Tampere University

Charitha Raghavaraju

ENHANCING DOMAIN-SPECIFIC AUTOMATED AUDIO CAPTIONING: A STUDY ON ADAPTATION TECHNIQUES AND TRANSFER LEARNING

Master of Science Thesis
Faculty of Information Technology and Communication Sciences
Examiners: Professor Tuomas Virtanen
Postdoctoral Research Fellow Toni Heittola
December 2023

ABSTRACT

Charitha Raghavaraju: Enhancing Domain-Specific Automated Audio Captioning: A Study on Adaptation Techniques and Transfer Learning
Master of Science Thesis
Tampere University
Master's in Signal Processing and Machine Learning
December 2023

Automated audio captioning is a challenging cross-modal task that takes an audio sample as input to analyze it and generate its caption in natural language as output. The existing datasets for audio captioning such as AudioCaps and Clotho encompass a diverse range of domains, with current proposed systems primarily focusing on generic audio captioning. This thesis delves into the adaptation of generic audio captioning systems to domain-specific contexts, simultaneously aiming to enhance generic audio captioning performance. The adaptation of the generic models to specific domains has been explored using two different techniques: complete fine-tuning of neural model layers and layer-wise fine-tuning within transformers. The process involves initial training with a generic captioning setup, followed by adaptation using domain-specific training data. In generic captioning, the process for training starts with training the model on the AudioCaps dataset followed by fine-tuning it using the Clotho dataset. This is accomplished through the utilization of a transformer-based architecture, which integrates a patchout fast spectrogram transformer (PaSST) for audio embeddings and a BART transformer. Word embeddings are generated using a byte-pair encoding (BPE) tokenizer tailored to the training datasets' unique words, aligning the vocabulary with the generic captioning task. Experimental adaptation mainly focuses on audio clips related to animals and vehicles. The results demonstrate notable improvements in the performance of the generic and domain adaptation systems. Generic captioning has demonstrated an improvement in SPIDEr scores, increasing from 0.291 during fine-tuning to 0.301 with layer-wise fine-tuning. Specifically, we observed a notable increase in SPIDEr scores, from 0.315 to 0.323 for animal-related audio clips and from 0.298 to 0.308 for vehicle-related audio clips.

Keywords: Automated audio captioning, transformer models, Clotho dataset, domain adaptation.

PREFACE

I want to express my heartfelt appreciation to Professor Tuomas Virtanen for granting me the invaluable chance to work as a research assistant within the Audio Research Group (ARG) at Tampere University. I would like to express my deep appreciation to Toni Heittola for his role as my mentor during my tenure as a research assistant. I am truly grateful for having such an approachable and patient mentor to answer all my queries. My heartfelt thanks to both of them for supervising and guiding me through the process of this thesis, from the research phase to the writing stage.

I would like to extend my acknowledgment and gratitude to my colleagues Dr. Konstantinos Drossos and Samuel Lipping for their substantial contributions during the initial phases of this project, which laid the groundwork for this research. I wish to thank my colleagues Irene Martin Morato and Manu Harju for helping me during the experiments and all the members of ARG for supporting me in several situations.

I express my gratitude to Tampere University and CSC-IT Center for Science, Finland, for furnishing the essential infrastructure and computational resources crucial for this research.

I wish to express my thanks to my dear friends, Sai Poojith, Nitin Chandra, and other close companions, for their unwavering emotional support during challenging times and for encouraging my personal growth throughout this journey. I am thankful to my close friend and colleague Parthasaarathy Sudarsanam for his technical and personal support, always ready to listen and assist me. I would also like to express my appreciation to my therapists for their invaluable role in helping me on my path to well-being.

Last but not least, I want to convey my heartfelt gratitude to my parents, Swarna Latha and R.P. Rama Raju, for their unwavering support and love, which enabled me to pursue my aspirations without any hesitation. I also wish to express my gratitude to my cousins for their contribution to making this journey possible. I would like to acknowledge my faithful companion, Suziee, for patiently awaiting my return.

Tampere, 7th December 2023

Charitha Raghavaraju

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AAC automated audio captioning

AI artificial intelligence

AMT amazon mechanical turk

ANN artificial neural networks

BART bidirectional and auto-regressive transformers

BPE byte-pair encoding

CBOW continuous bag Of words

CNN convolutional neural network

DNN deep neural network

FFN feed-forward neural network

GRU gated recurrent unit

LSTM long short-term memory

MLP multi-layer perception

MSE mean square error

NLP natural language processing

OOV out-of-vocabulary

PaSST patchout fast spectrogram transformer

ReLU rectified linear unit

RNN recurrent neural networks

STFT short-time Fourier transform

XE cross-entropy

1. INTRODUCTION

Captioning involves the generation of text-based descriptions through the analysis of input from any modality in a specific format. This input may take the form of audio (audio captioning, e.g., [1]), image (image captioning, e.g., [2]), or video (video captioning, e.g., [3]) content. Although captioning in image and video modalities has progressed significantly, the domain of audio captioning remains relatively less explored. In this thesis, we're diving deep into audio captioning, also referred to as automated audio captioning (AAC). It merges the concepts of natural language processing (NLP) and audio processing, which is a multi-modal task that blends the understanding of sound and language.

Automated audio captioning can be utilized in various scenarios, such as enhancing accessibility for individuals with hearing abilities, contributing to security surveillance in innovative city environments, facilitating efficient audio organization through indexing, etc. A basic AAC system is shown in Figure 1.1. Generating a caption in the natural language requires an advanced understanding of the input audio signal. For instance, it requires the ability to comprehend distinct sound events, discern their characteristics, and accurately capture the spatiotemporal associations among multiple sound sources. An example of a produced caption for an audio sample that has been given is "A woman sings while birds chirp and baby coos".

Figure 1.1. *Fundamental visualization of an automated audio captioning system.*

The research on AAC is mainly driven by deep neural networks (DNNs) and recent advancements in machine learning such as Transformers [4]. A sequence of audio samples is passed through the network that outputs the word embeddings in a sequence as a caption. In general, audio is processed to extract log-mel spectrogram features, and words are tokenized into word embeddings. The captioning system relies on the Clotho dataset [5]; however, its limited size may not comprehensively capture all real-life scenarios. To address this constraint, the largest available AAC dataset, AudioCaps [6] is

utilized. These datasets contain audio and captions related to various sets of domains. The captions in the current open datasets commonly cover a range of domains, such as weather conditions, geographical locations, animal vocalizations, and human behavior.

Automated audio captioning systems that have been proposed so far are based on generic captioning. It means that the systems are trained on generic datasets, that contain a diverse range of domains. In the context of applications such as traffic monitoring and detection of suspicious activities in parking lots, the use of a generic AAC model can create undue complexity. The incorporation of numerous domains within the model, when addressing the tasks specific to the domain of vehicles or traffic, can lead to a state of confusion and reduced accuracy. Therefore, it is essential to adapt the AAC system to match the specific domain it's being used in.

The motivation for domain adaptation of generic AAC lies in the need for improved accuracy and reduced complexity when focusing on domain-specific tasks. For domain adaptation, we choose a model that bases itself on a sequence-to-sequence architecture, formed by a Transformer encoder and Transformer decoder with additional improvements that include a pre-trained model for extracting audio embeddings. Domain adaptation is achieved through two techniques: layer-wise fine-tuning and transfer learning. Both techniques use a generic dataset to train the model initially and subsequently domain-specific dataset is utilized to fine-tune the model. In layer-wise fine-tuning, experiments are done by freezing different layers and fine-tuning the weights specific to a domain.

In this thesis, our main objective is to address the domain adaptation of generic audio captioning systems and overcome the data scarcity issue of Clotho by experimenting with different techniques of transfer learning. The system uses a transformer-based architecture that employs a patchout fast spectrogram transformer (PaSST) for audio embeddings and a BART transformer. For word embeddings, we utilize a byte-pair encoding (BPE) tokenizer trained on unique words from the training datasets to align the vocabulary with the generic captioning task. In the experiments, we adapt the generic system to work with audio clips such as animal or vehicle-related sounds. In addition to that, we also experiment with various transfer learning and fine-tuning techniques along with three different tokenizers.

1.1 Structure of the thesis

The remainder of this thesis is organized as follows: Chapter 2 presents a concise overview of the essential theoretical concepts employed in this study. Chapter 3 is dedicated to elucidating the proposed system and its comprehensive implementation. Chapter 4 offers a detailed account of the experiments, their evaluation, and the results derived from the preceding chapter. Lastly, Chapter 5 outlines the thesis's conclusion and potential avenues for future research.

2. THEORETICAL BACKGROUND

This chapter gives a comprehensive review of fundamental concepts in machine learning, audio processing, and natural language processing. Section 2.1 discusses the concepts of audio signal processing used in this thesis briefly. Section 2.2 delves into machine learning principles, while Sections 2.4 and 2.3 introduce the concepts associated with sequence-to-sequence models for captioning and natural language processing, respectively. Additionally, Sections 2.5 and 2.6 comprehensively explore audio captioning and domain adaptation, offering valuable insights into these specific areas.

2.1 Audio processing

In this thesis, pre-processing involved the application of audio signal processing algorithms especially, extracting audio features using mel-spectrogram from the audio input. The concepts required to understand the mel-spectrogram such as short-time Fourier transform (STFT) and mel scale are briefly presented in Sections 2.1.1 and 2.1.2, respectively. Section 2.1.3 provides a detailed discussion on the mel-spectrogram.

2.1.1 Short-Time Fourier Transform

The short-time Fourier transform (STFT) is a mathematical approach for analyzing a signal's time-varying frequency content. It depicts how a signal's frequency components vary over small, overlapping time intervals. The main idea of STFT is to decompose a signal into its individual frequency components as they change over time. This is done by splitting the signal into small time frames, performing the Fourier transform on each frame, and then stacking the results to generate a time-frequency representation.

The computation of the STFT varies for continuous-time and discrete signals. In practical scenarios, we deal with discrete signals, and therefore, we utilize the discrete form of STFT. For a given discrete signal denoted as $x[n]$, the computation of discrete STFT is as follows:

$$X[m,\omega] = \sum_{n=-\infty}^{\infty} x[n]\omega[n-m]e^{-j\omega n} \tag{2.1}$$

where $X[m, \omega]$ represents the STFT of the signal at the frame m and frequency ω, $x[n]$ is the discrete signal, $\omega[n - m]$ is the discrete window function, ω is the angular frequency. Some of the common window functions include the Hamming, Hanning, and Gaussian windows. The balance that exists between frequency and time resolution is determined by this window function. Figure 2.1 clearly shows how STFT can effectively capture both the frequency and the corresponding time in the signal.

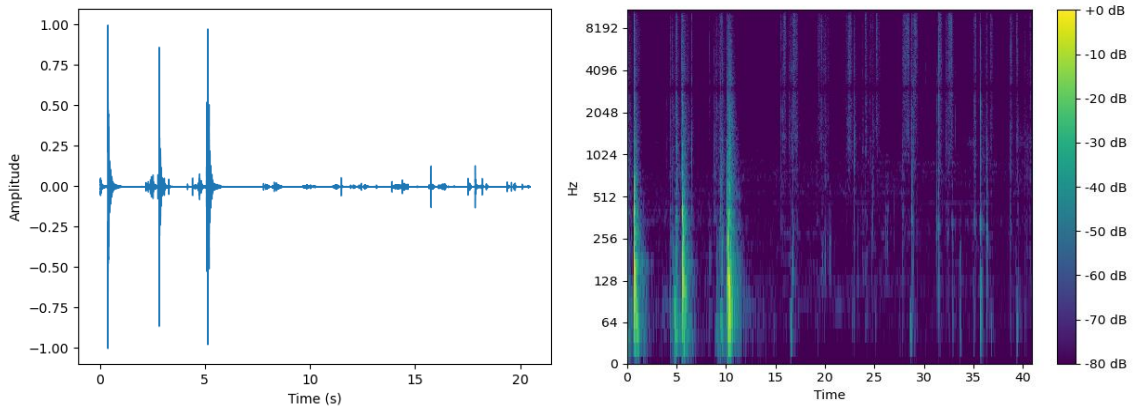

Figure 2.1. ***(Left)*** *Time-domain signal of an audio sample from the Clotho dataset.* ***(Right)*** *STFT of the corresponding audio sample.*

2.1.2 Mel Scale

The perception of sound frequencies by the human auditory system is not linear. In reality, the ability to discern between different frequencies gets less sensitive at higher frequencies and grows more sensitive at lower frequencies. For example, we can readily distinguish a variation in pitch between 1000 and 1500 Hz, but distinguishing the difference between 5000 and 5500 Hz becomes quite challenging, despite the fact that the frequency gap is the same. This phenomenon demonstrates that the perception of pitch follows a linear pattern at lower frequencies and a logarithmic pattern at higher frequencies.

To overcome this apparent non-linearity, the "mel frequency scale" was developed. This scale has been designed to make equal pitch intervals appear to be equally far from one another. At lower frequencies, the Mel frequency scale produces densely packed bands; at higher frequencies, the bands become more widely dispersed, as seen in Figure 2.2. This method provides a more precise representation of a signal, which is consistent with how humans hear sound.

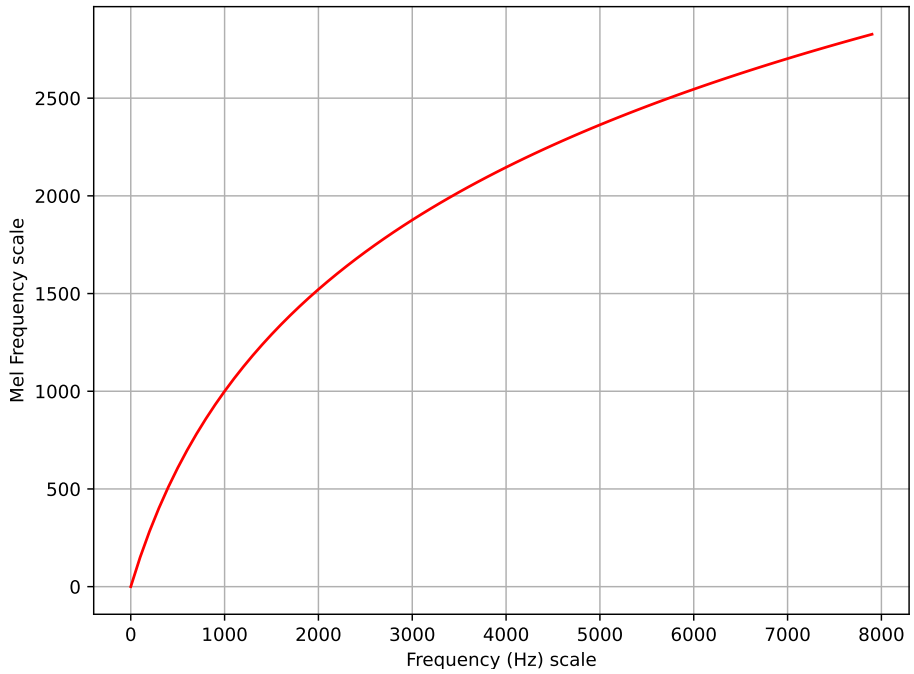

Figure 2.2. *The x-axis represents frequency in the Hz scale, while the y-axis represents the associated mel frequency scale.*

The following formula is used to convert conventional Hertz (Hz) frequency to the mel frequency scale:

$$f_{mel} = 1127\, log_e(1 + f/700), \tag{2.2}$$

Where the variable f represents the frequency in Hertz (Hz), and f_{mel} corresponds to the mel frequency.

2.1.3 Mel Spectrogram

The mel spectrogram of an audio signal is generated by calculating the power spectrogram, followed by a conversion of the frequencies within this power spectrogram to the mel frequency scale. Initially, a mel filter bank is constructed by partitioning the entire frequency range into a fixed number of mel frequency bins, usually denoted by "nmels" (typically set to 64 or 128). These frequency bins are evenly spaced along the mel scale. The power spectrogram is generated by squaring the absolute value of the STFT. This power spectrogram is then passed through the mel filter bank, where the output from each mel filter is summed and combined to produce the mel spectrogram of the audio input. The mel spectrogram of an audio signal is shown in Figure 2.3.

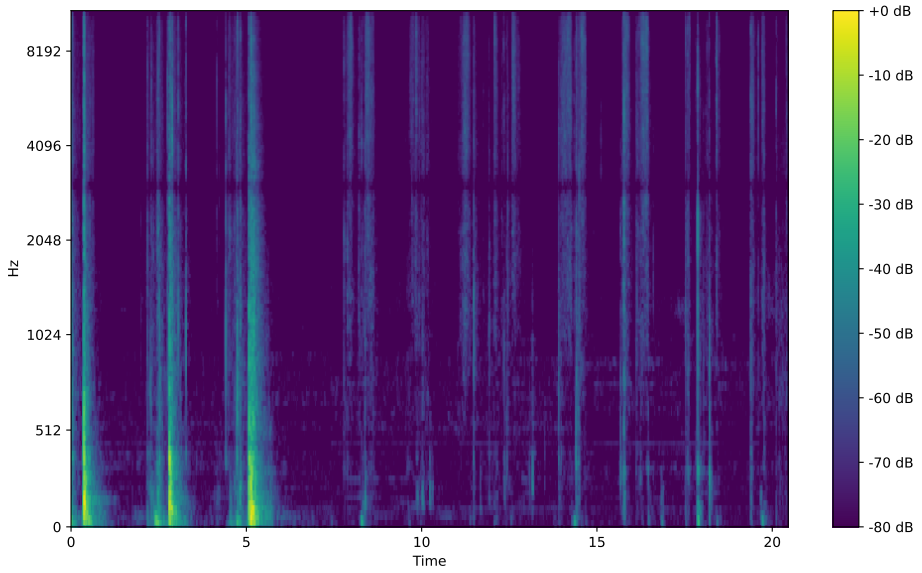

Figure 2.3. *Mel spectrogram of a sample audio signal containing walking steps on pavement.*

2.2 Machine Learning

Machine learning, a subset of artificial intelligence (AI), is dedicated to developing models and algorithms that empower computers to understand and make decisions or predictions based on data. Unlike traditional programming where tasks are explicitly defined, machine learning systems leverage data to identify patterns, generalize from examples, and enhance their performance through experience. Machine learning has a broad range of applications, from speech and image recognition to autonomous vehicles, natural language processing, and recommendation systems, and it plays a vital role in solving complex, data-driven problems.

Machine learning incorporates a range of techniques, including reinforcement learning (learning through interactions with an environment), unsupervised learning (identifying patterns in unlabeled data), and supervised learning (learning from labeled data) [7]. They are utilized in distinct use cases and applications. Supervised learning is used in applications that already have a target or labeled data i.e., automated audio captioning [1], image classification [8], sentiment analysis [9] etc. Clustering [10], anomaly detection [11], image compression [12] falls under unsupervised learning. Reinforcement learning is used for example, game playing [13], autonomous vehicles [14] etc.

In machine learning, multiple techniques can be used to train the systems that are customized to the unique requirements and features of the dataset. These techniques include unsupervised learning, supervised learning, self-supervised learning, and semi-supervised learning. In this work, we utilize supervised learning which is explained in 2.2.1. In the subsequent sections, we focus on providing a comprehensive introduction to fundamental machine learning algorithms that are the basis for this thesis.

2.2.1 Supervised Learning

Supervised learning is a machine learning approach through which the algorithm learns to interpret labeled input data into output data. To train a mapping or function that can produce precise predictions or classifications when given either fresh or unknown data is the aim of supervised learning.

In a supervised learning approach, the algorithm is provided with a training dataset consisting of input-output pairs, that can be denoted as (x_1, y_1), (x_2, y_2), (x_3, y_3), ..., (x_n, y_n). Here, x signifies the data given as input, y is the corresponding ground truth or target, and the total number of input data points is represented as n. To give an instance, in a dataset for forecasting housing prices, the input data would include features such as square footage, number of bedrooms, and so on, while the intended outputs would be the matching sale prices. Supervised learning is an effective and commonly used machine learning approach, but it necessitates a large quantity of labeled training data and may not be suitable for tasks where obtaining labeled data is challenging or costly.

2.2.2 Artificial Neural Networks

Artificial neural networks (ANNs), sometimes known as neural networks for short, were developed by the human brain's initial models of processing sensory data [15]. Each neuron is considered a computational unit that performs a simple task based on the given input. ANNs can be applied for the classification and regression of continuous target variables. The fundamental building block of any ANN contains a single neuron, where every input x = [x_0, x_1, x_2, .., x_{n-1}] to the neuron has an associated weight w = [w_0, w_1, w_2, .., w_{n-1}]^T that alters the strength of the input. The output y of a neuron is formulated as

$$y = f((\sum_{i=1}^{n-1} w_i x_i) + b) = f(wx + b) \tag{2.3}$$

where b is a bias and $f()$ is an activation function. Bias is a learnable parameter that allows neurons to introduce a shift or offset in their activation. It enables the model to find complex patterns and make accurate predictions. An activation function is a mathematical function that determines the output of a neuron based on the weighted sum of inputs. The activation function is often a logistic function that converts the output to a number between 0 and 1. It enables the neuron to learn non-linear relations between the input features. Common activation functions include rectified linear unit (ReLU), sigmoid, hyperbolic tangent (tanh), and softmax [16] functions that are explained in Section 2.2.4.

Figure 2.4 provides a visual depiction of an artificial neuron, aligning with the mathematical expression presented in Equation 2.3.

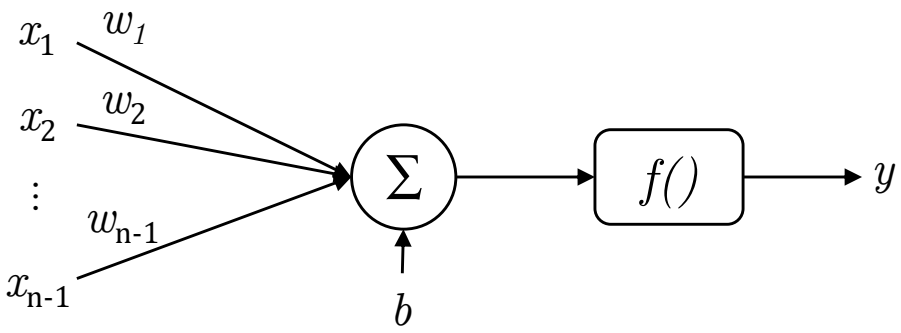

Figure 2.4. *An image representing an artificial neuron having n inputs.*

An artificial neural network (ANN) is built by layering a large number of artificial neurons. The preceding layer's outputs serve as inputs for each neuron in the subsequent layer in this architecture. The parameters of an artificial neural network (ANN), which are commonly denoted as Θ, are composed of the biases and weights corresponding to the neurons inside the network. Three layers make up a neural network in general: an input layer, either one or several hidden layers, and the output layer. The input layer of each neuron points to input features which are passed to hidden layers where the non-linearity is introduced to learn complex relationships. The configuration of the output layer in an artificial neural network, including the number of neurons and the choice of activation function, is determined by the nature of the task at hand. In binary classification, for example, it is typical to have only one neuron in the output stage with an activation function called a sigmoid. A general representation of ANN is visualized in Figure 2.5.

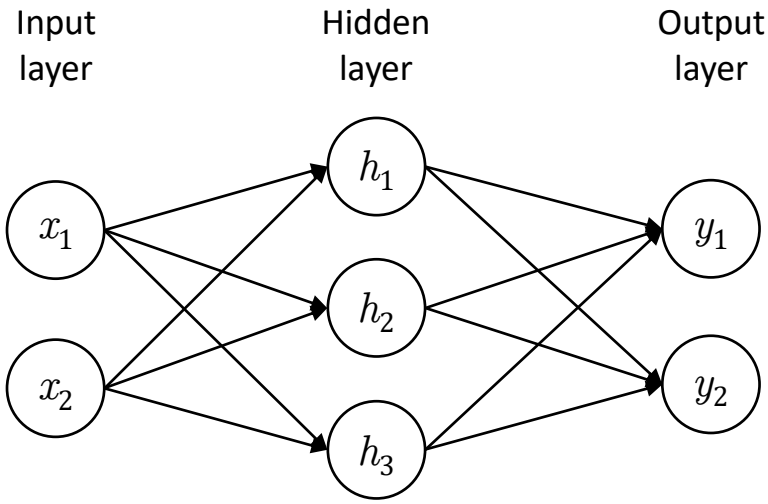

Figure 2.5. *A graphical illustration of the artificial neural network having two data points in the input layer, three nodes in the hidden layer of the network, and two nodes in the final layer. Each node in the hidden and output layer has an associated activation function $f()$ and biases b and each edge has an associated weight w.*

A neural network in which knowledge flows in a unidirectional manner, moving to the output layer from the input layer without any feedback loops, is referred to as a feed-forward neural network (FFN). Extending the concept of feed-forward networks, deep neural networks (DNNs) are characterized by their multiple hidden layers. An instance of a deep neural network is a convolutional neural network (CNN) employed in image classification. Such networks incorporate alternative designs that modify neural connections to suit specific machine-learning tasks. This is referred to as introducing an inductive bias into the model [17].

2.2.3 Training Neural Networks

To train a network, ANNs employ a learning process [18]. The learning is classified into two important categories such as unsupervised and supervised learning. This section explains how to train a neural network using supervised learning.

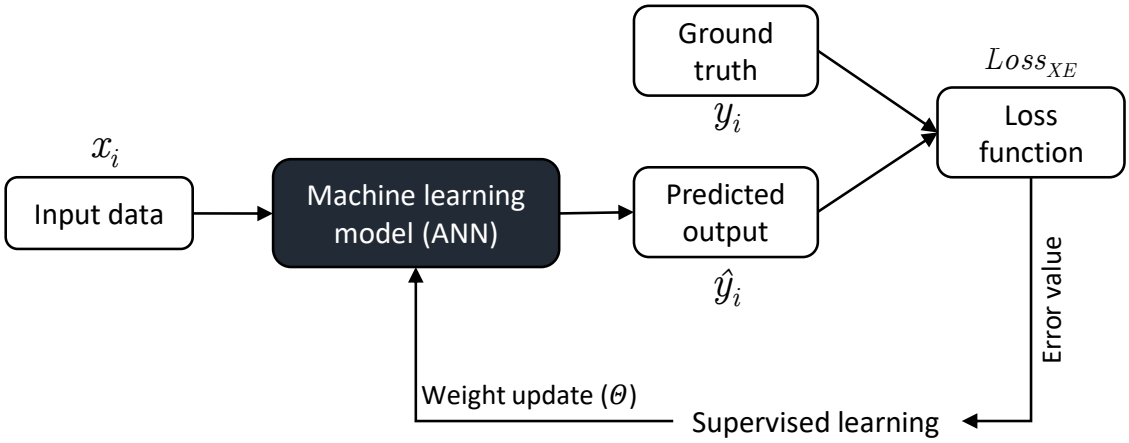

Figure 2.6. *A visualization of supervised learning and the stages involved in training a machine learning model (ANN).*

Figure 2.6 illustrates the various stages involved in supervised learning. In the context of training a neural network using a supervised learning algorithm, the model's parameters Θ are initially assigned random weights. The input x_i is fed into the model and processed through all the network's layers, ultimately generating an output prediction denoted as $\hat{y}_i$. This initial training phase is referred to as forward propagation. To assess the model's performance, a loss function L is employed. It measures the dissimilarity (error) between the ground truth label y_i and the predicted output $\hat{y}_i$. Examples of commonly used loss functions include cross-entropy (XE) loss, that can be often used for classification tasks and is defined as follows:

$$Loss_{XE} = -\sum_{i=1}^{n-1} y_i log(\hat{y}_i) \tag{2.4}$$

and mean square error (MSE) for regression is expressed as

$$Loss_{MSE} = \frac{1}{n}\sum_{i=1}^{n-1}(y_i - \hat{y}_i)^2 \tag{2.5}$$

where y_i is the ground truth, and $\hat{y}_i$ is the predicted output.

The primary objective during the training process is to find the appropriate parameter values for the model so that the difference between the predicted output and the actual ground truth is minimized. To achieve this, we calculate an error value using a loss function, which guides in optimizing the neural network's parameter values Θ through gradient descent techniques. This adjustment process is commonly referred to as back-propagation [18]. The weights for any input x_i in the parameter set Θ are updated as

$$\Theta' = \Theta - \lambda \nabla_{\Theta} L \tag{2.6}$$

where λ is the learning rate.

These processes are repeated across multiple epochs, encompassing all the data points until the model attains its minimal loss. An epoch in neural network training signifies a complete iteration through the entire training dataset, during which the model's parameters are adjusted. The model with the lowest loss is saved and subsequently utilized for evaluation and real-time applications.

One of the key challenges in training models is the issue of over-fitting, where the model becomes too specialized in learning from the training data and struggles to generalize to real-world data. To address this problem, regularization techniques such as dropout and early stopping are employed. Dropout involves randomly deactivating a subset of neurons during each training step, which mitigates over-fitting by preventing inter-dependencies among neurons [19]. On the other hand, early stopping involves terminating the training process when there is no further improvement in validation metrics.

2.2.4 Activation Functions

Neural networks consist of layers of neurons that process the input to extract features and produce an output. These layers in the network such as convolutional networks behave as a linear model, limiting its capacity to represent intricate patterns in the data. To enable the neural network to gain knowledge of complicated and non-linear patterns, activation functions are introduced to scale the outputs of each layer non-linearly and pass them on to the next layer.

Activation functions used in this thesis for developing the model have been discussed in detail in this section. Specifically, the softmax activation function along with the rectified linear unit (ReLU) are explained.

Rectified Linear Unit

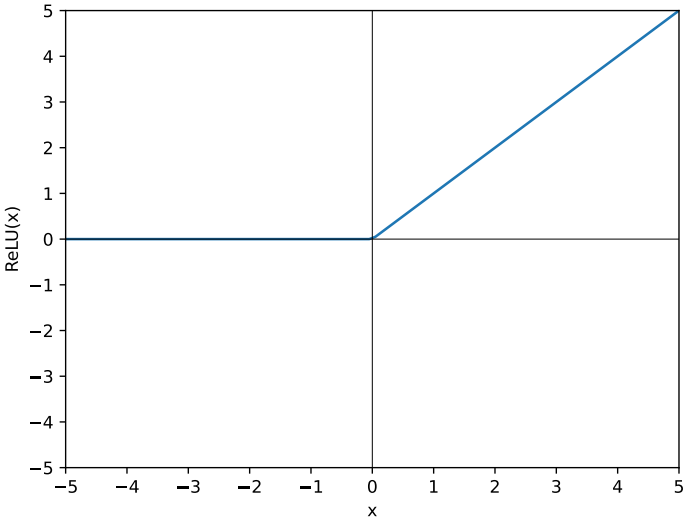

Figure 2.7. *Activation function ReLU.*

Rectified linear unit otherwise known as ReLU is one of the popular activation functions in neural networks. It is mathematically represented as

$$f(x) = \begin{cases} x & x > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{2.7}$$

As shown in Figure 2.7, the output of the activation is the same as the input if it is positive, and the output is 0 in other cases. ReLU does not impose an upper bound on its output values for positive inputs, thereby avoiding the vanishing gradient problem commonly associated with other activation functions.

Softmax Activation Function

The softmax activation function is commonly employed as the output layer in neural networks for tasks involving multi-class classification. For an input vector $[z_1, z_2, .., z_N]$, containing N real numbers, it transforms the vector into a probability distribution by exponentiating its elements and normalizing them, ensuring that the sum of the resulting probabilities equals 1. The softmax function is mathematically represented as

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \qquad z_i \in \mathbb{R}. \tag{2.8}$$

It scales the values generated by the final classification layer, creating a probability distribution for the predicted class. The index corresponding to the highest probability in the output of the softmax function is selected as the predicted class.

2.2.5 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a class of deep learning architectures primarily used for tasks involving image analysis and recognition. CNNs are first introduced in [20] for the purpose of hand-written zip code recognition. The term "convolutional neural network" signifies that the network utilizes a mathematical operation known as convolution. Mathematically, convolution between an input x and a learnable filter k is defined as

$$s(t) = (x * k)(t) = \sum_{i=-\infty}^{\infty} x(a)k(t-a) \tag{2.9}$$

where t is the time index which takes only integers. In machine-learning applications, the input to the network is multi-dimensional, and the kernel usually is a multi-dimensional array. Extending the above equation to the multi-dimensional case, for instance, a two-dimensional image X with a kernel K as

$$S(i,j) = (X * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} X(m,n)K(i-m, j-n) \tag{2.10}$$

Many machine learning models, including convolutional neural networks (CNNs), often implement a mathematical operation that is essentially equivalent to convolution but is referred to as cross-correlation. It is represented as

$$S(i,j) = (X * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} X(i+m, j+n)K(m,n) \tag{2.11}$$

where the output at position (i, j) in the feature map is represented by the symbol $S(i, j)$. The visualization of the convolution or cross-correlation mechanism is depicted in Figure 2.8.

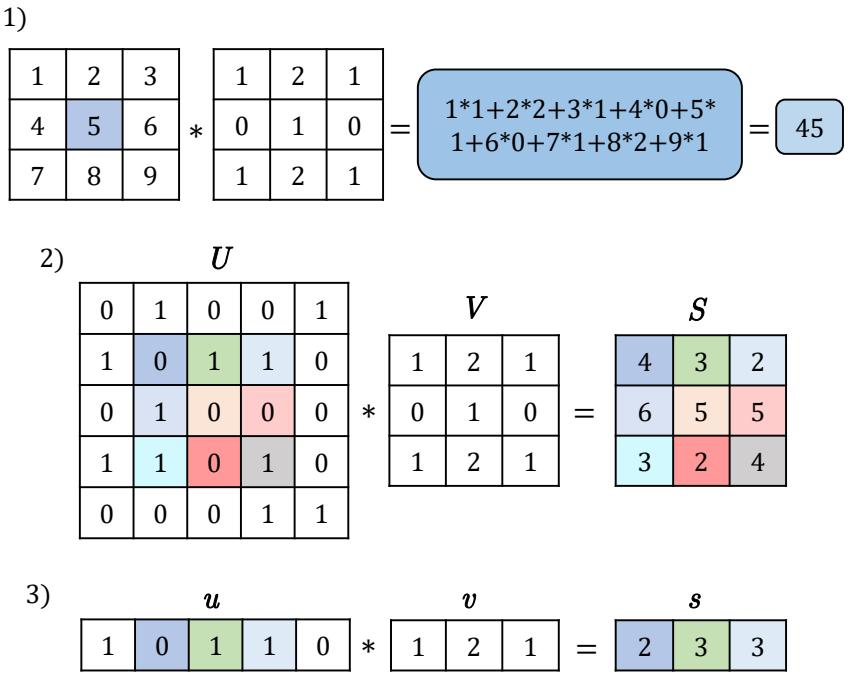

Figure 2.8. *Illustration of the cross-correlation involving a sliding window operation. Image 1 showcases a two-dimensional case with singular multiplication and summation operation using a 3x3 kernel. The process of cross-correlation between an input that is two-dimensional and a two-dimensional kernel is depicted in Image 2. A similar one-dimensional instance is shown in Image 3. The colored squares in each scenario represent the places where the kernel is centered to calculate the same color result.*

CNNs utilize these convolutional layers that contain multiple filters to capture different features from the input image, resulting in multiple feature maps. Following each convolutional layer are activation functions (ReLU) and pooling layers. Through downsampling, pooling layers decrease the spatial dimensionality of feature maps. A common pooling method called "max-pooling" [21] extracts the largest value from a local neighborhood. The final layer typically has single or multiple fully connected layers that resemble ANNs for high-level reasoning. Figure 2.9 shows the network configuration of a neural network based on convolution (CNN).

The foundational CNN is primarily employed in computer vision [20] and relies on two-dimensional convolution as its core operation. However, they are also used for one-dimensional sequential data in [22], [23]. Using pre-trained models, CNNs may be optimized for particular tasks or used as feature extractors in transfer learning.

Convolutional neural networks have been instrumental in driving progress in computer vision systems [24], [25] and have seen widespread applications in the field of audio processing [26].

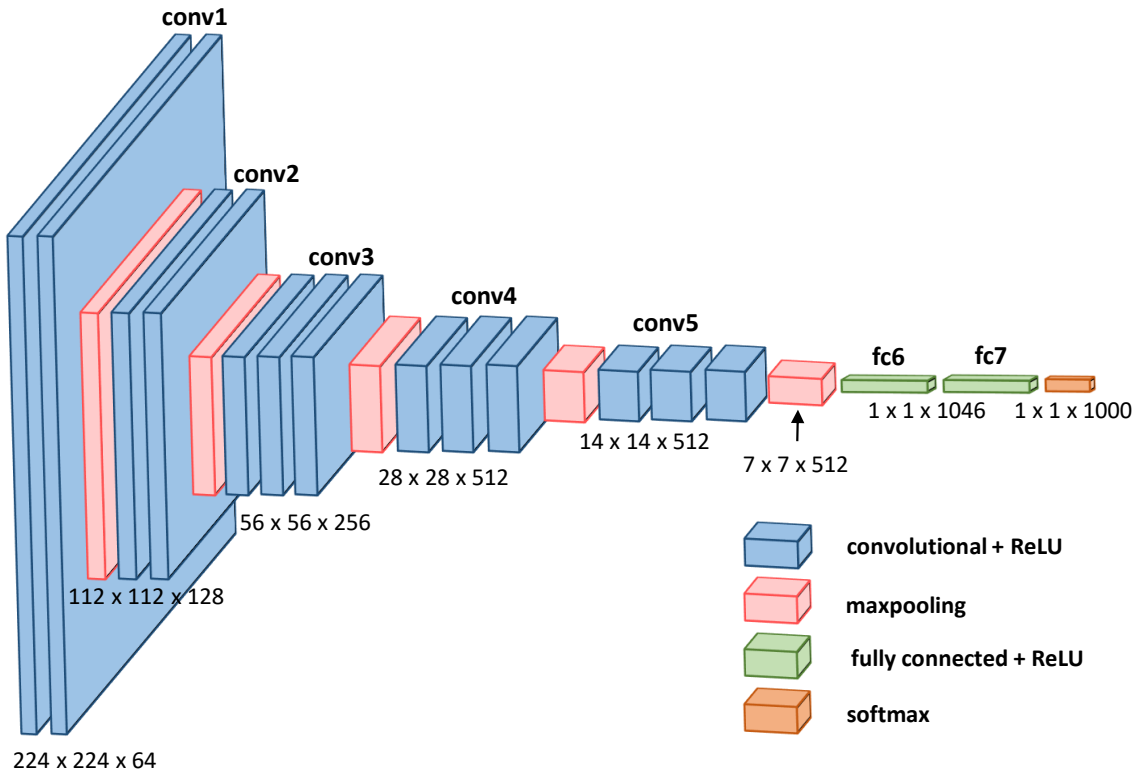

Figure 2.9. *This diagram illustrates the typical architecture of a Convolutional Neural Network, which commonly comprises a sequence of convolutional blocks, followed by several fully connected layers.*

2.2.6 Transformers

Transformers are a type of deep neural network model architecture that has a profound impact on natural language processing (NLP) and machine learning tasks after its first introduction in the paper "Attention is all you need" [4]. It is recognized as an essential component for many state-of-the-art NLP models and has been effectively employed in several computer vision tasks such as image classification [27], and audio processing tasks like sound event detection [28] and audio captioning [29].

Unlike other sequence-to-sequence models such as recurrent neural networks (RNNs), Transformers entirely rely on a novel architecture called attention mechanism. An attention function translates a set of pairs of keys and values and a query into an output, all of which are vectors: keys, values, and the output [4]. The simple attention layer mechanism, the scaled-dot product has been depicted in Figure 2.10. This mechanism of attention layer is mathematically formulated as

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax(\frac{\mathbf{Q}\mathbf{K}^{\mathbf{T}}}{\sqrt{d_k}})\mathbf{V}, \tag{2.12}$$

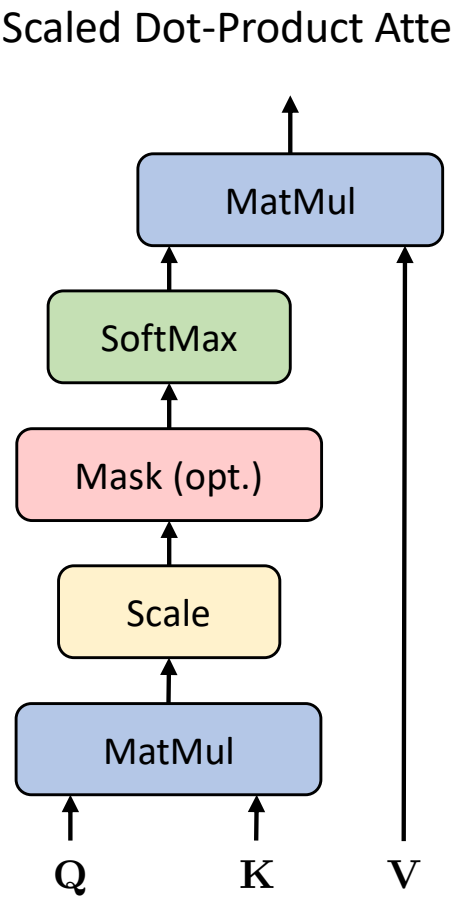

Figure 2.10. *Scaled dot product attention mechanism of the base Transformer. The optional masking in the attention graph prevents the decoder's self-attention from focusing on future time steps.*

where the input comprises of the feature matrices containing n_q queries $\mathbf{Q}$ and values $\mathbf{V}$ with dimensions, $\mathbf{Q} \in \mathbb{R}^{n_q \times d_k}$, $\mathbf{V} \in \mathbb{R}^{n_q \times d_v}$, and n_k keys $\mathbf{K}$ with dimensions $\mathbf{K} \in \mathbb{R}^{n_k \times d_k}$. To calculate the similarity between the key and query, the query matrix and the key matrix are multiplied. A higher degree of resemblance results in a higher value, and the other way around. Subsequently, the result is processed using a softmax operator, as elaborated in Section 2.2.4, in order to normalize the attention scores in the range of 0 to 1. The softmax operation is specifically applied over the dimension corresponding to the sequence length or query dimension, resulting in the generation of an attention map denoted as W. These weights determine how much focus each position should have on the current position during computation. The value vector and the attention weights are multiplied to calculate the attention feature.

There are two types of attention mechanisms: self-attention and multi-head attention. In self-attention and multi-head attention, the values, keys, and queries are all component parts of the same sequence, but there is a distinction in how the sequences are used. The mechanism is applied in parallel with different sets of parameters in multi-head attention, enabling more complex relationships to be captured. However, in the sequence-

to-sequence framework described in Section 2.4, encompassing both an encoder and a decoder, cross-attention is implemented. During this process, queries are generated from the decoder input, while values and keys are extracted from the encoder outputs at each time step [4].

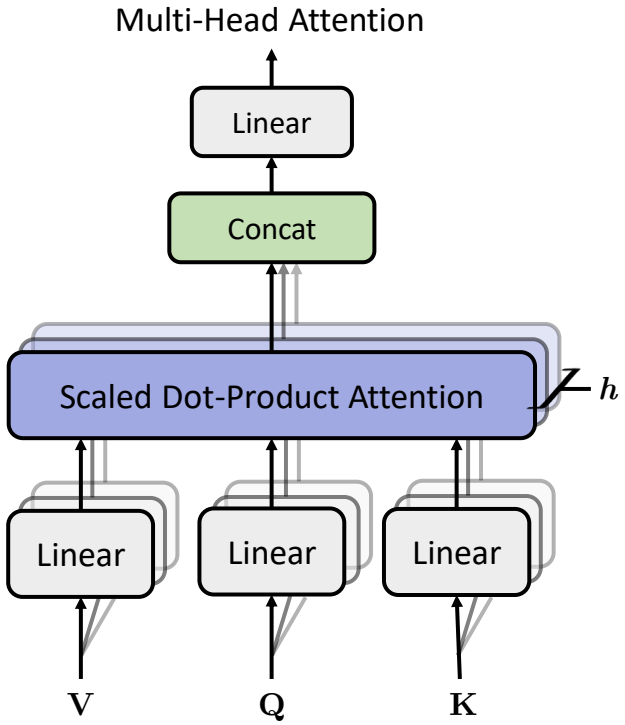

Figure 2.11. *Multi-head attention. It involves multiple attention layers operating simultaneously in parallel.*

In Transformers, the multi-head attention mechanism is employed. In this mechanism, when using h attention heads, the keys, queries, and values are processed by h separate attention mechanisms, each operating independently. The final result is usually obtained by concatenating and projecting the outputs from all attention heads again using a linear layer as depicted in Figure 2.11. It is represented mathematically as

$$\begin{aligned} MultiHead(\mathbf{Q},\mathbf{K},\mathbf{V}) &= Concat(head_1,...,head_h)\mathbf{W_O}, \\ head_i &= Attention(\mathbf{QW_i^Q},\mathbf{KW_i^K},\mathbf{VW_i^V}) \end{aligned} \tag{2.13}$$

where $\mathbf{W_i^Q} \in \mathbb{R}^{d_{model}\times d_k}$, $\mathbf{W_i^K} \in \mathbb{R}^{d_{model}\times d_k}$, $\mathbf{W_i^V} \in \mathbb{R}^{d_{model}\times d_v}$, and $\mathbf{W_O} \in \mathbb{R}^{hd_v\times d_{model}}$ are the weight vectors of the projections, and d_{model} refers to the dimension of the original queries, keys, and values. The dimensionality of each head is reduced to $d_k = d_v = \frac{d_{model}}{h}$ to match the computational cost of full-dimensional single-head attention [4]. Furthermore, all the attention layers can be run parallel as they have different parameters.

Transformers are typically utilized in an encoder-decoder architecture, characterized by key hyperparameters: the embedding size denoted as d_{model}, the number of encoder and decoder blocks represented as (N_e, N_d), and the dimension of inner layers referred to as

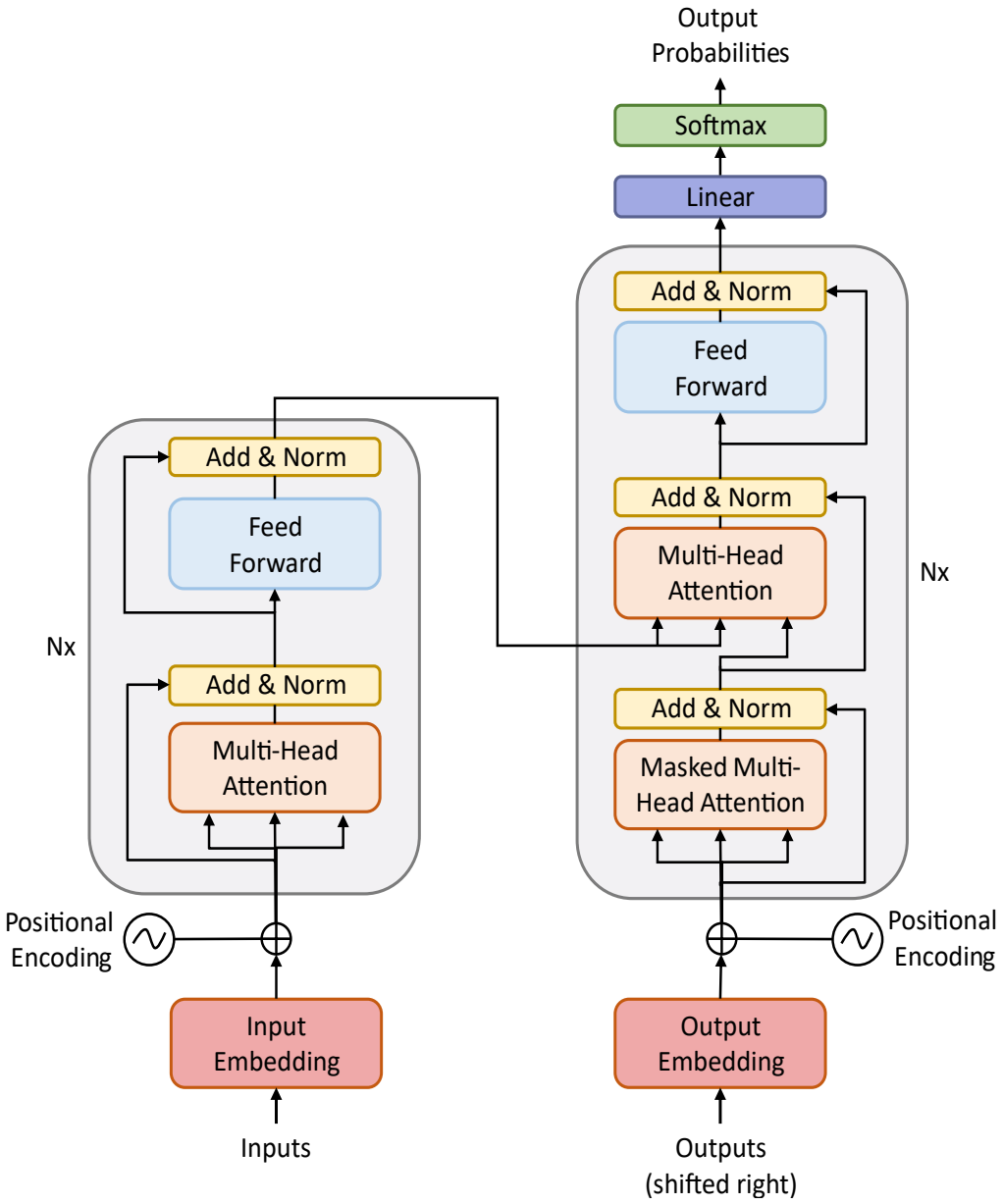

Figure 2.12. *The basic Transformer model's architecture. The encoder is positioned on the left block, while the decoder is positioned on the right block. Nx is the number of encoder-decoder layers in the Transformer. The figure is sourced from [4].*

d_{ff}. In the original Transformer architecture, the fundamental values are set as follows: $d_{model} = 512$, both N_e and N_d are 6, and $d_{ff} = 2048$. Figure 2.12 shows the Transformer architecture in a detailed way.

Inputs and ground truth sequences are initially projected into vectors called embeddings through a feed-forward neural network (FFN). Since the Transformer lacks any inherent notion of sequence order, positional information must be added to the embeddings. This is typically done by adding fixed positional encoding vectors to the input embeddings, which contain information about the position of each token in the sequence. Positional encodings can be approached in various ways, for instance, encodings can be learned

[30] or as introduced in [31]. The encoder outputs are obtained by passing these extracted positional embeddings through an encoder block that includes a feed-forward layer and a multi-head attention mechanism.

The positional embeddings of the ground truth are passed through the decoder block which contains an extra layer of masked multi-head attention along with multi-head attention and a feed-forward layer. Masked multi-head attention is a variant of the attention mechanism used in transformers. It involves multiple attention heads that operate in parallel, each attending to distinct segments of the input sequence. The masking aspect ensures that each position in the sequence can only attend to previous positions, preventing information leakage from future tokens during training or generation.

2.3 Natural Language Processing

Natural language processing (NLP) is a facet of artificial intelligence (AI) that empowers computers to comprehend, interpret, and generate human language. It performs tasks such as text summarization, language translation, sentiment analysis, etc. In the thesis, the datasets used to train contain a sequence of words in the natural language called captions. So, it is critical to express these words in the form of vectors so that machine learning algorithms can interpret them. In the following sections, the fundamental representation of words as vectors using one-hot encoding is explained. The byte-pair encoding (BPE) tokenization utilized in the present work is briefly discussed in section 2.3.2.

2.3.1 Word Vectors

One of the simplest methods for representing words as vectors is through one-hot encoding. In machine learning models, it is especially useful for representing categorical information as binary vectors, such as decision trees. Each class or category is represented as a unique binary vector, where all elements are assigned a value of 0 except for the element corresponding to the category, which is set to 1. For instance, a categorical word set contains X = {“*bird*”, “*chirp*”, “*tree*”}, it can be encoded as

‘bird’ = [1, 0, 0]
‘chirp’ = [0, 1, 0]
‘tree’ = [0, 0, 1]

However, one-hot encoding has two major limitations; high dimensionality and lack of semantic information. One-hot encoding yields high dimensional vectors, where the dimension is equal to the vocabulary size. This can result in highly lengthy and sparse feature spaces, particularly in languages with huge vocabularies such as English. Managing and analyzing such extensive volumes of data can be computationally and memory-intensive.

For example, if the above-mentioned word set has millions of words, it can be represented as

‘bird’ = [1, 0, 0, ..., 0]
‘chirp’ = [0, 1, 0, ..., 0]
‘tree’ = [0, 0, 1, ..., 0]

One-hot encoded vectors are orthogonal to one another, which means they have no intrinsic similarity or meaning. As a result, cosine similarity and other similarity measurements are always zero. This lack of semantic information makes it challenging for models to capture relationships and meanings between words. From the above word set, the cosine similarity is the same between the vector pairs ‘bird’, ‘chirp’, and ‘bird’, ‘tree’. it can be represented as

$$CosineSimilarity(\text{‘bird’, ‘chirp’}) = CosineSimilarity(\text{‘bird’, ‘tree’}) = 0$$

The aforementioned limitations have resulted in the development of alternative word representations, such as word embeddings (e.g., Word2Vec [32], GloVe [33], and FastText [34]), which try to solve both large dimensionality and semantic information challenges. Word2Vec is a word embedding technique that is most commonly used in natural language processing (NLP) and machine learning. It is designed to learn dense vector representations (embeddings) of words from a large set of corpora to capture semantic relationships between the words and has high cosine similarity. FastText is an extension of Word2Vec as it is designed to handle the subword information.

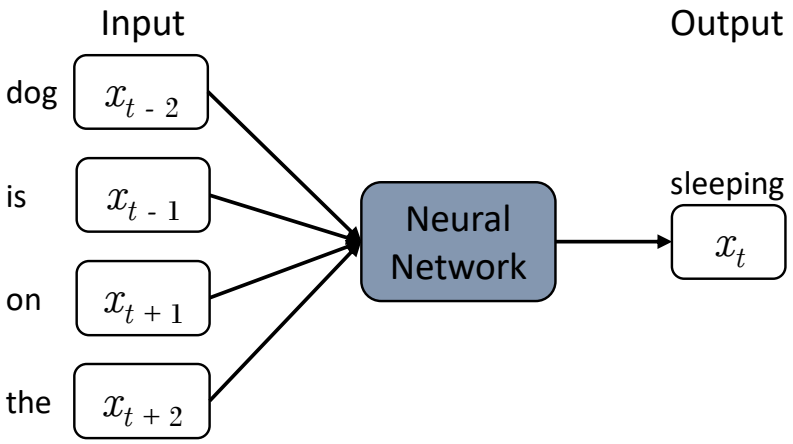

Figure 2.13. *Continuous bag of words (CBOW) architecture. Using the context words provided, it anticipates the target word.*

Word2Vec and FastText both employ two techniques to represent words as vectors: continuous bag of words (CBOW) [32] and skip-gram model [35]. Within the CBOW model, a neural network aims to forecast the target word based on its adjacent context words and a defined context window. The ’context window’ parameter specifies how many words to include as context on both sides of the target word. As an illustration, given the sentence

'The dog is sleeping on the bed', if the context words are 'dog', 'is', 'on', and 'the', then CBOW tries to predict the target word 'sleeping'. Here, the context window is set to two.

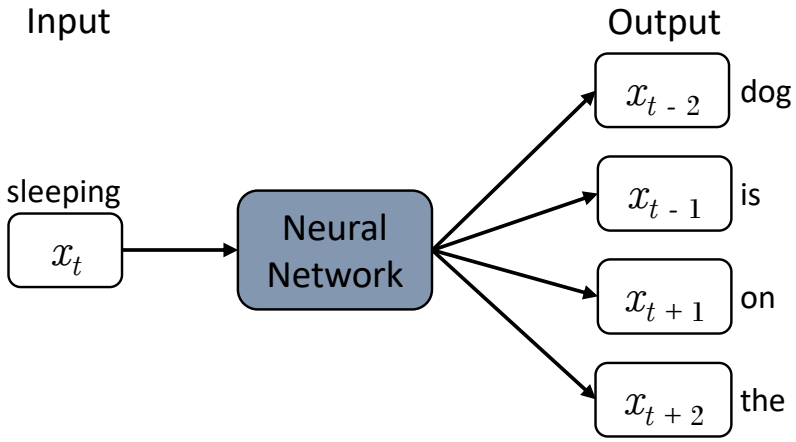

Figure 2.14. *Skip-gram architecture. It predicts the surrounding context words based on the given target word.*

The skip-gram model is the vice-versa of the CBOW model, where the target word is given and it predicts the context words. Using the aforementioned example, if the target word 'sleeping' is given, skip-gram tries to predict the surrounding context words, 'dog', 'is', 'on', and 'the'. The techniques CBOW and skip-gram model architectures are shown in Figures 2.13 and 2.14, respectively. While CBOW is faster to train and works well on small to medium datasets, skip-gram captures fine-grained semantic relationships and is preferred for larger datasets. Furthermore, FastText takes into account subword information such as character n-grams to capture meaningful relationships for the out-of-vocabulary (OOV) words. For instance, In the case of words like smaller, faster, and larger, FastText recognizes that they share the common character n-gram ($n = 2$) "er" at the end of the word.

2.3.2 Word Tokenization / Language Modelling

In this thesis, to encode the captions, byte-pair encoding (BPE) [36] is used. Byte-pair encoding (BPE) is a data compression and text encoding algorithm used in various natural language processing (NLP) tasks, including subword tokenization and text compression. It is extensively used for handling languages and creating subword tokenizers for machine translation and language modeling.

BPE was initially designed as a compression technique that combines common byte pairs and denotes them with unused bytes [37]. It segments words into subwords, ranging from as short as a single character to as long as an entire word. This segmentation is achieved by iteratively merging frequently occurring adjacent characters or character sequences (bi-grams) into new symbols, ultimately forming a subword vocabulary. It is crucial to

identify the ideal number of merges to achieve optimal performance. Vocabulary size can also be predefined to limit the number of merges while training.

Suppose, we have a vocabulary set containing words, V = { 'old', 'older', 'lower' }. A special end token is added at the end of each word i.e., V = { 'old</w>', 'older</w>', 'lower</w>' }. Initially, every character and the special token in the training set is considered a new token. So the basic vocabulary consists of individual characters: {'o', 'l', 'd', 'w', 'e', 'r', '</w>'}. The most common adjacent characters (bi-grams and tri-grams) from the corpus are 'er', 'ol', and 'old', which can be merged and added to the vocabulary. This process repeats for a few iterations to get a vocabulary set as {'old', 'l', 'o', 'w', 'er</w>'}. This can be further used to tokenize the sentences into subwords.

Byte-pair encoding is an unsupervised learning technique that learns patterns and subword representations of the text data. Other tokenization approaches, such as subword regularization [38] and BPE-dropout [39], have been proposed to increase robustness. The motivation to use BPE in this thesis is that it can handle morphologically complex languages effectively and has the ability to adapt to the training corpus while also handling out-of-vocabulary (OOV) words.

2.4 Sequence-to-Sequence Models for Captioning

Sequence-to-sequence (Seq2Seq) models are a type of neural network that was developed to process a sequence of data to generate an output sequence. In simple terms, the input and output are both sequences. They find extensive application in natural language processing tasks such as machine translation, speech recognition, and more. In machine translation, both the source and target are textual sequences, whereas in automated audio processing, an audio sequence is taken in as an input and the output is a textual representation. Sequence lengths in both tasks could vary not only across different input-output pairs but also within the same sample, emphasizing the necessity for specific deep neural network designs capable of processing variable-length data.

The basic architecture of Seq2Seq models comprises two primary components, namely the encoder and decoder, explained in the following sections 2.4.1 and 2.4.2 respectively. This method does not try to encode a whole input statement into a single fixed-length vector, which is the primary distinction between it and the conventional encoder-decoder. Instead, an adaptive subset of these vectors is chosen once the input text has been converted into a sequence of vectors [40].

2.4.1 Encoder

In the encoder, the input sequences are processed and transformed into fixed-length vector representations. It is often referred to as context vectors or embeddings. The information regarding the input sequence is captured in these embeddings, or vectors, that are used to generate the output sequences. The encoder can be implemented through the most popular deep neural networks such as recurrent neural networks (RNNs), long short-term memory (LSTM), or gated recurrent units (GRUs). For each timestep t in the input sequence, the mathematical representation of the hidden state of the encoder can be expressed as:

$$h_t = Encoder(x_t, h_{t-1}) \tag{2.14}$$

where x_t and h_t represent the input and hidden state of the encoder, respectively, at timestep t, and $Encoder$ denotes the selected recurrent unit (RNN, LSTM, or GRU). After T timesteps, the output of the last hidden state h_T of the encoder serves as a context vector c.

In recent times, to improve the performance of the models and address different types of data, various encoder architectures have been used. Some of them are Transformers and convolutional neural networks (CNNs). In particular, attention mechanisms like self-attention and multi-head attention are employed as encoders to capture dependencies between different parts of the input sequence. However, the encoder architectures are selected depending on the task requirements.

2.4.2 Decoder

The context vector generated by the encoder is given as input to the decoder, thus generating the output sequence in an auto-regressive manner. In simple words, it generates one element of the output sequence at each time step, conditioned on the previous elements and the context vector. RNNs and LSTM are often used as decoders to generate the output sequence. The previously generated outputs and the context vector are provided as inputs to the decoder. The formulation of the decoder at each timestep t in the output sequence is given as

$$\begin{aligned} s_t &= Decoder(s_{t-1}, y_{t-1}, c) \\ y_t &= softmax(W_s s_t) \end{aligned} \tag{2.15}$$

where s_t is the hidden state of the decoder at time step t, y_t is the output (probability distribution over the vocabulary) at time step t, y_{t-1} is the previously generated output,

c is the context vector from the encoder, W_s is the output weight matrix, $Decoder$ is the decoder's recurrent unit.

Seq2Seq models are trained by calculating the loss for each time step of the output and then using back-propagation to update the model's weights. During the training process, it can be beneficial to provide the actual correct output from the previous time step as input to the decoder rather than using the model's own generated output from the previous step. This technique, which can speed up training and stabilize learning, is referred to as teacher forcing [41].

Several decoders other than LSTM and GRU have been used in recent times, depending on the specific task. Transformers and attention-based decoders have become quite popular due to their strong performance in NLP tasks. At each decoding stage, an attention mechanism allows the decoder to weigh the relevance of different sections of the input sequence. Each element in the input sequence is given attention scores or weights, indicating how much attention the decoder should give that element while generating the following token.

When working with sequence-to-sequence models like the Transformer, two typical decoding methods used in sequence-generating tasks are greedy search and beam search. These techniques determine the process by which the model, given an input sequence, produces an output sequence. In a simple decoding algorithm like a greedy search algorithm, the model chooses a token with the highest predicted probability as the next element without considering future sequences. As it ignores the global context of the sequence, it may lead to improper results.

Instead of only choosing the top-scoring token at each step, exploring multiple sequences in parallel would be better for the global context. Beam search is one such sophisticated decoding algorithm that keeps track of the fixed number (*beam size*) of candidate sequences. It considers the top-k tokens at each step and preserves the k most likely sequences based on cumulative probabilities. This can be helpful in capturing long-range dependencies and producing coherent output. In beam search, the *beam size* is a hyperparameter that may be adjusted depending on the specific task. Beam search frequently incorporates length normalization as a standard step to minimize the issue of favoring shorter sequences [42].

2.5 Automated Audio Captioning

Automated audio captioning serves as a cross-modal task that bridges the gap between audio signal processing and natural language processing. The aim of this project is to provide a natural language written description, or caption, for a particular audio recording [1]. Audio captions usually provide one-sentence summaries of the most common audio occurrences and scenarios in the audio samples. The physical characteristics of sound objects and the acoustic environment, as well as the spatial-temporal interactions between audio occurrences and scenes, may all be included in these comprehensive descriptions [43]. An example of a caption that describes an audio clip can be "children yelling in the background and a voice complaining in the foreground at a car wash"[1].

AAC systems generally employ the sequence-to-sequence model as a foundational framework to generate captions for audio content. This model takes in a sequence of audio frames as input to the encoder and predicts an output caption from the decoder as explained in Section 2.4. In the thesis, we use a Transformer as an encoder to get the audio embeddings and a transformer-based neural network as a decoder to predict the output. Either one-hot or vectorization techniques are utilized to encode the words. We use a subword encoding technique i.e., byte-pair encoding (BPE) as mentioned in section 2.3.2.

2.6 Generic AAC and Domain Adaptation

All the automated audio captioning systems proposed so far have been focusing on developing a model for the generic dataset that contains a wide range of domains from environment, and traffic to animal sound events. Instead of focusing on all the domains, training the system with diverse audio content belonging to a specific domain can be valuable. For instance, in terms of real-time implementation of AAC systems in parking lots or in traffic, training with diverse traffic or vehicle data is useful. Due to the inherent limitation of data scarcity in a specific domain, it would be difficult to train a domain-specific model which may result in poor results. Furthermore, generic AAC can create unnecessary complexity due to a wide range of domains.

In order to solve this issue, we first train a general AAC system using generic captioning datasets that includes material from a wide range of domains, and then adapt this system to a domain-specific task. This is called domain adaptation. It can be achieved through two machine learning techniques: transfer learning and layer-wise finetuning which are explained in the following Sections 2.6.1 and 2.6.2 respectively.

[1] Caption is from the Clotho dataset.

2.6.1 Transfer learning

A machine learning technique called transfer learning leverages a model that has been developed on one task to be fine-tuned or modified for an alternate but associated task. It is particularly beneficial in situations when the target job has limited data availability because it makes use of the information gained from the source task to enhance the target task's performance. The knowledge transfer can include feature patterns, representations, or learned parameters. New tasks can benefit from the use of pre-trained models that have been trained on bigger datasets.

In this thesis, a pre-trained Transformer model is used as the encoder to extract the audio embeddings. For enhancing the performance of the generic AAC, the system is trained with the AudioCaps [6] dataset and then fine-tuned with the Clotho [5] dataset. While in domain adaptation, the Clotho dataset is initially split into 2 sets containing vehicle data and the rest of the data i.e., external data. The system is first trained with external data and fine-tuned with the vehicle data to obtain a domain-specific model. This is explained further in detail in Chapter 3.

2.6.2 Layer-wise Fine-Tuning

Layer-wise fine-tuning is a machine learning approach that involves fine-tuning a pre-trained neural network model by modifying or updating individual layers or sections of the model while allowing the remaining layers alone. The fine-tuning process has more control and specialization since different layers capture different layers of abstraction and domain-specific information. Several layer-wise fine-tuning techniques involve selective updating, freezing layers, fine-tuning strategy, and so on.

In selective updating, a group of layers or some specific layers are chosen for fine-tuning based on the experimental results and domain knowledge. While in freezing layers, the layers that are not chosen for fine-tuning are "frozen," which means that during training, their weights and parameters stay unchanged. This helps in preserving the knowledge and features learned in earlier training stages. In the fine-tuning strategy, layers chosen for fine-tuning are adjusted based on the target task's requirements. This might involve changing the learning rates, applying different optimization techniques, or making other modifications to the layer's architecture.

In this thesis, all the three above approaches of fine-tuning are utilized. In the encoder-decoder architecture used, selective updating of specific layers of the encoder and freezing the remaining layers has been experimented with. Along with that, while fine-tuning, learning rates are changed. By selectively updating and freezing layers, the model has retained valuable prior knowledge while specializing in the nuances of the new task.

3. PROPOSED SYSTEM

In this chapter, we have provided a comprehensive explanation of the deep learning system that was developed to implement the audio captioning system, along with insights into the datasets utilized. The detailed discussion of the models employed for audio embedding extraction, word processing, and the Transformer model's implementation, has been outlined in Section 3.1. Datasets in detail are explained in Section 3.2. In Section 3.3, how the model is trained with different strategies has been discussed for both generic captioning and domain adaptation systems.

3.1 Model Architecture

A traditional sequence-to-sequence encoder-decoder architecture has been adopted to implement this thesis. A baseline model automated audio captioning task[1] proposed in the Detection and Classification of Acoustic Scenes and Events (DCASE) 2022 challenge is taken as a reference model for the automated audio captioning system. VGGish [44] has been employed to extract audio features, while a Transformer serves as both the encoder and decoder. For acoustic scene classification, sound event recognition, and audio tagging tasks, patchout fast spectrogram transformer (PaSST) based audio embeddings were presented recently, which have shown considerable improvement in performance[2]. As a result, we replaced the VGGish-based audio feature extractor in the baseline system with a PaSST-based audio feature extractor. The input to the Transformer encoder consists of PaSST audio embeddings, which it uses to extract encoded audio feature outputs. In Transformers, specifically, auto-regressive decoders are used which generate output sequences step by step, where each step relies on previously generated elements, making it suitable for tasks like language translation and text generation. This auto-regressive decoder takes the encoder's output as input and generates a probability distribution over the vocabulary. This vocabulary is constructed through training on the distinct words in the datasets using a byte-pair encoding (BPE) trainer. The fundamental audio captioning system has been illustrated in Figure 3.1.

[1] https://dcase.community/challenge2022/task-automatic-audio-captioning
[2] https://github.com/kkoutini/passt_hear21

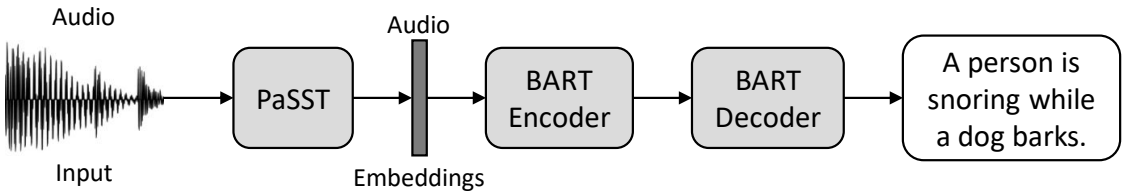

Figure 3.1. *Overview of an automated audio captioning system.*

3.1.1 Audio Embeddings

PaSST, the Patchout Fast Spectrogram Transformer, is employed for extracting audio features from a provided audio input in all of the experiments. It is one of the backbone models trained on the AudioSet [45] dataset and is initialized with the weights from the vision transformer pre-trained on ImageNet. Similar to Vision Transformer (ViT) proposed in [27], PaSST operates by extracting patches from the input audio spectrogram. The transformer architecture of the PaSST is summarized in Figure 3.2.

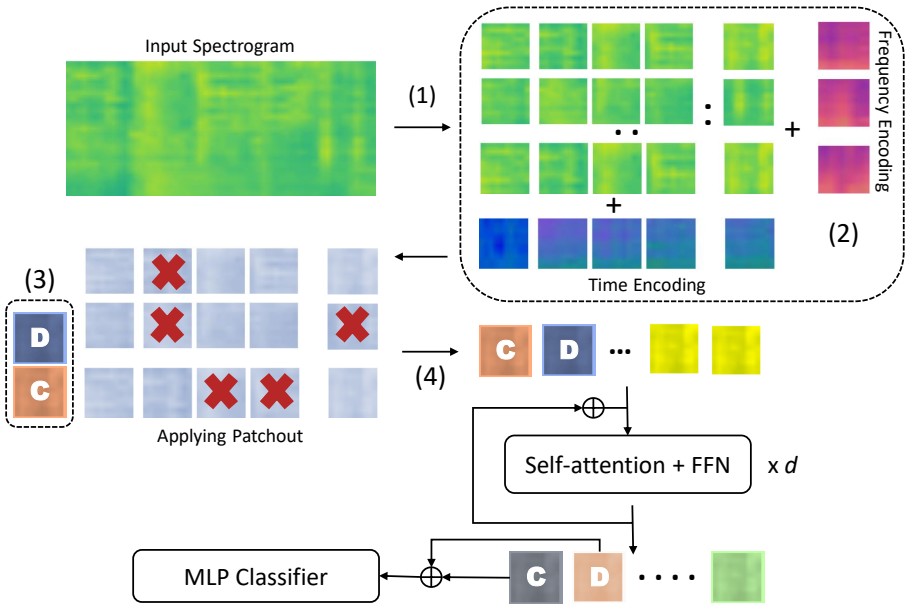

Figure 3.2. *The architecture of Patchout Transformer. Detailed visualization of the audio embedding extraction from the PaSST model is shown. The figure is adapted from [46].*

Initially, the mel-spectrogram is extracted from the audio signal. The process is initiated at the top-left corner of the figure by feeding the extracted audio spectrogram as the input to the model. As detailed in the reference [27], in (1), the fixed-size patches are extracted from the input audio spectrogram and are flattened to embed into a linear projection embedding. In (2), positional encodings for both frequency and time are incorporated. This simplifies the process of performing inference and fine-tuning pre-trained models for downstream tasks with shorter audio durations.

In step (3), *Patchout* is applied, where random patches are dropped out of the input sequence, reducing the sequence length. When the transformer is being trained, it is provided with this input sequence, which encourages the transformer to use an incomplete sequence to do the classification [46]. There are two types of Patchout: structured and unstructured Patchout. The entire column or row of retrieved patches is discarded after specific frequency bins or periods are selected in structured patchout. Unstructured patchout is the most basic type of patchout, where the patches to be discarded are chosen at random [46]. A classification token C is added to the patches at the end, and for models based on Dataefficient Image Transformers (Deit) [47], a distillation token D is added. In the thesis, we use unstructured Patchout, and only a classification token is added.

Finally, in step (4), the sequence is flattened and passed through d self-attention blocks, where d is the depth of the transformer. The output obtained from the transformer is sent to the multi-layer perception (MLP) classifier to obtain the final audio embeddings.

The PaSST embedding model comprises various configurations, including scene embeddings and time-stamp embeddings. In the scene embedding setup, a singular embedding is generated for a specific audio segment. Conversely, the timestamp embeddings setup generates multiple embeddings without pooling them over time. In the experiments conducted in this thesis, scene embeddings are employed.

To extract either the scene embeddings or time-stamp embeddings, PaSST utilizes various distinct modules for audio embedding extraction. In this thesis, specifically three modules namely *PaSST base, PaSST base2level, and PaSST base2levelmel* are used to extract audio embeddings. These three modules were chosen because of their state-of-the-art performance on the FSD50K [48] dataset for the audio tagging task in the HEAR21[3] challenge. The difference between these three modules lies in the dimensions of the time-stamp embeddings extracted during the process. In the case of *PaSST base*, the time-stamp embedding dimensions are set to 1295. However, for *PaSST base2level*, the dimensions are doubled to 1295×2, i.e., it concatenates a longer window (160 ms and 800ms), and for *PaSST base2levelmel*, the dimensions become $768 + 1295 \times 2$, where a raw mel spectrogram is concatenated additionally [49]. The derivation of scene embeddings involves computing the mean over the time-stamp embedding dimension mentioned above, yielding a resultant dimensionality of 1295.

In this thesis, we use audio scene embeddings that are specifically obtained using the *PaSST base2levelmel*[4] module. These embeddings have a dimensionality of $\mathbb{R}^{1295\times1}$, with 1295 being the embedding dimension.

[3] https://neuralaudio.ai/hear.html

[4] https://github.com/kkoutini/passt_hear21

3.1.2 BPE Tokenization and Vocabulary

A tokenizer is a natural language processing component that breaks down a text or sequence of text into smaller units, typically words, subwords, or tokens. These tokens are easier to handle than raw text because they are segmented into meaningful subunits, making it easier for machine learning models to understand and process them. In the baseline model, bidirectional and auto-regressive transformers (BART) tokenizers have been used to tokenize words. It is a tokenization method specifically designed for BART model architecture, a transformer-based neural network developed by Facebook AI [50]. It is trained on a diverse corpus of content, including news articles, books, and websites. A specific BART tokenizer named *facebook/bart-base* tokenizer is utilized which has a vocabulary size of 50265 tokens.

In the thesis, we have experimented by extending the vocabulary of the BART tokenizer. This is done by adding the unique words extracted from the Clotho dataset captions. In some contexts, if a word or token related to audio captioning datasets is not present in the language model or tokenizer, it might result in out-of-vocabulary (OOV) issues. It can pose challenges to NLP tasks as the tokenizer cannot handle the OOV word or token. To overcome this, a subword tokenization algorithm namely byte-pair encoding is utilized. The BPE algorithm is briefly explained in Section 2.3.2 of Chapter 2.

To train the BPE tokenizer for the AAC task, the process involves several general steps, including data pre-processing, vocabulary construction, subword tokenization using the BPE algorithm, and model training. Data pre-processing entails tasks such as removing punctuation, eliminating double spaces, and converting all characters to lowercase within each sentence from the dataset captions. Furthermore, each caption has two unique tokens inserted at the beginning and end: "<eos>" (end of sentence) and "<sos>" (start of sentence).

Furthermore, all the distinct words present in the training split dataset are gathered, resulting in 4,368 unique words for the Clotho dataset. For transfer learning purposes, vocabularies from both AudioCaps and Clotho datasets are merged, resulting in a combined vocabulary of 6,656 words. The tokenizer is then trained from scratch using these unique words from the dataset corpus, along with any special tokens required. Each token can be either a subword or an individual letter from the model vocabulary. The training process is explained briefly in Section 2.3.2 of Chapter 2

The length of the vocabulary in this tokenizer varies, with 6,316 words for the Clotho dataset and 8,989 words for the combined corpus of AudioCaps and Clotho.

3.1.3 Transformer

A traditional sequence-to-sequence backbone model of bidirectional and auto-regressive transformers is used for the experiments in this thesis. The basic visualization of BART is illustrated in Figure 3.3. The implementation consists of a sequence-to-sequence model that utilizes a left-to-right auto-regressive decoder and a bidirectional encoder over distorted text [50]. There are n decoder and encoder layers within it. In the baseline system, a base model of BART with $n = 6$ layers has been used. In the thesis, we experiment with $n = 4, 12$ layers, with twelve attention heads and affine layers with 3072 hidden units.

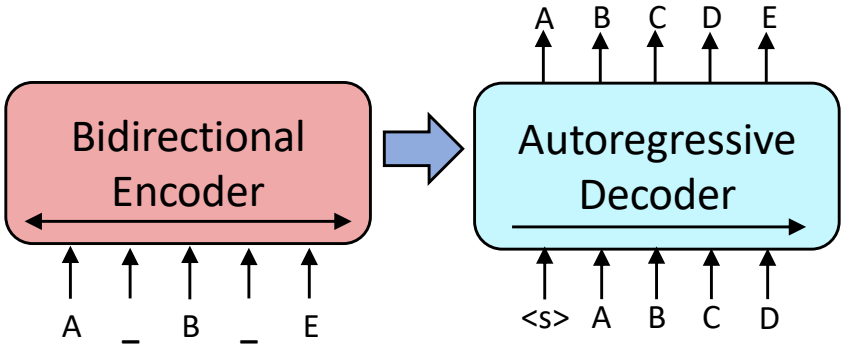

Figure 3.3. *The schematic illustration of BART. The encoder does not require alignment with decoder outputs, enabling arbitrary noise transformations. In this case, the text has been replaced by mask symbols, corrupting the page. A bidirectional model is utilized to encode the corrupted text on the left, and an autoregressive decoder is then employed to calculate the probability of the original document on the right [50]. The figure is adapted from [50].*

Audio embeddings from the PaSST model are passed as an input to the encoder of BART, which gives the encoded audio outputs. Each block of the encoder consists of a multi-head attention block and a feed-forward layer that outputs encoded 768-dimensional audio features. Each decoder block additionally contains a masked multi-head attention block along with a multi-head attention block and performs cross-attention across the audio features that are encoded from the final hidden layer. It does not use the additional feed-forward network before word prediction as in Figure 2.12 of Chapter 2. The auto-regressive decoder employs encoder outputs along with BPE-tokenized word embeddings. Each token within the word sequence is associated with a feature vector via an embedding map and then forwarded to the decoder. In each layer of the decoder, attention mechanisms are employed: self-attention considers previously generated tokens, and cross-attention considers the entire encoder output sequence. Each decoder layer within the transformer produces an output of a 768-dimensional embedding. The BART for audio captioning is depicted in the following Figure 3.4.

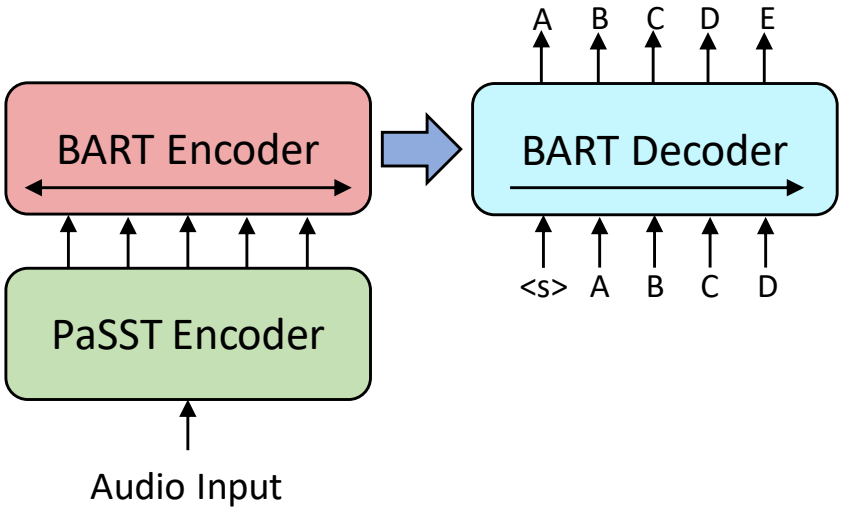

Figure 3.4. *BART schematic illustration for automated audio captioning. An additional PaSST encoder is used to extract audio embeddings and undergo training in the BART model to generate a caption in an auto-regressive way.*

Subsequently, a final layer, comprising a softmax activation function, produces probability scores for each vocabulary item. The final output is determined through beam decoding with a beam size of 4, which generates multiple probabilities and selects the most favorable one as the ultimate output. During training, standard cross-entropy loss is utilized to minimize the training loss.

3.2 Datasets

For all of the experiments in the thesis, two AAC datasets are utilized: Clotho (version 2)[5] [5] and AudioCaps[6] [6]. These datasets encompass a wide variety of generic audio samples paired with diverse textual captions spanning various domains. For the domain adaptation experiments, the dataset is sourced from the Clotho dataset, where the audio samples specific to animal and vehicle domains are segregated based on the metadata keywords.

3.2.1 AudioCaps

AudioCaps [6] is one of the easily available and largest datasets available for audio captioning tasks. It consists of $46,000$ audio samples comprising around 127 hours of data. It is gathered through crowdsourcing on AudioSet [45] which is an audio tagging dataset that contains audio and video samples from YouTube. The duration of each audio sample is 10s.

[5]https://github.com/audio-captioning/clotho-dataset
[6]https://github.com/cdjkim/audiocaps

The dataset is divided into three splits: *train*, *validation*, and *evaluation* splits. Every audio sample in the training split has one ground truth caption associated with it. On the other hand, every file in the validation and evaluation sets has a set of 5 captions associated with it. These captions are crowdsourced from Amazon Mechanical Turk (AMT). The audio samples and their corresponding captions contain a diverse range of acoustic environments and domains. These domains encompass nature, sports, music, and everyday life scenarios, making it a valuable and comprehensive resource for training and evaluating captioning models.

3.2.2 Clotho

Clotho [5] serves as a significant benchmark dataset in the field of captioning research and is designated for task 6 within the DCASE challenge. It comprises a total of 6972 audio samples with a total duration of approximately 43.6 hours. The data is sourced from the online platform Freesound [51], and the audio clip durations range from 15 to 30 seconds.

The dataset has been divided into four splits: *development, validation, evaluation*, and *test*. In this dataset, the development split comprises $3,839$ audio clips, with an equal distribution of $1,045$ clips each in the validation and evaluation splits. Additionally, the test data split consists of $1,043$ samples. Notably, each audio clip is manually annotated by human annotators, and this annotation includes the creation of 5 captions for each clip. The captions of the test split are not public and are not used for evaluation in the thesis. The captions are also crowdsourced from AMT and have a length of 8 to 20 words per caption.

For domain adaptation, the Clotho dataset is divided into two datasets: domain/target data and external data. Initially, the insights about the domains covered in the whole dataset are gathered through the statistical analysis of the keywords provided in the metadata of the training dataset. For the experiments in the thesis, specifically two domains are focused such as animals and vehicles.

By leveraging domain-specific keywords identified in the metadata, the dataset was partitioned into animal-related and vehicle-related data. Initially, a keyword list is pre-processed from the metadata, where terms associated with animals, such as 'bird', 'dog', and 'chirp,' from the metadata are flagged as 'yes.' Subsequently, files containing these positively marked keywords were extracted from the complete Clotho dataset. Similarly, for the vehicle data, metadata keywords were pre-processed, and keywords such as 'car,' 'motor,' and 'horn' from the metadata were labeled as 'yes.' Files containing these affirmative keywords were then isolated from the overall Clotho dataset. This process efficiently separates animal-related and vehicle-related data based on the specified keywords.

The remaining files from the Clotho dataset serve as external data in the case of the animal-specific dataset, and the same follows for the vehicle-specific domain. This division enabled the creation of subsets of data specifically tailored to our intended domain. In the animal-specific and vehicle-specific subsets, there were 721 and 114 audio clips respectively for training and evaluation in the animal domain, and 827 and 115 audio clips for training and evaluation in the vehicle domain, respectively.

3.3 Model Training

The proposed model serves a dual purpose, being employed for training both generic captioning systems and adapting to specific domains (domain adaptation). Both of these models take as input a 1295-dimensional audio embedding feature derived from the Patchout Fast Spectrogram Transformer (PaSST). PaSST, in turn, processes a mono audio file, meaning it deals with single-channel audio data sampled at 32 kHz. In the following sections, the techniques applied to train the models for both of these distinct systems are discussed in detail.

3.3.1 Generic Captioning

To train the proposed model for generic captioning, the entire Clotho dataset is utilized. The model is trained using the development and validation splits of the Clotho data, and its performance is evaluated using the evaluation split of the Clotho dataset. Various configurations of the BART model have been tested: a base model (BART-base) with $n = 6$, a large model (BART-large) with $n = 12$, and $n = 4$, where n is the number of encoder-decoder layers in the Transformer. Additionally, experimentation has been conducted with an extended vocabulary BART tokenizer, alongside the original BART tokenizer, which has a vocabulary size of 50,265 tokens. Furthermore, a BPE tokenizer, trained on the Clotho corpus, has been employed as a baseline for all subsequent experiments conducted in this thesis.

Due to the limited data availability in the Clotho dataset, the transfer learning technique is conducted using the AudioCaps dataset. The model is trained with AudioCaps [6] and then fine-tuned and evaluated with the Clotho [5] dataset. Furthermore, layer-wise fine-tuning has been performed which is briefly explained in Section 2.6.2 of Chapter 2. To facilitate transfer learning from the AudioCaps dataset to Clotho, four distinct strategies were implemented that aimed at freezing specific layers within the transformer architecture. These strategies encompass the following:

Freezing all attention layers: In this approach, all the attention layers in both the encoder and decoder of the transformer were frozen. This primarily transfers learned dependencies and relationships among input tokens.

Freezing the whole encoder: The weights of the entire encoder layers are frozen in this technique. It goes a step further by additionally transferring the extracted audio features and their representations to the target domain.

Freezing only attention layers in the encoder: Here, the focus is on preserving linguistic knowledge and contextual understanding of the input, making sure that these layers remain static during transfer.

Freezing attention layers in the decoder: In this strategy, we freeze the attention layers of the decoder within the transformer. This allows the model to maintain the capacity to generate captions based on the transferred knowledge obtained during pre-training.

These diverse freezing strategies enable us to tailor the transfer learning process and selectively retain specific aspects of the pre-trained knowledge, ultimately enhancing the model's adaptability to the target domain.

3.3.2 Domain Adaptation

The Clotho dataset has been selected as the foundation for domain adaptation, aiming to tailor the audio captioning system to operate effectively within a specific domain. To achieve this, the dataset has been divided into distinct target domains, as explained in section 3.2.2. The BART model with $n = 4$ encoder-decoder layer configuration has been utilized to train the system.

To adapt the AAC system to a particular domain, such as animal-related data, the initial step involves training the model using the animal data extracted from the development and validation splits of the Clotho dataset. Subsequently, the model's performance is assessed exclusively on the animal-related data. The same process is followed for the vehicle-related data domain as well. Given the limited availability of domain-specific data, we utilize external data from the Clotho dataset to perform fine-tuning. This process enables the transfer of knowledge to enhance the model's performance in the target domains.

The model undergoes a two-step training process. Initially, it is trained on the external data extracted from the Clotho dataset. Subsequently, fine-tuning takes place using domain-specific data related to animals or vehicles from the Clotho dataset. Additionally, the layer-wise fine-tuning strategies outlined in section 3.3.1 have been employed to examine and optimize the domain adaptation process.

In addition to the aforementioned training process, an alternative approach has been explored. In this scenario, the model is initially trained using the complete Clotho dataset. Subsequently, the model undergoes fine-tuning with domain-specific data, focusing on either animals or vehicles, which is separated from the Clotho dataset. Similar layer-wise fine-tuning strategies are also implemented in this case.

The primary objective during fine-tuning and layer-wise fine-tuning lies in transferring essential knowledge, enabling the model to leverage its prior learning experiences effectively and adapt more efficiently to the audio captioning task within the specific domain.

4. EXPERIMENTS AND EVALUATION

In this chapter, the experiments conducted in this thesis and the corresponding results are focused. The experimental configurations employed in the subsequent sections have been outlined in Section 4.1. Following that, Section 4.2 provides an explanation of the evaluation metrics that were utilized in our experiments. In the final Section 4.3, the performance of the both generic and domain adaptation captioning systems has been presented.

4.1 Experimental Setup

The implementation of all models is carried out using the Python library PyTorch[1] [52], a widely-used framework for deep learning. The identical set of hyperparameters employed in the baseline system of task 6 in the DCASE 2022 challenge has been utilized in this thesis. To minimize the cross-entropy loss, we employed the AdamW optimizer [53]. Throughout all stages of training, including fine-tuning and transfer learning, a constant learning rate of 10^{-5} was maintained. Additionally, a dropout rate of 0.1 was applied to the proposed model to mitigate overfitting issues.

For data processing, both input audio and text features were padded to fixed lengths of 32 and 64, respectively. Zero vectors were used for padding audio features, while a special "$< pad >$" token was employed for text feature padding.

The model undergoes training for a variable number of epochs, typically up to 20, depending on the specific experiment. After every 1000 iterations, we assess the loss on the validation set. The checkpoint associated with the lowest validation loss is saved and is used to evaluate during the inference stage. During the inference stage, we employ a beam search with a beam size of 4 to substantially enhance performance.

4.2 Metrics

In captioning tasks, such as image and audio captioning, the evaluation process usually involves a combination of translation metrics and metrics specifically tailored to the captioning task. This dual approach ensures a comprehensive assessment of the sys-

[1] https://github.com/pytorch/pytorch

tem's performance in accurately describing the audio content in meaningful and contextually appropriate textual captions. The most commonly employed translation metrics include BLEU [54], METEOR [55], and ROUGE-L [56]. Captioning metrics encompass CIDEr [57] and SPICE [58], in addition to a third metric, SPIDEr [59], which is a linear combination of the aforementioned two captioning metrics. A detailed explanation of the above-mentioned metrics is as follows:

BLEU (Bilingual Evaluation Understudy): It measures the quality of captions by comparing n-grams (unigrams, bigrams, etc.) in generated and ground truth captions. BLEU computes a weighted geometric mean of these n-gram precisions, that match with the ground truth captions [54].

METEOR (Metric for Evaluation of Translation with Explicit ORdering): Captions are evaluated based on precision, recall, stemming, and synonymy. Words in generated captions have been matched with ground truth captions, and an F-score is computed based on several matching alternatives [55].

ROUGE (Recall-Oriented Understudy for Gisting Evaluation): Caption quality is evaluated by calculating the overlap of n-grams in generated and ground truth captions. ROUGE-L emphasizes sentence-level similarity by focusing on the longest common subsequence [56].

CIDEr (Consensus-based Image Description Evaluation): Using term-frequency inverse-document-frequency (TF-IDF) weighting, CIDEr determines the weighted cosine similarities of n-grams [57]. CIDEr is especially useful during analyzing captions when several ground truth captions are available.

SPICE (Semantic Propositional Image Caption Evaluation): SPICE assesses captions on the basis of their semantic content. It determines whether captions have specified language components like objects, attributes, and relationships [58].

SPIDEr (SPICE-Derived): SPIDEr makes use of both CIDEr and SPICE by generating an average of these two metrics [59].

In addition to these conventional captioning metrics, the performance of the system is measured by two recently proposed metrics namely SentenceBert [60] and Fense [61]. SentenceBert is a neural network based on transformers, designed to generate meaningful sentence embeddings for capturing semantic information. For pairs of sentences, the distance between these embeddings has been used to determine their semantic similarity. Meanwhile, Fense combines the SentenceBert metric for caption similarity with an Error Detector component, which penalizes incorrect sentences.

In this thesis, SPIDEr stands out as the primary evaluation metric, following the evaluation methods employed in the DCASE audio captioning challenge. Alongside SPIDEr, we provide detailed insights into the system's performance through other metrics including CIDEr, SPICE, SentenceBert, and Fense scores in the subsequent sections. This comprehensive evaluation approach offers a well-rounded assessment of the system's capabilities, addressing both consensus and semantic content, as well as capturing semantic similarity and penalizing incorrect sentences.

4.3 Evaluation of Experiments

In this section, we present the results obtained from various experiments conducted for this thesis. The performance evaluation of different configurations of the PaSST and BART model has been reported in Sections 4.3.1 and 4.3.2, respectively. Different tokenizer performances while training the model has been presented in the Section 4.3.3. In the Section 4.3.4, the evaluation performance of the model when it is fine-tuned is summarized. Lastly, we report the domain adaptation results in the Section 4.3.5.

4.3.1 PaSST embeddings results

The baseline model for the audio captioning task in the DCASE 2022 challenge originally relies on VGGish audio embeddings, but the proposed system has introduced the use of PaSST audio embeddings as a replacement. As mentioned earlier, the PaSST model comprises three distinct modules designed for extracting scene embeddings from audio input.

The performance of these various PaSST modules is presented in Table 4.1. The results indicate that the *PaSST base2levelmel* module has outperformed the other modules *PaSST base* and *PaSST base2level* in terms of performance. Consequently, this module was selected to extract audio embeddings for all the future experiments conducted in this thesis.

Table 4.1. *Performance evaluation results of different audio embeddings.*

Setup	**CIDE$_r$**	**SPICE**	**SPIDE$_r$**	**FENSE**	**SentenceBERT**
PaSST base	0.392	0.114	0.253	-	-
PaSST base2level	0.392	0.114	0.253	-	-
PaSST base2levelmel	**0.425**	**0.120**	**0.278**	0.517	0.518

4.3.2 BART configurations results

Initially, the proposed system underwent training with a base model of BART, featuring $n = 6$ layers in both the encoder and the decoder. Furthermore, the experiment included testing the BART-large model with an extensive $n = 12$ layers and another variant with reduced complexity, employing $n = 4$ encoder-decoder layers.

A comprehensive assessment of the performance of these distinct BART model configurations is briefly presented in Table 4.2. The results from the table clearly indicate that the BART model featuring $n = 4$ encoder-decoder layers demonstrated superior performance compared to the other models. Consequently, this particular BART configuration was selected for utilization in future experiments.

Table 4.2. *Performance evaluation results of different BART configurations.*

Setup	CIDEr	SPICE	SPIDEr	FENSE	SentenceBERT
BART-base (n = 6)	0.434	0.123	0.278	0.518	0.517
BART-large (n = 12)	0.413	0.121	0.267	0.507	0.502
BART (n = 4)	**0.436**	**0.125**	**0.282**	0.525	0.526

4.3.3 Tokenizer performance results

Initially, the model was trained using both the standard BART tokenizer and its extended vocabulary version. Since the BART tokenizer is pretrained on a large corpus, it includes tokens unrelated to the specific captioning dataset. To address this, a Byte Pair Encoding (BPE) tokenizer was trained from scratch using the available training dataset corpus. The performance comparison of these tokenizers is summarized in Table 4.3.

Table 4.3. *Performance evaluation of different tokenizer setups.*

Setup	CIDEr	SPICE	SPIDEr	FENSE	SentenceBERT
BART	0.434	0.123	0.278	0.517	0.518
BART (extend vocabulary)	0.419	0.122	0.271	0.510	0.514
BPE (unique words only)	**0.439**	**0.127**	**0.283**	0.516	0.517

The results clearly indicate that the tokenizer trained on the specific dataset outperformed the one pretrained on a larger corpus containing a wide range of text from various domains. Subsequently, in the later experiments involving fine-tuning, this system, which incorporates PaSST as the audio embedding extractor, BART as the encoder-decoder, and the BPE tokenizer, was adopted as the baseline.

4.3.4 Layer-wise Fine-Tuning Results

The fine-tuning process was conducted on the two primary datasets, AudioCaps and Clotho, using two distinct strategies: fine-tuning involving all layers of the neural model and fine-tuning with selective layer freezing as explained in Section 3.3.1 of Chapter 3.

The initial phase involved training the model using the AudioCaps dataset. Subsequently, the fine-tuning phase was executed, during which the model's weights and parameters were updated through back-propagation. In contrast, for the latter set of experiments, the weights of the specific layers within the transformer were deliberately frozen and left unaltered during the fine-tuning process.

Comprehensive results detailing the outcomes of these varied fine-tuning experiments are presented in Table 4.4. These findings provide valuable insights into the model's adaptability and performance under different fine-tuning strategies.

Table 4.4. *Layer-wise fine-tuning performance evaluation.*

Setup	**CIDEr**	**SPICE**	**SPIDEr**	**FENSE**	**SentenceBERT**
Fine-tuning	0.453	0.130	0.291	**0.531**	**0.531**
Layer-wise fine-tuning the model across datasets (training with AudioCaps, and fine-tuning by freezing specific layers in the transformer with Clotho)					
Frozen layers					
All attention layers	0.456	0.130	0.293	0.529	0.527
Encoder layers	0.448	0.129	0.288	0.526	0.526
Encoder attention layers	0.468	0.133	**0.301**	0.530	0.528
Decoder attention layers	0.451	0.129	0.290	0.527	0.526

4.3.5 Domain Adaptation Results

Given the constraints of limited domain-specific data, we leveraged the insights gained from the preceding fine-tuning tests to optimize the approach for domain adaptation. Table 4.5 provides the results when the model is trained and evaluated with only a domain-specific dataset and when the model is trained with the entire Clotho dataset and evaluated on the domain-specific dataset. The table indicates that the SPIDEr score is higher when assessing the dataset with the generic model compared to the SPIDEr score obtained from the evaluation using the domain-specific model.

Table 4.6 offers a thorough comparative analysis of our experiments. It highlights the differences in results between domain adaptation with and without layer-wise fine-tuning, using a generic Clotho dataset that is split into external data and domain-specific data.

Table 4.5. *Domain adaptation results for vehicles and animals using domain-specific model and a generic model.*

Setup	$CIDE_r$	SPICE	$SPIDE_r$	FENSE	SentenceBERT
Vehicles					
Domain-specific model	0.324	0.101	0.213	0.476	0.497
Generic model	0.488	0.109	0.298	0.527	0.530
Animals					
Domain-specific model	0.294	0.139	0.217	0.566	0.568
Generic model	0.461	0.168	0.315	0.648	0.619

These results are generated by initially training the model with the external dataset from Clotho, followed by the application of fine-tuning and layer-wise fine-tuning using domain-specific datasets related to either animals or vehicles.

Table 4.6. *Domain adaptation results for vehicles and animals: Comparison of fine-tuning and layer-wise fine-tuning using external and domain-specific data.*

Setup	$CIDE_r$	SPICE	$SPIDE_r$	FENSE	SentenceBERT
Vehicles					
Fine-tuning	0.414	0.112	0.263	0.511	0.521
All attention layers	0.433	0.111	0.272	0.525	0.531
Encoder attention layers	0.407	0.113	0.260	0.514	0.520
Animals					
Fine-tuning	0.432	0.156	0.294	0.615	0.598
All attention layers	0.437	0.158	0.297	0.602	0.593
Encoder attention layers	0.389	0.153	0.271	0.612	0.598

The alternative training process mentioned in Section 3.3.2 of Chapter 3 results are presented in Table 4.7. Here the model is firstly trained with the entire Clotho dataset and then fine-tuning and layer-wise fine-tuning strategies are applied to the domain-specific data extracted from the Clotho dataset.

In cases where fine-tuning was absent (i.e., the model was trained from scratch), it was then evaluated using domain-specific data. The results presented in the table demonstrate a significant enhancement in SPIDEr scores for captioning related to vehicles and animals. These findings underscore the model's successful adaptation to domain-specific captioning tasks, signifying its proficiency in capturing the unique characteristics of the target domain.

For instance, when comparing the generic AAC model to the domain-adapted captioning model, we observe a notable shift in the generated captions. The generic model might

Table 4.7. *Domain adaptation results for vehicles and animals: comparison of fine-tuning and layer-wise fine-tuning using generic and domain-specific data.*

Setup	$CIDE_r$	SPICE	$SPIDE_r$	FENSE	SentenceBERT
Vehicles					
Fine-tuning	0.461	0.124	0.293	0.528	0.536
All attention layers	0.488	0.127	0.308	0.536	0.540
Encoder attention layers	0.460	0.122	0.291	0.528	0.536
Animals					
Fine-tuning	0.462	0.163	0.312	0.633	0.622
All attention layers	0.480	0.166	0.323	0.638	0.624
Encoder attention layers	0.450	0.163	0.306	0.632	0.621

produce captions like "a person is snoring in the background as a dog barks in the foreground" or "an engine is running at a steady pace." In contrast, the domain-adapted model generates captions such as "an animal is snoring while it is raining" and "the engine of a car is revving up and down" for the same audio segments. These results illustrate the discernible trend of captions becoming more coherent and domain-centric when domain adaptation is applied.

5. CONCLUSION AND FUTURE WORK

Automated audio captioning is a multi-modal translation task where a system takes in an audio signal as input and analyses it to generate a caption in natural language as its output. For training the system, we leveraged two widely available datasets: Clotho and AudioCaps.

As the baseline model, we introduced a sequence-to-sequence framework comprising a Transformer encoder and Transformer decoder based on the BART model. The input audio signal embeddings were extracted using the pre-trained Transformer network, Patchout Fast Spectrogram Transformer (PaSST), trained on the AudioSet dataset. Word tokenization was performed using subword tokenization, specifically byte-pair encoding. We conducted experiments with various configurations of the BART Transformer model, different tokenization methods, and different modules of the PaSST model.

Among the three PaSST modules, the *PaSST base2levelmel* module stood out, achieving a SPIDEr score of 0.278. Further improvements were observed by decreasing the BART encoder-decoder layers to $n = 4$, resulting in a SPIDEr score of 0.282. Additionally, by employing the byte-pair encoding tokenizer, the model's performance was enhanced, yielding a SPIDEr score of 0.283.

This thesis explores explicitly the adaptation of a generic audio captioning model into a domain-specific captioning model by leveraging knowledge transfer. Our approach involves initially training the model on large datasets and subsequently fine-tuning it using domain-specific training data. The model's performance is evaluated using three different settings such as training from scratch, fine-tuning, and layer-wise fine-tuning.

Fine-tuning involved training the model using both the source and target datasets without any layer freezing. On the other hand, layer-wise fine-tuning involved experimenting with various layer-freezing strategies within the transformer architecture. Notably, layer-wise fine-tuning consistently outperformed different approaches, showcasing substantial performance enhancements compared to training the model from scratch. The domain-specific model, under the layer-wise fine-tuning setting with all attention layers frozen, yielded remarkable SPIDEr scores of 0.308 for vehicles and 0.323 for animals. This approach not only improved performance but also deepened the model's comprehension, resulting in the generation of more precise and contextually appropriate captions for audio

samples within its specialized domain.

In the future, we aim to enhance the Clotho dataset by augmenting the diversity of both audio samples and captions. Moreover, we intend to expand the size of the dataset, as training with limited data can be challenging. Additionally, we have plans to elevate the overall quality of captions within the Clotho dataset. In addition to these explorations, we also intend to develop advanced model architectures that can harness the potential of multi-modal inputs for audio captioning tasks.

REFERENCES

[1] K. Drossos, S. Adavanne, and T. Virtanen, “Automated audio captioning with recurrent neural networks,” in *2017 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 2017, pp. 374–378.

[2] S. Zhao, P. Sharma, T. Levinboim, and R. Soricut, “Informative image captioning with external sources of information,” in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 6485–6494.

[3] L. Sun, B. Li, C. Yuan, Z. Zha, and W. Hu, “Multimodal semantic attention network for video captioning,” in *2019 IEEE International Conference on Multimedia and Expo (ICME)*, 2019, pp. 1300–1305.

[4] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, 2017.

[5] K. Drossos, S. Lipping, and T. Virtanen, “Clotho: An audio captioning dataset,” in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020.

[6] C. D. Kim, B. Kim, H. Lee, and G. Kim, “AudioCaps: Generating captions for audios in the wild,” in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 119–132.

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[8] W. Rawat and Z. Wang, “Deep convolutional neural networks for image classification: A comprehensive review,” *Neural Computation*, vol. 29, no. 9, pp. 2352–2449, 2017.

[9] R. Prabowo and M. Thelwall, “Sentiment analysis: A combined approach,” *Journal of Informetrics*, vol. 3, no. 2, pp. 143–157, 2009.

[10] A. K. Jain, M. N. Murty, and P. J. Flynn, “Data clustering,” *ACM Computing Surveys*, vol. 31, no. 3, 1999-09.

[11] S. Gavel, R. Charitha, P. Biswas, and A. S. Raghuvanshi, “A data fusion based data aggregation and sensing technique for fault detection in wireless sensor networks,” *Computing*, vol. 103, no. 11, pp. 2597–2618, Nov. 2021.

[12] N. Krishnaraj, M. Elhoseny, M. Thenmozhi, M. M. Selim, and K. Shankar, “Deep learning model for real-time image compression in internet of underwater things

(iout)," *Journal of Real-Time Image Processing*, vol. 17, no. 6, pp. 2097–2111, Dec. 2020.

[13] A. Goldwaser and M. Thielscher, "Deep reinforcement learning for general game playing," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 1701–1708, Apr. 2020.

[14] D. Isele, A. Nakhaei, and K. Fujimura, "Safe reinforcement learning on autonomous vehicles," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 1–6.

[15] A. Krogh, "What are artificial neural networks?" *Nature Biotechnology*, vol. 26, no. 2, pp. 195–197, Feb. 2008.

[16] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *International Journal of Engineering Applied Sciences and Technology*, vol. 04, pp. 310–316, May 2020.

[17] T. M. Mitchell, "The need for biases in learning generalizations," Department of Computer Science, Laboratory for Computer Science Research, 1980.

[18] J. Zou, Y. Han, and S.-S. So, "Overview of artificial neural networks," in *Artificial Neural Networks: Methods and Applications*, D. J. Livingstone, Ed. Totowa, NJ: Humana Press, 2009, pp. 14–22.

[19] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[20] Y. LeCun, B. Boser, J. S. Denker, *et al.*, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, Dec. 1989.

[21] Zhou, Y. T. and Chellappa, R., "Computation of optical flow using a neural network," in *IEEE 1988 International Conference on Neural Networks*, 1988, 71–78 vol.2.

[22] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, "1d convolutional neural networks and applications: A survey," *Mechanical Systems and Signal Processing*, vol. 151, p. 107 398, 2021.

[23] S. Abdoli, P. Cardinal, and A. Lameiras Koerich, "End-to-end environmental sound classification using a 1d convolutional neural network," *Expert Systems with Applications*, vol. 136, pp. 252–263, 2019.

[24] N. Zakaria and Y. M. Mohmad Hassim, "Improved vgg architecture in cnns for image classification," in *2022 IEEE International Conference on Artificial Intelligence in Engineering and Technology (IICAIET)*, 2022, pp. 1–4.

[25] J. Chai, H. Zeng, A. Li, and E. W. Ngai, "Deep learning in computer vision: A critical review of emerging techniques and application scenarios," *Machine Learning with Applications*, vol. 6, p. 100 134, 2021.

[26] N. Ono, N. Harada, Y. Kawaguchi, *et al.*, *Proceedings of the 5th Workshop on Detection and Classication of Acoustic Scenes and Events (DCASE 2020)*. Zenodo, Nov. 2020.

[27] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, “An image is worth 16x16 words: Transformers for image recognition at scale,” in *International Conference on Learning Representations*, 2021.

[28] K. Miyazaki, T. Komatsu, T. Hayashi, S. Watanabe, T. Toda, and K. Takeda, “Convolution-augmented transformer for semisupervised sound event detection,” in *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2020 Workshop (DCASE2020)*, 2020.

[29] X. Mei, Q. Huang, X. Liu, *et al.*, “An encoder-decoder based audio captioning system with transfer and reinforcement learning,” in *Proceedings of the 6th Detection and Classification of Acoustic Scenes and Events 2021 Workshop (DCASE2021)*, Barcelona, Spain, Nov. 2021, pp. 206–210.

[30] Y.-A. Wang and Y.-N. Chen, “What do position embeddings learn? an empirical study of pre-trained language model positional encoding,” in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Online: Association for Computational Linguistics, Nov. 2020, pp. 6840–6849.

[31] O. Press, N. Smith, and M. Lewis, “Train short, test long: Attention with linear biases enables input length extrapolation,” in *International Conference on Learning Representations*, 2022.

[32] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space", *International Conference on Learning Representations*, 2013.

[33] J. Pennington, R. Socher, and C. Manning, “GloVe: Global vectors for word representation,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543.

[34] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, “Enriching word vectors with subword information,” *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

[35] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in *Advances in Neural Information Processing Systems*, C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, Eds., vol. 26, Curran Associates, Inc., 2013.

[36] R. Sennrich, B. Haddow, and A. Birch, “Neural machine translation of rare words with subword units,” in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1715–1725.

[37] P. Gage, “A new algorithm for data compression,” *The C Users Journal archive*, vol. 12, pp. 23–38, 1994.

[38] T. Kudo, “Subword regularization: Improving neural network translation models with multiple subword candidates,” in *Proceedings of the 56th Annual Meeting of the*

Association for Computational Linguistics (Volume 1: Long Papers), Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 66–75.

[39] I. Provilkov, D. Emelianenko, and E. Voita, "BPE-dropout: Simple and effective subword regularization," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Online: Association for Computational Linguistics, Jul. 2020, pp. 1882–1892.

[40] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *CoRR*, vol. abs/1409.0473, 2014.

[41] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27, Curran Associates, Inc., 2014.

[42] Y. Wu, M. Schuster, Z. Chen, *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[43] X. Mei, X. Liu, M. D. Plumbley, and W. Wang, "Automated audio captioning: An overview of recent progress and new challenges," *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2022, no. 1, p. 26, Oct. 2022.

[44] S. Hershey, S. Chaudhuri, D. P. W. Ellis, *et al.*, "Cnn architectures for large-scale audio classification," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 131–135.

[45] J. F. Gemmeke, D. P. W. Ellis, D. Freedman, *et al.*, "Audio set: An ontology and human-labeled dataset for audio events," in *Proc. IEEE ICASSP 2017*, New Orleans, LA, 2017.

[46] K. Koutini, J. Schlüter, H. Eghbal-zadeh, and G. Widmer, "Efficient training of audio transformers with patchout," in *Interspeech 2022, 23rd Annual Conference of the International Speech Communication Association, Incheon, Korea, 18-22 September 2022*, H. Ko and J. H. L. Hansen, Eds., ISCA, 2022, pp. 2753–2757.

[47] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jegou, "Training data-efficient image transformers amp; distillation through attention," in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., ser. Proceedings of Machine Learning Research, vol. 139, PMLR, 18–24 Jul 2021, pp. 10 347–10 357.

[48] E. Fonseca, X. Favory, J. Pons, F. Font, and X. Serra, "FSD50K: An open dataset of human-labeled sound events," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 30, pp. 829–852, 2022.

[49] J. Turian, J. Shier, H. R. Khan, *et al.*, "Hear: Holistic evaluation of audio representations," in *NeurIPS 2021 Competitions and Demonstrations Track*, PMLR, 2022, pp. 125–145.

[50] M. Lewis, Y. Liu, N. Goyal, *et al.*, “BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension,” in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Jul. 2020, pp. 7871–7880.

[51] F. Font, G. Roma, and X. Serra, “Freesound technical demo,” in *Proceedings of the 21st ACM International Conference on Multimedia*, ser. MM ’13, Barcelona, Spain: Association for Computing Machinery, 2013, pp. 411–412.

[52] A. Paszke, S. Gross, F. Massa, *et al.*, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc., 2019.

[53] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” *arXiv preprint arXiv:1711.05101*, 2017.

[54] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “Bleu: A method for automatic evaluation of machine translation,” in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, Jul. 2002, pp. 311–318.

[55] S. Banerjee and A. Lavie, “METEOR: An automatic metric for MT evaluation with improved correlation with human judgments,” in *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, Ann Arbor, Michigan: Association for Computational Linguistics, Jun. 2005, pp. 65–72.

[56] C.-Y. Lin, “ROUGE: A package for automatic evaluation of summaries,” in *Text Summarization Branches Out*, Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81.

[57] R. Vedantam, C. L. Zitnick, and D. Parikh, “Cider: Consensus-based image description evaluation,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 4566–4575.

[58] P. Anderson, B. Fernando, M. Johnson, and S. Gould, “Spice: Semantic propositional image caption evaluation,” in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part V 14*, Springer, 2016, pp. 382–398.

[59] S. Liu, Z. Zhu, N. Ye, S. Guadarrama, and K. Murphy, “Improved image captioning via policy gradient optimization of spider,” in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 873–881.

[60] N. Reimers and I. Gurevych, “Sentence-bert: Sentence embeddings using siamese bert-networks,” in *Conference on Empirical Methods in Natural Language Processing*, 2019.

[61] Z. Zhou, Z. Zhang, X. Xu, Z. Xie, M. Wu, and K. Q. Zhu, “Can audio captions be evaluated with image caption metrics?” In *2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 981–985.