الجمهورية الجزائرية الديمقراطية الشعبية

وزارة التعليم العالي والبحث العلمي

جامعة سعيدة د. مولاي الطاهر

كلية العلوم

قسم: الإعلام الآلي

Mémoire de Master

Spécialité : MICR

Thème

Virtual Assistant For The Visually Impaired

Présenté par :

ADJIR Asmaa Naziha

Dirigé par :

Dr. ADJIR Noureddine

Promotion 2023 - 2024

Acknowledgment

I would like to express my sincere gratitude to my supervisor, Mr. Adjir Noureddine, for his invaluable guidance, support, and expertise throughout this dissertation journey. His insightful feedback and unwavering encouragement were instrumental in shaping this work.

I extend my heartfelt thanks to the jury members, Dr. Derkaoui and Dr. Kouidri, for their time, effort, and valuable insights during the evaluation of this dissertation. Their constructive feedback and critical evaluations were crucial in enhancing the quality of this work.

Beyond the academic realm, I am incredibly grateful to my family, especially my parents for their consistent support and belief in me. Their love and encouragement provided the foundation upon which I could achieve this goal. I would also like to extend my thanks to my friends, teachers, and everyone who has offered support and encouragement along the way.

ملخص

تستكشف هذه الرسالة تطوير AIRIS ، وهو مساعد افتراضي مصمم لمساعدة الأفراد المكفوفين وضعاف البصر. من خلال دمج الذكاء الاصطناعي(AI) ، لا سيما عبر الإجابة البصرية على الأسئلة(VQA) ، يتصدى AIRIS لتحدي فهم الصور للمستخدمين المكفوفين وضعاف البصر من خلال توفير واجهة مدعومة بـ VQA للاستفسار عن الصور في الوقت الفعلي باستخدام الأوامر الصوتية. تدرس الدراسة التحديات التي تواجه هذه الفئة وتحقق في كيفية تعزيز الذكاء الاصطناعي لإمكانية الوصول والاستقلالية. على الرغم من أن تطوير AIRIS كشف عن إمكانيات الحلول المدعومة بالذكاء الاصطناعي، فإن القيود الناتجة عن قيود الموارد (مثل قوة المعالجة) أثرت على القدرات الحالية للنظام. تختتم الرسالة بمناقشة هذه القيود وتقترح اتجاهات للعمل المستقبلي لتحسين AIRIS بشكل أكبر واستكشاف الإمكانات الأوسع للذكاء الاصطناعي في تعزيز الوصول والشمولية للأفراد المكفوفين وضعاف البصر.

Abstract

This dissertation explores the development of AIRIS, a virtual assistant designed to assist visually impaired individuals. By integrating Artificial Intelligence (AI), particularly through Visual Question Answering (VQA), AIRIS addresses the challenge of image comprehension for visually impaired users by providing a VQA-powered interface for real-time image inquiry using voice commands. The study examines the challenges faced by this population and investigates how AI can enhance accessibility and independence. While the development of AIRIS revealed the potential of AI-powered solutions, limitations due to resource constraints (e.g., processing power) impacted the system's current capabilities. The dissertation concludes by discussing these limitations and proposes future work directions to further refine AIRIS and explores the broader potential of AI in advancing accessibility and inclusivity for visually impaired individuals.

Résumé

Cette dissertation explore le développement d'AIRIS, un assistant virtuel conçu pour aider les personnes malvoyantes. En intégrant l'Intelligence Artificielle (IA), en particulier à travers la réponse visuelle aux questions (VQA), AIRIS relève le défi de la compréhension des images pour les utilisateurs malvoyants en fournissant une interface alimentée par VQA pour l'interrogation d'images en temps réel à l'aide de commandes vocales. L'étude examine les défis rencontrés par cette population et enquête sur la manière dont l'IA peut améliorer l'accessibilité et l'indépendance. Bien que le développement d'AIRIS ait révélé le potentiel des solutions alimentées par l'IA, les limitations dues aux contraintes de ressources (par exemple, la puissance de traitement) ont impacté les capacités actuelles du système. La dissertation conclut en discutant de ces limitations et propose des pistes de travail futures pour affiner davantage AIRIS et explorer le potentiel plus large de l'IA pour faire progresser l'accessibilité et l'inclusivité pour les personnes malvoyantes.

Contents

List of Figures

List of Tables

List of Abbreviations

AI Artificial Intelligence. 12, 62

CNN Convolutional Neural Network. 12, 19, 23, 24

LSTM Long Short Term Memory. 12

VQA Visual Question Answering. 4, 5, 12, 27–34, 41, 44–47, 49–52, 54–56, 59, 62

General Introduction

General Introduction

Vision impairment, or visual impairment, refers to a range of conditions that affect a person's ability to see, from mild vision loss to complete blindness.

Low vision refers to significant visual impairment that cannot be fully corrected with standard glasses, contact lenses, medication, or surgery. Individuals with low vision experience difficulties with daily tasks. This category includes:

- **Moderate Visual Impairment:** Reduced vision that affects everyday activities, such as reading and recognizing faces.
- **Severe Visual Impairment:** More pronounced vision loss that significantly hinders daily activities, even with the best possible corrective measures.

Blindness involves a severe loss of vision that impacts a person's ability to perform most visual tasks. It can be further classified into:

- **Legally Blind:** Defined in many countries as having a visual acuity of 20/200 or worse in the better eye with the best possible correction, or a visual field of 20 degrees or less. Individuals who are legally blind may still have some usable vision.
- **Totally Blind:** The complete lack of light perception, meaning no vision at all.

Vision impairment can result from a variety of factors, including:

- Uncorrected refractive errors (nearsightedness, farsightedness, astigmatism, presbyopia)
- Eye diseases (cataracts, glaucoma, diabetic retinopathy, age-related macular degeneration)
- Other causes (injuries, strokes, neurological conditions)

Vision impairment can result from a variety of factors, such as refractive errors, eye diseases, and other causes. Refractive errors include nearsightedness, farsightedness, astigmatism, and presbyopia, which are uncorrected by standard methods. Eye diseases like cataracts, glaucoma, diabetic retinopathy, and age-related macular degeneration can also lead to vision impairment. Additionally, injuries, strokes, and neurological conditions may damage the eyes or the parts of the brain responsible for vision, causing impairment.

At least 2.2 billion people worldwide experience some form of vision loss [15]. This condition can be challenging and isolating, making daily activities such as reading, cooking, and navigating unfamiliar environments difficult. Furthermore, it can impact one's mental and emotional well-being, leading to feelings of frustration, anxiety, and depression. Vision loss can also make it difficult to perform tasks like managing finances, pursuing educational or employment opportunities, and participating in social activities.

Challenges and Barriers

Despite technological advancements, several challenges persist for visually impaired people (VIPs):

- **Accessibility and Affordability:** Many assistive technologies are expensive and inaccessible to everyone, particularly in low-income areas.

- **Training and Awareness:** Effective use of adaptive devices necessitates adequate training, which is often unavailable. Awareness of available technology is likewise restricted. Some assistive aids require significant effort and training, which may discourage VIPs from using them.
- **Technological Limitations:** Current technologies may not be appropriate for all environments or scenarios, necessitating the development of more reliable and user-friendly alternatives. Fully blind individuals often need more sophisticated devices with built-in cameras and sensors for object detection and recognition.
- **Universal Design Challenges:** The lack of a universal design approach hinders researchers from improving assistive technologies, as each design methodology is different and not easily adaptable.
- **Social and Psychological Barriers:** Stigma and misconceptions about vision impairment may hinder the use of assistive technology and the integration of visually impaired people into society.
- **Ease of Access:** The devices currently available may not meet the diverse needs of visually impaired individuals, making ease of access a continuing challenge.

Solutions to Overcome These Challenges
To tackle these challenges, several solutions have been developed, ranging from low-tech to high-tech:
Low-Tech Solutions

- **White Canes:** These are used to detect obstacles in the path of the user, providing tactile feedback about the environment.
- **Braille:** A tactile writing system that enables blind individuals to read and write through touch.
- **Braille Labels and Tactile Markers:** These are used to label items such as medication bottles, appliances, and personal belongings for easy identification.
- **Magnifying Glasses:** Simple handheld lenses that magnify text and images, making them easier to see for those with low vision.
- **Bold Lined Paper and Large Print Books:** Specially designed paper with thick, dark lines and books with larger text to aid readability.
- **Talking Watches and Clocks:** Devices that announce the time audibly, helping visually impaired individuals keep track of time.
- **Raised Dot Stickers:** These are applied to household appliances, phones, and keyboards to help identify buttons and keys by touch.
- **Color-Contrasting Tools:** Items such as cutting boards, measuring cups, and utensils in high-contrast colors improve visibility during daily tasks.

High-Tech Solutions

- **Screen Readers:** Software that converts text on a computer screen into synthesized speech or braille output, allowing blind users to access digital content.
- **Optical Character Recognition (OCR):** Devices or apps that scan and convert printed text into digital text, which can then be read aloud by screen readers.
- **Smart Glasses:** Wearable devices equipped with cameras and sensors to provide audio descriptions of the environment, text recognition, and object identification.
- **Voice-Activated Assistants:** Devices like Amazon Echo, Google Home, and Apple's Siri that respond to voice commands, assisting with tasks like setting reminders, controlling smart home devices, and providing information.
- **Braille Displays:** Electronic devices that translate on-screen text into braille characters in real-time, allowing blind users to read digital text through touch.

- **Navigation Apps:** Smartphone applications like Seeing AI, Be My Eyes, and Google Maps offer features such as audio navigation, object recognition, and volunteer assistance.
- **Video Magnifiers:** Electronic devices that use a camera to capture and magnify text and images onto a screen, providing adjustable magnification and contrast settings.
- **Text-to-Speech (TTS) Software:** Converts written text into spoken words, enabling visually impaired individuals to listen to digital content, such as books, articles, and websites.
- **Electronic Braille Notetakers:** Portable devices that allow users to take notes, write documents, and access digital information using braille input and output.
- **AI-Powered Apps:** Applications that leverage Artificial Intelligence (AI) to provide advanced features such as scene description, facial recognition, and handwriting recognition.

Travel and Navigation Aids

- **Travel Aids:** Devices that gather information about the surrounding environment and transfer it to the user.
- **Orientation Aids:** Provide directions to pedestrians in unfamiliar places.
- **Position Location Devices:** Determine the precise location of the user.
- **Vision Assistive Caps:** Utilized for VIP navigation.
- **Smartphone Camera Systems:** Used for navigation, object detection, and contacting emergency services.

While these solutions have greatly enhanced the quality of life for many visually impaired people, there are still gaps that require innovative approaches. Current technologies frequently have limits in terms of accessibility, affordability, usability, and applicability across many environments and tasks. Many existing tools focus on specialized functions such as navigation, object detection, and image captioning, but there is a need for a more comprehensive approach that can address a broader range of questions.

Motivated by this need, we have developed a virtual assistant named "AIRIS." AIRIS explores a new path by incorporating Visual Question Answering (VQA), which allows users to inquire about the content of images using voice commands.

Following this introduction, Chapter 1 will cover various deep learning techniques, namely Convolutional Neural Network (CNN) and Long Short Term Memory (LSTM). The second chapter delves into recent advances in VQA, evaluating various models, datasets, and approaches while emphasizing both progress and challenges in the field. Chapter 3 describes the creation of AIRIS, a virtual assistant designed to assist visually impaired people, including the system's architecture, voice interface mechanisms, and the integration of a VQA model, as well as the training process and performance assessment. The dissertation concludes with a summary of the key findings, the impact of AIRIS on improving accessibility for visually impaired individuals, and potential directions for future research.

Following this introduction, Chapter 2 will delve deeper into the Chapter 3 will present the methodology used in this research

Deep Learning

Chapter 1

Deep Learning

1.1 Introduction

Deep learning, a subfield of artificial intelligence, employs sophisticated artificial neural networks modeled after the human brain to tackle complex problems. This burgeoning field has revolutionized research across diverse disciplines, particularly in artificial intelligence, by augmenting its power and broadening its reach.

In this chapter, we will introduce artificial neural networks with different types of algorithms for deep learning, and we will focus on convolutional neural networks and recurrent neural networks.

1.2 Artificial Neural Networks

The age-old dream of intelligent machines fueled a quest to understand the human brain. Inspired by its workings, researchers developed artificial neural networks, mimicking the brain's structure to create machines capable of learning.

1.2.1 The Biological Neuron

With over 1000 billion interconnected neurons [16], the nervous system is a complex network. These fundamental cells, called neurons, exhibit diverse shapes and behaviors depending on their location within the brain.

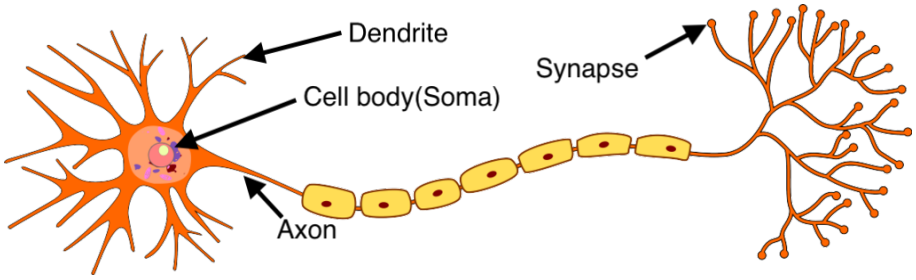


Figure 1.1: Neuron Biology [1]

As illustrated in 1.1, a neuron typically has a cell body, star-shaped structures called dendrites that receive signals, and a long extension called an axon that transmits signals throughout the body.

1.2.2 The Artificial/Formal Neuron

Formal or artificial neural networks are modeled after natural (biological) neural networks. They are typically optimized using statistical learning methods. Modeling involves characterizing the neuron (basic unit) and its connections to other biological neurons. McCulloch and Pitts [17] introduced the first formal neuron in 1943 (see 1.2).

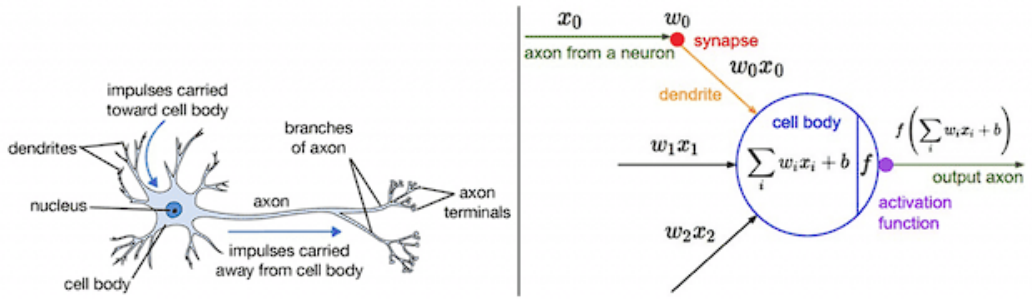


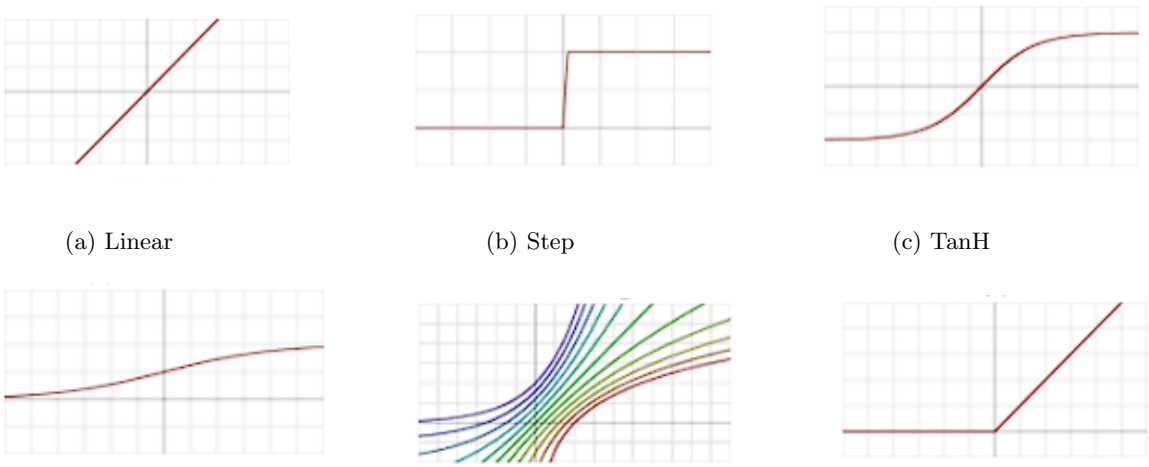
Figure 1.2: Formal neuron equivalent to a biological neuron [2]

The formal neuron is a mathematical model that mimics the behavior of biological neurons, specifically the sum of inputs. The authors developed an algorithm that weights inputs based on synaptic weights w_i (the sum of input values multiplied by weighting coefficients), recognizing that not all synapses have the same value due to differences in neuron strength.

The choice of an activation function is a critical component of neural networks. The most commonly used functions are [18]:

- **Linear:** serves as the output layer for a regression. The output units are equal to their input level. Output interval: $]-\infty, \infty[$ (Figure 1.3a).
- **Step:** it always returns 1 for positive signals and 0 for negative signals (Figure 1.3b).
- **TanH:** is generally used in LSTMs for continuous data. The output interval is $[-1, +1]$ (Figure 1.3c).
- **Sigmoid:** (logistic) the most popular function for decades; its output interval is $[0, +1]$(Figure 1.3d).
- **Softmax:** is utilized for multi-classification in the output layer. Output interval: $[0, +1]$ (Figure 1.3e).
- **ReLU:** (Rectified Linear Unit) These are the most often used functions nowadays. Compared to sigmoid and tanh functions, they offer faster training and are lighter. Widely used for CNNs. Output interval: $]0, \infty[$ (Figure 1.3f).

The activation function adds non-linearity to neuron function, similar to how a system's transfer function varies output based on input. It transfers the weighted sum from the previous layer's neurons to form the neuron's output.



(a) Linear (b) Step (c) TanH

(d) Sigmoid (e) Softmax (f) ReLU

Figure 1.3: Activation functions [3]

1.2.3 The Multilayer Perceptron (MLP)

The multilayer perceptron is an artificial neural network with layers that transmit information in a single direction, from input to output. Frank Rosenblatt designed the perceptron in 1957 at Cornell Aeronautical Laboratory, drawing inspiration from Friedrich Hayek and Donald Hebb's cognitive theories.

Initially, the perceptron was a monolayer with a single output connected to all inputs.

Figure 1.4 illustrates a network with an input layer, two hidden levels, and an output layer. The input layer is always a virtual layer that deals with system inputs. It has no neurons. The following layers are composed of neurons.

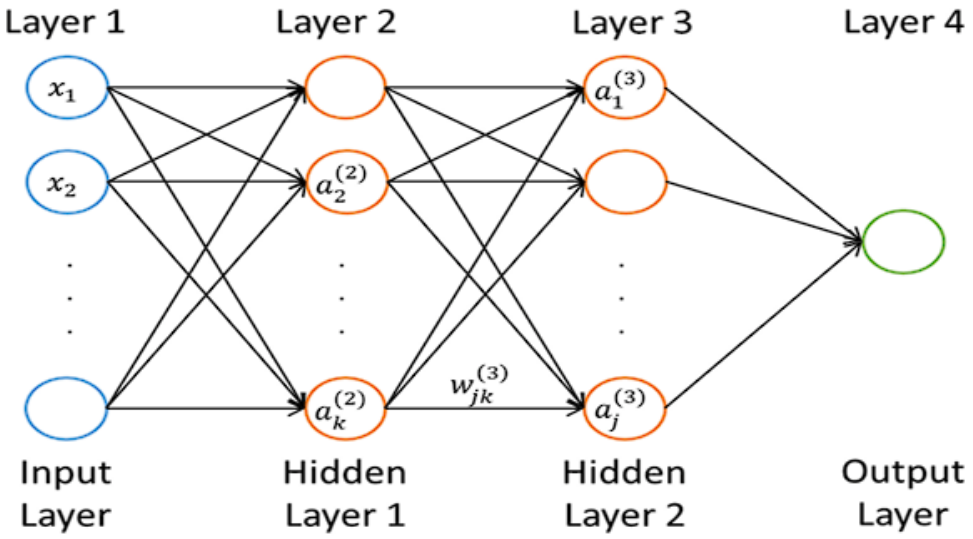


Figure 1.4: Multi-layer perceptron [4]

Figure 1.4 shows an example with more than three inputs, k neurons on the first hidden layer, j on the second, and one on the output layer. The neurons in the final layer always produce the system's outputs. A multilayer perceptron can contain any number of layers and inputs. Neurons are linked through weighted connections w_{ij} (Figure 1.2). The network's weighted connections program an application from input to output using a nonlinear transformation. The activation function yields this result.

1.2.4 Learning formal neural networks

An artificial neural network can learn by modifying the weights of its connections based on training data. Over time, it develops the ability to generalize. Learning is the process of modifying a neural network's behavior to achieve desired results [19].

Paul Werbos suggested backpropagation of the error gradient in multilayer systems in 1984 [20] and David Rumelhart improved it two years later in 1986 [21], allowing neural networks to handle nonlinear problems. This algorithm aims to minimize errors between computed and desired outputs. Back-propagation of the gradient refers to how the error calculated at the output is sent in the opposite way to the input.

1.2.5 Gradient backpropagation algorithm

The back-propagation multilayer perceptron connects neurons from one layer to all adjacent layers. A coefficient influences the information's impact on the destination neuron through these connections. Using a multilayer perceptron to solve a problem requires calculating the optimal weights for each inter-neuronal connection. This is determined by a backpropagation technique [22].

The algorithm starts by propagating inputs x_i from the input layer to the output layer, passing through hidden layers. To attain the intended output, compare the computed output to the desired output. If an error is found, alter the weights and biases of the network using back-propagation to adapt and learn [22].

The back propagation algorithm is represented as follows:

Step 1:

1. Initialize the weights w_{ij} and the internal thresholds of the neurons to small random values.
2. Choose the activation functions.
3. Choose the number of hidden layers.
4. Fix the learning rate α.
5. Set an error threshold Emax
6. Set the maximum number of epochs Itmax.

Step 2: Activate the perceptron by applying an input vector and calculating the output of each layer.

Step 3: Calculate the signal error terms of the output layer and the hidden layer.

Step 4: Update the output layer and hidden layer weights.

Step 5: Test if there are more patterns in the training set. Go to step 2.

Step 6: If the condition of convergence towards the fixed threshold is pending go to step 8.

Step 7: If the maximum number of epochs is reached go to step 8. Else, go to step 2 and start a new iteration.

Step 8: End.

1.3 Types of Algorithms Used in Deep Learning

Deep learning models use multiple algorithms. Although no network is considered perfect, some algorithms are better suited for specific tasks. To choose the right algorithm, it is best to fully understand the details of each one.

1.3.1 Convolutional Neural Networks (CNN)

A Convolutional Neural Network (CNN or ConvNet) is a multi-layer neural network mainly used for image processing and target detection. Yann LeCun created the first CNN in 1988, known as LeNet. It can recognize manuscripts like postal codes and numerals [23].

1.3.2 Recurrent Neural Networks (RNN)

RNN works with either sequential or time series data. These deep learning techniques are commonly used for sequential or temporal tasks, including language translation, natural language processing (NLP), speech recognition, and image captioning [24].

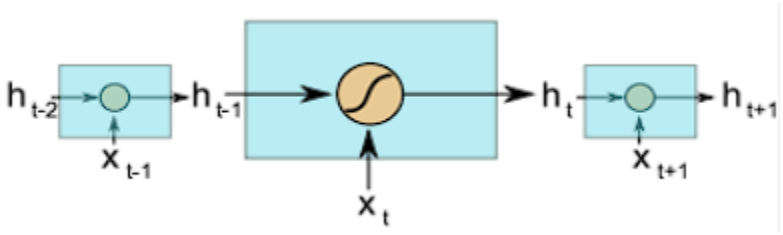


Figure 1.5: Simple RNN internal operation [5]

Figure 1.5 shows a simple recurrent neural network that computes internal memory (h_t) as follows:

$$h_t = g(Wx_t + Uh_t + b)$$

Figure 1.6: Internal memory equation[6]

However, a major challenge with traditional RNNs is the vanishing gradient problem. During training, the error signal's gradients become very small as they propagate backward through the network. This makes it difficult for the network to learn effectively from long-term dependencies.

1.3.2.1 Long Short-Term Memory (LSTM)

LSTM networks address the vanishing gradient problem through additional gating mechanisms within the network architecture. These gates control the flow of information, allowing the network to selectively remember or forget information from the past. LSTMs have three specific gates:

- **Input Gate:** Controls the flow of new information into the cell.
- **Forget Gate:** Decides what information from the previous cell state to retain.
- **Output Gate:** Determines what information from the current cell state to expose.

This gating mechanism enables LSTMs to effectively learn long-term dependencies, making them a powerful tool for various tasks involving sequential data.

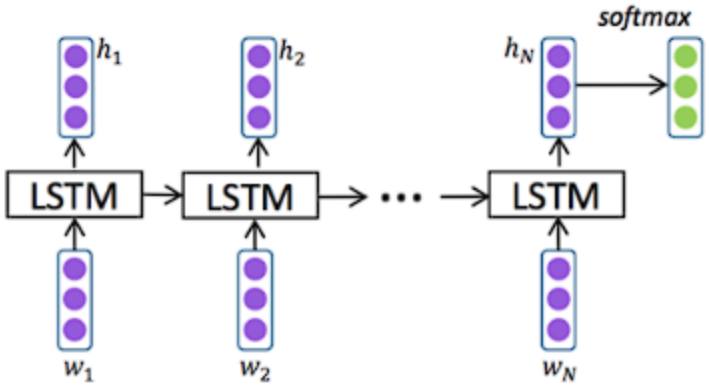


Figure 1.7: High-level LSTM architecture [6]

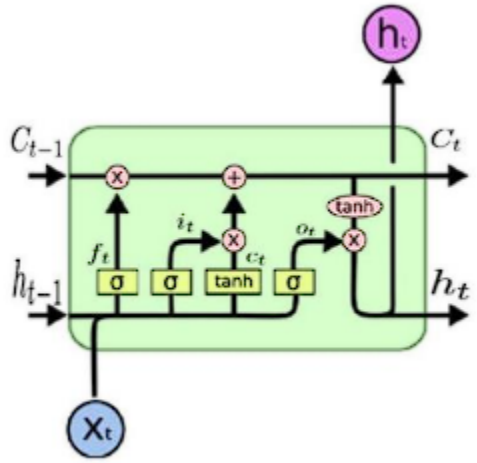


Figure 1.8: The inner architecture of an LSTM cell [7]

1.3.2.2 Gated Recurrent Unit (GRU)

Similar to LSTMs, GRUs are another type of RNN that tackles the vanishing gradient problem. They also utilize gating mechanisms to control information flow but with a simpler architecture compared to LSTMs. GRUs have two gates:

- **Reset Gate:** Controls the influence of the previous cell state on the current cell state.
- **Update Gate:** Determines how much of the previous cell state to keep and how much new information to incorporate.

Despite having fewer gates, GRUs have shown impressive performance and are often a good choice when computational efficiency is a concern.

Both LSTMs and GRUs offer significant advantages over traditional RNNs for processing sequential data. Due to their effectiveness, they are among the most popular RNN architectures used today.

1.3.3 Generative Conflict Networks (GANs)

GAN is a deep learning algorithm that generates new data based on previous training data. The GAN is made up of two parts: a generator that produces false data and a discriminator that learns from it [25].

1.3.4 Radial Basis Function Networks (RBFN)

RBFN is a predictive neural network that uses a radial basis function as an activation function. The model consists of three layers: input, hidden, and output layer, and is mostly used for classification, regression, and time series prediction [26].

1.3.5 Multilayer Perceptrons (MLP)

MLP is a feedforward neural network consisting of numerous layers of perceptrons and activation functions. MLP consists of fully connected input and output layers. They have the same number of input and output layers but can have many hidden layers. Can be used for voice translation and automatic image processing software [27].

1.3.6 Self-Organizing Maps (SOM)

Teuvo Kohonen created self-organizing neural networks (SOMs) to visualize data and reduce its dimensions. Data visualization seeks to solve the problem of humans' inability to visualize high-dimensional data. SOMs are designed to assist users comprehend this multidimensional data [28].

1.3.7 Restricted Boltzmann Machines (RBM)

RBMs, created by Geoffrey Hinton, are stochastic neural networks capable of learning based on a probability distribution of inputs. This deep learning technique is used for several tasks, including dimensionality reduction, classification, regression, collaborative filtering, feature learning, and modeling [29].

1.3.8 Auto-encoders (AE)

Autoencoders are feedforward neural networks whose input and output are identical. Geoffrey Hinton created autoencoders in the 1980s to handle unsupervised learning problems. These trained neural networks duplicate data from input to output layers. Autoencoders were used for drug discovery, popularity prediction, and image processing [30] and now are used in different domains *e.g.* NLP.

1.4 OVERVIEW ON CNNs

The convolutional neural network is a class of deep learning methods that has become dominant in various computer vision tasks and is attracting interest in a variety of fields.

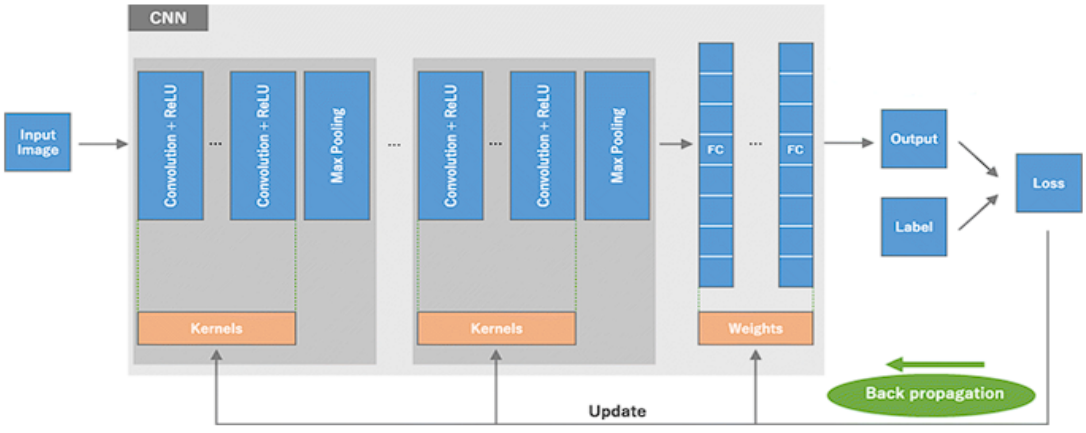


Figure 1.9: Overview of CNN architecture [8]

1.4.1 The building blocks of a CNN

The convolutional neural network is composed of several building blocks, such as convolution layers, pooling layers, and fully connected layers [31], and are designed to automatically and adaptively learn the spatial hierarchies of entities via a backpropagation algorithm (see Figure 1.9).

1.4.1.1 Convolution layers

The term "convolution" refers to the mathematical combination of two functions (addition and multiplication) to form a third function. When this happens, two sets of data are merged. In the context of CNNs, a convolutional layer (called a filter or kernel) is applied to the input data to then produce a feature map [31] as shown in Figure 1.10.

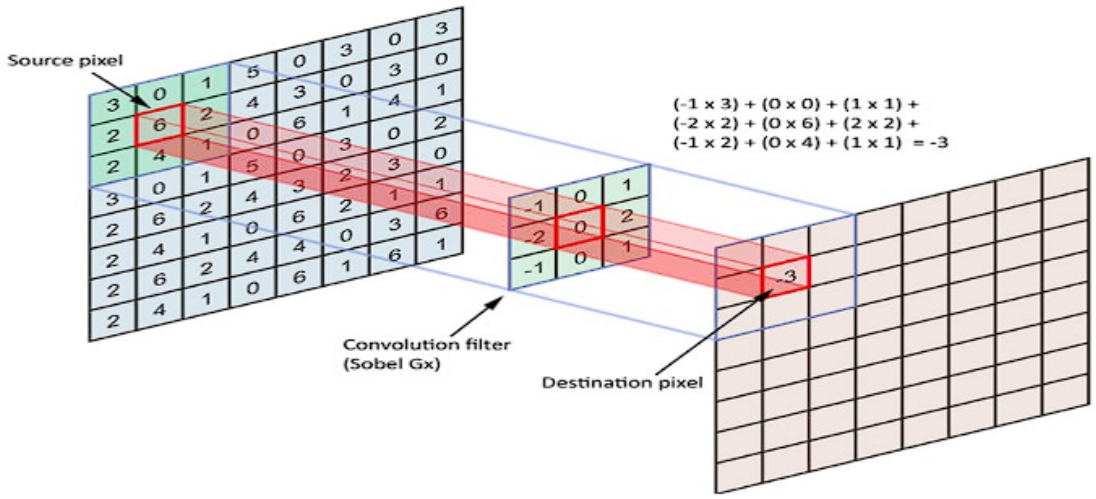


Figure 1.10: Representation of the convolutional layer in CNN [9]

The figure depicts the dot product multiplication of a 3 x 3 filter matrix with a 3 x 3 input image matrix. The sum of the matrix elements determines the output value ("target pixel") on the feature map. After sliding over the input matrix, the filter performs dot product multiplication on each remaining combination of the 3x3 size area to finish the feature map.

Three parameters are used to size the volume of the convolution layer: the depth, the step, and the margin.

- **Layer depth:** The number of convolution nuclei (or the number of neurons associated with the same receptive field).
- **Pitch:** Controls the overlap of receptive fields. The smaller the pitch, the more the receptive fields overlap the greater the output volume will be.
- **The margin (at 0) or zero padding:** Sometimes it is convenient to add zeros to the borders so that the filter adapts to the input matrix. The size of this zero-padding is the third hyperparameter. This margin controls the spatial dimension of the output volume. In particular, it is sometimes desirable to keep the same area as the input volume.

1.4.1.2 Correction layers (RELU)

To improve processing efficiency, a layer that performs a mathematical function (activation function) on output signals is added. The Rectified Linear Units (ReLU) function, F(x)=max (0, x) (Figure 1.3f), causes neurons to return positive values. The RELU layer introduces nonlinearity into the network, resulting in a corrected feature map [18] (see Figure 1.3f).

1.4.1.3 Pooling layers (POOL)

Pooling, often known as picture downsampling, is a crucial aspect of CNN algorithms. Pooling reduces the dimensionality of feature maps, including height and width, while keeping depth. This method reduces the computational power needed to process data and extracts key features from feature maps [31].

The most common pooling layer has 2x2 tiles (width/height) and outputs the maximum input value (see diagram). In this case, we refer to "MaxPool 2x2" (compression by a factor of four).

Figure 1.11 illustrates how "average pooling" can reduce computing time and optimize power calculations. However, reducing the size of the representation may cause information loss. To address this issue, it is advisable to utilize small filters (2x2).

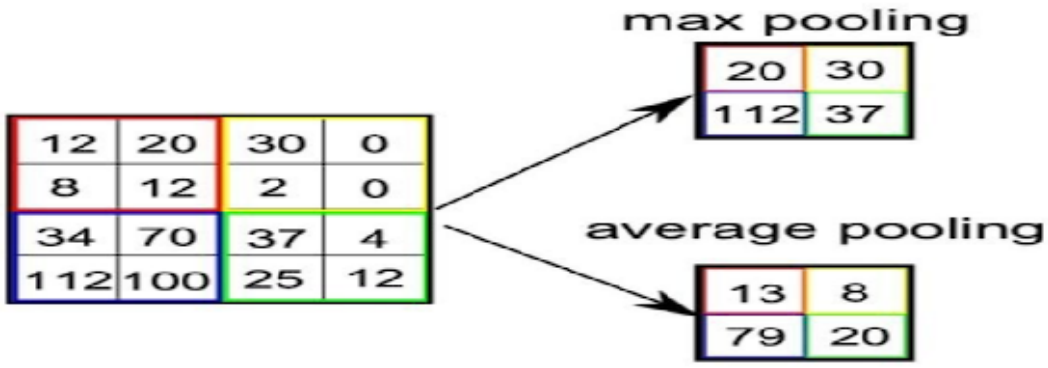


Figure 1.11: Pooling layer representation [10]

1.4.1.4 Fully-connected layer (FC)

This layer takes a vector as input and returns a new vector as output. The algorithm uses a linear combination and potentially an activation function to process the input values. The fully connected layer classifies the input image of the network, returning a vector of size n, where n represents the number of classes in the image classification problem.

Each element of the vector represents the probability that the input image belongs to a specific class. Each value in the input array "votes" in favor of a specific class [31].

1.4.1.5 Loss layer (LOSS)

The loss layer determines how network training penalizes deviations between predicted and actual signals. The deep learning model aims to minimize the loss function, which is achieved through optimization. Various loss functions can be applied. The "Softmax" function calculates probability distributions for output classes [32].

A CNN's completely connected layers can be replaced by convolutional layers, resulting in a fully convolutional network (FCN) [33].

1.4.2 Advantages of convolutional neural networks

Convolutional neural networks have the following main advantages over other techniques [34]:

- **Local connection:** Each neuron is only connected to a few neurons from the previous layer. This reduces several parameters.
- **Weight sharing:** allows several connections to share the same weight instead of having separate weights for each.
- **Down sampling:** Pooling layer uses downsampling to simplify data processing while maintaining important information. Remove unnecessary samples from the feature map to reduce parameter count.

1.4.3 Visual Geometry Group 16

VGG (Visual Geometry Group) is an advanced and deeper CNN architecture, developed by Karen Simonyan and Andrew Zisserman in 2014[35]. VGG was designed to recognize natural images and achieved a high performance of 92.7% top-5 accuracy on the ImageNet dataset, which consists of 1.2 million images of 1000 classes. VGG consists of 16 or 19 layers, depending on the version. VGG was originally designed to recognize natural images, but it can also be used to recognize other types of images, such as artworks, logos, or scenes.

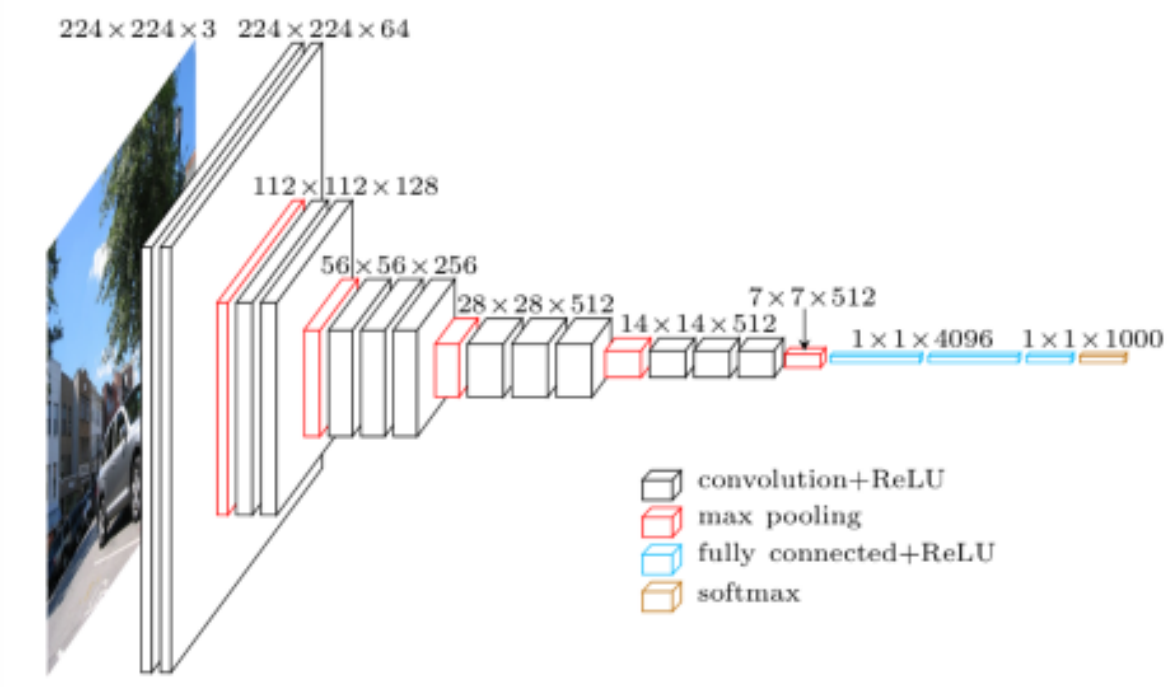


Figure 1.12: VGG 16 architecture [11]

Visual Question Answering

Chapter 2

Visual Question Answering

2.1 Introduction

The multidisciplinary field of VQA demands both computer vision (CV) skills and natural language processing (NLP) capabilities. For instance, computer vision is necessary to identify objects and their attributes (e.g., color) within the image, while NLP helps understand the semantics and context of the question related to the image content.

This chapter explores VQA, its history, applications, and the technical core: feature extraction, fusion techniques, and answer generation. We'll then examine datasets and evaluation metrics before concluding with future directions.

2.2 History

The field of VQA has its roots in early research on integrating computer vision and language processing. While the exact origin isn't attributed to a single person or company, advancements in deep learning have significantly boosted VQA research in recent years.

Early research on integrating computer vision and language for tasks like question answering emerged in the 1970s with systems like SHRDLU [36], developed by Terry Winograd in the 1970s, was a landmark system. It could understand simple natural language questions about a block world and manipulate objects within that world. While not a complete VQA system in the modern sense, SHRDLU demonstrated the potential for this type of integration.

However, significant progress in VQA awaited the development of deep learning techniques. Advancements in Convolutional Neural Networks (CNNs) for image recognition and Recurrent Neural Networks (RNNs) for language processing laid the groundwork for training powerful VQA models on large datasets. These models can now process complex images and answer intricate questions about their content.

Early VQA (2010s):

The field of VQA emerged in the 2010s with the creation of big image datasets accompanied by questions and answers. These datasets enabled researchers to train computers to comprehend the relationship between what they saw and the questions they might answer about it.

Earlier approaches focused on extracting features from the image and text independently before merging them to answer the query.

Modern VQA (2010s - Present):

The field has shifted toward the use of large pre-trained models that learn to recognize both images and text simultaneously. This has resulted in significant improvements in accuracy.

VQA has grown beyond static images to encompass videos, 3D settings, and other visual inputs.

2.3 Applications

VQA has the potential to change accessibility for the blind. In the future, it may be used to answer questions about their surroundings, from identifying objects to understanding signs and labels. This technology goes beyond simple accessibility, with applications in a variety of fields. VQA systems can be used to assess medical images and help medical professionals make diagnoses. They can also be used with satellite imagery to improve our understanding of the world around us. Furthermore, VQA can be used to answer questions based on data visualizations, making difficult material more approachable.

2.4 Comparison with other CV tasks

The topic is considered AI-complete, meaning it addresses the Artificial General Intelligence problem of making computers as intelligent as humans. The problem has also been suggested to be used as a Visual Turing Test [37]. The AI-dream of visual dialogue is a step closer to being achieved with VQA. VQA

differs from other methods in that it requires searching and reasoning on an image's content. For example, to determine the presence of humans, the system needs to detect objects, and to determine if it's raining, a scene needs to be classified. The system requires global knowledge to answer questions about team identities. To determine which player possesses the ball, both commonsense and knowledge reasoning are necessary. Computer Vision (CV) has made significant progress in addressing many problems such as object recognition, counting, and scene classification.

CV task	Representative VQA question
Object recognition	What is in the image?
Object detection	Are there any dogs in the picture?
Attribute classification	What color is the umbrella?
Scene classification	Is it raining?
Counting	How many people are there in the image?
Activity recognition	Is the child crying?
Spatial relationships among objects	What is between cat and sofa?
Commonsense reasoning	Does this person have 20/20 vision?
Knowledge-base reasoning	Is this a vegetarian pizza?

Figure 2.1: The sub-problems the task of visual question answering entails[12]

2.5 VQA

In general, any VQA system includes mainly three phases:

- **Phase 1:** Extraction of question and image features.
- **Phase 2:** Combining question features with image features for joint comprehension.
- **Phase 3:** Generation of an answer.

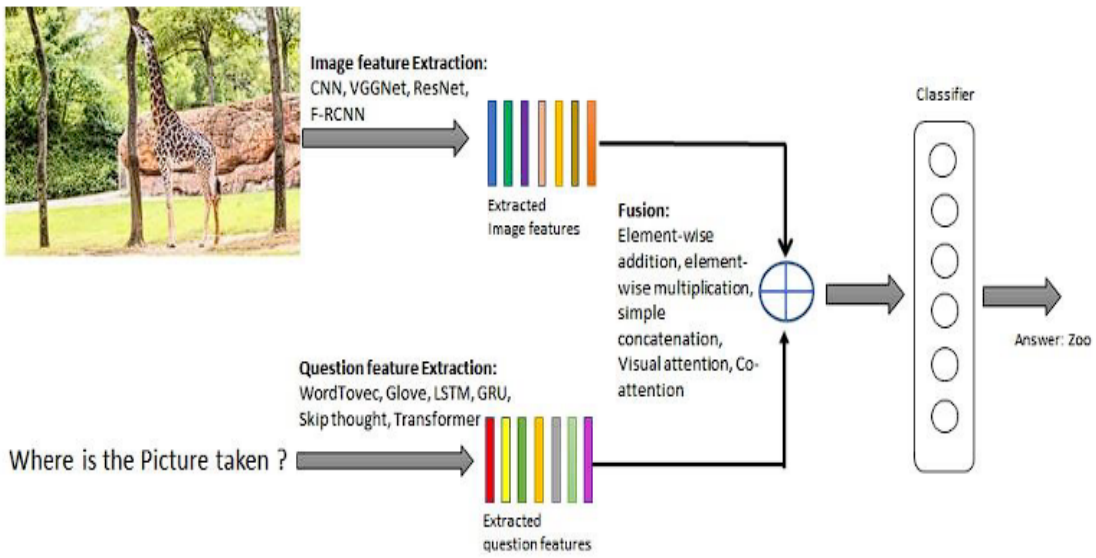


Figure 2.2: VQA task

A variety of techniques have been investigated in the past for each of the aforementioned phases. The most popular method for phase one is to extract visual features using Convolutional Neural Networks and their derivatives, such as GoogleNet ResNet, VGGNet. And Recurrent Neural Networks and their variations like LSTM and GRU for question feature extraction. Phase 2 is a little difficult since it fuses the image features and the question elements to understand how they relate to one another. The literature has examined a variety of techniques, including basic concatenation, element-wise multiplication, and complicated attention networks.

2.6 Feature Extraction

As mentioned earlier, the first stage of a VQA model involves extracting visual features from the image and textual features from the question.

Visual feature extraction is the process of extracting important characteristics from an image and converting them to numerical data for neural network processing. Convolutional Neural Networks (CNNs) are the dominant architecture for image feature extraction in VQA.

Initially, many researchers used VGGNet for the extraction of image features [38, 39, 40, 41, 42]. They used the final hidden layer of VGG-Net as image features as most of the spatial information is retained in the last hidden layer. Sometimes features are extracted from the last pooling layer instead of the final fully-connected layer[43].

Following its introduction, ResNet became a powerful network for image feature extraction [44, 45, 46, 47]. One of the reasons why VGGnet is still preferred over ResNet is because of its simple and lightweight architecture and fast convergence compared to ResNet which is four times heavier than VGGNet. However, with the increased availability of powerful computational resources, several ResNet architectures can be used such as ResNet-18, ResNet-101, and ResNet-152 for VQA tasks.

While prior studies focused on capturing global image features, recent studies suggest that including local information can boost VQA model performance even more. This local information refers to specific objects or regions within the image.

Faster R-CNN, a deep learning architecture, has been employed by several studies [46, 48, 49, 50] to extract these local, object-level features. These local features allow the model to focus on more detailed and relevant aspects of the image that might be crucial for answering the question.

For example, Ilievski et al. [42] went beyond basic object features. They used a ResNet to extract features specifically from objects related to the keywords in the question. This approach combines both object-level and image-level information, resulting in richer visual features for the VQA model.

Since 2018, transformers have revolutionized the field of Vision + Language tasks, including VQA. Their success stems from setting new performance benchmarks.

[50] proposed using separate BERT-based models for image and language features.

LXMERT [51] further pushed the boundaries by building a fully transformer-based VQA model, utilizing BERT for both language and visual modeling. This approach achieved significant improvements over previous transformer-based models.

UNITER [52] introduced a novel method for aligning text and image regions during pre-training, along with conditional masking for pre-training tasks.

[53] explored using object tags as anchor points to facilitate text-image alignment, employing multi-layer transformers built on BERT to integrate image and question features.

Table 2.1 summarizes deep learning models used in the past for image feature extraction.

Model	Paper
VGGNet	LSTM+Q+I [38], AVWAN [39], SAN [43], Facts-VQA [54], DPP [55], QAM [56], RegionSel [57], NMN [58]
ResNet	FDA [42], Bayesian [44], Dense-Sym [45], Code-Mix VQA [46], Hei-Co-atten [47], Rich-img-Region [59], MCB [60], MRN [61], FVTA [62], MUTAN [63],Meta-VQA [57], QTA [64], DCN [45]
GoogleNet	Neural Image QA [64], Multi-Modal QA [65], i-Bowing [66], Smem [67]
F-RCNN	Code-Mixed VQA [46], CAQT [49], QLOB [68], BAN [69], MFB [70], explicit-know-Based [71], Know-Base Graph [72]
BERT	VilBERT [50], LXMERT [51], UNITER [52], Oscar [53], MPC [73], Semantic VLBERT [74]

Table 2.1: Summary of deep learning models used for image feature extraction

The next step in the VQA model is to extract question features. Several word embedding techniques have been used in NLP. Some of these are one-hot encoding, Continuous bag of words (CBOW), co-occurrence matrix, word2vec, etc.

Although the above word embedding techniques extract the contextual information from a given text, new recurrent networks such as LSTM and GRU have proven better at extracting the contextual meaning of the question. But these networks don't exist independently. The basic idea is to create a word vector using any of the word embedding methods discussed above (e.g. word2vec) and these vectors are then fed to LSTM or GRU network.

VQA models extract question features by first converting each word in the question into a numerical representation (word embeddings) using techniques like word2vec. This allows the model to understand relationships between words. While simpler embedding methods exist, VQA often utilizes powerful Recurrent Neural Networks (RNNs) like LSTMs or GRUs. These RNNs excel at capturing the context and meaning within a question by processing words sequentially, considering how each word relates to the others. Importantly, VQA models use both strategies - word embeddings offer individual word meaning and RNNs capture the whole question context - to achieve a more comprehensive understanding.

Some early efforts used simpler methods like Bag-of-Words (BoW) to extract word co-occurrence patterns in questions and responses ([38, 75]).

Others, including [76], used word2vec to represent question words in multiple-choice VQA.

Semantic parsing and word embedding are used in more advanced techniques. For example, [77] used GloVe embeddings in conjunction with LSTMs for question feature extraction, whereas [78] proposed structure (PRS) to categorize question elements (main object, relation, secondary object) and used LSTMs and word2vec for feature engineering.

LSTMs are a popular method for capturing sequential information in questions. Works such as [40, 43, 69] used LSTMs and word embeddings (one-hot or word2vec) to extract semantic information. Skip-thought models, as an alternative to LSTMs, have also been investigated for question encoding ([44, 79]).

Hierarchical attention models have emerged for jointly attending to images and questions at different levels (word, phrase, question) ([44, 47]). These models use embedding matrices, convolutional layers for phrases, and recurrent layers for the entire question.

Beyond LSTMs, Bidirectional LSTMs (Bi-LSTMs) have been used for question encoding, with some studies employing all hidden states from the Bi-LSTM as final question features ([48]). Additionally, some works explore unidirectional GRUs ([53]) while others utilize bidirectional GRUs ([80]) for question embedding.

Transformers have become a dominant force in NLP due to their ability to efficiently process long sequences like questions. This has led to their widespread adoption in VQA models for question feature extraction.

Several studies have successfully employed transformers for this task. [73, 81] utilized the BERT transformer architecture to extract language features from questions. In [72], the authors went a step further by concatenating outputs from multiple BERT layers to capture hierarchical features within the question. Beyond BERT, other transformer models are being explored. For instance, [50] investigated the potential of transformers for both visual and language feature extraction, while [82] leveraged OpenAI's GPT-2 transformer as the language model.

One example of this trend is [59], where researchers replaced the BANs language model with BERT and employed ensemble learning with 20 models. This highlights the effectiveness of transformers and the potential for further advancements in VQA.

Table 2.2 summarizes techniques used for question feature extraction.

Model	Paper
One hot encoding	Ask your Neuron [40], FDA [42], CAQT [49]
BoW	VQA [38], Simple Baseline [75]
GloVe	KAN [77], BAN [69], TipsNTracks [83]
Word2Vec	Where- to-look [76],Yin And Yang [78]
Skip thought	Answer Type pred [44], RSVQA [84]
LSTM	CAQT [49], KAN [77], Yin and Yang [78], Know-AugmentedVQA [85]
GRU	Coarse to fine reasoning [86], DMN [80], BAN [69], DPP [55]
Transformers	VilBERT [50],LXMERT [51], UNITER [52], Oscar [53], MPC [73], Coarse to fine reasoning [23], Hie-Alternation coattention [81], Rich Image region VQA [59], KRISP [87]

Table 2.2: Summary of deep learning models used for question feature extraction

2.7 Fusion Techniques

Understanding the relationship between the image and the question is crucial for VQA models. This process, called information fusion, combines the extracted visual and textual features into a single representation. While basic techniques like concatenation or element-wise operations can achieve this [38, 43, 76], more recent approaches leverage the power of attention mechanisms. Here, the model assigns weights to different elements in each feature vector based on their importance, allowing it to focus on the most relevant aspects of the image and question for answering the query. This attention-based fusion has become a dominant technique for effectively merging image and question features in VQA models.

So, in that context, we can classify attention-based fusion into the following three categories:

1. Visual attention-based joint representation
2. Language attention-based joint representation
3. Co-attention (Visual + Language) based joint representation

VQA models use attention mechanisms to focus on key features of both the image and the question that are significant to answering the query. VQA explores three main types of attention:

Visual Attention: The model prioritizes specific portions of the image based on the question. question features direct the model to attend to key entities in the image, such as objects or places referenced in the inquiry ([42, 76, 61, 88]). For example, for the question "What color is the car?" the model may focus on the car region in the image.

Language Attention: This approach emphasizes important words within the question sentence that are most relevant to the image.

Co-attention: This is a more advanced strategy in which both visual and question characteristics impact one another. Visual features can be utilized to focus on specific parts of the question, whereas question features can direct the model's focus to relevant sections of the image ([47, 69, 62, 89]). Some research looks into different types of co-attention, such as parallel co-attention (producing attention for both the image and the question at the same time) or alternating co-attention (focused on the image and the question sequentially)[47].

Several research have looked into different aspects of attention mechanisms in VQA models. For example, [42] used stacked attention networks for multi-step reasoning, whereas [88] focused on question-guided object attention. [90] developed co-attention, which takes into account features other than spatial information, and [63] provided a novel fusion method based on tensor decomposition.

[91] creates semantic feature vectors by identifying relationships (Subject-Predicate-Object) in the image and using them for feature extraction. Additionally, [92] extracts various features (image, image predicate, question, question predicate) and employs a coarse-to-fine-grained reasoning module to filter irrelevant information. These refined features are then used for a multi-modal learning block, where the model learns the relationships between them before performing semantic reasoning to answer the question.

2.8 ANSWER GENERATION

Once VQA models create a combined representation of image features and question features, they need to generate the most relevant answer. Here's a look at some common answer-generation approaches:

- **Open-Ended Answers:** These can be single words, phrases, or even full sentences.
- **Counting Answers:** This approach focuses on counting objects within the image to answer the question.
- **Multiple-Choice Answers:** The model selects the most fitting answer from a set of pre-defined choices.
- **Binary (Yes/No) Answers:** For these questions, a sigmoid layer (a type of activation function) is often used at the end. The combined representation is passed through one or two fully-connected layers before reaching the classification layer, which typically has a single neuron that outputs the answer ("yes" or "no").

Multiple choice VQA models use the top k most frequent answers in a dataset to forecast the top 5 most matched responses, turning the answer generation problem into a classification task. For VQA 2.0, the majority of researchers used k=1000.

MLP (Multi-layer Perceptron) has continued to be the method of choice for researchers to solve the VQA problem. [38, 39, 43, 44, 46].

Recurrent networks, such as LSTMs, are commonly used to transform joint feature representations into responses to open-ended inquiries. Malinowski et al. [40] used a combination of visual features and LSTMs to represent words.

All of the preceding literature used VQA as an answer selection model. However, in real life, VQA is a generation problem.

Few attempts have been made to tackle this challenge. [93] provided a generative model for Med-VQA and introduced PMC-VQA, a large-scale dataset for medical images. MedVInT was trained by combining visual input from a pre-trained vision encoder and language models.

The authors of [82] combined classification and generation in a single model using a masked transformer. The model can classify images and linguistic traits, as well as generate images, questions, and masked replies.

Although the previous two pieces of literature successfully converted VQA into a generation problem, there remains a challenge when open-domain questions demand external knowledge.

2.9 Datasets

Several datasets are available to experiment with visual question answering. This section discusses some of the most important VQA datasets, as well as their advantages and disadvantages.

Almost all VQA datasets include a triplet of questions, images, and answers. Some datasets include additional information such as image captions, object-related facts, OCR tokens for text in an image, bounding boxes for an image's objects, etc.

The majority of publicly available datasets have been created via a crowdsourcing approach [38, 94, 95] that used Amazon Mechanical Turk to collect image-related questions and answers. In [54], the authors assigned 38 human volunteers to produce the dataset.

2.9.1 DAQUAR

The first VQA dataset is DAQUAR (DAtaset for QUestion Answering on Real-world images) [96], which contains 1,449 photos with 6,797 and 5,674 training and test question-answer pairs, respectively. Questions and answers were created utilizing both automated and manual annotations. The photos were generated using the NYU depthV2 dataset [97].

The authors made the DAQUAR dataset available in two forms: Full DAQUAR and Reduced DAQUAR.

The Reduced version only contains images from 37 classes, including 3,825 training QA pairs and 297 testing QA pairs, whereas the Full dataset includes images from all 894 classes. There are just 25 test images in total. The main drawback of this dataset is its small size, which is insufficient for training complicated deep learning models. The dataset features indoor scenes with inadequate lighting, making it difficult to answer the question. Human accuracy on this dataset is roughly 50%.

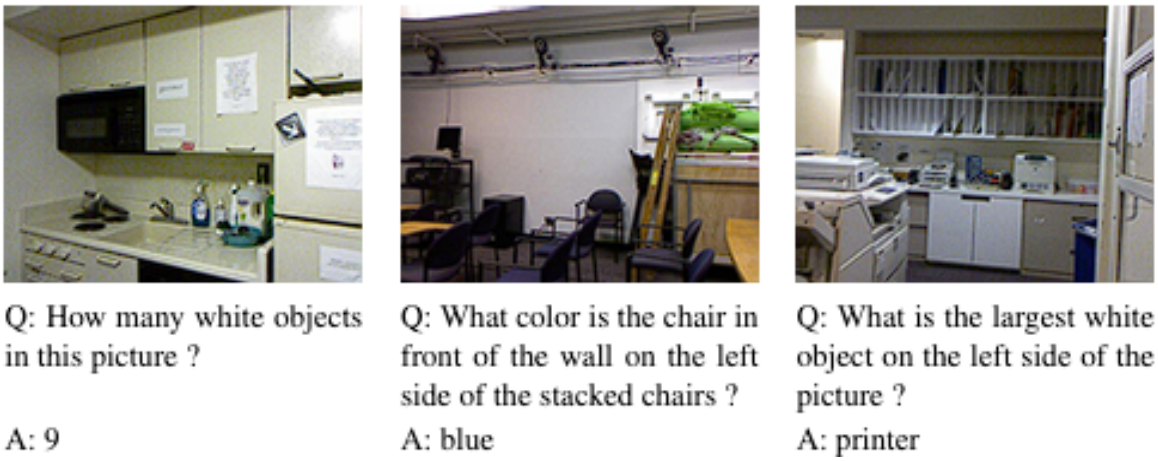


Figure 2.3: Examples from The DAQUAR dataset

2.9.2 COCO-QA

The COCO-QA dataset [98] was created from Microsoft's COCO pictures [99](see Figure 2.4).

Figure 2.4: Examples of annotated images from the COCO dataset

The datasets contain 1,23,287 pictures. There are 1,17,684 questions generated automatically from the image captions, divided into 78,786 training questions and 38,948 testing questions.

Figure 2.5: Examples from The COCO-QA dataset

The dataset has only four sorts of questions: object, counting, color, and location. Also, because the questions are generated automatically, they are not grammatically correct, and all of the responses are one word(Figure 2.6 shows an example of a grammatical error).

COCO-QA: What does an intersection show on one side and two double-decker buses and a third vehicle,?
Ground Truth: Building

Figure 2.6: Examples of grammatical errors in the question [13]

2.9.3 VQA 1.0

The VQA 1.0 dataset [38] consists of both real and abstract images. The total number of pictures is 204721, with 123287 used for training and validation and 81434 preserved for testing. All photos are from the Microsoft COCO dataset. It also includes 50,000 abstract graphics. For abstract images, the train, validation, and test splits are 20k, 10k, and 20k, respectively.

All the questions and annotations were human-generated. Each image has three questions and ten human-generated solutions.

VQA V1 offers two sorts of answers: multiple-choice and open responses. In open-answer mode, the model selects the answer with the highest probability from all k potential responses. In multiple-choice mode, the model selects the answer with the highest probability from the given options.

There are a few issues with the dataset. Many questions can only be answered by asking more questions. Many questions lack a definitive answer because they are too subjective. There are no questions in the dataset that require strong reasoning or common sense.

2.9.4 VQA 2.0

Compared to other datasets, the VQA 2.0 dataset [14] is relatively larger. In addition to 204,721 images from the COCO dataset, it includes 50,000 abstract cartoon images. There are three questions per image and ten answers per question. To achieve this, a team of Amazon Mechanical Turk workers generated the questions and another team wrote the answers.

To make VQA 1.0 more balanced, the authors added more images.

They provided a complementary image for each image in VQA 1.0 so that the question Q still makes sense, but the answer is different.

For example, they added the complementary picture I', for which the query Q still makes sense but the answer is A', to each triple (I,Q,A):(Image, Question, and Answer) (see Figure 2.7.

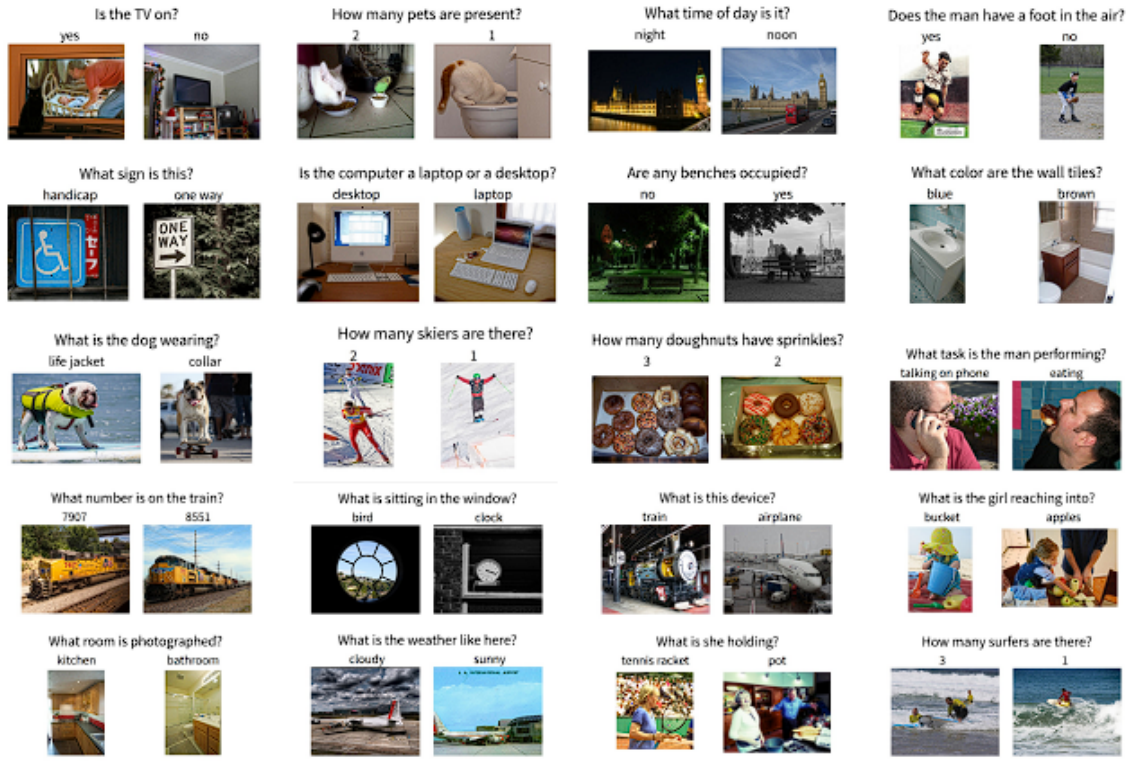


Figure 2.7: Examples from the VQA 2.0 where each question has two similar images with different answers to the question[14].

In essence, this eliminates the language bias found in the initial VQA sample. The balanced VQA dataset has 443K train, 214K validation, and 453K test (question, picture) pairings, for a total of 1.1 million (image, question) pairs, approximately 13 million associated replies, and 200K images.

2.9.5 CLEVR

CLEVR [100] is a dataset of synthetic images of 3D objects of various shapes. The training set consists of 70,000 images and 699,989 questions. A validation set consists of 15,000 photos and 149,991 question-answer pairs, whereas a test set consists of 15,000 images and 14,988 question-answer pairs.

The dataset was created to assess the model's ability to perform complicated visual reasoning, as the questions in the dataset require complex reasoning. While useful for performing high-level complex reasoning, it is not suitable for real images as real images are natural and noisy.

2.9.6 VizWiz

Vizwiz [79], was primarily released to encourage the community to develop VQA systems to assist blind individuals in their daily lives.

The images in this collection were captured by genuine blind people using a mobile app. It has 32842 photos and one question per image. Answers are generated by humans.

The issue with this dataset is that the photos are of low quality. Additionally, some questions remain unanswerable because of insufficient quality.

2.9.7 Visual Genome

Visual Genome [101] contains 108,249 photos from both YFCC100M [102] and COCO images. There are 1.7 million QA pairs for photos, with an average of 17 per image. As of this writing, Visual Genome has the largest VQA dataset. Due to its recent introduction, no methodologies have been examined beyond the authors' set baselines.

Q: What shape is the bench seat ?

A: oval, semi circle, curved, curved, double curve, banana, curved, wavy, twisting, curved

Q: What color is the stripe on the train ?

A: white, white, white, white, white, white, white, white, white

Q: Where are the magazines in this picture ?

A: On stool, stool, on stool, on bar stool, on table, stool, on stool, on chair, on bar stool, stool

Figure 2.8: Examples from The VQA-real dataset

Q: Who looks happier ?.

A: old person, man, man, man, old man, man, man, man, man, grandpa

Q: Where are the flowers ?

A: near tree, tree, around tree, tree, by tree, around tree, around tree, grass, beneath tree. base of tree

Q: How many pillows ?

A: 1, 2, 2, 2, 2, 2, 2, 2, 2, 2

Figure 2.9: Examples from The VQA-abstract dataset

2.9.8 SHAPES

The SHAPES dataset [[58]] includes 15,616 synthetic images of colorful 3D shapes with 244 questions categorized as attribute, relationship, and position. Although the dataset is useful for testing mode's reasoning capability, it is limited to yes/no questions and does not generalize well to real-world images.

2.9.9 Visula7w

This dataset consists of 47,300 images [103] taken from the COCO dataset. This collection includes 327,929 QA pairs, 1,311,756 human-created multiple-choice questions, and 561,459 object groundings from 36,579 categories. The dataset includes 7W questions: what, when, where, why, who, how, and which. It only provides multiple-choice responses.

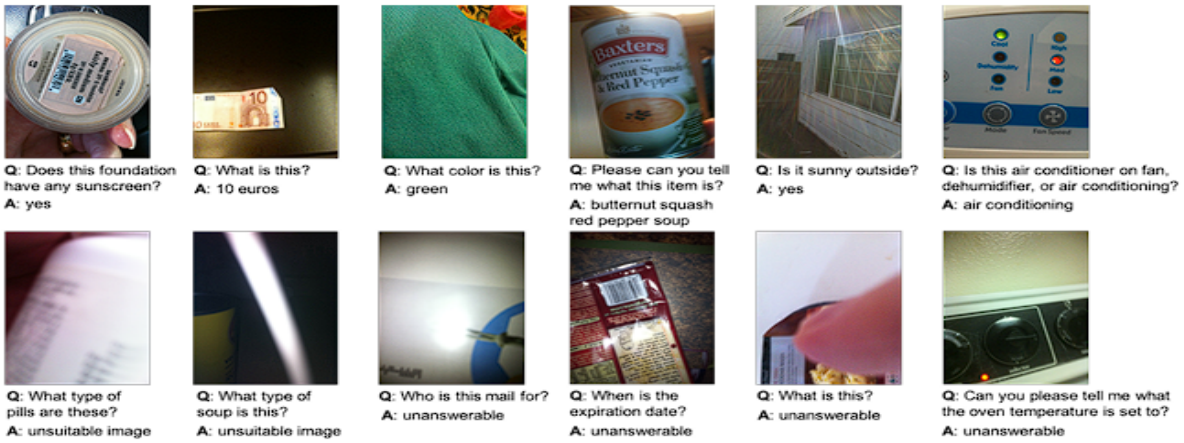


Figure 2.10: Examples from The VizWiz dataset

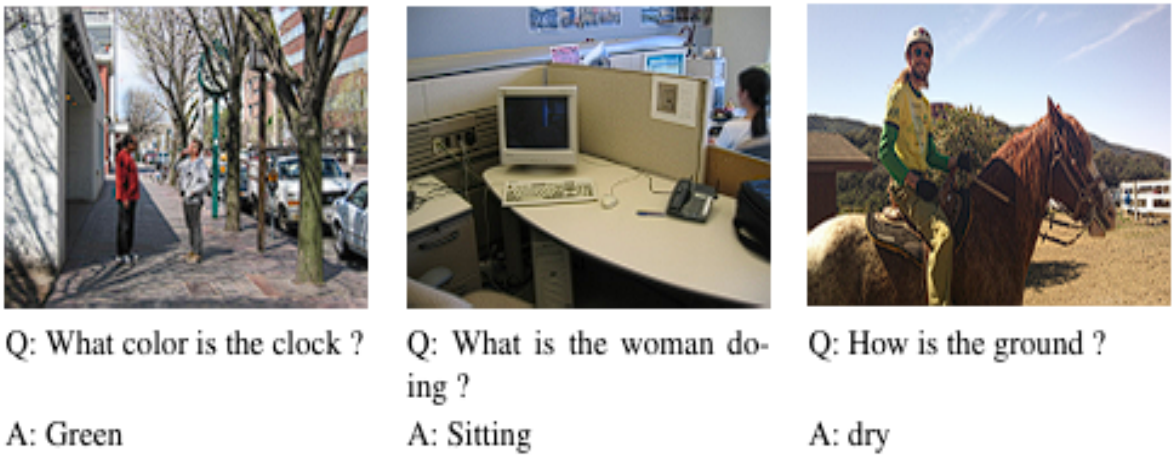


Figure 2.11: Examples from The Visual Genome dataset

2.10 Evaluation Metrics

Several evaluation measures have been proposed in the literature to assess VQA models. The most commonly used metric is VQA accuracy. Other useful metrics include BLUE, WUPS, METEOR, and human judgment.

2.10.1 Accuracy

[38] proposed a new accuracy metric to evaluate the VQA model that accounts for variation in human answers. This is also known as accuracy based on human consensus. The formula for calculating accuracy is:

$$\text{Accuracy}_{\text{VQA}} = \min\left(\frac{n}{3}, 1\right), \tag{2.1}$$

For an output to be considered accurate, it needs to be consistent with the judgments of at least three humans.

2.10.2 Wu and Palmer Similarity (WUPS)

Wu et al. [104] proposed an alternative metric called WUPS that can be used to assess the accuracy of the VQA model. The method compares the generated and actual answers based on semantic similarity to determine the accuracy of the prediction.

WUPS takes the model-generated and ground truth answers as inputs and returns a score ranging from 0 to 1 based on the List Common subsumer between the two. It is assumed that the path length and depth in path-based metrics determine how closely two concepts are related to one another. So, terms that are completely different yet have the same semantic meaning will be penalized less in this context. For instance, "chair" and "wooden chair" have a similarity of 0.96, whereas "chair" and "furniture" share 0.8 similarity. However, WUPS has a disadvantage in that it may assign a high value to distant concepts. For example, "sea" and "water" may be ascribed a 0.4 similarity.

A solution to this is to utilize a threshold WUPS score, in which a score less than a particular threshold is reduced by certain parameters. A threshold of 0.9 and a scale of 0.1 was proposed by [96].

Several authors [40, 105, 106] have utilized this threshold WUPS, in addition to normal accuracy, to evaluate model performance.

Another issue with WUPS is that it returns high scores for comparable concepts, which makes it difficult to measure the answer connected to an object's attributes. For example, replies "Red" and "Yellow" will obtain high scores because they are semantically similar.

2.10.3 Bilingual Evaluation Understudy (BLUE)

BLEU [107] is another important metric that multiple authors [45, 65] used. It also generates 0 to 1 output, similar to WUPS. With values closer to one, the expected and actual results are more similar, as represented by this number. The method determines the fraction of matching n-grams between the ground truth answer and the anticipated response. The comparison is made regardless of the order of words.

$$P_n = \frac{\sum_{n\text{-grams}} \text{count}_{\text{clip}}(n\text{-gram})}{\sum_{n\text{-grams}} \text{count}(n\text{-gram})} \tag{2.2}$$

2.10.4 METEOR

The METEOR [108] (Metric for evaluation of translation with explicit ordering) measure considers several characteristics of translation quality, including precision, recall, stemming, and synonymy. It is intended to overcome some of the shortcomings of the BLEU metric, such as its reliance on n-gram precision and insensitivity to word order.

To compute the METEOR, first determine the precision and recall for all unigrams. Then it calculates a harmonic mean of the precision and nine times the recall.

$$M_{\text{mean}} = \frac{10P}{P + 9R} \tag{2.3}$$

The following explains how METEOR estimates a penalty for a specific alignment to allow for longer matches. To begin, all the unigrams in the system translation that corresponds to unigrams in the reference translation are grouped into the fewest number of chunks possible, ensuring that each chunk's unigrams are in adjacent positions in both the system translation and the reference translation, where they are also mapped to unigrams in adjacent positions.

The second portion is a penalty function, which is defined as follows:

$$\text{Penalty} = 0.5 \times \left(\frac{\text{no. of chunks}}{\text{no. of unigrams}} \right)^3 \tag{2.4}$$

Finally, the score is calculated as follows:

$$\mathrm{s} = M_{\mathrm{mean}} \times (1 - penalty) \tag{2.5}$$

2.10.5 Human judgment

Human judgment is the most reliable and accurate evaluation approach. Humans evaluate all answers generated by the model. However, this procedure is both expensive and time-consuming. This can serve as a secondary technique of evaluation.

2.11 Conclusion

In conclusion, this chapter has given a thorough overview of VQA. We investigated the history of VQA research, its varied applications across fields, and the technical features that allow machines to answer questions about images. Advances in feature extraction, fusion approaches, and answer generating methodologies have allowed VQA to become a powerful tool with great promise.

Contribution

Chapter 3

Contribution

3.1 Introduction

In this chapter, we present the design, development, and implementation of a virtual assistant named 'AIRIS', focusing on its architecture, voice interaction mechanisms, and the integration of a VQA model. We will discuss the training procedure of the VQA model and evaluate its performance. The primary contribution of this study is the integration of the VQA model, which allows users to inquire about the content of images using voice commands.

The motivation behind this project stems from the increasing demand for intelligent systems capable of understanding and responding to human language in a natural and meaningful way. While many virtual assistants exist today, few are capable of accurately interpreting and answering questions about visual content. This project addresses this gap by developing a VQA system trained on the VQA 2.0 dataset, enabling the assistant to understand and answer questions about images.

AIRIS integrates various essential libraries to function effectively. The SpeechRecognition library handles voice input, pyttsx3 manages text-to-speech output, and pywhatkit facilitates music playback via YouTube. Additionally, the datetime module is used to display the current time.

3.2 Tools and Environment

3.2.1 Programming Environment

Python 3.8 was the programming language used to develop and test the model. This version of Python provides a stable and comprehensive library ecosystem, which is critical for deep learning research.

3.2.2 Jupyter Notebook

For an interactive coding environment, I used Jupyter Notebook. This tool allowed for the smooth execution of code in a cell-based structure, making it easy to debug, visualize outcomes, and document the experimental process.

3.2.3 Libraries

PyTorch: the primary deep learning framework, was chosen because to its dynamic computation graph, ease of use, and extensive community support. PyTorch allowed for quick building models, training, and evaluation.

pyttsx3: This text-to-speech conversion package enabled voice outputs. It is a platform-independent library that works offline and offers a simple interface for speech synthesis.

SpeechRecognition: Voice input capabilities were integrated using the SpeechRecognition 3.10.4 library. This library provides support for a variety of speech-to-text engines and APIs, allowing spoken language to be converted into text data for use in VQA tasks.

PyWhatKit: is a Python library with various helpful features. It's easy to use and does not require any additional setup. Currently, it is one of the most popular libraries for WhatsApp and YouTube automation.

datetime: The datetime module supplies classes for manipulating dates and times. While date and time arithmetic is supported, the focus of the implementation is on efficient attribute extraction for output formatting and manipulation.

3.2.4 Hardware Specifications

DELL Inspiron 7415 2-in-1 Operating system: Windows 11 Home Processor: AMD Ryzen 7 5700U with Radeon Graphics 1.80 GHz RAM: 16.0 GB (15.4 GB usable) Storage: 262 GB

3.3 System architecture

This section provides a detailed explanation of the system architecture of AIRIS, the virtual assistant. The architecture is designed to seamlessly integrate voice command processing, natural language understanding, and a VQA system. The system is made up of important components that work together to deliver a rich and interactive user experience.

3.3.1 High-Level Architecture

The high-level architecture of AIRIS is made up of the following components:

- **Voice Command Interface:** Captures and processes the user's voice inputs.
- **Command Understanding:** This module interprets and parses the user's voice commands
- **Task Execution Modules:** Perform specialized activities such as telling the time, playing music on YouTube, and handling image inquiries.
- **VQA Model:** A custom-trained model that answers image-related queries.
- **Response Generation:** Converts system replies into speech output.

3.3.2 Component descriptions

- **Voice Command Interface:**
  - Utilizes the SpeechRecognition library to capture voice input.
  - Converts voice input into text for further processing.
- **Command Understanding:**
  - Analyzes and parses the text command to determine the user's intent.
  - Routes the command to the appropriate task execution module.
- **Task Execution Modules:**
  - **Time-Telling Module:** Uses the datetime module to retrieve and vocalize the current time.
  - **Music Playback Module:** Utilizes the pywhatkit library to play music on YouTube based on user requests.
  - **Image Inquiry Module:** Processes questions about images using the VQA model.

- **VQA Model:**
  - Trained on the VQA 2.0 dataset.
  - Uses VGG16 for visual feature extraction and GloVe embeddings for text processing.
  - Employs LSTM layers for question encoding and an element-wise multiplication for modality fusion.
  - Outputs an answer to image-related questions.
- **Response Generation**
  - Converts text responses into speech using the pyttsx3 library.
  - Provides vocal feedback to the user.

3.3.3 Data flow

1. **Voice Input:** The user speaks a command.
2. **Speech Recognition:** The voice command is captured and converted to text.
3. **Command Processing:** The text is analyzed to identify the command.
4. **Command routing:** The command is routed to the appropriate module.
5. **Task Execution:** The appropriate module processes the command (*e.g.*, fetching time, playing music, answering a question about an image).
6. **Response Generation:** The system generates a text response and converts it to speech.
7. **Voice Output:** The response is spoken to the user.

3.3.4 Example Scenarios

3.3.4.1 Scenario 1: Playing Music on YouTube

User Command: "Play 'Bohemian Rhapsody' by Queen on YouTube."

1. **Voice Input:** The user says, "Play 'Bohemian Rhapsody' by Queen on YouTube".
2. **Speech Recognition:** The voice command is captured and converted to text.
3. **Command Processing:** The system identifies the command as a request to play a specific song on YouTube.
4. **Command routing:** The command is routed to the Music Playback Module.
5. **Task Execution:** The system uses the pywhatkit library to search for and play "Bohemian Rhapsody" by Queen on YouTube.
6. **Response Generation:** The system generates a confirmation response.
7. **Voice Output:** AIRIS says, "Playing 'Bohemian Rhapsody' by Queen on YouTube."

3.3.4.2 Scenario 2: Telling Time

User Command: "What time is it?"

1. **Voice Input:** The user says, "What time is it?".
2. **Speech Recognition:** The voice command is captured and converted to text.
3. **Command Processing:** The system identifies the command as a request for the current time.
4. **Command routing:** The command is routed to the Time-Telling Module.
5. **Task Execution:** The system retrieves the current time using the datetime module.
6. **Response Generation:** The system generates a text response with the current time.
7. **Voice Output:** AIRIS says, "The current time is 3:45 PM."

3.3.4.3 Scenario 3: Image Inquiry

User Command: "What is in this picture?" (User shows a picture of a cat.)

1. **Voice Input:** The user says, "What is in this picture?" while showing an image of a cat.
2. **Speech Recognition:** The voice command is captured and converted to text.
3. **Command Processing:** The system identifies the command as a request for image inquiry.
4. **Command routing:** The command is routed to the Image Inquiry Module.
5. **Task Execution:** The system captures the image shown by the user.
   The VQA model processes the image and the question.
   The VQA model outputs the answer: "This is a cat."
6. **Response Generation:** The text response is converted to speech.
7. **Voice Output:** AIRIS says, "A cat."

3.4 Voice command processing

Voice command processing is an essential component of AIRIS, allowing the virtual assistant to interpret and respond to input from users in natural language. This section describes the procedures involved in processing voice commands, from recording the voice input to producing a response.

1. **Voice Input Capture:**
   - The technology uses a microphone to record the user's voice.
   - The audio input is streamed in real-time to facilitate quick processing.
2. **Speech-to-Text Conversion:**
   - The SpeechRecognition library handles the audio input.
   - It transforms spoken words into text via Google's Web Speech API or another speech recognition service.
   - The text output is afterward routed to the Natural Language Understanding module.
3. **Natural Language Understanding:**
   - The parsed text is analyzed to determine the user's intention.
   - The command is routed based on the recognized intent.
4. **Command Execution:**
   - The command is sent to the relevant task execution module depending on the recognized intent.
   - Task execution modules are responsible for specific operations such as timekeeping, music playback, and VQA model querying.
5. **Response Generation:**
   - The output of the task execution module is translated to a text response.
   - The pyttsx3 library translates text responses to speech.
   - The generated speech is played back to the user via speakers.

3.5 Telling time functionality

The telling time functionality is a key element of AIRIS, allowing the virtual assistant to provide the current time upon request. This section describes the implementation and processing flow of the time-telling feature.

1. **Voice Input Capture:**
   - The system captures the user's voice command using a microphone.
2. **Speech-to-Text Conversion:**
   - The SpeechRecognition library processes the audio input and converts it into text.
3. **Intent Recognition:**
   - The Command Processing module analyzes the text to determine the user's intent.
   - The system identifies that the user is requesting the current time.
4. **Time Retrieval:**
   - The system uses the datetime module to get the current time from the system clock.
   - The time is formatted into a user-friendly string.
5. **Response Generation:**
   - A text response is generated that includes the current time, *e.g.*, "The current time is 3:45 PM."
6. **Text-to-Speech Conversion:**
   - The pyttsx3 library converts the text response into speech.
7. **Voice Output:**
   - The system outputs the generated speech through the speakers.

3.6 Playing Music on YouTube

AIRIS's functionality includes the ability to play music on YouTube. This function allows users to request certain songs or artists using voice commands, which the system subsequently plays on YouTube. This section describes the implementation and processing flow for the music playback capability.

1. **Voice Input Capture:**
   - The system captures the user's voice command using a microphone.
2. **Speech-to-Text Conversion:**
   - The SpeechRecognition library processes the audio input and converts it into text.
3. **Intent Recognition:**
   - The Command Processing module analyzes the text to determine the user's intent.
   - The system identifies that the user is requesting to play music on YouTube.
4. **YouTube Search and Playback:**
   - The pywhatkit library is used to search for the requested song or artist on YouTube.
   - The library then initiates playback of the first result matching the search query.

5. **Response Generation:**
    - A text response is generated to confirm the action.
6. **Text-to-Speech Conversion:**
    - The pyttsx3 library converts the text response into speech.
7. **Voice Output:**
    - The system outputs the generated speech through the speakers.

3.7 Image Inquiry Functionality

1. **Voice Input Capture:**
    - The system captures the user's voice command using a microphone.
2. **Speech-to-Text Conversion:**
    - The SpeechRecognition library processes the audio input and converts it into text.
3. **Intent Recognition:**
    - The Command Processing module analyzes the text to determine the user's intent.
    - The system identifies that the user is asking a question about an image.
4. **Image Capture or Selection:**
    - If the user needs to provide an image, the system captures it using a camera.
    - Alternatively, the user can select an existing image from a gallery.
5. **Question Processing:**
    - The VQA model takes the image and the text question as inputs.
    - The model processes the visual features (extracted using VGG16) and the text (embedded using GloVe embeddings) to generate an answer.
6. **Answer Generation:**
    - The VQA model produces a text answer based on its understanding of the image and the question.
7. **Text-to-Speech Conversion:**
    - The pyttsx3 library converts the text answer into speech.
8. **Voice Output:**
    - The system plays the generated speech through the speakers.

3.8 Integration of VQA Model with the Virtual Assistant

1. **Image Acquisition:**
    - The system captures a new image using a camera or allows the user to select an existing image from a gallery.
2. **Voice Command Processing:**
    - **Voice Input Capture:** The user's voice command is captured using a microphone.
    - **Speech-to-Text Conversion:** The SpeechRecognition library converts the audio input into text.
    - **Intent Recognition:** The Command Processing module analyzes the text to identify the user's intent to inquire about the image.
3. **Feature Extraction:**
    - **Visual Features:** The VQA model uses VGG16 to extract visual features from the image.
    - **Text Features:** The text question is processed using GloVe embeddings to convert it into a meaningful representation. This embedding is then passed through 2 LSTM layers.
4. **Model Inference:**
    - **Feature Fusion:** The visual and text features are fused using element-wise multiplication.
    - **Classification:** The fused features are passed through a classifier to generate the answer to the user's question.
5. **Response Generation:**
    - **Text-to-Speech Conversion:** The generated answer is converted into speech using the pyttsx3 library.
    - **Voice Output:** The response is delivered to the user through speakers.

3.9 VQA model

3.9.1 Dataset

The VQA 2.0 Dataset is available to download at [1]. The dataset is structured as follows:

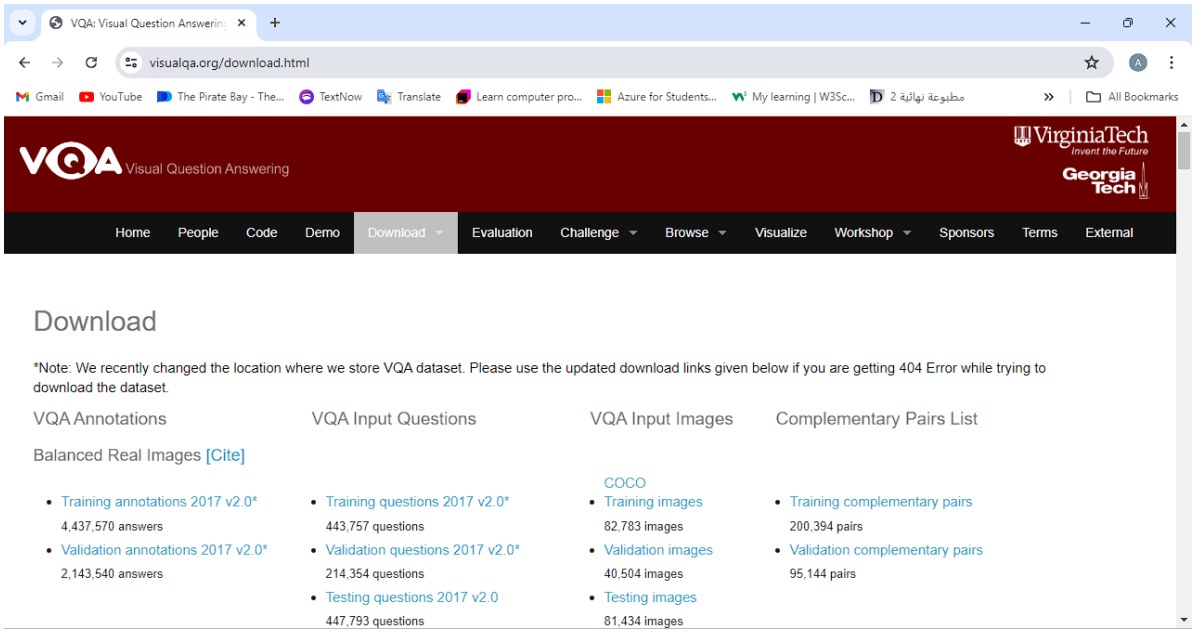


Figure 3.1: The Official VQA 2.0 Website

Images: There are 204,721 COCO images. These are split into:

- 82,783 images for training.
- 40,504 images for validation.
- 81,434 images for testing.

Questions: It has 1,105,904 questions, divided as follows:

- 443,757 questions for training.
- 214,354 questions for validation.
- 447,793 questions for testing.

Annotations: 11,059,040 ground truth answers. These are divided into:

- 4,437,570 training answers.
- 2,143,540 validation answers.

However, the annotations for the test set are not provided to ensure fair evaluation and benchmarking. Withholding the test annotations enables a standardized evaluation method. Researchers can submit their model predictions to a centralized evaluation server, which will calculate the performance measures. This ensures that all models are tested under the same conditions, allowing for a fair comparison among models.

3.9.2 Data preprocessing

After downloading and extracting the data, It is organized in the following directory structure:

[1] https://visualqa.org/download.html

Name	Date modified	Type	Size
train2014	4/26/2024 12:52 PM	File folder	
val2014	4/18/2024 2:29 PM	File folder	
v2_mscoco_train2014_annotations.json	6/15/2024 11:09 AM	JSON File	347,185 KB
v2_mscoco_val2014_annotations.json	6/15/2024 11:09 AM	JSON File	167,737 KB
v2_OpenEnded_mscoco_train2014_questions...	6/15/2024 11:10 AM	JSON File	40,975 KB
v2_OpenEnded_mscoco_val2014_questions.j...	6/15/2024 11:10 AM	JSON File	19,774 KB

Figure 3.2: Downloaded files

There are at least 3 questions (5.4 questions on average) per image, 10 ground truth answers per question, and 3 plausible (but likely incorrect) answers per question.

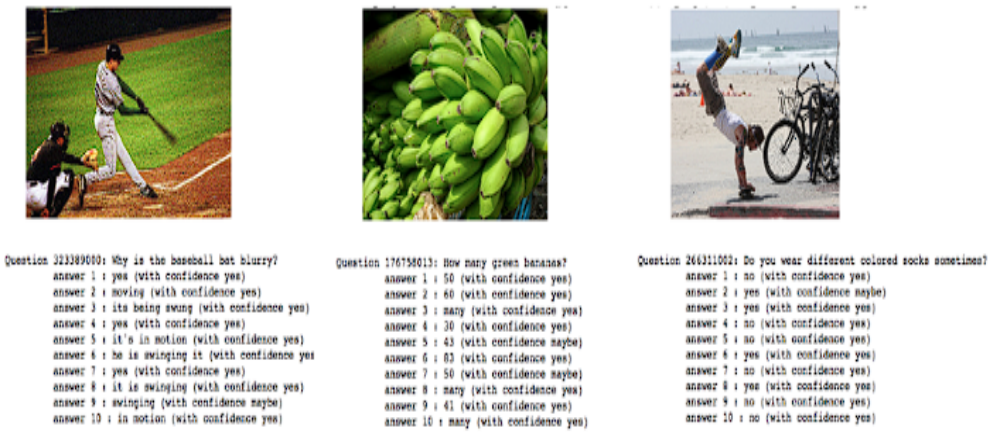


Figure 3.3: Examples from the VQA dataset. Each image contains three questions and each question has 10 candidates answers along with a confidence rate

3.9.2.1 Preprocessing images

Preprocessing is necessary to prepare the images in a format that is compatible with VGG16, which requires input images to be normalized in a specified way. This guarantees that the input data matches the data used to train the model.

- **Resize:** Ensures the shorter side of the image is 256 pixels while maintaining the aspect ratio. This step helps standardize the input size for the subsequent operations.
- **CenterCrop:** After resizing, the image is cropped to a 224x224 pixel square from the center. This step ensures the final input size is compatible with the VGG16 model, which expects 224x224 pixel images.
- **ToTensor:** Converts the image to a PyTorch tensor, changing the pixel values from the range [0, 255] to [0.0, 1.0]. This conversion is necessary because PyTorch models require tensor inputs.
- **Normalize:** Standardizes the pixel values for each channel (Red, Green, Blue) based on the predefined mean and standard deviation (mean=[0.485, 0.456, 0.406] and std=[0.229, 0.224, 0.225]). This normalization step is crucial because the VGG16 model was trained on images normalized in this manner, ensuring consistent input data distribution.

The preprocessing pipeline is implemented using the PyTorch 'transforms' module as follows:

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

3.9.2.2 Tokenizing Questions

The first step in the text preprocessing pipeline is to tokenize and convert the questions to lowercase. This provides consistency and helps reduce the vocabulary size. The tokenization removes punctuation and splits the text into words.

```
def tokenize(text):
    text = text.lower()
    text = re.sub(r'[^a-z0-9\s]', '', text) # Remove punctuation
    tokens = text.split()
    return tokens
```

A vocabulary is then built based on the training set questions. Only words with a frequency greater than a specific threshold are included in the vocabulary. This helps in managing the size of the vocabulary and excludes rare words that might not contribute significantly to the model's performance.

```
train_tokens = [tokenize(q['question']) for q in train_questions['questions']]
word_counter = Counter([token for tokens in train_tokens for token in tokens])
vocab = [word for word, freq in word_counter.items() if freq >= 5]
word_to_idx = {word: idx for idx, word in enumerate(vocab, 1)}
word_to_idx['<PAD>'] = 0 # Add padding token
```

3.9.2.3 Encode Answers

Answers are converted to lowercase, and rare answers are filtered out to ensure the answer space is manageable.Each answer is then assigned a unique integer for simple comparison and classification during training.

```
def preprocess_answers(annotations):
    answers = []
    for annotation in annotations['annotations']:
        answer = annotation['multiple_choice_answer'].lower()
        answers.append(answer)
    return answers

train_answers = preprocess_answers(train_annotations)
answer_counter = Counter(train_answers)
answer_vocab = [answer for answer, freq in answer_counter.items() if freq >= 5]
answer_to_idx = {answer: idx for idx, answer in enumerate(answer_vocab)}
```

3.9.2.4 Create Input-Output Pairs

Each question is mapped to its corresponding image and encoded answer, forming the input-output pairs required for training the model. The questions are then converted to sequences of integers based on the vocabulary created earlier, and these sequences are padded to ensure uniform length.

```
def question_to_sequence(question, word_to_idx, max_len=24):
    tokens = tokenize(question)
    sequence = [word_to_idx[token] for token in tokens if token in word_to_idx]
    return sequence

train_input_output_pairs = [(q['image_id'], q['question'], train_qid_to_ans[q['question_id']])
                        for q in train_questions['questions'] if q['question_id'] in
                            train_qid_to_ans]
train_sequences = [question_to_sequence(q[1], word_to_idx) for q in train_input_output_pairs]
train_sequences_padded = pad_sequences(train_sequences, maxlen=24, padding='post')

train_image_ids = [q[0] for q in train_input_output_pairs]
train_labels = [q[2] for q in train_input_output_pairs]
```

3.9.3 Feature extraction

3.9.3.1 Image features

Feature extraction is a critical step in the VQA model, as it converts raw image data into a form that the model can process and learn from. In this study, we used the VGG16 network, a well-established convolutional neural network pre-trained on the ImageNet dataset, to extract features from both the training and validation images.

The VGG16 model, loaded with pretrained weights from ImageNet, was modified to output features from the penultimate layer rather than the final classification layer. This modification is achieved by altering the classifier part of the network:

```
from torchvision import models
import torch.nn as nn

# Load the VGG16 model with pretrained weights
model = models.vgg16(weights=models.VGG16_Weights.IMAGENET1K_V1)
model.classifier = nn.Sequential(*list(model.classifier.children())[:-1])
model = model.eval() # Set to evaluation mode
```

The following function was used to extract features from each image:

```
from PIL import Image
import torch

def extract_features(image_path, model, preprocess):
    # Open and preprocess the image
    image = Image.open(image_path).convert('RGB')
    image = preprocess(image).unsqueeze(0) # Add batch dimension

    # Extract features using the model
    with torch.no_grad():
        features = model(image)

    return features
```

In this function, '$image_path$' refers to the path of the image file, 'model' is the modified VGG16 model, and 'preprocess' is the preprocessing pipeline described earlier. The image is opened and converted to RGB format, ensuring compatibility with the VGG16 model. The preprocessing steps are applied, and the resulting tensor is passed through the model to obtain the feature representation.

By extracting features from the second-to-last layer of the VGG16 model, we capture high-level image representations that are rich in semantic information. These features serve as the input to the subsequent stages of the VQA model, enabling it to correlate visual content with textual questions effectively.

This feature extraction process was applied uniformly to all images in both the training and validation datasets, ensuring consistency and enabling the model to learn from a standardized set of visual inputs.

3.9.3.2 Question Features

GloVe embeddings are used to extract relevant information from questions. These embeddings capture semantic relationships between words by representing them in a high-dimensional space. The pre-trained GloVe embeddings are loaded, and an embedding matrix is created, where each word in the vocabulary is mapped to its corresponding GloVe vector.

```
def load_glove_embeddings(glove_file):
    embeddings_index = {}
    with open(glove_file, 'r', encoding='utf-8') as f:
        for line in f:
            values = line.split()
            word = values[0]
            coefs = np.asarray(values[1:], dtype='float32')
            embeddings_index[word] = coefs
    return embeddings_index

glove_embeddings = load_glove_embeddings('glove.6B//glove.6B.300d.txt')
embedding_dim = 300
embedding_matrix = np.zeros((len(word_to_idx) + 1, embedding_dim))

for word, idx in word_to_idx.items():
    embedding_vector = glove_embeddings.get(word)
    if embedding_vector is not None:
        embedding_matrix[idx] = embedding_vector
```

Each sequence of tokenized questions is then transformed into a sequence of GloVe embeddings, providing a dense representation of the questions that the model can process more effectively.

```
def sequence_to_embedding(sequence, embedding_matrix):
    return np.array([embedding_matrix[idx] for idx in sequence])

train_sequences_embedded = [sequence_to_embedding(seq, embedding_matrix) for seq in
    train_sequences_padded]
val_sequences_embedded = [sequence_to_embedding(seq, embedding_matrix) for seq in
    val_sequences_padded]
```

3.9.4 Dataset Creation

In this section, we describe the dataset used for training the VQA model. The dataset includes the features extracted from images, the question embeddings, and the corresponding labels.

3.9.5 Custom Dataset Class

To handle the data efficiently, we implemented a custom dataset class in PyTorch:

```
from torch.utils.data import Dataset, DataLoader

class CustomDataset(Dataset):
    def __init__(self, features, labels, question_embeddings, image_ids):
        self.features = [features[id_] for id_ in image_ids]
        self.labels = labels
        self.question_embeddings = question_embeddings

    def __len__(self):
        return len(self.features) # This should be 78749

    def __getitem__(self, idx):
        feature = self.features[idx]
        label = self.labels[idx]
        question_embedding = self.question_embeddings[idx]
        return feature, label, question_embedding
```

3.9.6 Model Architecture

The model used for the VQA task is based on a combination of LSTM for question embeddings and fully connected layers for image features.

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class VQAModel(nn.Module):
    def __init__(self, embedding_dim, hidden_dim, feature_dim, num_classes):
        super(VQAModel, self).__init__()
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, num_layers=2, batch_first=True)
        self.fc_question = nn.Linear(hidden_dim, feature_dim)
        self.fc_image = nn.Linear(feature_dim, feature_dim)
        self.fc_fusion = nn.Linear(feature_dim, feature_dim)
        self.classifier = nn.Linear(feature_dim, num_classes)

    def forward(self, image_features, question_embeddings):
        # Process question embeddings through LSTM
        _, (h_n, _) = self.lstm(question_embeddings)
        question_features = h_n[-1] # Use the last hidden state
        question_features = self.fc_question(question_features)

        # Process image features
        image_features = self.fc_image(image_features.squeeze())

        # Fusion: Element-wise multiplication
        fused_features = image_features * question_features

        fused_features = self.fc_fusion(fused_features)
        output = self.classifier(fused_features)
        return output
model = VQAModel(embedding_dim=300, hidden_dim=1024, feature_dim=4096, num_classes=num_answers)
```

Description: The `VQAModel` consists of two LSTM layers for processing question embeddings, fully connected layers for processing image features, and a fusion mechanism that combines these features before passing them through a classifier to produce the final output.

3.9.7 Training and Validation

The model was trained and validated using the following process:

```
def train_and_validate(model, train_dataloader, val_dataloader, criterion, optimizer,
    num_epochs, device='cpu'):
    model.to(device)

    # Initialize lists to store losses and accuracies
    train_losses = []
    val_losses = []
    val_accuracies = []

    for epoch in range(num_epochs):
        # Training phase
        model.train()
        running_loss = 0.0
        for image_features, labels, question_embeddings in train_dataloader:
            image_features = image_features.to(device).float()
            question_embeddings = question_embeddings.to(device).float()
            labels = labels.to(device).long()
```

```
            optimizer.zero_grad() # Zero the gradients
            outputs = model(image_features, question_embeddings) # Forward pass
            loss = criterion(outputs, labels) # Compute loss
            loss.backward() # Backward pass
            optimizer.step() # Update model parameters

            running_loss += loss.item()

        avg_train_loss = running_loss / len(train_dataloader)
        train_losses.append(avg_train_loss) # Store the average training loss

        # Validation phase
        model.eval()
        val_running_loss = 0.0
        correct = 0
        total = 0
        all_predictions = []
        all_labels = []
        with torch.no_grad():
            for image_features, labels, question_embeddings in val_dataloader:
                # Move data to the appropriate device and dtype
                image_features = image_features.to(device).float()
                question_embeddings = question_embeddings.to(device).float()
                labels = labels.to(device).long()

                outputs = model(image_features, question_embeddings) # Forward pass
                loss = criterion(outputs, labels) # Compute loss
                val_running_loss += loss.item()

                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

                all_predictions.extend(predicted.cpu().numpy())
                all_labels.extend(labels.cpu().numpy())

        avg_val_loss = val_running_loss / len(val_dataloader)
        val_losses.append(avg_val_loss) # Store the average validation loss
        accuracy = correct / total
        val_accuracies.append(accuracy) # Store the validation accuracy

        print(f'Epoch [{epoch+1}/{num_epochs}], Train Loss: {avg_train_loss:.4f}, Val Loss: {
            avg_val_loss:.4f}, Val Accuracy: {accuracy * 100:.2f}%')

    return train_losses, val_losses, val_accuracies, all_labels, all_predictions

# Train and validate the model
train_losses, val_losses, val_accuracies, all_labels, all_predictions = train_and_validate(
    model, train_dataloader, val_dataloader, criterion, optimizer, num_epochs, device)
```

```
import matplotlib.pyplot as plt

def plot_metrics(train_losses, val_losses, val_accuracies):
    # Plot the training and validation loss
    plt.figure(figsize=(10, 5))
    plt.plot(train_losses, label='Training Loss')
    plt.plot(val_losses, label='Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Training and Validation Loss Over Epochs')
    plt.legend()
    plt.show()

    # Plot the validation accuracy
    plt.figure(figsize=(10, 5))
    plt.plot(val_accuracies, label='Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.title('Validation Accuracy Over Epochs')
    plt.legend()
    plt.show()

# Plot metrics
plot_metrics(train_losses, val_losses, val_accuracies)
```

3.10 Challenges and Limitations

While developing and deploying AIRIS, we encountered several challenges and limitations that impacted its current functionalities. This section discusses these key issues and explores potential areas for improvement in future iterations.

3.10.1 Overview

The challenges and limitations can be broadly categorized into:

- **Resource Constraints:** The limited computational power of the development computer made it difficult to employ complicated algorithms for training the model.
- **Data Limitations:** Using a smaller training dataset caused overfitting, limiting the system's capacity to generalize to new data.
- **User Interaction Challenges:**
  - Ensuring seamless and intuitive user interactions is still an area for improvement.
  - The existing speech implementation has limits in terms of naturalness and emotional range,

3.10.2 Resource Constraints

This section explores the challenges encountered due to limitations in computational resources during development.

- **Integration Challenges:** Combining several libraries and models (e.g., SpeechRecognition, pyttsx3, pywhatkit, VGG16, GloVe embeddings) caused integration challenges due to resource limits.

  The development machine's limited processing capability made it impossible to ensure seamless synchronization across components, which occasionally resulted in delays or errors in voice command processing and response creation.

- **Limited Training:** The chosen VQA model's architecture, while potentially capable, was limited in training due to resource constraints on the development computer. Insufficient processing power limited the amount and diversity of the training dataset, reducing the model's ability to learn successfully. This led to poor performance on image analysis tasks.

  Running the model on a typical laptop with low resources may not have offered the best training environment, resulting in increased CPU and memory utilization during the training phase.

3.10.3 Data Limitations

Limited Training Dataset: The size and diversity of the training data used for the VQA model were limited due to resource restrictions.

This reduced the model's exposure to different image types and question formats, resulting in overfitting. For example, questions about objects that were underrepresented in the training data frequently yielded incorrect or nonsensical answers.

3.10.4 User Interaction Challenges

Limited Command Processing Capabilities: The current implementation of command processing focuses on interpreting basic instructions and questions using predefined structures.

This restricted the system's ability to understand more complicated or nuanced user requests. Misinterpretation of user intent can result in inaccurate or irrelevant responses. For example, variations in phrasing or open-ended inquiries may confuse the system, resulting in failures in recognizing the user's intent.

3.11 Conclusion

AIRIS, a virtual assistant capable of voice-controlled tasks, was designed, developed, and implemented. This chapter detailed its architecture, voice command processing, and functionalities like music playback and answering image-related questions using a VQA model. While the VQA integration is a significant step forward for virtual assistants, challenges like technical issues, limited data, and user interaction difficulties were encountered.

General Conclusion

General Conclusion

Future Work

The developments of AIRIS have created several possibilities for future research and development. This section explores potential enhancements and extensions to AIRIS' capabilities.

Improved understanding Using modern Natural Language Processing (NLP) techniques to better understand user inquiries and requests, including variations in phrasing. Training the system on a larger dataset to improve its capacity to understand natural language and eliminate misinterpretations.

Enhanced Speech Recognition Incorporating noise-cancellation techniques to improve voice recognition accuracy in noisy environments.

Faster Processing Improving system performance by researching lightweight models or using cloud computing for challenging tasks.

Expanded Functionality AIRIS's functionality can be expanded by integrating with various services, like reminders and smart home device control.

VQA Model Refinement Collecting and utilizing a larger and more diverse dataset to train the VQA model, to improve its ability to handle different image types and inquiries.

Multilingual Support Develop the capability to interpret and respond to user inquiries in many languages. This could entail training the system with multilingual datasets and incorporating language translation algorithms.

In conclusion, this dissertation has explored the intersection of AI and assistive technology for visually impaired individuals.

It began by looking at the fundamental notions of AI and how it can help the visually impaired gain more accessibility and freedom. The study next looked at the state of the art in VQA, highlighting advances, limitations, and the possibility of incorporating VQA into assistive technologies. The main contribution of this work, AIRIS, a comprehensive virtual assistant that uses VQA, was provided along with a detailed examination of its design, implementation, and performance evaluation. AIRIS takes a huge step toward closing the accessibility gap by providing natural voice-based interaction and real-time image interpretation capabilities. Looking ahead, the findings highlight the necessity of ongoing research and innovation in AI-driven solutions to empower visually impaired people and promote technological inclusivity.

Bibliography

[1] Sentdex. Deep learning with python and keras - full course. `https://www.youtube.com/watch?v=iwOieOFeUKM`, 2023. Accessed: 2024-07-03.

[2] Udit Agarwal, Shikhar Makhija, Varun Tripathi, and Kunwar Singh. An investigation into neuromorphic ics using memristor-cmos hybrid circuits, 08 2022.

[3] Wikipedia contributors. Activation function. `https://en.wikipedia.org/wiki/Activation_function`, 2024. Accessed: 2024-07-03.

[4] Mohsen Riahi Manesh, Jonathan Kenney, Wen Hu, Vijay Devabhaktuni, and Naima Kaabouch. Detection of gps spoofing attacks on unmanned aerial systems. pages 1–6, 01 2019.

[5] Frederik Kratzert, Daniel Klotz, Claire Brenner, Karsten Schulz, and Mathew Herrnegger. Rainfall–runoff modelling using long short-term memory (lstm) networks. *Hydrology and Earth System Sciences*, 2018.

[6] Farhad Mortezapour Shiri, Thinagaran Perumal, Norwati Mustapha, and Raihani Mohamed. A comprehensive overview and comparative analysis on deep learning models: Cnn, rnn, lstm, gru, 2023.

[7] Wei Fang, Yupeng Chen, and Qiongying Xue. Survey on research of rnn-based spatio-temporal sequence prediction algorithms. *Journal on Big Data*, 3:97–110, 01 2021.

[8] Ahmed Barnawi, Shivani Gaba, Anna Alphy, Abdoh Jabbari, Ishan Budhiraja, Vimal Kumar, and Neeraj Kumar. A systematic analysis of deep learning methods and potential attacks in internet-of-things surfaces. *Neural Computing and Applications*, 35:1–16, 06 2023.

[9] Gaurav Sen. Deep learning personal notes - part 1: Lesson 3 - cnn theory, convolutional filters, max pooling. *Towards Data Science*, 2020. Accessed: 2024-07-03.

[10] Mohammad Anwarul Islam. *Reduced Dataset Neural Network Model for Manuscript Character Recognition*. PhD thesis, 07 2020.

[11] Haoyuan Li and Qing Yin. Beam detection based on machine learning algorithms, 07 2023.

[12] Sruthy Manmadhan and Binsu C. Kovoor. Visual question answering: a state-of-the-art review. *Artificial Intelligence Review*, 53:5705 – 5745, 2020.

[13] Kushal Kafle and Christopher Kanan. Visual question answering: Datasets, algorithms, and future challenges. *Computer Vision and Image Understanding*, 163:3–20, October 2017.

[14] Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the v in vqa matter: Elevating the role of image understanding in visual question answering, 2017.

[15] Vision impairment and blindness.

[16] Bernard Gosselin. *Application de réseaux de neurones artificiels à la reconnaissance automatique de caractères manuscrits*. PhD thesis, Faculté polytechnique de Mons, 1996.

[17] Samantha Hayman. The mcculloch-pitts model. *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No.99CH36339)*, 6:4438–4439 vol.6, 1999.

[18] Siddharth Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *International Journal of Engineering Applied Sciences and Technology*, 2020.

[19] Fionn Murtagh. Multilayer perceptrons for classification and regression. *Neurocomputing*, 2:183–197, 1991.

[20] Juan-Manuel Torres-Moreno. Apprentissage et généralisation par des réseaux de neurones : étude de nouveaux algorithmes constructifs. (machine learning and generalization by neural networks: new constructive algorithms). 1997.

[21] Paul J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1:339–356, 1988.

[22] Richard Durbin and David E. Rumelhart. Product units: A computationally powerful and biologically plausible extension to backpropagation networks. *Neural Computation*, 1(1):133–142, 1989.

[23] Mehmet Ardiclioglu, Ozgur Kisi, and Tefaruk Haktanir. Suspended sediment prediction using two different feed-forward back-propagation algorithms. *Canadian Journal of Civil Engineering*, 34:120–125, 2007.

[24] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.

[25] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization, 2015.

[26] Zhaoqing Pan, Weijie Yu, Xiaokai Yi, Asifullah Khan, Feng Yuan, and Yuhui Zheng. Recent progress on generative adversarial networks (gans): A survey. *IEEE Access*, 7:36322–36333, 2019.

[27] Joydeep Ghosh and A. C. Nag. An overview of radial basis function networks. 2001.

[28] Dubravko Miljkovic. Brief review of self-organizing maps. *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1061–1066, 2017.

[29] Asja Fischer and C. Igel. An introduction to restricted boltzmann machines. In *Iberoamerican Congress on Pattern Recognition*, 2012.

[30] Xuedan Du, Yinghao Cai, Shuo Wang, and Leijie Zhang. Overview of deep learning. *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, pages 159–164, 2016.

[31] John H. Murphy. An overview of convolutional neural network architectures for deep learning. 2016.

[32] Assaad MOAWAD. Neural networks and back-propagation explained in a simple way, 2023.

[33] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation, 2015.

[34] Rikiya Yamashita, Mizuho Nishio, Richard K. G. Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9:611 – 629, 2018.

[35] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[36] Terry Winograd. Understanding natural language. *Cognitive Psychology*, 3(1):1–191, 1972.

[37] Donald Geman, Stuart Geman, Neil Hallonquist, and Laurent Younes. Visual turing test for computer vision systems. *Proceedings of the National Academy of Sciences*, 112(12):3618–3623, 2015.

[38] Aishwarya Agrawal, Jiasen Lu, Stanislaw Antol, Margaret Mitchell, C. Lawrence Zitnick, Dhruv Batra, and Devi Parikh. Vqa: Visual question answering, 2016.

[39] Nelson Ruwa, Qirong Mao, Liangjun Wang, and Ming Dong. Affective visual question answering network. In *2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 170–173, 2018.

[40] Mateusz Malinowski, Marcus Rohrbach, and Mario Fritz. Ask your neurons: A neural-based approach to answering questions about images, 2015.

[41] Geonmo Gu, S. T. Kim, and Yong Man Ro. Adaptive attention fusion network for visual question answering. *2017 IEEE International Conference on Multimedia and Expo (ICME)*, pages 997–1002, 2017.

[42] Ilija Ilievski, Shuicheng Yan, and Jiashi Feng. A focused dynamic attention model for visual question answering, 2016.

[43] Zichao Yang, Xiaodong He, Jianfeng Gao, Li Deng, and Alex Smola. Stacked attention networks for image question answering. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21–29, 2016.

[44] Kushal Kafle and Christopher Kanan. Answer-type prediction for visual question answering. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4976–4984, 2016.

[45] Duy-Kien Nguyen and Takayuki Okatani. Improved fusion of visual and language representations by dense symmetric co-attention for visual question answering, 2018.

[46] Deepak Gupta, Pabitra Lenka, Asif Ekbal, and Pushpak Bhattacharyya. A unified framework for multilingual and code-mixed visual question answering. In Kam-Fai Wong, Kevin Knight, and Hua Wu, editors, *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 900–913, Suzhou, China, December 2020. Association for Computational Linguistics.

[47] Jiasen Lu, Jianwei Yang, Dhruv Batra, and Devi Parikh. Hierarchical question-image co-attention for visual question answering, 2017.

[48] Chao Yang, Mengqi Jiang, Bin Jiang, Weixin Zhou, and Keqin Li. Co-attention network with question type for visual question answering. *IEEE Access*, 7:40771–40781, 01 2019.

[49] Lianli Gao, Liangfu Cao, Xing Xu, Jie Shao, and Jingkuan Song. Question-led object attention for visual question answering. *Neurocomputing*, 391, 04 2019.

[50] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[51] Hao Tan and Mohit Bansal. Lxmert: Learning cross-modality encoder representations from transformers, 2019.

[52] Yen-Chun Chen, Linjie Li, Licheng Yu, Ahmed El Kholy, Faisal Ahmed, Zhe Gan, Yu Cheng, and Jingjing Liu. Uniter: Universal image-text representation learning, 2020.

[53] Xiujun Li, Xi Yin, Chunyuan Li, Pengchuan Zhang, Xiaowei Hu, Lei Zhang, Lijuan Wang, Houdong Hu, Li Dong, Furu Wei, Yejin Choi, and Jianfeng Gao. Oscar: Object-semantics aligned pre-training for vision-language tasks, 2020.

[54] Peng Wang, Qi Wu, Chunhua Shen, Anton Hengel, and Anthony Dick. Fvqa: Fact-based visual question answering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP, 06 2016.

[55] Hyeonwoo Noh, Paul Hongsuck Seo, and Bohyung Han. Image question answering using convolutional neural network with dynamic parameter prediction, 2015.

[56] Kan Chen, Jiang Wang, Liang-Chieh Chen, Haoyuan Gao, Wei Xu, and Ram Nevatia. Abc-cnn: An attention based convolutional neural network for visual question answering, 2016.

[57] Damien Teney and Anton van den Hengel. Visual question answering as a meta learning task, 2017.

[58] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks, 2017.

[59] Bei Liu, Zhicheng Huang, Zhaoyang Zeng, Zheyu Chen, and Jianlong Fu. Learning rich image region representation for visual question answering, 2019.

[60] Akira Fukui, Dong Huk Park, Daylen Yang, Anna Rohrbach, Trevor Darrell, and Marcus Rohrbach. Multimodal compact bilinear pooling for visual question answering and visual grounding, 2016.

[61] Jin-Hwa Kim, Sang-Woo Lee, Dong-Hyun Kwak, Min-Oh Heo, Jeonghee Kim, Jung-Woo Ha, and Byoung-Tak Zhang. Multimodal residual learning for visual qa, 2016.

[62] Junwei Liang, Lu Jiang, Liangliang Cao, Yannis Kalantidis, Li-Jia Li, and Alexander G. Hauptmann. Focal visual-text attention for memex question answering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(8):1893–1908, August 2019.

[63] Hedi Ben-younes, Rémi Cadene, Matthieu Cord, and Nicolas Thome. Mutan: Multimodal tucker fusion for visual question answering, 2017.

[64] Yang Shi, Tommaso Furlanello, Sheng Zha, and Animashree Anandkumar. Question type guided attention in visual question answering, 2018.

[65] Haoyuan Gao, Junhua Mao, Jie Zhou, Zhiheng Huang, Lei Wang, and Wei Xu. Are you talking to a machine? dataset and methods for multilingual image question answering, 2015.

[66] Bolei Zhou, Yuandong Tian, Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Simple baseline for visual question answering, 2015.

[67] Huijuan Xu and Kate Saenko. Ask, attend and answer: Exploring question-guided spatial attention for visual question answering, 2016.

[68] Jin-Hwa Kim, Jaehyun Jun, and Byoung-Tak Zhang. Bilinear attention networks, 2018.

[69] Zhou Yu, Jun Yu, Jianping Fan, and Dacheng Tao. Multi-modal factorized bilinear pooling with co-attention learning for visual question answering, 2017.

[70] Pan Lu, Hongsheng Li, Wei Zhang, Jianyong Wang, and Xiaogang Wang. Co-attending free-form regions and detections with multi-modal multiplicative feature embedding for visual question answering, 2017.

[71] Peng Wang, Qi Wu, Chunhua Shen, Anton van den Hengel, and Anthony Dick. Explicit knowledge-based reasoning for visual question answering, 2015.

[72] Wenfeng Zheng, Lirong Yin, Xiaobing Chen, Zhiyang Ma, Shan Liu, and Bo Yang. Knowledge base graph embedding module design for visual question answering model. *Pattern Recognition*, page 108153, 07 2021.

[73] Mario Dias, Hansie Aloj, Nijo Ninan, and Dipali Koshti. Bert based multiple parallel co-attention model for visual question answering. In *2022 6th International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 1531–1537, 2022.

[74] Prashan Wanigasekara, Kechen Qin, Emre Barut, Fan Yang, Weitong Ruan, and Chengwei Su. Semantic vl-bert: Visual grounding via attribute learning. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2022.

[75] Ze Hu, Jielong Wei, Qingbao Huang, Hanyu Liang, Xingmao Zhang, and Qingguang Liu. Graph convolutional network for visual question answering based on fine-grained question representation. *2020 IEEE Fifth International Conference on Data Science in Cyberspace (DSC)*, pages 218–224, 2020.

[76] Kevin J. Shih, Saurabh Singh, and Derek Hoiem. Where to look: Focus regions for visual question answering, 2016.

[77] Liyang Zhang, Shuaicheng Liu, Donghao Liu, Pengpeng Zeng, Xiangpeng Li, Jingkuan Song, and Lianli Gao. Rich visual knowledge-based augmentation network for visual question answering. *IEEE Transactions on Neural Networks and Learning Systems*, 32:4362–4373, 2020.

[78] Peng Zhang, Yash Goyal, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Yin and yang: Balancing and answering binary visual questions, 2016.

[79] Danna Gurari, Qing Li, Abigale J. Stangl, Anhong Guo, Chi Lin, Kristen Grauman, Jiebo Luo, and Jeffrey P. Bigham. Vizwiz grand challenge: Answering visual questions from blind people, 2018.

[80] Caiming Xiong, Stephen Merity, and Richard Socher. Dynamic memory networks for visual and textual question answering, 2016.

[81] Dipali Koshti, Ashutosh Gupta, and Mukesh Kalla. Bert based hierarchical alternating co-attention visual question answering using bottom-up features. *International Journal of Intelligent Systems and Applications in Engineering*, 10(3s):158–168, Dec. 2022.

[82] Fuji Ren and Yangyang Zhou. Cgmvqa: A new classification and generative model for medical visual question answering. *IEEE Access*, 8:50626–50636, 2020.

[83] Damien Teney, Peter Anderson, Xiaodong He, and Anton van den Hengel. Tips and tricks for visual question answering: Learnings from the 2017 challenge, 2017.

[84] Sylvain Lobry, Diego Marcos, Jesse Murray, and Devis Tuia. Rsvqa: Visual question answering for remote sensing data. *IEEE Transactions on Geoscience and Remote Sensing*, 58(12):8555–8566, December 2020.

[85] Maryam Ziaeefard and Freddy Lécué. Towards knowledge-augmented visual question answering. In *International Conference on Computational Linguistics*, 2020.

[86] Binh X. Nguyen, Tuong Khanh Long Do, Huy Tran, Erman Tjiputra, Quang D. Tran, and A. Nguyen. Coarse-to-fine reasoning for visual question answering. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 4557–4565, 2021.

[87] Kenneth Marino, Xinlei Chen, Devi Parikh, Abhinav Gupta, and Marcus Rohrbach. Krisp: Integrating implicit and symbolic knowledge for open-domain knowledge-based vqa, 2020.

[88] Alberto Bellini. Towards open-ended vqa models using transformers. 2020.

[89] Ahmed Osman and Wojciech Samek. Dual recurrent attention units for visual question answering, 2019.

[90] Sheng Zhang, Min Chen, Jincai Chen, Fuhao Zou, Yuan-Fang Li, and Ping Lu. Multimodal feature-wise co-attention method for visual question answering. *Inf. Fusion*, 73:1–10, 2021.

[91] Moshiur Farazi, Salman Khan, and Nick Barnes. Attention guided semantic relationship parsing for visual question answering, 2020.

[92] Guohao Li, Hang Su, and Wenwu Zhu. Incorporating external knowledge to answer open-domain visual questions with dynamic memory networks, 2017.

[93] Xiaoman Zhang, Chaoyi Wu, Ziheng Zhao, Weixiong Lin, Ya Zhang, Yanfeng Wang, and Weidi Xie. Pmc-vqa: Visual instruction tuning for medical visual question answering, 2023.

[94] Noa Garcia, Chentao Ye, Zihua Liu, Qingtao Hu, Mayu Otani, Chenhui Chu, Yuta Nakashima, and Teruko Mitamura. A dataset and baselines for visual question answering on art, 2020.

[95] Kenneth Marino, Mohammad Rastegari, Ali Farhadi, and Roozbeh Mottaghi. Ok-vqa: A visual question answering benchmark requiring external knowledge, 2019.

[96] Mateusz Malinowski and Mario Fritz. A multi-world approach to question answering about real-world scenes based on uncertain input, 2015.

[97] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. pages 746–760, 10 2012.

[98] Mengye Ren, Ryan Kiros, and Richard Zemel. Exploring models and data for image question answering, 2015.

[99] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[100] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning, 2016.

[101] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Fei-Fei Li. Visual genome: Connecting language and vision using crowdsourced dense image annotations, 2016.

[102] Bart Thomee, David A. Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. Yfcc100m: the new data in multimedia research. *Communications of the ACM*, 59(2):64–73, January 2016.

[103] Yuke Zhu, Oliver Groth, Michael Bernstein, and Li Fei-Fei. Visual7w: Grounded question answering in images, 2016.

[104] Zhibiao Wu and Martha Palmer. Verb semantics and lexical selection. In *Annual Meeting of the Association for Computational Linguistics*, 1994.

[105] Sanket Shah, Anand Mishra, Naganand Yadati, and Partha Pratim Talukdar. Kvqa: Knowledge-aware visual question answering. In *AAAI Conference on Artificial Intelligence*, 2019.

[106] Bin He, Meng Xia, Xinguo Yu, Pengpeng Jian, Hao Meng, and Zhanwen Chen. An educational robot system of visual question answering for preschoolers. *2017 2nd International Conference on Robotics and Automation Engineering (ICRAE)*, pages 441–445, 2017.

[107] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Annual Meeting of the Association for Computational Linguistics*, 2002.

[108] Satanjeev Banerjee and Alon Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *IEEvaluation@ACL*, 2005.