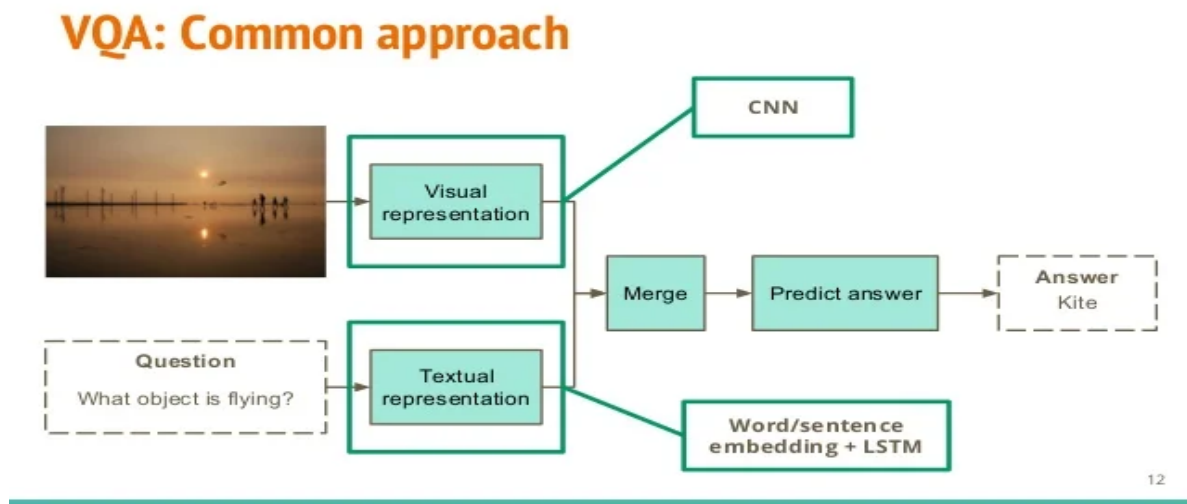
In terms of issue formulation, existing datasets, assessment criteria, and algorithms, we intend to critically analyze the current status of VQA. We're particularly interested in learning about the limits of existing datasets in terms of their capacity to properly train and evaluate VQA algorithms. Then, instead of using typical LSTM-CNN, we aim to thoroughly analyze current VQA algorithms such as Vision Transformer(Vit) and other transformers. Finally, we'd want to see what future avenues VQA and image comprehension research may go. In this study, we will attempt to acquire access to ViT's self-attention layer, which allows us to embed information globally over the whole picture. The model will also be able

to learn from training data in order to encode the upward revision of picture patches in order to rebuild the image's structure.

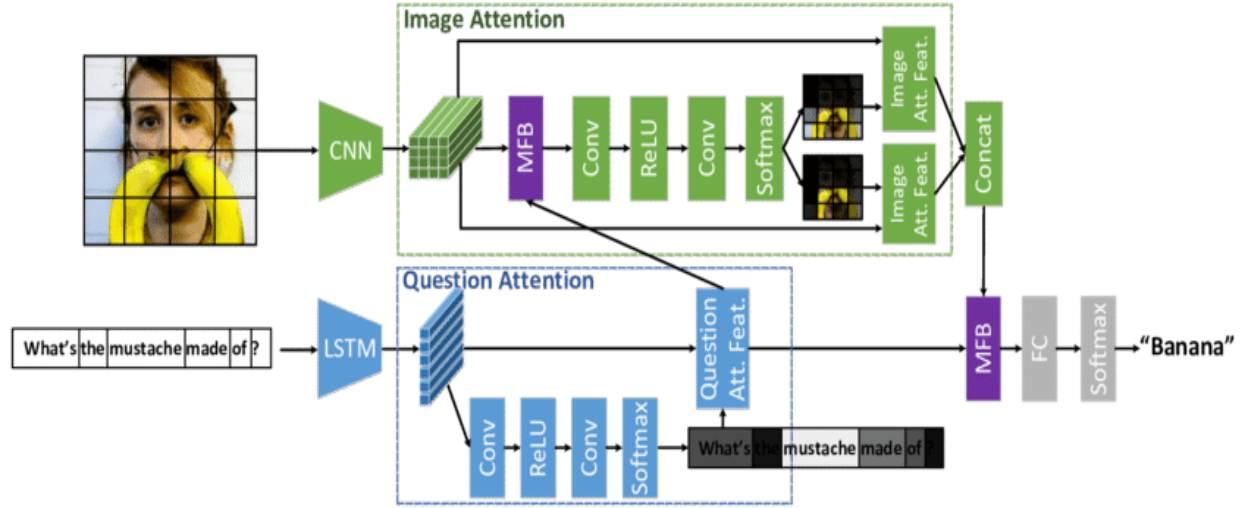
Our ambition is to create a visual Question Answering (VQA) system that will allow you to obtain an answer from a picture when you ask a question about it. It is simple, efficient, and trustworthy to create the VQA model. Our VQA project will assist you in better understanding computer vision and implementing it more rigorously. We will first study our dataset before attempting to preprocess it. Finally, we use Vision Transformer(Vit) to create a model for the VQA that will be able to provide as accurate an answer as possible to the query regarding the provided image. Taking into consideration all of the factors, our objective is to create a VQA system that is as efficient as feasible, GPU-light, user-friendly, and dependable in order to provide you with an appropriate answer.

### 3.3 Workflow

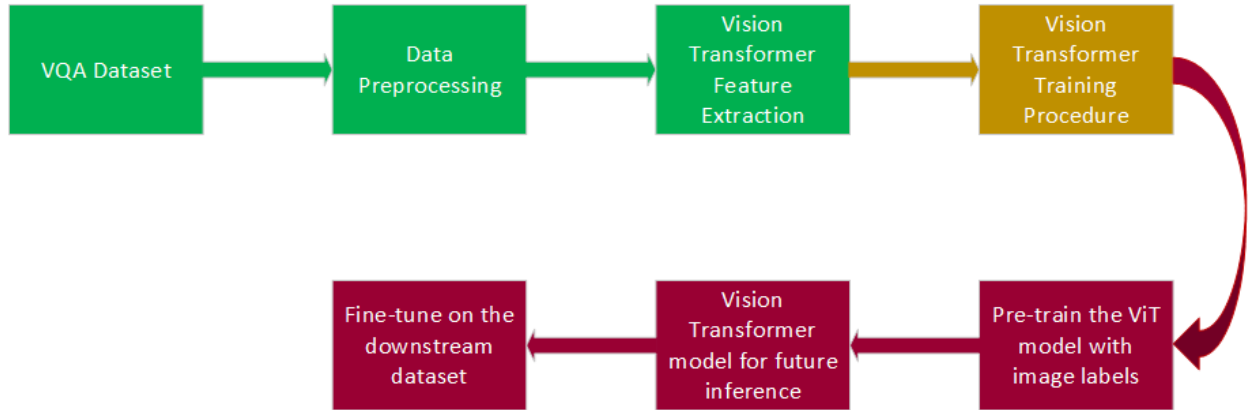
The most common approach to VQA has been ABC-CNN or VAnilla VQA, or to put it simply, an implementation of CNN and LSTM. This model uses convolution kernels, also known as configurable convolutions, and the kernels are set using the LSTM-extracted question characteristics. The convolution kernels are then convolved with the image to produce a feature map that lends more weight to the relevant sections of the image.



**Figure 4:** Visual question answering common approach using CNN-LSTM.



**Figure 5:** Visual question answering approach using CNN-LSTM attention.



**Figure 6:** Block diagram of the proposed approach of VQA.

We were not happy with the image's attention-grabbing part. Hence we opted to work on a different approach. So we used a Vision Transformer.

### 3.4 Dataset

VQA is a new dataset that contains open-ended picture questions. To answer these questions, you'll need a basic understanding of the eyesight, language, and common sense. The dataset's first version was released in October of 2015. In April 2017, VQA v2.0 was launched. The VQA dataset is a bit bigger. It provides 50,000 abstract cartoon pictures in addition to the 204,721 images from the COCO dataset.[1]The VQA

system uses image features of the picture and inference gained from textual questions to try to determine the proper answer to a question in the natural language given an image and a question in the natural language.[2] There are three questions for each image and ten answers per question, for a total of around 760K questions and over 10 million answers. The COCO dataset from Microsoft is the gold standard for measuring the performance of cutting-edge computer vision models. The COCO dataset is less well known among general practitioners, despite its widespread use in the computer vision research field. The COCO dataset, which stands for Common Objects in Context, is intended to represent a wide range of objects that humans come across on a daily basis. The COCO dataset is labeled, giving information for training supervised computer vision models that can recognize the dataset's common items. Of course, these models are far from flawless, thus the COCO dataset serves as a baseline for measuring the models' progress over time as a result of computer vision research.

The VQA's official website provides us with a dataset. The dataset was then downloaded to our local PC workstation. Then we uploaded the dataset to GoogleDrive and worked on Google Collab Pro. These are the zipped datasets, and the dataset is enormous. So, we have three types of images for training: 82,783 images for training and 40,504 images for validation. We have 81,434 images for the final one to test our models. To boost our model's training, we randomly selected 3000 photos from the training images and manually extracted questions and annotations from the JSON file. Then, for the VQA input questions, we have got some training questions in our dataset. The dataset contains almost 443,757 questions for the training images, 214,354 questions for the validation questions, and 447,793 questions for the testing questions. The annotations follow, which are essentially answers to the questions. There are 4,437,570 responses for annotations, and 2,143,540 entries for valid annotations. [3]

### 3.4.1 Train/Test/Validation Input Images Format:

The images in the dataset are all .jpg files. A unique image id is assigned to each image. JPG is a digital picture format that stores image data in compressed form. JPG pictures are quite small, with a compression ratio of 10:1. The JPG format saves vital image information. This is the most widely used picture format for exchanging photos and other images over the internet, as well as between mobile and desktop users.

### 3.4.2 Train/Test/Validation Input Question Format:

The JSON file format is used to hold the questions. The data structure for the questions is as follows:

```
{
  "info" : info,
  "task_type" : str,
  "data_type": str,
  "data_subtype": str,
  "questions" : [question],
  "license" : license
}
```

```
info {
  "year" : int,
  "version" : str,
  "description" : str,
```



```
"contributor" : str,
"url" : str,
"date_created" : datetime
}
```

```
license{
"name" : str,
"url" : str
}
```

```
question{
"question_id" : int,
"image_id" : int,
"question" : str
}
```

data\_subtype: the type of data subtype (for example, mscoco's train2014/val2014/test2015, abstract v002's train2015/val2015).[3]

### 3.4.3 Train/Test/Validation Input Annotations Format:

The JSON file format is used to hold the annotations. The data structure of the annotations format is as follows:

```
{
"info" : info,
"data_type": str,
"data_subtype": str,
"annotations" : [annotation],
"license" : license
}
```

```
info {
"year" : int,
"version" : str,
"description" : str,
"contributor" : str,
"url" : str,
"date_created" : datetime
}
```

```
license{
"name" : str,
"url" : str
}
annotation{
"question_id" : int,
"image_id" : int,
"question_type" : str,
"answer_type" : str,
"answers" : [answer],
"multiple_choice_answer" : str
}
```

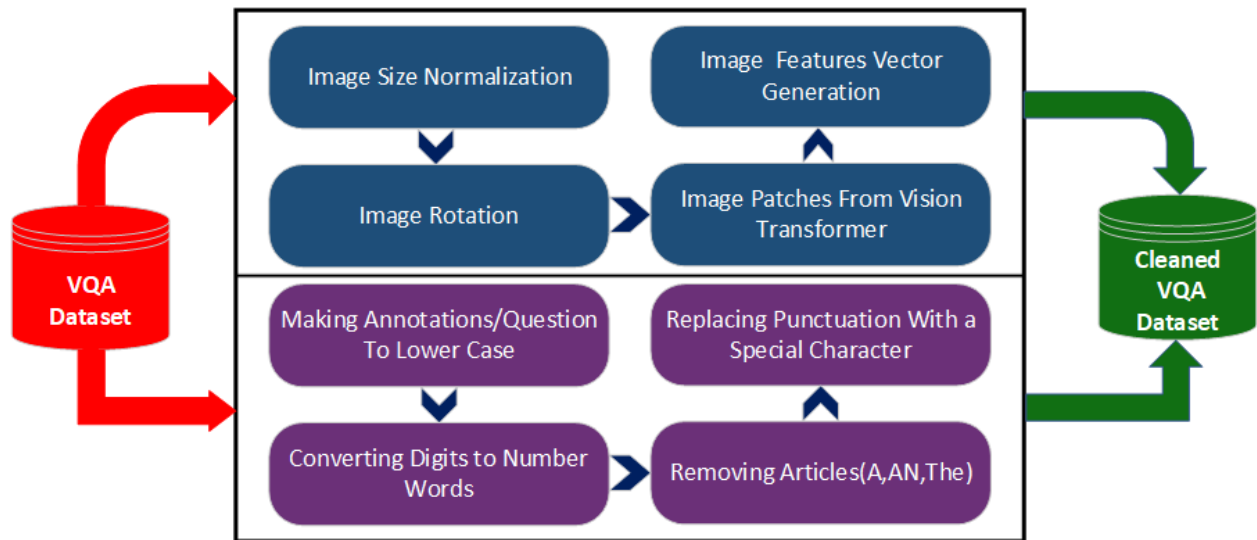
```

answer{
  "answer_id" : int,
  "answer" : str,
  "answer_confidence": str
}

```

data\_type: the image's source (ms coco or abstract v002).  
data\_subtype: the type of data subtype (for example, mscoco's train2014/val2014/test2015, abstract v002's train2015/val2015).  
question\_type: the question's type is specified by the question's first few words.  
answer\_type: the answer's type. "Yes/no," "number," and "other" are the only choices currently accessible.  
multiple\_choice\_answer: most frequent ground-truth answer.  
answer\_confidence: the subject's belief in his or her ability to answer the question.

### 3.5 Data Preprocessing



**Figure 7:** The overview of the data preprocessing process.

#### 3.5.1 Train/Validate and Test setup

We didn't divide the dataset since we have already got a train, test, and validation sets. Before transmitting the data for training and validation, it was preprocessed. Our initial approach is to train for 80% of the dataset and test for the remaining 20%, however, we have not finalized yet. The images were rescaled to a height of 32 pixels and width to 128 pixels and all images were transformed to grayscale. The train and validation batch size was set to 100. Optimizer was set to Adam with a learning rate: of  $1e-4$ , and weight decay: of  $1e-5$ .

#### 3.5.2 Evaluation Metrics

Our evaluation was based upon some factors like the format of the Results and Evaluation Code:

### 3.5.3 Overview of the Format of the Results:

Only one sort of work exists: the open-ended task. For the v2.0 releases of the VQA dataset, we employed an open-ended job with Python API and Evaluation Code. We did, in fact, create a directory called ms coco within this directory. Created directories within the ms coco directory with the names train2014, val2014, and test2015, respectively, and downloaded relevant pictures from the MS COCO website and stored them in related folders. The question files for v2.0 were obtained from the VQA download website, extracted, and placed in this folder.

```
results = [result]
result {
  "question_id": int,
  "answer": str
}
```

### 3.5.4 Evaluation Code

We developed a new assessment metric that is resistant to inter-human variation in response phrasing: Machine accuracies are averaged over all 10 chosen 9 sets of human annotators in order to be consistent with human accuracies.

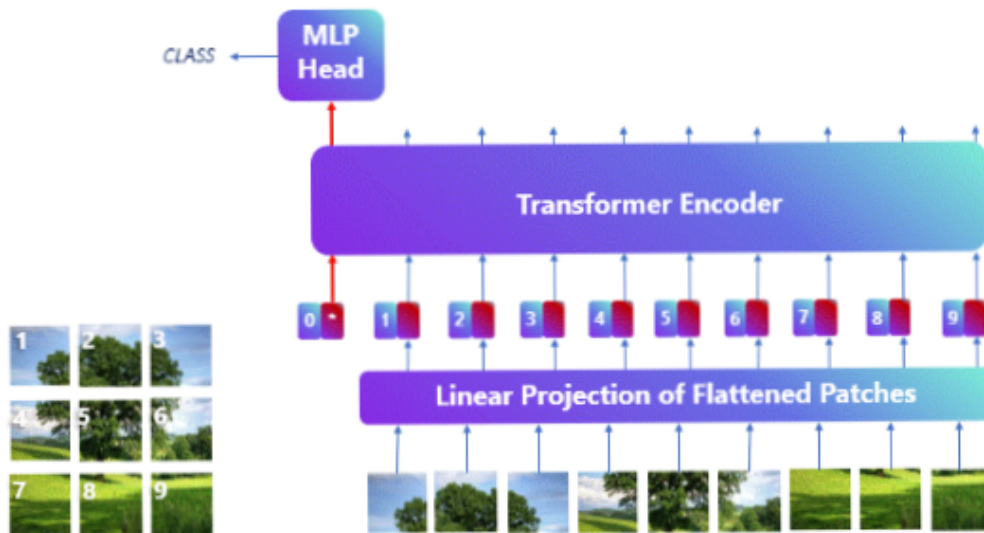
$$Accuracy(answer) = \min\left[\frac{\#humans\ who\ answer}{3}, 1\right]$$

We performed the following steps before analyzing machine-generated answers:

- Changing all letters to lowercase.
- Leaving out periods unless it's a decimal.
- Converting digits to number words.
- Articles to be removed (a, an, the).
- If an apostrophe is lacking from a contraction (for example, convert "didn't" to "didn't"), add it.
- All punctuation (save the apostrophe and colon) is replaced with a space character. We leave apostrophes in because they might wrongly transform possessives to plural, e.g., "boy's" to "boys," and colons in because they frequently allude to time, e.g., 1:35 pm. If a comma appears between numbers, no space is added, for example, convert 204,145 to 204145. (This processing step is also applied to ground truth responses.)

### 3.6 Vision Transformer

## Vision Transformers



**Figure 8:** The overview of Vision Transformer(ViT).

The Vision Transformer (ViT) has emerged as a viable alternative to convolutional neural networks (CNNs), which are the current state-of-the-art in computer vision and are widely utilized in image identification applications. In terms of computing efficiency and accuracy, ViT models exceed the present state-of-the-art (CNN) by almost a factor of four. When it comes to NLP models, these transformers have a high success rate, and they're currently being used on photos for image recognition tasks. ViT separates the pictures into visual tokens, whereas CNN employs pixel arrays. The visual transformer separates a picture into fixed-size patches, embeds each one appropriately, and passes positional embedding to the transformer encoder as an input. Furthermore, ViT models beat CNNs in terms of computing efficiency and accuracy by nearly four times.

ViT's self-attention layer allows you to embed information globally throughout the entire image. The model also uses training data to represent the relative locations of picture patches in order to recreate the image's structure.

---

## CHAPTER 4: RESULTS

In this section, we'll be looking at the test results. This part will focus on the ViT model in particular. We implemented four ViT models.

1. ViT: MLP MIXER
2. ViT: Using Hugging Face
3. ViT: Implementation with Pytorch 1
4. ViT: Implementation With Pytorch 2

Every model was able to identify the object in the image. We got an accuracy of about 85%-95% in our models.

We also wanted to implement a VQA model, so we used a Vanilla VQA model to see how our dataset works and whether a simple model can process it.

### 4.1 VQA Model

We first did a simple VQA model run on some data to see if our dataset was processable or not. We did do much in coding; however, the results we got were not very encouraging. We constructed a predictive model because this is simply for testing purposes. We don't want all of the terms in our vocabulary, just the ones that are more likely to appear or are familiar. As a result, we set the upper limit at 1000, corresponding to the glossary's first 1000 most often occurring words.

```
def freq_answers(questions, answers, image_id, upper_lim):
    freq_ans = defaultdict(int)
    for ans in answers:
        freq_ans[ans] +=1

    sort = sorted(freq_ans.items(), key=operator.itemgetter(1), reverse=True)[0:upper_lim]
    #print(sort)
    top_ans, top_freq = zip(*sort)
    #print(top_ans, top_freq)
    new_answers_train = list()
    new_questions_train = list()
    new_images_train = list()
    for ans, ques, img in zip(answers, questions, image_id):
        if ans in top_ans:
            new_answers_train.append(ans)
            new_questions_train.append(ques)
            new_images_train.append(img)
    return (new_questions_train, new_answers_train, new_images_train)

upper_lim = 1000
questions, answers, image_id = freq_answers(questions, answers, image_id, upper_lim)
questions_len, questions, answers, image_id = (list(t) for t in zip(*sorted(zip(questions_len, questions, answers, image_id))))
print (len(questions), len(answers), len(image_id))

105175 105175 105175
```

Figure 9: Selecting the first 1000

```

model = Dense(num_hidden_nodes_mlp, kernel_initializer='uniform', activation = 'tanh')(combined)
#model = Activation('tanh')(model)
model = Dropout(0.5)(model)

model = Dense(num_hidden_nodes_mlp, kernel_initializer='uniform', activation = 'tanh')(model)
#model = Activation('tanh')(model)
model = Dropout(0.5)(model)

model = Dense(num_hidden_nodes_mlp, kernel_initializer='uniform', activation = 'tanh')(model)
#model = Activation('tanh')(model)
model = Dropout(0.5)(model)

model = Dense(upper_lim)(model)
model = Activation("softmax")(model)

model = Model(inputs=[image_model.input, language_model.input], outputs=model)

```

Figure 10

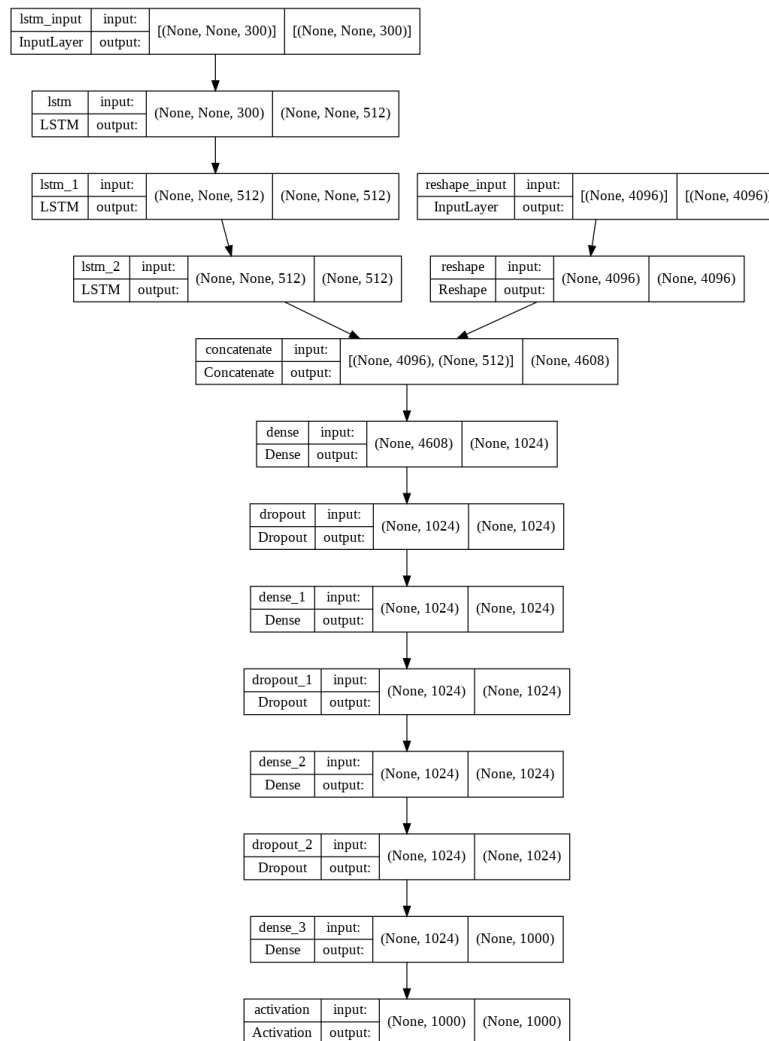


Figure 11

Layer (type)	Output Shape	Param #	Connected to
lstm_input (InputLayer)	[(None, None, 300)]	0	[]
lstm (LSTM)	(None, None, 512)	1665024	['lstm_input[0][0]']
reshape_input (InputLayer)	[(None, 4096)]	0	[]
lstm_1 (LSTM)	(None, None, 512)	2099200	['lstm[0][0]']
reshape (Reshape)	(None, 4096)	0	['reshape_input[0][0]']
lstm_2 (LSTM)	(None, 512)	2099200	['lstm_1[0][0]']
concatenate (Concatenate)	(None, 4608)	0	['reshape[0][0]', 'lstm_2[0][0]']
dense (Dense)	(None, 1024)	4719616	['concatenate[0][0]']
dropout (Dropout)	(None, 1024)	0	['dense[0][0]']
dense_1 (Dense)	(None, 1024)	1049600	['dropout[0][0]']
dropout_1 (Dropout)	(None, 1024)	0	['dense_1[0][0]']
dense_2 (Dense)	(None, 1024)	1049600	['dropout_1[0][0]']
dropout_2 (Dropout)	(None, 1024)	0	['dense_2[0][0]']
dense_3 (Dense)	(None, 1000)	1025000	['dropout_2[0][0]']
activation (Activation)	(None, 1000)	0	['dense_3[0][0]']
Total params: 13,707,240			
Trainable params: 13,707,240			
Non-trainable params: 0			

Figure 12

```

for k in range(num_epochs):
    print("Epoch Number: ",k+1)
    progbar = generic_utils.Progbar(len(train_questions))
    for question_batch, ans_batch, im_batch in zip(grouped(train_questions, batch_size, fillvalue=train_questions[-1]),
                                                    grouped(train_answers, batch_size, fillvalue=train_answers[-1]),
                                                    grouped(train_image_id, batch_size, fillvalue=train_image_id[-1])):
        timestep = len(nlp(question_batch[-1]))
        X_ques_batch = get_questions_tensor_timeseries(question_batch, nlp, timestep)
        #print(X_ques_batch)
        X_img_batch = get_images_matrix(im_batch, id_map, features)
        Y_batch = get_answers_sum(ans_batch, le)
        #print(X_img_batch.shape)
        loss = model.train_on_batch(({ 'lstm_input' : X_ques_batch, 'reshape_input' : X_img_batch}), Y_batch)
        progbar.add(batch_size, values=[('train loss', loss)])

Epoch Number: 1
95232/95000 [=====] - 1627s 17ms/step - train loss: 4.9355
Epoch Number: 2
95232/95000 [=====] - 1600s 17ms/step - train loss: 4.5245
Epoch Number: 3
95232/95000 [=====] - 1621s 17ms/step - train loss: 4.4429
Epoch Number: 4
95232/95000 [=====] - 1583s 17ms/step - train loss: 4.4357
Epoch Number: 5
95232/95000 [=====] - 1565s 16ms/step - train loss: 4.3203

```

Figure 13

```

[ ] print ("Accuracy: ", round((correct_val/total)*100,2))

Accuracy: 15.86

```

Figure 14

## 4.2 ViT: MLP MIXER

Our main goal was to focus on image processing and how efficiently image processing can do it. ViT was our obvious choice, but to study ViT better, we used an already-ready model which was based on the papers.

[An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale](#) ,  
[MLP-Mixer: An all-MLP Architecture for Vision](#) , [How to train your ViT? Data Augmentation, and Regularization in Vision Transformers](#),  
[When Vision Transformers Outperform ResNets without Pretraining or Strong Data Augmentations](#).  
 These papers were very substantial for ViT; the papers had all been published very recently.

Our Colab Notebook for this part:

<https://colab.research.google.com/drive/1frV-UGzi1ChrQzDD2WWjgfmFqVUJ8BnF?usp=sharing>



Load and convert pretrained checkpoint. This involves loading the pre-trained model results and modifying the parameters, e.g., changing the final layers and resizing the positional embeddings.

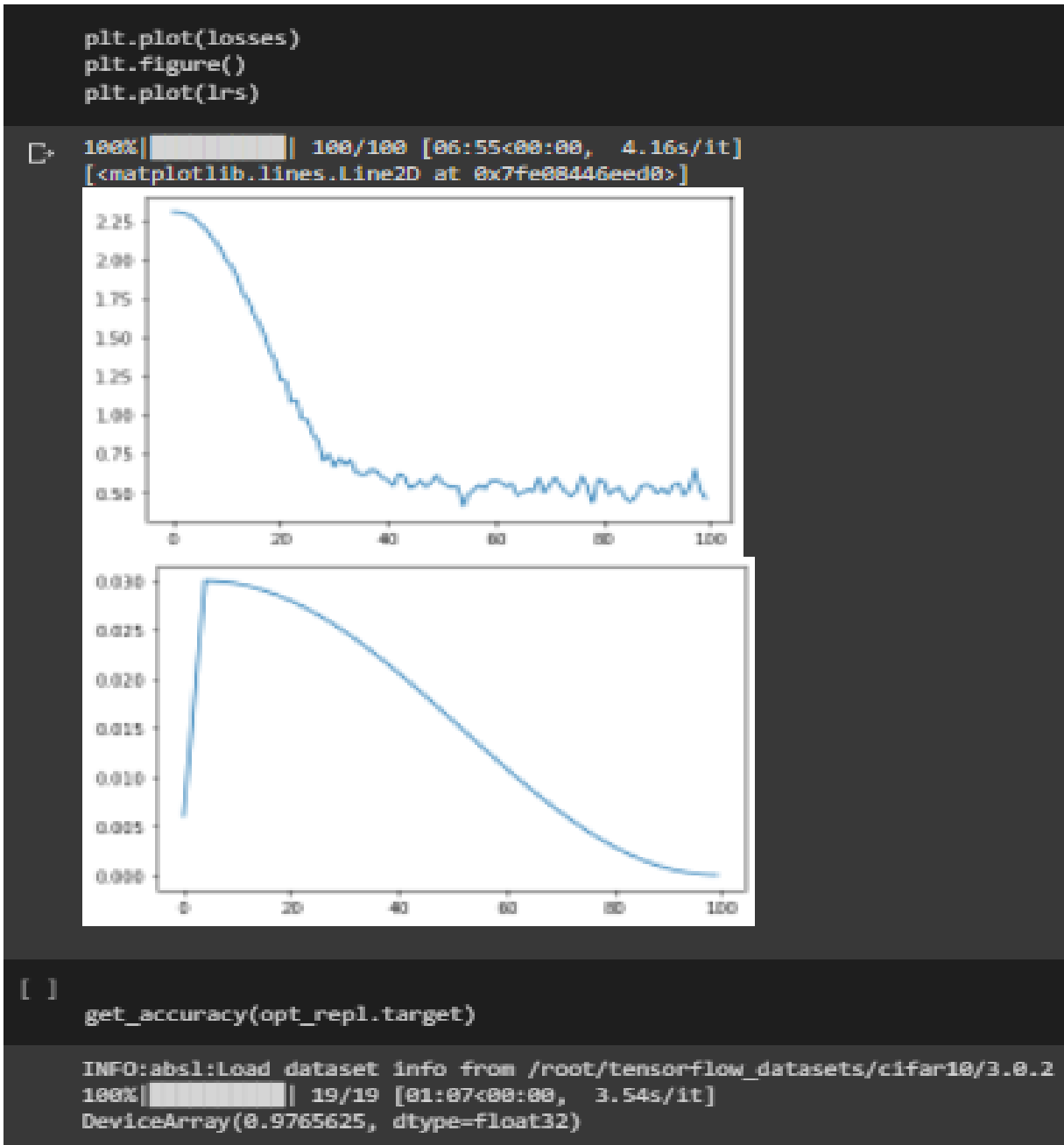


Figure 15: Loss and Accuracy


```

# Get a random picture with the correct dimensions.
resolution = 224 if model_name.startswith('Mixer') else 384
!wget https://picsum.photos/$resolution -O picsum.jpg
import PIL
img = PIL.Image.open('picsum.jpg')
img

--2022-04-15 18:53:29-- https://picsum.photos/384
Resolving picsum.photos (picsum.photos)... 104.26.4.30, 172.67.74.163, 104.26.5.30, ...
Connecting to picsum.photos (picsum.photos)|104.26.4.30|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://i.picsum.photos/id/81/384/384.jpg?hmac=9HikF-vJTS-IM6Sp3GHu3FUGYnx0XrgNdn-DVYa2gyA [following]
--2022-04-15 18:53:29-- https://i.picsum.photos/id/81/384/384.jpg?hmac=9HikF-vJTS-IM6Sp3GHu3FUGYnx0XrgNdn-DVYa2gyA
Resolving i.picsum.photos (i.picsum.photos)... 172.67.74.163, 104.26.4.30, 104.26.5.30, ...
Connecting to i.picsum.photos (i.picsum.photos)|172.67.74.163|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 27659 (27K) [image/jpeg]
Saving to: 'picsum.jpg'

picsum.jpg      100%[=====] 27.01K  --.-KB/s   in 0s

2022-04-15 18:53:29 (73.3 MB/s) - 'picsum.jpg' saved [27659/27659]



# Predict on a batch with a single item (note very efficient TPU usage...)
logits, = model.apply(dict(params=params), (np.array(img) / 128 - 1)[None, ...], train=False)

preds = np.array(jax.nn.softmax(logits))
for idx in preds.argsort()[::-1:-1]:
    print(f'{preds[idx]:.5f} : {imagenet_labels[idx]}', end='')

0.12542 : park_bench
0.05578 : megalith, megalithic_structure
0.03214 : maze, labyrinth
0.01982 : buckeye, horse_chestnut, conker
0.01493 : wild_boar, boar, Sus_scrofa
0.01431 : swing
0.01321 : stone_wall
0.01063 : earthstar
0.00862 : mountain_bike, all-terrain_bike, off-roader
0.00855 : bolete

```

Figure 16: The inference

### 4.3 ViT: Using Hugging Face


Hugging Face, which started as a chat app for bored teenagers, now offers open-source NLP technologies and received \$15 million last year to construct a comprehensive NLP library. Hugging Face has been able to build language processing expertise since its chat app rapidly.

Our Colab Notebook for this part:

[https://colab.research.google.com/drive/1wfoUTn7q2aqsDd07-i1VVijFY6pe9jy\\_?usp=sharing](https://colab.research.google.com/drive/1wfoUTn7q2aqsDd07-i1VVijFY6pe9jy_?usp=sharing)

Our hugging face website link: [https://huggingface.co/AhmedSayeem/ViT\\_Basic](https://huggingface.co/AhmedSayeem/ViT_Basic)

For public use our model and test it for themselves.

```
 class Classifier(pl.LightningModule):

    def __init__(self, model, lr: float = 2e-5, **kwargs):
        super().__init__()
        self.save_hyperparameters('lr', *list(kwargs))
        self.model = model
        self.forward = self.model.forward
        self.val_acc = Accuracy()

    def training_step(self, batch, batch_idx):
        outputs = self(**batch)
        self.log(f"train_loss", outputs.loss)
        return outputs.loss


    def validation_step(self, batch, batch_idx):
        outputs = self(**batch)
        self.log(f"val_loss", outputs.loss)
        acc = self.val_acc(outputs.logits.argmax(1), batch['labels'])
        self.log(f"val_acc", acc, prog_bar=True)
        return outputs.loss


    def configure_optimizers(self):
        return torch.optim.Adam(self.parameters(), lr=self.hparams.lr)


[ ] pl.seed_everything(42)
    classifier = Classifier(model, lr=2e-5)
    trainer = pl.Trainer(gpus=1, precision=16, max_epochs=4)
    trainer.fit(classifier, train_loader, val_loader)
```

Figure 17

```
feature_extractor = ViTFeatureExtractor.from_pretrained('google/vit-base-patch16-224-in21k')
model = ViTForImageClassification.from_pretrained(
    'google/vit-base-patch16-224-in21k',
    num_labels=len(label2id),
    label2id=label2id,
    id2label=id2label
)
collator = ImageClassificationCollator(feature_extractor)
train_loader = DataLoader(train_ds, batch_size=8, collate_fn=collator, num_workers=2, shuffle=True)
val_loader = DataLoader(val_ds, batch_size=8, collate_fn=collator, num_workers=2)
```

Downloading: 100%  160/160 [00:00<00:00, 3.43kB/s]

Downloading: 100%  502/502 [00:00<00:00, 13.9kB/s]

Downloading: 100%  330M/330M [00:09<00:00, 37.4MB/s]

Some weights of the model checkpoint at google/vit-base-patch16-224-in21k were not used when initializing ViTForImageClassification:

Figure 18

```
Global seed set to 42
Using 16bit native Automatic Mixed Precision (AMP)
GPU available: True, used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs
Missing logger folder: /content/lightning_logs
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
```

	Name	Type	Params
0	model	ViTForImageClassification	85.8 M
1	val_acc	Accuracy	0

85.8 M Trainable params  
0 Non-trainable params  
85.8 M Total params  
171.605 Total estimated model params size (MB)


Epoch 39: 21%  20/94 [1:07:11<4:08:37, 201.59s/it, loss=0.0161, v\_num=0, val\_acc=0.875]

Figure 19

With just 39 epochs, we reached 87.5% accuracy, and the loss had been reduced significantly from 0.205 with just 4 epochs to 0.0161 with 39 epochs. This model is easy to implement and was made from just five classes. More importantly, our model identified the object from a different image on the internet, which is not part of the training dataset.

```
display(image)

[ ] #feature_extractor = ViTFeatureExtractor.from_pretrained('google/vit-base-patch16-224')
    # model = ViTForImageClassification.from_pretrained('google/vit-base-patch16-224')

    feature_extractor = ViTFeatureExtractor.from_pretrained("AhmedSayeem/VIT_Basic")
    model = ViTForImageClassification.from_pretrained("AhmedSayeem/VIT_Basic")

    Downloading: 100% ██████████ 228/228 [00:00<00:00, 4.06kB/s]
    Downloading: 100% ██████████ 855/855 [00:00<00:00, 20.4kB/s]
    Downloading: 100% ██████████ 327M/327M [00:17<00:00, 32.3MB/s]

[ ] inputs = feature_extractor(images=image, return_tensors="pt")

[ ] outputs = model(**inputs)

[ ] logits = outputs.logits

[ ] predicted_class_idx = logits.argmax(-1).item()
    print("Predicted class:", model.config.id2label[predicted_class_idx])

Predicted class: ladders
```

Figure 20: The inference

## 4.4 ViT: Implementation with Pytorch 1

The ViT PyTorch implementation 1 has been carefully constructed. Keeping factors like:

- Patches Embeddings:
  - CLS Token
  - Position Embedding
- Transformer

- Attention: The attention matrix is computed using queries and values to "attend" to the values, and it takes three inputs: famous questions, keys, and values. We use multi-head attention in this scenario, which means that the processing is distributed over n heads with smaller input sizes.
- Residuals
- MLP: The attention output is sent to a fully linked layer of two layers that upsample the input by a factor of 'expansion.'
- TransformerEncoder
- Head
- ViT

Our Colab Notebook for this part:

<https://drive.google.com/file/d/1YJmOW23iM0Icgx25bcGUlmqXwVgKNirW/view?usp=sharing>

## Vi(sual) T(ransformer)

We can compose `PatchEmbedding`, `TransformerEncoder` and `ClassificationHead` to create the final ViT architecture.

```
[ ] class ViT(nn.Sequential):
    def __init__(self,
                  in_channels: int = 3,
                  patch_size: int = 16,
                  emb_size: int = 768,
                  img_size: int = 224,
                  depth: int = 12,
                  n_classes: int = 1000,
                  **kwargs):
        super().__init__(
            PatchEmbedding(in_channels, patch_size, emb_size, img_size),
            TransformerEncoder(depth, emb_size=emb_size, **kwargs),
            ClassificationHead(emb_size, n_classes)
        )
```

We can use `torchsummary` to check the number of parameters

Figure 21: The inference

## 4.5 ViT: Implementation with Pytorch 2

The ViT PyTorch implementation 2 uses 'deit\_tiny\_patch16\_224' pretrained model. With just ten epochs, it reached 94% accuracy with loss reduced to 1.05. We do the training and validation phase per epoch.

Our Colab Notebook for this part:

[https://colab.research.google.com/drive/1hojKFUcsDTmfV3RtcEr\\_SgN--P\\_0TmXJ?usp=sharing](https://colab.research.google.com/drive/1hojKFUcsDTmfV3RtcEr_SgN--P_0TmXJ?usp=sharing)

```
for param in model.parameters(): #freeze model
    param.requires_grad = False

n_inputs = model.head.in_features
model.head = nn.Sequential(
    nn.Linear(n_inputs, 512),
    nn.ReLU(),
    nn.Dropout(0.3),
    nn.Linear(512, len(classes))
)
model = model.to(device)
print(model.head)
```

```
Sequential(
  (0): Linear(in_features=192, out_features=512, bias=True)
  (1): ReLU()
  (2): Dropout(p=0.3, inplace=False)
  (3): Linear(in_features=512, out_features=50, bias=True)
)
```

Figure 22

```

def train_model(model, criterion, optimizer, scheduler, num_epochs=10):
    since = time.time()
    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print(f'Epoch {epoch}/{num_epochs - 1}')
        print("-"*10)

        for phase in ['train', 'val']: # We do training and validation phase per epoch
            if phase == 'train':
                model.train() # model to training mode
            else:
                model.eval() # model to evaluate

            running_loss = 0.0
            running_corrects = 0.0

            for inputs, labels in tqdm(dataloaders[phase]):
                inputs = inputs.to(device)
                labels = labels.to(device)

                optimizer.zero_grad()

                with torch.set_grad_enabled(phase == 'train'): # no autograd makes validation go faster
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1) # used for accuracy
                    loss = criterion(outputs, labels)

                    if phase == 'train':
                        loss.backward()
                        optimizer.step()
                    running_loss += loss.item() * inputs.size(0)
                    running_corrects += torch.sum(preds == labels.data)

            if phase == 'train':
                scheduler.step() # step at end of epoch

            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects.double() / dataset_sizes[phase]

            print("{} Loss: {:.4f} Acc: {:.4f}".format(phase, epoch_loss, epoch_acc))

            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict()) # keep the best validation accuracy model
            print()

    time_elapsed = time.time() - since # slight error
    print('Training complete in {:.0f}m {:.0f}s'.format(time_elapsed // 60, time_elapsed % 60))
    print("Best Val Acc: {:.4f}".format(best_acc))

    model.load_state_dict(best_model_wts)
    return model

```

Figure 23



```
Epoch 6/9
-----
100%|██████████| 39/39 [05:20<00:00, 8.21s/it]
train Loss: 1.1204 Acc: 0.9045
100%|██████████| 8/8 [00:16<00:00, 2.04s/it]
val Loss: 1.0507 Acc: 0.9280

Epoch 7/9
-----
100%|██████████| 39/39 [05:21<00:00, 8.23s/it]
train Loss: 1.0950 Acc: 0.9154
100%|██████████| 8/8 [00:16<00:00, 2.03s/it]
val Loss: 1.0434 Acc: 0.9320

Epoch 8/9
-----
100%|██████████| 39/39 [05:20<00:00, 8.22s/it]
train Loss: 1.0707 Acc: 0.9269
100%|██████████| 8/8 [00:16<00:00, 2.08s/it]
val Loss: 1.0397 Acc: 0.9360

Epoch 9/9
-----
100%|██████████| 39/39 [05:32<00:00, 8.52s/it]
train Loss: 1.0562 Acc: 0.9312
100%|██████████| 8/8 [00:19<00:00, 2.42s/it]val Loss: 1.0171 Acc: 0.9480

Training complete in 57m 20s
Best Val Acc: 0.9480
```

Figure 24: The results

---

## CHAPTER 5: CONCLUSION

### 5.1 DISCUSSION

With our experiment, we aim to be able to tackle the problem in the VQA with large datasets and provide a proper comparative study. Even though we have yet to test our recommended technique, the literature we researched and the material we studied indicate a high likelihood of success. Even though numerous ways have been demonstrated to tackle the attention-grabbing part of the image problem in a dataset, we believe that fixing the problem via Vision Transformer will be a much more efficient procedure. The final results we obtain will be compact and efficient.

### 5.2 SUMMARY

In this semester, we first introduced our datasets that we will use to benchmark the performance of VQA algorithms; the datasets introduced included real-world images, synthetic images, and additional annotations such as supporting world facts necessary to answer some visual questions. We then introduced different Vision Transformer algorithms, which we grouped into specific categories based on the main contribution of the algorithm.

### 5.3 FUTURE WORK

Instead of having models that perform well on some datasets but badly on others, future work would include more holistic models capable of answering issues that require external world knowledge, complicated reasoning, and dealing with synthetic data. Our focus will be on more common VQA datasets and the linguistic part, which hopefully we will tackle with NLP.

### 5.4 Social and ethical concerns

Even Though computer vision allows artificial intelligence systems to recognize faces, objects, places, and motions, it also raises a number of ethical and privacy concerns. Fraud, bias, inaccuracy, and a lack of informed consent are examples of these. There are no privacy or criminal problems because we are working with text and selected images. However, there is a problem inaccuracy, which means that the model may predict a word wrongly in cases where the test image presented is excessively blurry or deformed.

## REFERENCES

- [1]. Visual Question Answering. 2022. Visual Question Answering. [online] Available at: <<https://visualqa.org/index.html>> [Accessed 27 April 2022].
- [2]. Y. Srivastava, V. Murali, S. Dubey, & S. Mukherjee (2020, December 23). Visual question answering using deep learning: A survey and performance analysis. Retrieved April 20, 2022, from <https://doi.org/10.48550/arXiv.1909.01860>
- [3]. Y. Goyal, T. Khot, D. Summers-Stay, D. Batra, & D. Parikh (2017, May 15). Making the V in VQA Matter: Elevating the role of image understanding in visual question answering. Retrieved April 21, 2022, from <https://arxiv.org/abs/1612.00837>
- [4]. A. Agrawal, J. Lu , S. Antol , M. Mitchell , C. Zitnick ,D. Batra, & D. Parikh (2016, October 27). VQA: Visual question answering. Retrieved April 23, 2022, from <https://arxiv.org/abs/1505.00468>
- [5]. M. Ren , R. Kiros, & R. Zeme, (2015, November 29). Exploring models and data for image question answering. Retrieved April 25, 2022, from <https://arxiv.org/abs/1505.02074>
- [6]. Fukui, D. H. Park, D. Yang, A. Rohrbach, T. Darrell, and M. Rohrbach (2016, September 24). Multimodal compact bilinear pooling for visual question answering and visual grounding. Retrieved April 26, 2022, from <https://arxiv.org/abs/1606.01847>
- [7]. A. Dosovitskiy, L. Beyer., A. Kolesnikov, D. Weissenborn,X. Zhai,T. Unterthiner, N. Houlsby, (2021, June 03). An image is worth 16x16 words: Transformers for image recognition at scale. Retrieved April 27, 2022, from <https://arxiv.org/abs/2010.11929>