Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Telecommunication Engineering

Lightweight Parameter-Efficient Neural Network for Action Recognition on Edge Device

by

Tomáš Zámostný

A master thesis submitted to
the Faculty of Electrical Engineering, Czech Technical University in Prague

Prague, January 2024

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Zámostný Tomáš**	Personal ID number:	**466202**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Telecommunications Engineering**		
Study program:	**Electronics and Communications**		
Specialisation:	**Communications Networks and Internet**		

II. Master’s thesis details

Master’s thesis title in English:

Lightweight Parameter-efficient Neural Network for Action Recognition on Edge Device

Master’s thesis title in Czech:

Rozpoznání akce pomocí neuronové sítě optimalizované pro běh na edge zařízení

Guidelines:

This work aims to design and implement a neural network for action recognition on edge devices. Follow the following guidelines:
1) Perform research on state-of-the-art approaches to action recognition from video sequences
2) Design the network so that it can be trained on a limited dataset of kinetics and optimize it for real-time performance
3) Implement the solution on an edge device like NVIDIA Jetson Nano
4) Find the trade-off between performance and real-time latency

Bibliography / sources:

[1] CHEN, Shoufa, et al. Adaptformer: Adapting vision transformers for scalable visual recognition. Advances in Neural Information Processing Systems, 2022, 35: 16664-16678.
[2] MAAZ, Muhammad, et al. Edgenext: efficiently amalgamated cnn-transformer architecture for mobile vision applications. In: European Conference on Computer Vision. Cham: Springer Nature Switzerland, 2022. p. 3-20.

Name and workplace of master’s thesis supervisor:

doc. Ing. Stanislav Vítek, Ph.D. Department of Radioelectronics FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **22.09.2023** Deadline for master's thesis submission: **09.01.2024**

Assignment valid until: **16.02.2025**

doc. Ing. Stanislav Vítek, Ph.D.		prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Supervisor:

Assoc. Prof. Stanislav Vítek, Ph.D.
Department of Radio Engineering
Faculty of Electrical Engineering
Czech Technical University in Prague
Technická 2
166 27 Prague 6
Czech Republic

Co-Supervisor:

Prof. Yie-Tarng Chen, Ph.D.
Department of Electronic and Computer Engineering
National Taiwan University of Science and Technology
No.43 Keelung Rd, Da'an District
Taipei City 106335
Taiwan (R.O.C)

Abstract

AI becoming plays a crucial role in IoT, specifically in processing Human activity recognition (HAR). Development driven in this direction focuses on the automation of HAR. The emerging trend in the development of automation HAR systems. However, the effectiveness of these systems relies heavily on robust deep-learning algorithms and improved hardware technologies. Vision transformers show absolute dominance performance in computer vision. Moreover, enhancing performance by using methods such as parameter-efficient transfer-learning with large-scale pre-trained models. Unfortunately, these transformer-based models have the common drawback of having many parameters and a large memory footprint, causing them to be difficult to deploy on edge devices as lightweight convolutional neural networks. My challenge is to develop a model able HAR deployable on edge device in real time with low budget training. In the thesis, I propose the best trade-off between performance and number of parameters. The model emerged from the architecture of MobileViT developed by Apple researchers. MobileViT is powerful in extracting and learning spatial features. My set requirement is process video – I have to be able to capture temporal components. I redevelop the architecture by inserting ST-Adapters. Transforming the model to a downstream task, MobileViT is pretrained and frozen. Adapters help introduce image-related bias into the model and extract temporal features from the video stream. The proposed model is small enough to be deployed on edge devices. The model achieved promising performance with an accuracy of 74.94% on Kinetics-400, utilizing only $5.3M$ parameters. During training, 15% of the parameters were updated. Furthermore, the model demonstrates an inference speed of 16.45 frames per second on a Jetson Nano device, using 2.58GB RAM, with a prediction accuracy of 71.07%.

Keywords:
Vision Transformers, Edge Device, Surveillance, Action Recognition, Parameter-Efficient Transfer Learning

Acknowledgements

I would like to express my sincere gratitude to all those who have contributed to the completion of this master's thesis. This academic journey has been a challenging yet rewarding experience, and I am thankful for the support and encouragement I have received.

First and foremost, I extend my deepest appreciation to my Taiwanese supervisor, Prof. Yie-Tarng Chen, and Czech supervisor, Assoc. Prof. Stanislav Vítek for their invaluable guidance, insightful feedback, and unwavering support throughout the entire research process. Their expertise and commitment played a pivotal role in shaping the direction of this thesis.

I would like to express my gratitude to my family for their love, encouragement, and patience. Their unwavering support provided me with the strength and motivation needed to navigate the challenges of academic life.

Special thanks go to Patrik Patera for his fellowship and collaborative spirit. Our discussions and shared experiences greatly enhanced the intellectual environment in which this thesis was developed.

Lastly, I want to thank all those friends who provided emotional support and understanding during the ups and downs of this academic journey. Your friendship has been a source of inspiration. To everyone who has played a part in this endeavor, whether mentioned explicitly or not, I am truly grateful. This thesis would not have been possible without your support.

Thank you.

Declaration

I declare that I have obtained and presented all information in this document according to the academic Methodical guidelines on compliance with ethical principles. I have also fully cited and referenced any material and results not original to this work.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Rovněž jsem plně citoval a uvedl odkazy na veškeré materiály a výsledky, které nejsou původní v této práci.

In Taipei, January 2024

....
Tomáš Zámostný

Dedication

To my Dad

Contents

List of Figures

List of Tables

Abbreviations

Number Sets

$\mathbb{N}$	Natural numbers set
$\mathbb{Z}$	Integer numbers set
$\mathbb{S}_m$	Symmetric residue number set with a module of m
$\mathbb{Q}$	Rational numbers set
$\mathbb{R}$	Real numbers set

Common Mathematical Functions and Operators

10_2	Numbers' radices are designated with a subscript
$\mathbf{b}$	Vector $\mathbf{b}$
b_i	the i^{th} element of vector $\mathbf{b}$
$\|\|\mathbf{b}\|\|$	Norm of vector $\mathbf{b}$
$\dim \mathbf{b}$	Dimension of vector $\mathbf{b}$
$\mathbf{A}$	Matrix $\mathbf{A}$
$a_{i,j}$	Element of matrix $\mathbf{A}$ at the i^{th} row, and the j^{th} column
$\mathbf{A}^{-1}$	Inverse matrix to matrix $\mathbf{A}$
$\mathbf{A}^T$	Transposed matrix to matrix $\mathbf{A}$
$\|\|\mathbf{A}\|\|$	Norm of matrix $\mathbf{A}$

Terminology

IoT	Internet of Things
OS	Operating System
GPU	Graphics Processing Unit
TPU	Tensor Processing Unit
MLP	Multilayer Perceptron
CNN	Convolutional Neural Network
NN	Neural Network
NLP	Natural Language Processing
AI	Artificial Intelligence
DL	Deep Learning
DNN	Deep Neural Networks
PETL	Parameter Efficient Transfer Learning
GRU	Gated Recurrent Unit
HAR	Human Activity Recognition
DM	Depth Multiplier
FPS	Frames per Second
CNN	Convolutional Neural Networks
RNN	Recurrent Neural Networks
DCNN	Deep Convolutional Neural Networks
GPU	Graphics Processing Unit
WRN	Wide Residual Networks
FCN	Fully Convolutional Network
CRF	Conditional Random Field
FLOP	Floating-Point Operations
NPU	Neural Processing Unit
2D	Two-Dimensional
3D	Three-Dimensional
MV2	MobileNetV2
ViT	Vision Transformer
SDK	Software Development Kit

CHAPTER 1

Introduction

The rapid growth of deployments in surveillance systems and home security cameras has resulted in an exponential increase in data generation, posing significant challenges for traditional manual analysis methods. The need for automated recognition systems that can efficiently process large volumes of data and provide accurate outputs has become increasingly urgent.

However, existing intelligent systems face limitations due to their reliance on external computation resources, leading to high computational costs and latency issues [9]. To address these challenges, this thesis proposes integrating intelligent surveillance systems into edge devices, specifically utilizing the Jetson Nano platform. By moving computation towards the camera, I aim to reduce computational costs, enhance real-time performance, and ensure data privacy. My primary objective is to find a balance between model performance and computational efficiency in human activity recognition (HAR) tasks [10].

In recent years, machine learning has played a significant role in developing intelligent surveillance systems. In the last few years, Convolutional Neural Networks (CNNs) have been widely used for feature extraction and final classification. However, they have been recently replaced by Vision Transformers (ViT) [4] in various computer vision tasks. Vision Transformers (ViT) show great potential and outperform convolutional-based models. The key component of success lies in the multiple-head self-attention mechanism. Unfortunately, their computation cost and the number of parameters grow exponentially, which remains a major drawback.

To overcome these limitations, the thesis explores innovative methods that efficiently adapt large pre-trained models to my edge device architecture. I employ parameter-efficient transfer learning and ST-Adapter [11] modules to minimize computational costs while preserving the temporal information essential for human activity recognition. My approach allows us to train only a small fraction of the model's parameters, reducing the training budget while maintaining competitive performance.

1.1 Contributions

The main contributions of the thesis are summarized as follows:

1. Rebuilding and adapting original architecture for action recognition on video downstream task.
2. Achieving very low parametric computation cost, with only 15% of parameters required for training.
3. Reaching promising performance in Adapter Fine-Tuning with an accuracy of 74.94% and Full Fine-Tuning with an accuracy of 76.43%.
4. Proposing a model that recognizes unseen actions in real-time with excellent accuracy and latency.
5. Deploying a small-size model on the edge device Jetson Nano.
6. Achieving an inference speed of 16 frames per second on Jetson Nano with 2.6GB RAM usage.

1.2 Structure of the Thesis

The thesis is organized into five chapters as follows:

1. *Introduction*: This section describes the motivation and goals of our efforts. It also includes a list of contributions made by this thesis.
2. *Background*: This section introduces the reader to the theoretical background and surveys current state-of-the-art methods related to the thesis.
3. *Methodology*: Outline the research design in-depth and used techniques to investigate our goals.
4. *Experimental Results*: Present the findings of the thesis, followed by figures and tables.
5. *Conclusions*: This section summarises the findings of our research, proposes potential areas for further investigation, and concludes the thesis.

Chapter 2

Background

In this chapter will provide the reader with sufficient technical and theoretical background.

2.1 Edge Computing

The primary goal of edge computing is to bring computational capabilities and data storage closer to the periphery of a decentralized system. Edge computing focuses on achieving low latency and higher bandwidth and can process vast amounts of data.

Increasing the trend of producing edge and IoT devices is timely fashion. According to Cisco's projection, by the year 2023, mobile connectivity is anticipated to be accessible to in excess of 70 percent of the worldwide populace. Edge computing aims to leverage existing data processing capabilities and exploit the proximity of data in an embedded system to solve the bottleneck computation problem commonly found in distributed systems.

Edge computing has arisen as a novel paradigm with the objective of enhancing computational efficiency and data storage by distributing processing capabilities in closer proximity to the origin of data. This approach deviates from the traditional practice of transmitting data to a centralized server for analysis. The growing ubiquity of intelligent devices and the surge in data generation at the periphery of the network have created opportunities for the efficient utilization of edge computing in data gathering and processing. While edge computing offers lower latency, better privacy, and mobility benefits, heterogeneity, and resource management challenges persist.

Reduced latency is a crucial advantage of edge computing because it eliminates the need to transfer data over long distances, resulting in faster processing and response times. By storing and processing data closer to its origin, edge computing minimizes delays associated with network congestion and improves the overall user experience. In addition, privacy concerns are addressed by edge computing because it reduces reliance on uploading data, thereby reducing data traffic and the associated risk of data hacking. However, privacy issues remain a persistent concern, especially in the context of Internet of Things (IoT) devices and networks.

Integrating different devices and components with different protocols is a significant challenge in edge computing. Incompatible protocols and different hardware configurations can prevent seamless interactions and communication between devices, hinder data processing efficiency, and introduce complexities during system integration. Addressing heterogeneity requires standardized protocols, compatible hardware, and robust interoperability mechanisms to ensure seamless data exchange and efficient processing. Another challenge in edge computing is to manage device mobility effectively. With the proliferation of edge devices that can move frequently, managing multiple devices and their relocation poses significant difficulties. The dynamic nature of device movement requires effective resource management strategies for optimal allocation of computing resources and seamless transition of processing tasks between devices. Streamlining resource management in mobile edge computing environments is essential to ensure continuous processing and uninterrupted services.

Through the relocation of computational and storage resources in proximity to the data origin, edge computing offers notable advantages, including reduced latency and enhanced data privacy. However, challenges arising from heterogeneity and mobility need to be addressed to exploit the potential of edge computing fully. Simplified interactions between devices, standardized protocols, and efficient resource management are essential for seamless and efficient data processing in a mobile edge computing environment. As edge computing evolves, addressing these challenges will be imperative to realize its full potential in optimizing computational efficiency and revolutionizing data analytics [12].

Deep Learning on Cloud

- Higher computational power (GPU, TPU).
- Flexible and scalable in terms of computing resources.
- Millions – Billions of parameter models.
- Usually used for training.
- Require stable power supply.
- Larger datasets and more complex models.
- High-speed internet connection (xGB/s).
- TFLOPs.
- Stationary.

- High initial and operating costs.

Deep Learning on Edge Device

- Limited computational power (NPU).
- No possibility of upgrading computing resources.
- Millions of parameter models.
- Usually used for inference.
- Battery-driven power supply.
- Small datasets and more complex models.
- Low-speed internet connection (xMB/s).
- Can operate offline without internet connectivity.
- Under GFLOPs.
- Data storage up to 128GB.
- Mobility.
- Cheaper initial and operating costs.
- Allows real-time processing and shorter response time.
- Better privacy, the device is located on the user site.

One key advantage of utilizing the cloud for deep learning is its significantly higher computational power. By leveraging the capabilities of GPUs and TPUs in the cloud, we can achieve faster results and more accurate models. On Server based solution we can easily extend computtation power by replacing outdated GPU/CPU units with newer one. Deep learning model often consists of millions to billions of parameters. That is the reason why we using cloud solution for these kind of task. One of the major demands is a stable power supply network with additional backup against power cuts and ensures uninterrupted computing and prevents any data loss during training. Usually cloud is connected to the internet network thought the optic fiber cable reaching speed over 10 GB per second. Large-scale data transfer between the cloud and local machines needs to be seamless, ensuring quick access to data for training and evaluation. Futermore we can enable training in highly complex models which also put requriment on data storage and memory. For example, If we use comprehensive dataset, we need Terabyte of memory for it.
For evaluating the performance of enumerous power of cloud based solution we commonly use a metric called Floating Point Operations per Second (FLOPs). Clouds are capable

to provide at least Trillions of these operation. This metric shows a superiority of clouds capabilities. Cloud-based infrastructure often supports high TFLOP capabilities.

Whilst cloud-based solutions provide numerous of advantages, it is important take in account a pricy initial cost following with operating cost. Also clouds require a special environment of location. Deep learning on edge devices has gained notable attention in recent years. In comparison to edge device has a limited computation resourses. Usually we could talk around less than giga Floating Point Operations per Second (GLOPs). Edge device usually consists Neural Procesing unit (NPU) which is specifically designed to solve operations for deep learning tasks. Edge device also provide just a certain amound of data storage around 64 GB. Which is necessary to store model locally on device. Local storage capacity also determines a capability of how many parameters can model consist. Models deployed on edge device often have a millions of parameters and used for interference running model.

Batteries, necessitating energy-efficient algorithms, typically power edge devices. One of the key demand for edge device is low power consumption. Furthermore, edge devices may operate in areas with low-speed internet connections, often limited to a few megabytes per second (MB/s). Consequently, these devices prioritize offline operation and can function without relying on an internet connection. When it comes to cost, edge devices have several advantages. Firstly, initial cost are not exceed a 100 USD for device, and secondly, operating cost is negligible in comparison with cloud based solution. The edge device usually have only MB to a few GB allocated RAM memory. It is necessary to optimize the model as much as possible.

2.2 Nvidia Jetson Nano

The Jetson Nano Developer Kit 2.1 is a highly capable computer that incorporates artificial intelligence. With the aim of providing convenience, this developer kit seamlessly connects with standard peripherals, including those from Raspberry Pi and Adafruit. This ensures effortless integration into your existing setup. The device is equipped with a custom Linux operating system alongside the latest Nvidia Jetpack SDK for a comprehensive and user-friendly development environment [13].

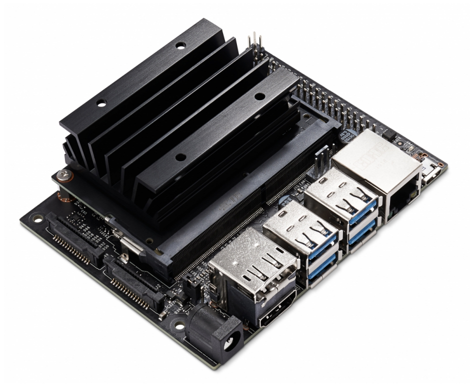


Figure 2.1: Jetson Nano Board Developer Kit [1].

The Jetson Nano Developer Kit supports popular artificial intelligence frameworks like TensorFlow, PyTorch, Caffe, and MXNet to facilitate the swift development and deployment of AI applications. This empowers users to harness the capabilities of these frameworks for rapid AI project creation, such as the Jet Bot – an open-source deep-learning robot that capitalizes on the features of the Jetson Nano Developer Kit [14].

Regarding hardware, the developer kit encompasses a 40-pin expansion header compatible with the newly incorporated Jetson GPIO Python library. This library simplifies the interface with external devices. Additionally, the kit features a micro-USB connector that enables power supply using a 5V and 2A power source, gigabit Ethernet, USB 3 Type-A connectors, HDMI and DP display connectors, and a cylindrical connector that can provide up to 4A of power. The camera connector supports IMX 219 camera modules. By default, the device has a passive radiator that can handle a 10-watt module power output at an ambient temperature of 25 degrees Celsius. A 40mm PWM fan can be attached if additional cooling is required [13, 15]. The following Table 2.1 presents a comparative analysis among various edge devices.

Table 2.1: Comparison between other edge devices [7].

	Raspberry Pi 4	**Jetson Nano**	**Jetson TX2**
Performance	13.5 GFLOPS	472 GFLOPS	1.3 TFLOPS
CPU	Cortex-A72@1.5 GHz	Cortex-A57@1.42 GHz	Cortex-A57@2GHz+NVIDIA Denver2@2GHz
GPU	Broadcom Core VI	NVIDIA Maxwell 128	NVIDIA Pascal 256
Memory	8GB LPDDR4	4GB LPDDR4	8GB LPDDR4
Networking	Gigabit Ethernet / WiFi 802.11ac	Gigabit Ethernet / M.2 Key E	Gigabit Ethernet, 802.11ac WLAN
Display	2x micro-HDMI (up to 4Kp60)	HDMI 2.0 and DP 1.4	2x DSI,2x DP 1.2 /HDMI 2/DP 1.4
USB	2x USB 3.0, 2x USB 2.0	4x USB 3.0, USB 2.0 Micro-B	USB 3.0 + USB 2.0
Other	40-pin GPIO	40-pin GPIO	40-pin GPIO
Encoder	H264(1080p30)	H.264/H.265(4Kp30)	H.264/H.265(4Kp60)
Decoder	H.265(4Kp60), H.264(1080p60)	H.264/H.265 (4Kp60, 2x 4Kp30)	H.264/H.265 (4Kp60)
Storage	Micro-SD	16 GB eMMC	32GB eMMC
Power	2.56W-7.30W	5W-10W	7.5W-15W

2.3 Neural Networks

Artificial neural networks (ANNs) are intricately designed to mimic the operational dynamics of the human cerebral cortex, comprising a multifaceted web of interconnected neurons mediated by synaptic connections. ANNs play a crucial role in the realm of artificial intelligence, particularly in statistical tasks such as classification and regression. Remarkably, by 2012, ANNs had demonstrated exceptional performance in these tasks, occasionally exceeding human capabilities in visual pattern recognition [16].

In the context of supervised learning, and more specifically within the domain of deep learning, the fundamental aim is to approximate the latent function that maps input data to desired outputs. This is achieved through a hierarchical arrangement of layers, including the input layer that transmits raw data to the concealed layers, where it undergoes a multi-stage learning process, culminating in the output layer that generates predictive outputs. These outputs vary depending on the specific statistical challenge being tackled. To optimize the precision of the model, a loss function is utilized to quantify the dissimilarity between forecasted and actual values, triggering adjustments in layer weights according to the prescribed loss function. This iterative weight modification serves to enhance the model's accuracy, ultimately facilitating its ability to generalize unseen data [16, 17].

2.4 Machine Learning

One of the most significant inventions in 20s century was a Turin machine. The history of machine learning is dated back to the early 1950s. Alan Turing, British scientist who break enigma code during World War II. Later he created a Turing test which is test could prove if the machine (computer) can think. Following years start a competition to achieve this goal. The biggest breakthrough came in machine learning in 2006. Researchers discovered a the backpropagation algorithm. It was one of the most significant achievements in machine learning. The term Machine learning is often interchanged with Artificial intelligence of deep learning. The general public often use term Artificial intelligence to describe a solution which have some sort of intelligence and could learn something meaningful [18].

Following Venn diagram 2.2 describing relationship between specific terms. As we can see machine learning is show as part of subset of artificial intelligence. If we want to compare machine learning with traditional algorithm, we can define a few key differences. In machine learning we trying to achieve higher dimensional complexity to solve a complex task. Furthermore for this we usually using big datasets.

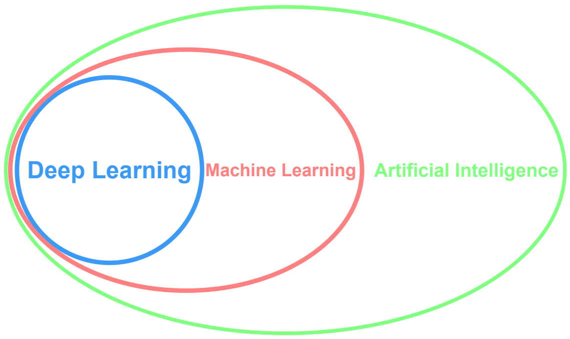


Figure 2.2: Venn diagram.

2.5 Deep Learning

The first scientists who laid the foundations of modern deep learning can be considered Yann LeCun a Yoshia Bengio. In 1998, these researchers created an architecture called LeNet [19], which was trained on the MNIST dataset. Deep learning belongs to a subset of machine learning. This is the term for an architecture that contains more than three layers. One of the goals of deep learning was to simulate the functioning of the human brain. When deepening the site or adding more hidden layers, we usually increase the accuracy of the results. In general, deep learning is used to solve complex tasks. One of the main characteristics of deep learning is that it can automatically extract hierarchical features and is, therefore suitable for tasks with unstructured data and complex patterns [20, 21, 22].

2.6 A Single Neuron

In 1943, researchers McCulloch and Pitts introduced a mathematical paradigm for understanding the functioning of neurons, which laid the groundwork for the development of artificial neural networks [23]. At the core of these networks lies the single neuron, alternatively referred to as a unit rather than a node. The neuron's operation is governed by a straightforward set of principles. First, it necessitates inputs endowed with suitable weights. Subsequently, a bias term is incorporated into the neuron, furnishing it with an initial impulse that precludes the output from being zero at the outset. By definition, a neuron is a nonlinear function. Mathematically speaking, the operation of a neuron can be illustrated by the following equation:

$$y = \sum_{i=1}^{n} w_i x_i + b_0 \ . \tag{2.1}$$

Mathematical notation is a linear combination of weights and input and final bias assignment. Where y denotes the output, x represents the vector of inputs, w symbolizes the weight matrix, and b stands for the bias term. The first step of the transformation is to multiply the inputs by a weighting parameter w that simulates the synaptic strength in biological networks. All weighted inputs are then summed to obtain the total input.

While individual perceptrons possess limited capacity to address intricate problems due to their restriction to linear decision boundaries, they constitute the foundation for more sophisticated neural network architectures. Multilayer perceptrons (MLPs) and deep neural networks, for instance, are capable of handling nonlinear challenges and find application in various machine learning domains [23].

2.6.1 Perceptron

Perceptron is a basic implementation of artificial neural networks inspired by the human brain [24]. In this paper, we will look at an example of classification to understand how perceptron works. The underlying equation governing the functioning of a perceptron can be expressed as follows:

The non-linear function $f(x)$ is defined as:

$$f(X) = \begin{cases} 1, & \text{if } W \cdot X + b > 0 \\ 0, & \text{otherwise} \end{cases} ,$$

$$X = \{x_1, x_2 \dots x_n\} \qquad X\ is\ feature\ input\ vector, \tag{2.2}$$

$$W = \{w_1, w_2 \dots w_n\} \qquad W\ are\ the\ weights, \tag{2.3}$$

$$\sum_{i=1}^{n} (w_i x_i) + b\ . \tag{2.4}$$

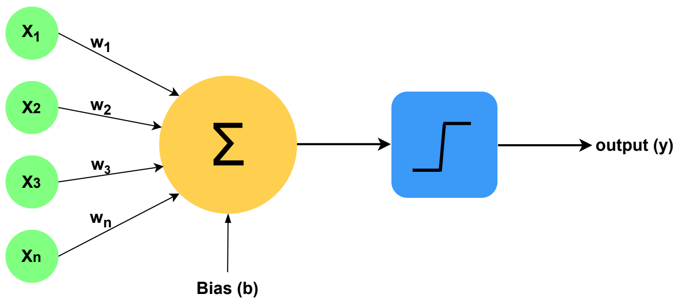


Figure 2.3: Single Layer Perceptron

In our case, we will consider the sigmoid function as the non-linear activation function for light and simplicity. The sigmoid activation function [25] produces an output between 0 and 1 and is ideal for binary classification [26].

Sigmoid activation function $f(x)$ is given by:

$$f(x) = \frac{1}{1 + e^{-x}}\ . \tag{2.5}$$

2.6.2 Multi-layer Perceptron

The defining characteristic of MLPs is the presence of one or more intermediate layers between the input and output layers, comprising perceptrons arranged in a hierarchical structure. The depicted Figure 2.4 illustrates an MLP with a single hidden layer, showcasing the network's ability to learn and solve nonlinear tasks. However, the higher complexity of the model also means that the weight update procedure becomes more complicated and time-consuming. An efficient weight-updating technique is called the back-propagation algorithm [27].

Each perceptron within a layer is connected to every perceptron in the subsequent layer via weights w_i, establishing a complex of interactions that enable the network to capture intricate patterns in the data. The computation performed by each perceptron involves the summation of weighted inputs, followed by passing the result through an activation function, typically a sigmoid function. This process allows the network to introduce non-linearity and approximate complex decision boundaries. For example, the output layer contains the same number of perceptrons as classes, and the perceptron with the highest activation is designated as the classification of the input sample. This arrangement enables the MLP to perform recognition tasks accurately.

The MLP can be categorized as either a fully connected or a partially connected network. A fully connected network depicted in Figure 2.4 is distinguished by having all nodes interconnected between its layers, in contrast to a partially connected network where there are no interconnections between nodes from the preceding layer to the subsequent layer. While partially connected networks conserve weight memory, research studies have consistently demonstrated that fully connected models achieve superior performance. Moreover, partial connectivity often leads to increased difficulty in training the network [28].

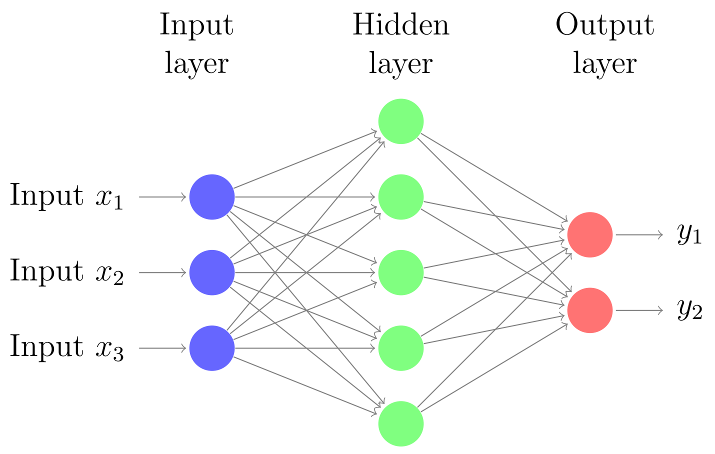


Figure 2.4: Multi Layer Neural Network Architecture.

2.7 Deep Learning Frameworks

2.7.1 TensorFlow

TensorFlow is an open-source deep learning framework. It was developed by Google in 2015. It is primarily used for numerical calculations. It is considered one of the cornerstones of modern AI. It is considered by many to be a game-changer in machine learning. It provides very diverse possibilities for developers with a user-friendly interface. It is optimized for both CPU and GPU computations, and it is possible to distribute computations to multiple devices at the same time [29].

2.7.2 Pytorch

Facebook developed the Pytorch [30] framework to replace the Numpy [31] math library. It has a large number of characterization properties similar to TensorFlow. It can also efficiently distribute calculations to the GPU. One of the very efficient functions is the gradient calculation Autograd. Autograd can calculate the models backpropagation without manual implementation. Currently, Pytorch is one of the most used frameworks for deep learning research and development. Based on data from papers with code, the majority of scientific implementations are based on this framework [32].

2.8 Convolutional Neural Network

In CNN's [33] early beginnings in the computer vision industry, it was revolutionary and achieved excellent results. Convolutional neural networks (CNNs) are one of the significant building blocks of modern deep learning.

Shallow neural networks have only one input layer, one hidden layer, and one output layer, while deep learning neural networks have multiple layers. Each layer in a deep learning neural network acts as a training layer for the following layers. In summary, the more hidden layers a model has, the better it can extract and cluster information from previous layers. Convolutional networks, which are composed of convolutional layers and use sliding windows, are commonly used in computer vision for their ability to extract spatial information from images. While CNNs were specifically designed for image recognition, they remain a state-of-the-art method for computer vision tasks. Convolutional neural networks (CNNs) have been in existence since the 1960s. However, the first significant breakthrough was the LeNet [19] architecture, followed by AlexNet [34] in 2012. The AlexNet architecture won a competition on the ImageNet dataset and was considered state-of-the-art. The subsequent Table 2.2 shows a comparison between various types of most influential CNN architectures.

Table 2.2: Comparison of various types CNNs implementation [8]

Architecture	Size (M)	Top-5 error (%)	No. Layers	Parameters	Year
LeNet [19]	-	-	8	60 k	1998
AlexNet [34]	238	18.00	8	6 M	2012
VGGNet [35]	540	9.33	16	138 M	2014
GoogleNet [36]	40	10.04	22	4 M	2013
ResNet-50 [37]	100	6.71	50	26 M	2015
ResNet-152 [37]	235	3.57	152	60 M	2017

The main difference between CNNs and classical neural networks is that a kernel is trained instead of connecting an input neuron to a hidden layer neuron. This kernel is moved in small steps, as shown in the Figure 2.5. The principle behind this technique is based on the mathematical operation of convolution [38].

It involves learning the grid pixel by pixel, mainly from the surrounding points. This principle, known as spatial, is described in detail in the following subsection.

The depth of the location can be indicated based on the number of kernels used. By using these convolutional layers, the model or the number of parameters can be reduced. Each layer in the CNN produces a feature map. Similar to regular neural networks, CNN feature maps also go through activation functions that add nonlinear components to the network [39, 40].The Figure 2.5 illustrates an example of a convolutional neural network and its corresponding feature maps.

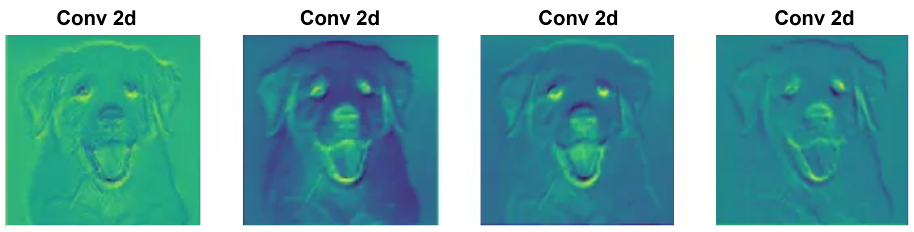


Figure 2.5: Visualization of feature maps in a Convolutional Neural Network (CNN) with a concealed layer [2].

2.8.1 Padding and Stride

Convolutional neural networks (CNNs) rely on two critical parameters that determine the size and behaviour of element maps during convolution. Accurate setting of these parameters is essential for an efficient CNN architecture.

Padding is a specialised CNN technique that adds extra pixels, typically zeroes, around the input image or element map before convolution is applied. Padding is mainly used to regulate the spatial dimensions of object maps. Padding is used to ensure that convolution operations are performed on the edges of the input, preventing information loss if the convolution were limited to the central pixels.

The stride value determines how much the convolution kernel (filter) moves across the input or feature map during each step of the convolution operation. A larger value results in a smaller output size, while a smaller value results in a larger output size. Decreasing the step size in a convolutional network increases the amount of information and spatial dimensions. This, in turn, raises the computational cost in these layers.

2.8.2 Pooling

The pooling method is typically used to down-sample or sub-sample an image in order to reduce its spatial dimension. In convolutional networks, these layers are often placed after the convolutional layer to preserve only the relevant information in the image, such as sharp features of a face. There are two types 2.6 of pooling that are frequently used in deep learning: max pooling and average pooling. Max pooling is the most frequently used method. Pooling is usually applied in CNNs after several layers. Like the previous method, pooling is used to reduce computational complexity. This is particularly important in computer vision and when training on video data.

2.8.2.1 Max Pooling

Max Pooling selects the highest value in each related image area. This technique helps to preserve the most salient features within each local region while discarding less relevant information.

2.8.2.2 Average Pooling

Similar to Max Pooling, Average Pooling defines a local region and calculates the average value of that region.

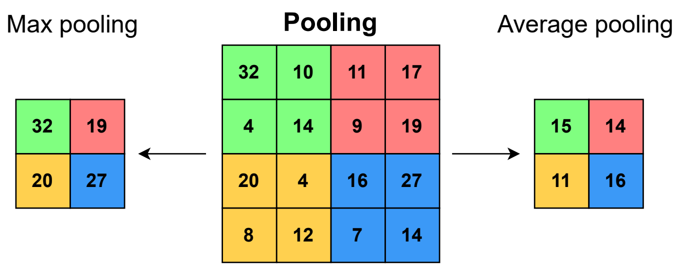


Figure 2.6: Illustration of pooling mechanisms: max pooling and average pooling.

2.9 Transformers

The architecture overcomes the basic limitation of sequential computation, which constrained the use of recurrent neural networks (RNN) in previous models that utilized GPUs. The paper presents a novel deep learning architecture called Transformer, which does not rely on sequential processing and exhibits significant potential for parallel computation. Figure 2.7 displays the encoder-decoder framework adopted in the models.

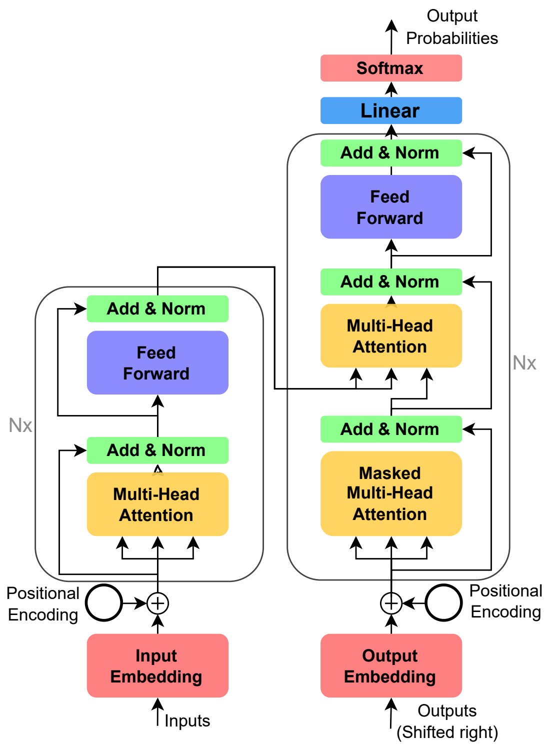


Figure 2.7: Visualization of a Transformer model architecture [3].

The Transformer architecture, shown in Figure 2.7, consists of self-attention and point-wise fully connected layers in both the encoder and decoder. The left side depicts the initial stage, converting inputs into input embeddings. Originally designed for translation, the model converts input words into numerical representations through the embedding layer, producing 512-dimensional vectors for each word in the sentence. These vectors are then passed to the next stage of the model.

Advancing natural language processing involves incorporating positional encoding, which informs the encoder of the location of words in a sentence. Recently, researchers proposed a novel approach to positional encoding that involves generating a d-dimensional vector that shares specific position information for each element of the sentence. This technique provides more details about the position of words, potentially leading to improved model performance. Since word embeddings have 512 dimensions, the coding dimension of the current model will also be 512, and the positional coding vector will also have 512 dimensions. The researchers proposed a sinusoidal function to solve this problem. The position coding vector contains sine and cosine pairs for each frequency w_k.

The next component of the encoder is a multi-headed self-attention mechanism that helps computers to understand finer sentence details. The paper encodes a word at a particular location, and the score determines the amount of attention required in other parts of the input sentence.

As the dimension in the paper was 64, each score value was divided by 8 before undergoing the softmax operation. The scores are normalized using Softmax, ensuring that all scores are positive and add up to 1. The resulting Softmax score determines the degree to which each word is expressed in its position, with the word at this specific position holding the highest Softmax score. The next step involves using value vectors, which are then applied by multiplying each value vector with its corresponding Softmax score. The aim is to maintain the importance of relevant words and reduce the significance of others by multiplying them with small values. In the final step of self-attention, the value vectors are added together to produce the output of the self-attention layer for the initial word. This resulting vector can then be passed on to the feed-forward neural network.

The next step is multi-head attention. Instead of using a single attention function with d-dimensional keys, values, and queries, the authors found it beneficial to project the queries, keys, and values h times using separate learned linear projections. In this study, 8 attention layers or heads were employed. Each head maintains distinct weight matrices for query, key, and value, leading to separate Q, K, and V matrices. To obtain these matrices, multiply the input embedding X by W_Q, W_K, and W_V. The attention function is then performed simultaneously on each of these projected versions, producing the final output values [3].

The attention block is followed by an add and norm layer. In this layer, the output vector of the attention block, which was just calculated, is summed with the input embedding vector retrieved in the first step. The resulting aggregate is then subjected to layer normalization. This normalization technique reduces training time, prevents bias towards higher value features, and restricts weights to a specific range. Overall, it is not recommended to train a model using gradient descent with non-normalized features. Additionally, both the encoder and decoder utilize fully connected feed-forward networks in each layer, which are identical and applied individually to each position. Along with attention sub-layers, this architecture guarantees optimal performance. This component consists of two linear transformations with a ReLU activation in between. The main aim is to enhance the output of a single attention layer to smoothly integrate it as input for the next attention layer.

The decoder generates text sequences using sub-layers similar to the encoder, including multi-headed attention, feed-forward, and normalization layers. Each attention layer performs a specific function. The decoder is autoregressive, taking the previous output and encoder outputs as inputs. It stops when an end token is produced.The decoding process involves several stages. First, the input is embedded and positionally encoded. The resulting embeddings are fed into the first multi-headed attention layer, which computes the attention scores for the decoder input. Unlike other attention layers, this one operates differently due to the decoder's autoregressive nature, necessitating the exclusion of future tokens in the attention computation.

In the decoding process, a unique masking technique is used to prevent the computation of attention scores on upcoming words. A look-ahead mask is applied before computing the softmax and scaling the scores, which fills the top right triangle of the attention scores with negative infinity. This masking process is the only difference in calculating attention scores in the initial multi-headed attention layer. Multiple heads are used to apply masks, and the masked outputs are then concatenated and processed by a linear layer. The result is a masked output vector that guides the model in addressing the decoder's inputs.

The secondary multi-headed attention layer utilizes the encoder's output as queries for its keys, while the encoder's output serves as values. This allows the decoder to selectively focus on relevant input. The output is then processed by a point-wise feed-forward layer and a final linear layer with a classifier, followed by a softmax layer that generates probability scores for each class. The predicted word is the index with the highest probability score. The decoder can be stacked with multiple layers, each taking inputs from the encoder and preceding layers, allowing the model to extract and combine various attention patterns, potentially improving its predictive ability [3].

Transformers utilize the attention mechanism's strength to make more accurate predictions. Although recurrent networks aim to accomplish similar tasks, transformers are typically more effective because they do not suffer from short-term memory, making them particularly useful for encoding or generating extended sequences [3].

2.9.1 Vision Transformers

The Vision Transformers operates similarly to the previously mentioned transformers. The distinction lies in data preprocessing before feeding it into the Vision Transformer. It processes image patches, and then the Reminder Transformer operates similarly. Figure 2.8 depicts the original architecture of ViT [4].

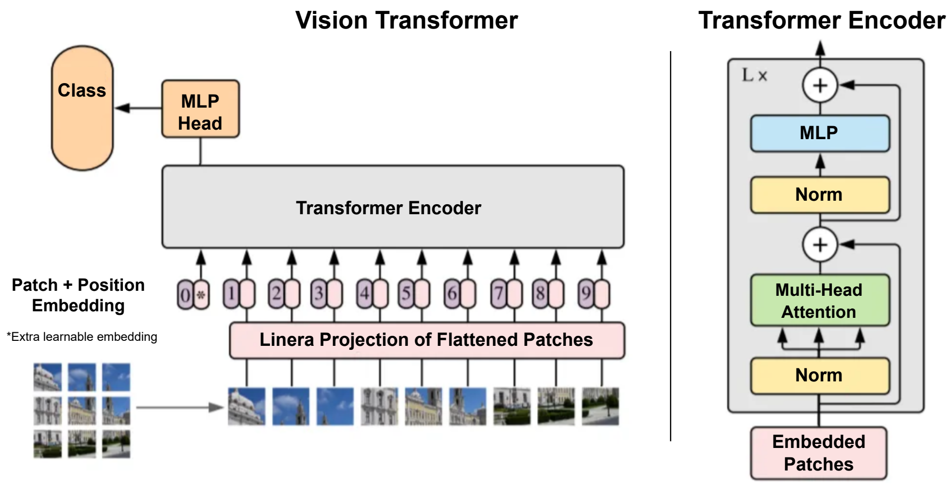


Figure 2.8: Vision Transformer architecture [4].

The first task for Vision Transformers is to divide an image into image patches illustrated on Figure 2.9. The architecture divides a 224 × 224 pixel image into image patches, where each patch is 14 × 14 pixels, resulting in a total of 256 of these patches. The benefit of a single attention layer requires a much more manageable 9.8 million comparisons. A linear projection layer is used to map the image patch arrays into image patch vectors by mapping these patches to the patch embeddings.

The next stage involves a learnable embedding, also known as a class token. This idea comes from BERT and has two unique features that make CLS special. First, it does not represent a real word. Therefore, the improved version is: Second, it is input into the classification head, which is used as a part of the pre-training process. This text already adheres to the principles or lacks context. Essentially, embedding is a general representation of the full sentence into a single token embedding because it helps the model make a good prediction.

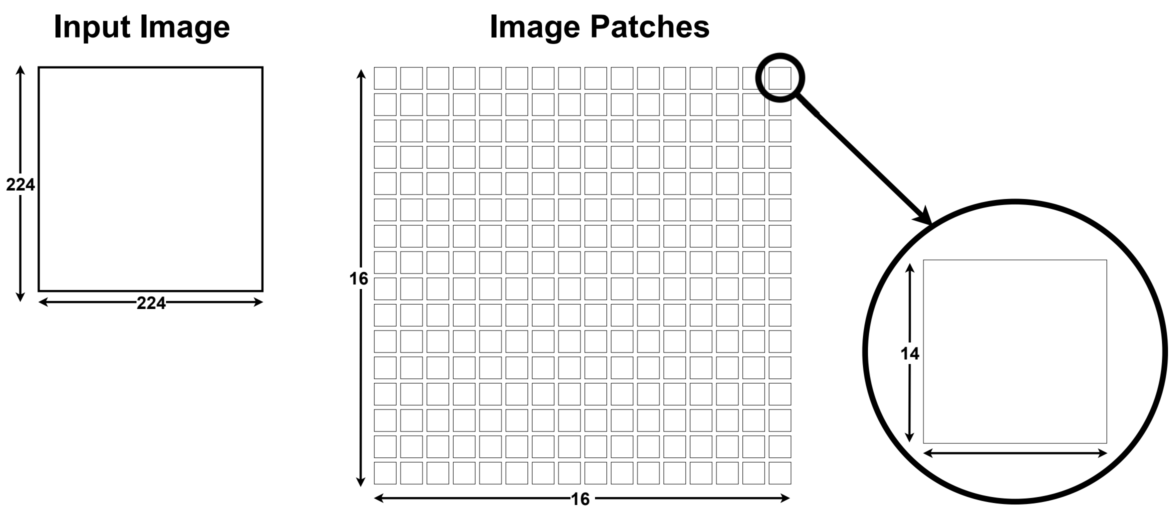


Figure 2.9: Schematic representation of the Vision Transformer pipeline: the input image is partitioned into 16 x 16 patches during the processing sequence.

The final step requires positional embeddings. Positional embedding is a crucial component of Vision Transformers (ViT). It helps the model understand the relative position of different regions in the image by providing positional information. Fixed-size vectors encode spatial information for each patch in positional embedding. These embedding vectors are typically acquired during the training process and are represented by sine and cosine functions. Spatial information assists the Vision Transformers (ViT) model in capturing the global context and dependencies between different image regions [4].

2.10 Computer Vision - Action Recognition

Computer vision is a wide-ranging discipline, with action recognition being one of its subfields. This field deals with the identification of human actions in video sequences. Additionally, there is the more computationally demanding area of action recognition, where we try to determine both the action and its location in the image. Action recognition is a crucial component of advanced observation systems and sports analysis. It is swiftly becoming a crucial component and performs a significant function in the sphere of computer vision. As advancements in technology within this field progress, it ushers in thrilling prospects and practical implications.

2.11 Datasets

2.11.1 ImageNet-1K

ImageNet-1k is a distinctive dataset comprising videos and images used for image recognition. It incorporates a total of 1.2 million images divided into 1000 classes [41].

2.11.2 Kinetics-400

Kinetics-400 is a dataset that is commonly employed for action recognition tasks. The dataset contains of 400 distinct actions and a total of $306,245$ videos that are hosted on YouTube. Researchers frequently use this dataset to evaluate their state-of-the-art work against benchmark standards. The total disk space occupied by the dataset amounts to approximately 443 GB [42]. A detailed list of all classes can be found in the Appendix section A.

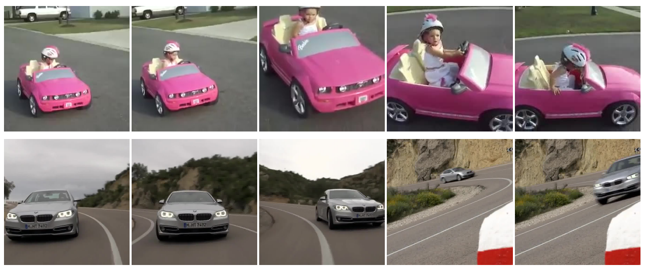


Figure 2.10: Kinetics-400 - class "Car Driving".

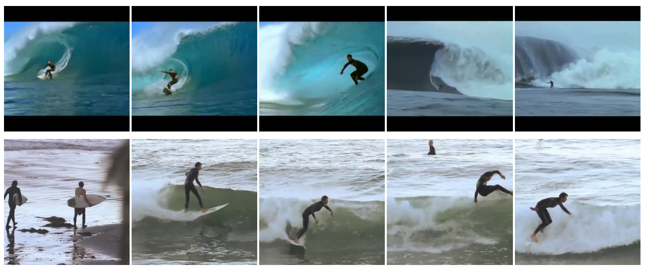


Figure 2.11: Kinetics-400 - class "Surfing".

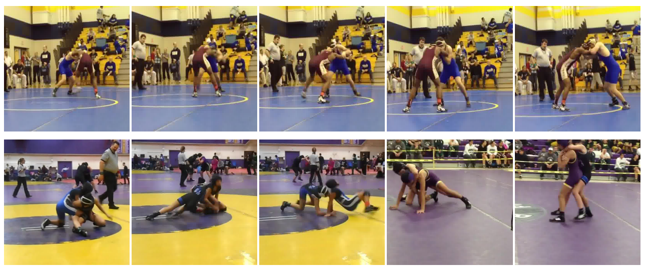


Figure 2.12: Kinetics-400 - class "Wrestling".

Chapter 3

Methodology

3.1 Goal

The main research objective of the thesis is to design a lightweight model for human action recognition running in a real-time setting on an edge device. It emerged a challenge to find the optimal trade-off between performance, efficiency, and computational cost.

3.2 Search Methodology

In order to conduct a comprehensive literature review, we selected a search platform that offered a wide range of academic publications. The majority of the searched articles were retrieved from Google Scholar [43] and IEEE Xplore Web [44]. To acquire the necessary knowledge, We thoroughly examined between 40 to 50 academic papers, with a focus on articles that presented cutting-edge approaches in the field of computer vision. The largest amount of valuable information was found in academic papers and conference publications, which provided us with the required insights into the latest developments in the domain.

To ensure a thorough search, We employed a strategic combination of keywords, including "PETL", "Lightweight", "ViT", "MobileViT", "Edge Device", "Action Recognition", "Real-time Edge Device" and "Computer Vision on Edge". This approach allowed us to uncover all relevant articles and filter out irrelevant content. The selected papers provided a rich source of knowledge during the thesis writing process.

During the review process, it was imperative to assess the eligibility of each article based on specific criteria. The examination process placed particular emphasis on the abstract, methodology, architecture, and conclusions, as well as the contributions made by each study. This rigorous evaluation ensured that only the most pertinent and high-quality articles were included in the literature review.

3.3 Enhancing CNN Capabilities

Creating a lightweight model has always been a significant challenge in computer vision. Until recently, depthwise separable convolutional methods have been the predominant approach. These methods successfully reduce the number of parameters and computation costs (FLOPS) required for training and inference by performing depthwise convolution in a regular $2D$ or $3D$ convolutional network. $3D$ depthwise convolutional networks are particularly effective at extracting spatial-temporal features while using minimal parameters and FLOPS.

In computer vision tasks, Vision Transformers (VIT) have demonstrated superiority over conventional CNN architectures in terms of accuracy. VIT uses a self-attention mechanism to effectively capture distant semantic relevance in images. Recent studies [45, 46] have explored the synergistic integration of computational CNN and VIT. The findings indicate that convolutional networks assist VIT in capturing local spatial information, complementing its limitations. The MobileViT [47] architecture was chosen as it combines the strengths of CNN and VIT, resulting in an improved action recognition structure.

The hybrid architecture of MobileViT, prior to employing self-attention, aggregates features from equally spaced pixel positions into the same group. Notably, patches within attention heads are not equally divided in the H and W dimensions. In addition, the T dimension enables the model to capture temporal information and sequences.

3.4 Feature Extraction Approach

Human activities are a sequence of movements where both spatial and temporal aspects are equally important. Temporal features are responsible for describing the relations between multiple frames and capturing motion patterns. However, incorporating these features can be challenging, especially when dealing with body movements that alter the spatial layout.

One approach to address this issue is to create a custom model for feature extraction from scratch. Unfortunately, this technique necessitates a substantial number of parameters and extensive computational resources, rendering it impractical for applications with strict constraints, such as running on edge devices with minimal interference rates and latency. To overcome this limitation, we drew inspiration from recent advances in literature and leveraged parameter-efficient transfer learning techniques in conjunction with large pre-trained models. By adapting existing architectures, we can effectively perform human action recognition tasks while maintaining computational efficiency.

In the context of parameter-efficient transfer learning, we employed MobileViT [47] as the foundation of our architecture for feature extraction. MobileViT was originally designed for mobile devices, such as the iPhone 12, and offers a variant with a limited number of parameters. Our newly proposed architecture builds upon this foundation and incorporates enhancements to capture additional temporal information in videos, thereby improving the representation of actions on a future map. The relationships between frames play a vital role in understanding and representing actions, making temporal features crucial for accurate action recognition. The detailed description of our proposed architecture can be found in Section 3.10.

3.5 Preprocessing

Before beginning training, it is necessary to prepare the datasets in the appropriate format. In this case, we used Kinetic-400, a commonly used dataset for human action recognition. As a first step, we converted the videos into a sequence of images to aid the model's learning process. We used a frame rate of 8 frames per second to balance reducing redundancy and preserving the temporal information necessary for accurate recognition.

It is important to note that the frame rate per second (FPS) chosen has a significant impact on the learning curve and model size. Altering the FPS can affect the model's performance and efficiency. This observation emphasizes the significance of considering the temporal aspect in human action recognition, as discussed in the previous chapter. There is a correlational relationship between FPS and the learning curve, which must be managed carefully for optimal results. In our model design, we added a final classification layer that can handle 400 classes. This layer has dense connections and uses the softmax function for activation, following established practices in the field [48, 49].

3.6 Alternative Architecture

During our research, we examined various architectural alternatives, as shown in Figure 3.1. However, after closer examination and testing, it became clear that this specific design iteration did not fulfill our performance expectations. We did not reach an accuracy over 32%. Therefore, we had to consider alternative approaches.

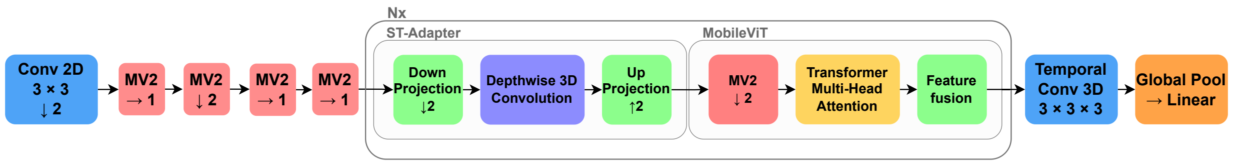


Figure 3.1: It explored architectural alternatives that show limited potential for additional training. Consider incorporating the ST-Adapter either before or after the MobileViT block in a serial configuration.

3.7 MobileViT

Lightweight convolutional neural networks prove to be a practical tool for mobile vision tasks. Their spatial induction bias enables them to acquire representations with fewer parameters in diverse visual cues. However, these networks exhibit spatial localization. To acquire global representations, self-attention-based vision transformers Q (ViTs) come into play. ViT, in contrast to CNN, is a heavyweight. MobileViT presents an alternative approach by using transformers as convolutions to process information. The data displays that MobileViT outperforms CNN and ViT based networks considerably on diverse tasks and datasets.

Despite its effectiveness, the practical application of ViT is impeded by the substantial number of network parameters and delays it necessitates. As a substitute for convolutional neural networks (CNNs), vision transformers - which are self-attention-based models - have arisen for visual learning. Succinctly stated, ViT fragments an image into patches and employs multi-head self-attention in transformers to comprehend the interconnections between these patches. To optimize performance, it is now customary to augment the parameters of ViT networks. Nonetheless, this advancement results in a larger model size and slower processing speed. Many real-world situations require rapid visual recognition tasks on mobile devices with constrained resources. The size of the ViT model has been reduced to accommodate the resource limitations of mobile devices, its performance remains markedly inferior to that of lightweight CNNs.

Many vision tasks on mobile devices use lightweight CNNs, but ViT-based networks are currently unsuitable for these devices due to their heavier weight, difficulty in optimization, and the need for significant data augmentation and L2 Regularization to prevent overfitting. For example, a ViT-based segmentation network with 345 million parameters performs similarly to a CNN-based network with only 59 million parameters. ViT-based models require a greater number of parameters as they lack the image-specific inductive bias present in CNNs.

The combination of CNN and Transformer to produce a ViT model for mobile vision tasks remains uncertain. In order to meet the requirements of mobile vision tasks, models need to be light, have low latency, and be resource-efficient, accurate, and adaptable. Floating point operations (FLOPs) alone are inadequate for achieving low latency on mobile devices because they fail to consider crucial inference-related factors such as memory access, parallelism, and platform characteristics. This paper utilizes MobileViT to merge the advantages of CNN and ViT. MobileViT presents the MobileViT block, which efficiently encodes both local and global information. Unlike ViT and its variations, MobileViT adopts a transformer to substitute local processing in convolution with global processing. This enables the MobileViT block to possess traits similar to both CNN and ViT, facilitating more efficient representation learning with fewer parameters and simpler training methods [5].

3.7.1 Architecture

The MobileViT module, as shown in Figure 3.2, intends to convey both local and global information from the input tensor while reducing the number of parameters. To accomplish this, MobileViT merges a typical convolutional layer with a pointwise (1x1) convolutional layer for generating the input tensor. Convolutional layers capture adjacent spatial details, whereas pointwise convolutions project tensors $X_L \in \mathbb{R}^{H \times W \times d}$ into a high-dimensional space by acquiring linear combinations of input channels C.

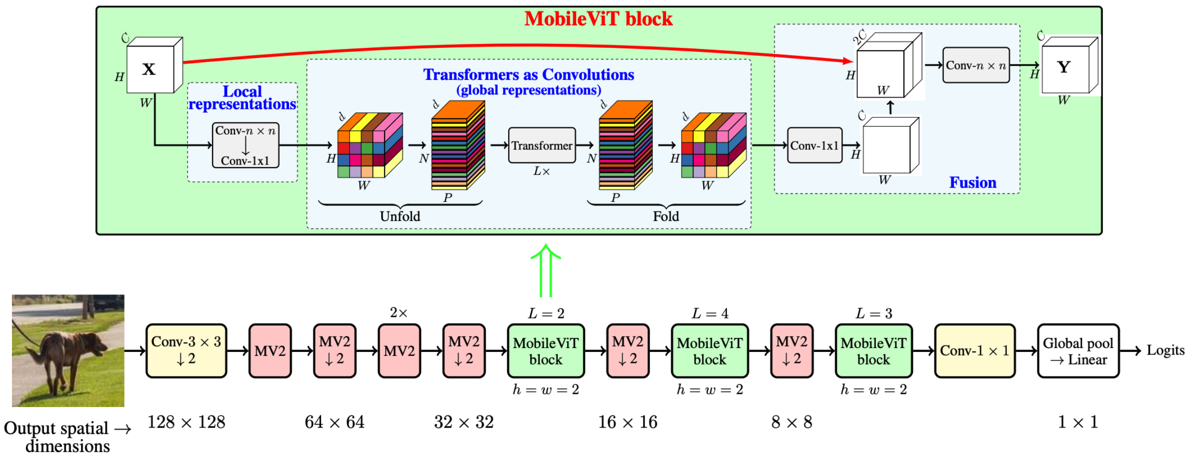


Figure 3.2: Architectural representation of the MobileViT model [5].

MobileViT aims to create models with long-range non-local dependencies and is capable of an effective receptive field of $H \times W$. Accomplishing this involves using the dilated convolution method with carefully chosen dilation rates. It's also important to apply weights to valid spatial regions, rather than just padded zeros. ViTs that utilize multi-head self-attention have proven effective in visual recognition tasks. Nonetheless, ViTs are quite heavy and exhibit subpar optimizability, largely due to their lack of spatial inductive bias. To achieve spatial induction bias and enable MobileViT to learn a global representation, it is necessary to expand it into N non-overlapping flattened patches, as shown in Figure 3.3.

Unlike ViT, which loses the spatial order of pixels, MobileViT preserves both the patch order and the spatial order of pixels within each patch. This means that $X_F \in \mathbb{R}^{P \times N \times d}$ can be folded from $X_G \in \mathbb{R}^{P \times N \times d}$. Finally, Tensor X_F is reduced to dimension C through point-wise convolution. Another convolutional layer with $n \times n$ filters is subsequently employed to merge the regional and global characteristics in the concatenated tensors. The overall efficient receptive field of MobileViT is $H \times W$.

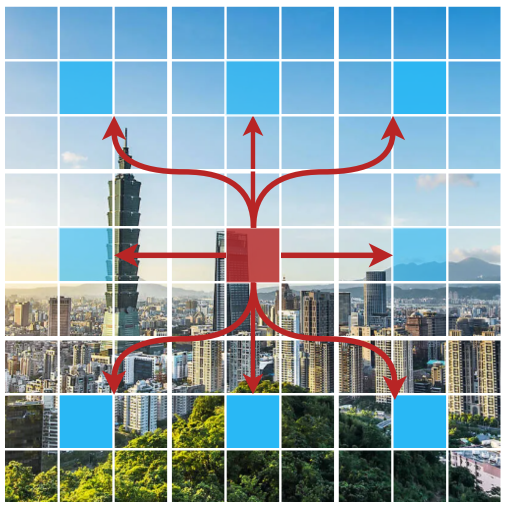


Figure 3.3: Patch Attention Mechanism: Each pixel undergoes a comparison with every other pixel, exemplified by a selection of connection arrows depicted here for simplicity. This attention mechanism draws inspiration from Transformers, where interconnected relationships extend across the entire system.

A standard convolution follows a three-step process of unrolling, matrix multiplication and folding. The MobileViT block operates similarly to convolutions, utilizing comparable elements. However, unlike convolutions which process locally, the MobileViT block processes globally, resulting in a more extensive and meticulous outcome. Due to this approach, the MobileViT block shares some commonalities with convolutions such as spatial bias. In fact, the MobileViT block is a combination of a transformer and a convolution. Our intentionally simple design allows for efficient implementation of these two techniques on various devices, eliminating the need for extra work.

3.7.2 Lightweight

The MobileViT block integrates conventional convolutions and transformers for capturing local and global representations. This combination bestows the MobileViT blocks with convolution-like properties as well as the competence of global processing. Such exceptional modeling capacity facilitated the development of a lightweight MobileViT model that is intentionally shallow and lighter in design [5].

3.8 Spatio-Temporal Adapter

ST-Adapters [50] operate by adding additional layer layouts to the target architecture. The original paper proposed ST-Adapters for a method called image-to-video transfer learning, which aimed to reduce competition costs. Adapters can take the tokens for all frames and significantly aid in capturing temporality in the video stream [50].

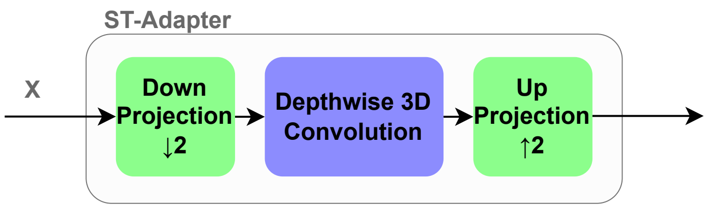


Figure 3.4: Architecture of Spatio-Temporal Adapter

Figure 3.4 of architecture uses 3D depthwise CNN and it composed by three important components. The first component is a down projector then a nonlinear layer followed by up projector. The features adaptation process is formally written as:

$$\text{ST-ADAPTER} = X + f\left(DW\_3D\_CNN\left(X\ W_{DOWN}\right)\right)\ W_{UP}\ . \tag{3.1}$$

Where formally given input is a feature matrix $X \in \mathbb{R}^{N\times d}$ at the i-th layer and $f(\cdot)$ activation function. $W_{DOWN} \in \mathbb{R}^{d\times r}$ refers to the down projection layer, and $W_{UP} \in \mathbb{R}^{d\times r}$ refers to the up-projection layer. Mentioned projectors are used to decrease and increase dimensions. These two components create together a lower-dimensional space-bottleneck. DW_3D_CNN denotes depth-wise 3D convolutional network placed in the bottleneck. The adapter operates in a low-dimensional feature space. Layers are highly efficient in parameter computation and capturing temporal features. 3D depth-wise CNN is implemented with the residual connection [50].

ST-Adapter has the following advantages:

- Achieves comparable performance compared to full fine-tuning.
- Requires a small training cost.
- Reduce the number of trainable parameters.

3.9 MobileNet-V2

MobileNetV2 (MV2) has become a crucial convolutional neural network architecture in the academic field, transforming research in computer vision and deep learning. The architecture was introduced by Google researchers in 2018. MobileNetV2 is designed to be deployable on mobile devices due to its inverted residual blocks, linear bottlenecks, and incorporation of squeeze-and-excitation mechanisms. Architecture is a popular model in academic circles due to its adaptability and success in transfer learning scenarios.

The block contains a 1×1 convolutional layer and a 3×3 depth-wise convolutional layer with batch normalization. This structure functions as a down-sampler. Figure 3.5 show architecture of MV2 block [51].

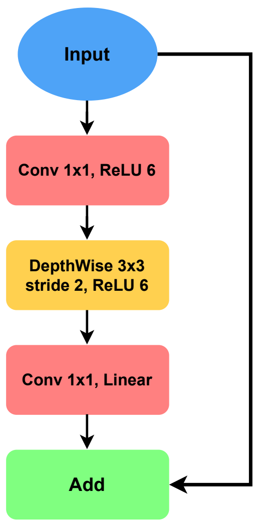


Figure 3.5: MobileNetV2 architecture [6].

3.10 Adapting MobileViT with ST-Adapters

The original input image can be described as $H \times W$, where H represents the height and W the width of the image. The input is extended by the parameter C representing the number of channels. The formula for input data is described as $x \in \mathbb{R}^{H \times W \times C}$. In our case, the input has the following values: $W = 256$, $H = 256$, $C = 3$. C is equal to 3 because we have a color image with 3 color spaces.

In the first layer, data is processed by a standard 1×1 convolutional layer to capture local spatial information, followed by a 3×3 pointwise convolutional layer. The main purpose of the pointwise convolutional layer is to project the feature tensor into a high-dimensional space, creating $x_L \in \mathbb{R}^{H \times W \times d}$, where d stands for a dimension greater than C. The original publication of MobileViT [47] describes this mentioned block as MV2, denoting the MobileNet-V2 block 3.9.

The following layer plays a crucial role in the architecture to enable the model to learn global representation with spatial inductive bias. Firstly, the modulo-unfold tensor $x_L \in \mathbb{R}^{H \times W \times d}$ into N non-overlapping patches. These patches represent small areas in the image, resulting in the unfolded tensor with a new shape: $x_{\text{UNFOLD}} \in \mathbb{R}^{P \times N \times d}$, where P is the size of patches, N is the number of patches, and d is the number of channels. Formally, $P = h_1 \times w_1$, and $N = \frac{HW}{P}$, where h_1 and w_1 denote the height (h) and width (w) of each token. The product of this operation yields non-overlapping patches.

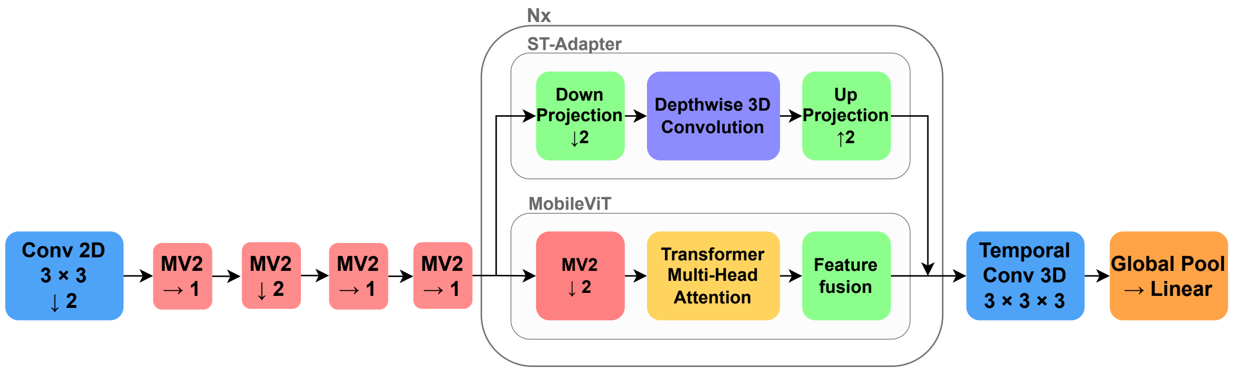


Figure 3.6: Proposed Architecture.

Following the unfolding, the vector $x_{\text{UNFOLD}} \in \mathbb{R}^{P \times N \times d}$ is then passed through a standard Vision Transformer (VIT) block. Vision Transformers use a multi-head attention mechanism to capture global relationships between patches. Mathematically, $x_G(p) = \text{ViT}(x_U(p))$, where $1 \leq p, p \leq P$. VIT calculates correlations between every patch (token) $p \in \{1, 2, .., P\}$. The output vector $x_G(p)$ is then unfolded back into a vector with the following shape: $\mathbb{R}^{H \times W \times d}$. The new vector $x_{\text{FOLD}} \in \mathbb{R}^{H \times W \times d}$ is projected into an MV2 block. The main purpose of the MV2 block is downsampling (decreasing dimension space).

Figure 3.6 illustrates a crucial element of the architecture, specifically a parallel ST-Adapter 3.8. The input vector is partitioned before entering the MV2 $\downarrow$ 2, Transformer Multi-Head Attention, and Feature Fusion modules. The secondary vector flows into the down-projection ($W_{\text{DOWN}} \in \mathbb{R}^{d \times r}$), resulting in a lower-dimensional feature space (bottleneck). Subsequently, a depth-wise 3D convolutional network is applied within the bottleneck to capture temporal features. Following this, the dimensionality is increased through an up-projection operation ($W_{\text{UP}} \in \mathbb{R}^{d \times r}$). This is performed by the function $fc2$, which is crucial for aligning the dimensionality with the output of the Transformer Multi-Head Attention block. It is worth noting that the network's dimensionality is reduced, as shown in Table 3.1, contributing to improved performance.

Finally, there is a $3 \times 3 \times 3$ temporal convolutional network with a linear activation function to capture spatial-temporal features.

Table 3.1: Architecture.

No.	Layer	Output Size	Output Stride	In Channels	Out Channels	Dim.	Attention Head	Activation
0	Image	256×256						
1	Conv 2D, 3×3, $\downarrow 2$ MV2	128×128	2	16	32			Swish
2	MV2, $\downarrow 2$ MV2 MV2	64×64	4	32	64			Swish
3	MV2, $\downarrow 2$ MobileViT block Adapter 3D, $3\times1\times1$	32×32	8	64	96	144	2	Swish
4	MV2, $\downarrow 2$ MobileViT block Adapter 3D, $3\times1\times1$	16×16	16	96	128	192	4	Swish
5	MV2, $\downarrow 2$ MobileViT block Adapter 3D, $3\times1\times1$	8×8	32	128	160	240	3	Swish
6	Conv 3D, $3 \times 3 \times 3$ Temporal	8×8	32	160	640			Swish
7	Global pool Linear	1×1	256	640	400			

3.11 Target Edge Device

The NVIDIA Jetson Nano [52] is employed as the edge device for deployment, offering computational resources at the edge. Tailored for machine learning applications, especially those necessitating inference acceleration, the Jetson Nano is a notable member of the NVIDIA family. Its primary component is the graphics processing unit (GPU), dedicated to accelerating AI tasks. Being one of the smallest devices in the NVIDIA family, the Jetson Nano proves to be an excellent selection for edge computing applications. Table 3.2 outlines the technical specifications of the Jetson Nano.

Table 3.2: Technical specification of Jetson Nano.

Item	**Specification**
Release Date	2020
GPU	128 CUDA cores NVIDIA Maxwell
CPU	Quad-core ARM Cortex-A57 MPCore processor
Memory	4 GB 64-bit LPDDR4, 1600MHz 25.6 GB/s
Storage	16 GB eMMC 5.1
Connectivity	Gigabit Ethernet

Camera

Type of the camera USB 2.0 Camera: HD USB Camera (USB-0000:00:14.0-9) [53].

3.12 Optimization Tool for Edge Device

We use Torch-TensorRT [54] as an application tool, which is a software development kit (SDK) provided by Nvidia. The primary purpose of the SDK is to create a lightweight and optimized module that can be easily deployed on edge devices, such as the Jetson Nano. The builder calibrates the model to operate with lower precision (FP16 or INT18), reducing computation time. Subsequently, the authors combine multiple layers into a single operation and use Dynamic Tensor Memory to reduce memory usage. These steps allow Torch-TensorRT to improve performance on edge devices.

3.13 Training Approach

In the field of training methodologies, we used two distinct strategies. The first strategy is based on the principles of Parameter Efficient Transfer Learning (PETL) [55] and ST-Adapters [50], while the second approach involves fully fine-tuning the model.

3.13.1 Full Fine-Tuning

The second strategy involves fully fine-tuning the model and optimizing all of its weights for the target task. This approach provides greater flexibility in addressing diverse tasks, especially those that differ significantly from the pre-trained models' original objectives. Figure 3.7 shows all parts of the model are set to trainable. Full fine-tuning requires significant computational resources and time for training.

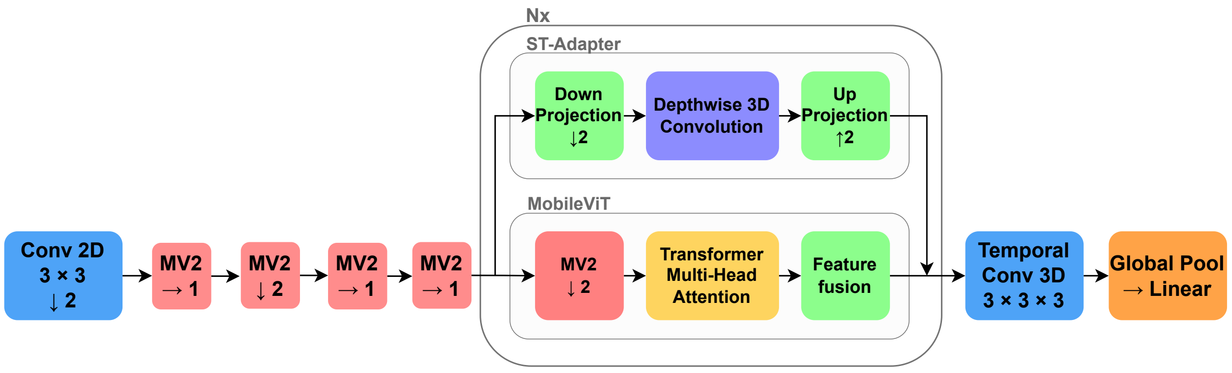


Figure 3.7: Full Fine-Tuning training strategy.

3.13.2 Adapter Fine-Tuning

The first strategy leverages the knowledge gained from Parameter Efficient Transfer Learning [55] and ST-Adapters [50]. This approach involves selectively freezing certain components of the DNN architecture, specifically the majority of the layers while allowing the remaining parts to be trained and adapted for the target task. The frozen layers are depicted in Figure 3.8 with a gray color and a padlock icon in the corner.

This methodology significantly reduces the computational expenses of training the model by constraining the updateable parameters to 15% of the total 5.3 million parameters. The incorporation of ST-Adapters further enhances the efficacy of the model by modifying the input embeddings to better conform to the temporal and spatial dimensions of the new task. The combination of PETL and ST-Adapters creates a robust and adaptable model that can achieve high performance with minimal training requirements.

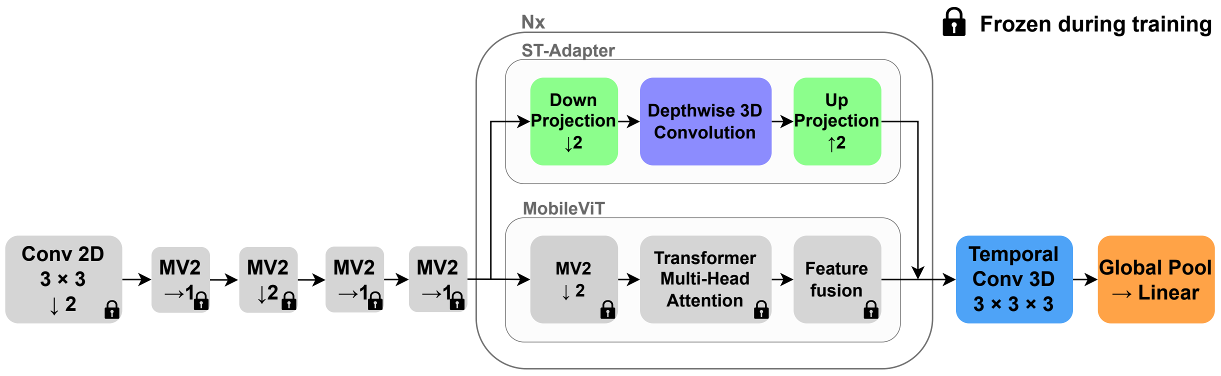


Figure 3.8: Adapter Fine-Tuning training strategy with frozen section.

Chapter 4

Experimental Results

4.1 Experimental Settings

Firstly, we utilize a vanilla MobileViT [56] model as the backbone architecture and integrate it with specially designed adapters to constitute a novel framework. The MobileViT component is pre-trained on the upstream task, and the weights are kept frozen during training. The MobileViT backbone is trained on the ImageNet-1K dataset [41]. We employ image-to-video transfer learning techniques [50] and initialize the newly added adapters modules with weights initialized using the Kaiming normal distribution function [57]. Through extensive experimentation, across diverse scenarios, we observe that if the initialization deviates excessively from the identity function, the model exhibits instability. From the beginning, our model is designed to perform HAR video recognition.

4.1.1 Training Approach

In order to evaluate the performance of our newly proposed architecture, we conduct a comparative analysis against two widely used strategies 3.13 - Adapter Fine-Tuning (involving the training of only the newly added layer) and Full Fine-Tuning (where all parameters are learnable and are subject to update during the training epochs).

4.1.2 Data Augmentation

Using data augmentation techniques is essential for improving the robustness of machine learning models. By randomly resizing and horizontally flipping video streams, we can diversify the training dataset and reduce overfitting. The choice of bilinear interpolation when resizing to 288×288 provides the preservation of spatial features. This strategic use of data augmentation optimises model performance and generalisability in video analysis tasks.

4.2 Implementation

PyTorch [30] serves as the principal toolkit in our experiments. The training process takes place on a server equipped with an Intel i9 7900X @ 3.3 GHz CPU, 64 GB RAM, and Ubuntu 20.04 operating system along with 4×Nvidia GeForce GTX 1080 Ti 12 GB RAM graphics cards, as displayed in Table 4.1.

Table 4.1: Technical specification of server.

Item	**Specification**
CPU	Intel i9 7900X @ 3.3 GHz CPU
GPU	4x Nvidia GeForce GTX 1080 Ti 12 GB RAM
RAM	64 GB
OS	Ubuntu 20.04
Connectivity	Gigabit Ethernet

Our initial step involves converting the video inputs into a sequence of frames, with each image resized to a resolution of 256x256 pixels. We utilize 64 GRU nodes from a total of 1024 nodes, as a larger number of nodes tends to increase both the accuracy and the number of parameters. The model consists of 7 layers in total, as shown in Table 4.2.

We employ the AdamW [58] optimizer and set our batch size to 64. The initial learning rate is set to 0.001, while the step decay is 0.01. To prevent overfitting, we introduce a dropout parameter with a value of 0.1. Before training, we warm up [59] the model for 1000 iterations, equivalent to five epochs. The model undergoes training for a total of 50 epochs, culminating in satisfactory accuracy. Notably, the final model size does not exceed 20 MB, fulfilling the requirement of deploying the model on an edge device. The SoftMax function is used as the final layer for classification.

A notable aspect of our model is the sensitivity of its performance to hyperparameter tuning. Specifically, even slight alterations to the hyperparameters can yield considerable variations in the ultimate experimental outcome. This highlights the importance of meticulous optimization and careful consideration of hyperparameter settings in order to achieve optimal model performance.

Table 4.2: Hyperparameters settings.

Description	**Value**
Input Video Size	256×256
Batch size	124
Frames per clip	8
Clips per video	2
Weight decay	0.01
Optimizer	AdamW
Scheduler	Cosine
Warmup iterations	1000
Warmup Initial Learning Rate	0.0001
Maximal Learning Rate	0.001
Minimal Learning Rate	0.0001
Number of Epoch	50
Activation Function	Swish
Momentum	0.1
Number of classes	400

4.3 Kinetics-400 Dataset

The training process utilized the Kinetics-400 (Kin-400) dataset [42, 41], which consists of $240,000$ training samples and $20,000$ validation samples. The dataset is categorized into 400 distinct classes, as presented in Table A.1, which describe various human activities. Each sample is 10 seconds long and contains approximately 300 frames, corresponding to a frame rate of around 30 FPS. The dataset records three main types of interactions: human activity, human-object interaction, and human-human interaction. For more information about the dataset, please refer to Section 2.11.2.

4.4 Evaluation Metrics

For the evaluation performance of our proposed model, we used the following metrics. These four situations describe how many positive results occur among all positive samples.

- **TP**: True Positive
- **TN**: True Negative
- **FP**: False Positive
- **FN**: False Negative

Based on the terms above, we calculate Accuracy, Precision, Recall, and F1 score as follows:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} , \tag{4.1}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} , \tag{4.2}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} , \tag{4.3}$$

$$\text{F1 score} = \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} . \tag{4.4}$$

FLOPs

In addition to accuracy and F1 score, another key metric for evaluating a model's performance is the number of floating-point operations per second (FLOPs). This measure provides insight into the computational complexity of the model and the amount of resources required to execute a single instance. Generally, a lower number of FLOPs indicates fewer computational resources are needed, which can be advantageous for deployment on resource-constrained devices. The typical range for this value is around 10^6 operations.

4.5 Visualisation of Training

A training curve is a graphical representation that illustrates the performance of a deep learning model on a given task. It visualizes how well the model is learning from the data as training progresses. The training curve can help us identify issues such as overfitting, underfitting, and convergence.

The validation curve, on the other hand, is a graphical representation that evaluates the performance. However, unlike the training curve, the validation curve provides a more realistic estimate of how well the model will perform on new, unseen data.

Figure 4.1 depicts the training and validation curves for a deep learning model on a particular task, with the x-axis representing the number of epochs and the y-axis displaying the accuracy in percentages, where 100% represents the best performance. The blue line represents the training curve, while the orange line represents the validation curve. As evident from the Figure, the validation curve furnishes a more realistic estimation. The model achieved the highest accuracy of 74.94%.

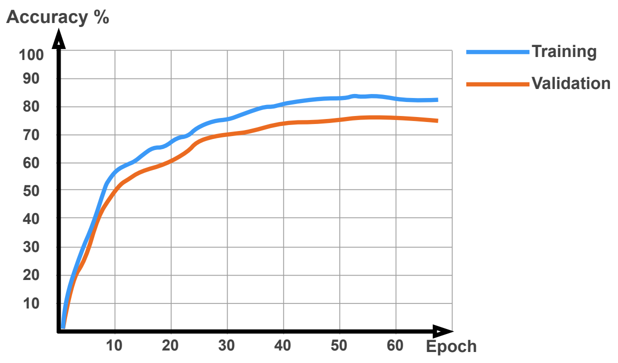


Figure 4.1: Training and Validation Accuracy.

Figure 4.2 interprets the loss curves, offering insights into the model's learning dynamics. From the graphical representation, it is comprehensible that the learning process exhibits a “good fit”. Both the training and validation curves demonstrate a consistent decrease, telling of the model's reach point of stability. The difference between the training and validation curves is quantified as the “generalization gap”, signifying the difference in performance between the model's training and its ability to generalize to unseen data.

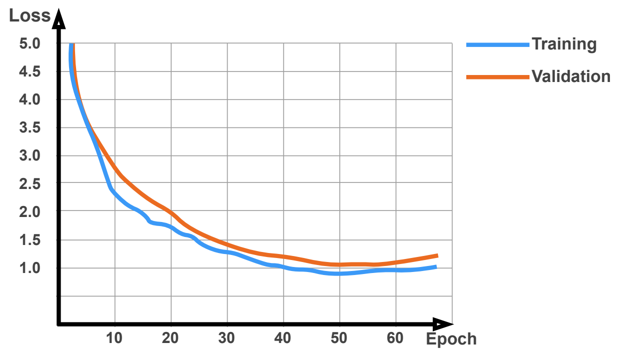


Figure 4.2: Training and Validation Loss.

To avoid the risk of overfitting during training, the application of “early stop” techniques are employed, as denoted in the provided in Table 4.2. The optimal number of epochs is determined as 50. The graphical illustration further explains the potential outcomes of continued training. The noticeable upward trend at the end in both the training and validation curves indicates a tendency towards overfitting. This emphasizes the importance of terminating the training process at the optimal epoch to prevent the model from overfitting.

4.6 Results

We employed two distinct training strategies 3.13 to evaluate the performance of our proposed architecture. Firstly, we present the results of our new architecture and compare it with various benchmark models on Kinetics-400. Table 4.3 presents a comprehensive overview of different computer vision models and architectures, providing insights into their specifications and performance metrics on distinct tasks. Each entry in the table includes details such as the model name, backbone architecture, pre-training dataset, input size, number of parameters, and performance scores. Performance is evaluated on tasks such as action recognition, with metrics such as Top-1 accuracy and Top-5 accuracy being reported. Additionally, the table showcases a variety of datasets used for pre-training, including ImageNet-1k [41], ImageNet-21k [41], CLIP [60], and custom datasets.

Notable models such as ViT and Swin are featured, each demonstrating competitive results in terms of accuracy and model complexity. The inclusion of models with different backbone architectures, input sizes, and pre-training datasets provides a comprehensive comparison, offering valuable insights for researchers in the computer vision field. Well-established models like CoCa [61], UniFormerV2-L [62], and VideoSwin-L [63] show superiority in terms of performance Top-1 accuracy. However, its biggest drawback is the number of parameters. For instance, CoCa [61] has over 1000 million, and UniFormerV2-L [62] 354 million parameters.

The second part of Table 4.3 shows architectures considered to be lightweight. Our proposed Adapter Fine-Tuning method outperforms the majority of other methods in terms of Top-1 accuracy, achieving a score of 74.94%. This indicates that the proposed approach is effective in adapting the pre-trained ImageNet-1k model to the Kinetics-400 dataset. The Full Fine-tuning method achieves higher accuracy than the Adapter Fine-Tuning method, with a Top-1 score of 76.43%. This suggests that fine-tuning the entire MobileViT model can lead to better performance gains compared to adapting only the pre-trained weights. The model demonstrates the best trade-off between the number of parameters, Top-1 accuracy, and computation costs (GFLOPs).

As shown in the Table 4.3, the combination of a 3D CNN and ViT model demonstrates superiority over previous architectures. The CNN extracts temporal features effectively, while the ViT model accurately calculates spatial information between all patches with its self-attention mechanism. However, our model does not achieve the highest accuracy due to several reasons. Our initial goal was to find a balance between various parameters, prioritizing the ability to deploy the model on an edge device with real-time performance. Additionally, we faced limitations in computational resources, aiming to optimize the model with a few-shot training. The proposed model has approximately 5.3 million parameters. During the training phase of Full Fine-tuning, all parameters are trainable and updated. This is reflected in the achieved performance. The proposed architecture for Adapter Fine-Tuning updates only 15% of the total parameters. Consequently, our final trained models are compact enough to operate on low-processing stations, such as the Jetson Nano.

Table 4.3: Performance comparisons for action recognition on the Kinetics-400.

Method	Backbone	Pretrain	Input Size	GFLOPs	Param	Top-1 %	Top-5 %	Year
CoCa [61]	ViT-g	JFT-3B+ALIGN-1.8B	16×576×576	N/A×3×4	1000 M	88.9	-	2022
UniFormerV2-L [62]	ViT-L	CLIP-400M+K710	64×336×336	12550×3×2	354 M	90.0	98.4	2022
MViTv2-L [64]	-	ImageNet-21k	N/A	42420	218 M	86.1	-	2022
VideoSwin-L [63]	Swin-L	ImageNet-21k	32×224×224	7248	197 M	83.1	95.9	2022
TimeSformer-L [65]	ViT-B	ImageNet-21k	96×224×224	7140	121 M	80.7	95.7	2021
DUALPATH w/ ViT-B/16 [66]	ViT-B/16	CLIP	N/A	710	96 M	85.4	-	2023
ST-Adapter w/ ViT-B/16 [50]	ViT-B/16	CLIP	N/A	1821	93 M	82.7	-	2022
UMT-B800e [67]	ViT-B	K710	8×224×224	180×3×4	87 M	85.7	97.0	2023
R(2+1)D RGB [68]	ResNet-34	-	32×3×112×112	1524	63.8 M	72.0	-	2018
SlowFast101 [69]	R101+NL	-	80×224×224	255×1×5	60 M	79.8	93.9	2019
UniFormer-B [70]	UniFormer-B	ImageNet-1k	32×224×224	3108	50 M	83.0	95.4	2021
ECO [71]	BN-Inception+3D ResNet-18	-	16×3×224×224	64	47.5 M	69.0	-	2018
MViTv2-B [72]	MViTv2-B	-	32×224×224	259×3×4	37 M	81.2	95.1	2021
Two-Stream I3D [73]	3D BN-Inception	ImageNet-1k	3×256×256	544.44	25 M	71.1	89.3	2017
TSM [74]	ResNet-50	-	8×3×224×224	33	24.3 M	70.6	-	2019
MF-Net [75]	-	ImageNet-1k	16×3×224×224	11.1	8 M	72.8	90.4	2018
VTN-EFF [76]	Effnet-B0	ImageNet-1k	32×224×224	-	6.8 M	68.2	88.1	2021
T-STFT [77]	BNInception	-	64×224×224	41.2	6.27 M	75.0	91.1	2022
MoViNet-A2 [78]	-	-	32×224×224	-	4.8 M	75.0	92.3	2021
Ours Adapter Fine-Tuning	**MobileViT**	**ImageNet-1k**	**8×3×256×256**	**16.53**	**5.3 M**	**74.94**	**93.2**	**2023**
Ours Full Fine-tuning	**MobileViT**	**ImageNet-1k**	**8×3×256×256**	**16.59**	**5.27 M**	**76.43**	**94.06**	**2023**

4.6.1 Comparison of Different Optimizers

We present a comparative analysis of the performance of distinct optimizers, specifically AdamW [58], AdamGrad [79], and Stochastic Gradient Descent (SGD) [80]. To ensure consistency in our evaluation, we employed identical hyperparameter settings as depicted in Table 4.2.

Table 4.4 showcases a comparison of the model's accuracy. Notably, the AdamW optimizer attains an accuracy of 76.43%, followed by the AdamGrad optimizer with 75.28%. Finally, the SGD optimizer achieves an accuracy of 72.64%. These findings suggest that the AdamW optimizer performs supremely among the three optimizers considered in our study.

Table 4.4: Comparison of Top-1 accuracy of different optimizers.

Optimizer	**Full Fine-tuning**	**Adapter Fine-Tuning**
AdamW [58]	76.43%	74.94%
Adagrad [79]	75.28%	72.51%
SGD [80]	72.64%	69.36%

4.6.2 Inference on Server

Before deploying the model on an edge device, it was crucial to validate its performance on the server using a test dataset. We executed the inference model on the server and analyzed the results for a video sequence. The following Figures 4.3,4.4 illustrate the model's predictions, with Figures 4.4 highlighting instances of misclassification. Notably, there were cases where the model confused similar actions, such as mistaking 'motorcycling' for 'riding a scooter'. This confusion may arise due to the similarity of these actions in terms of their features, increasing the probability of confusion. The mean inference speed reach on the server is approximately 144 FPS.

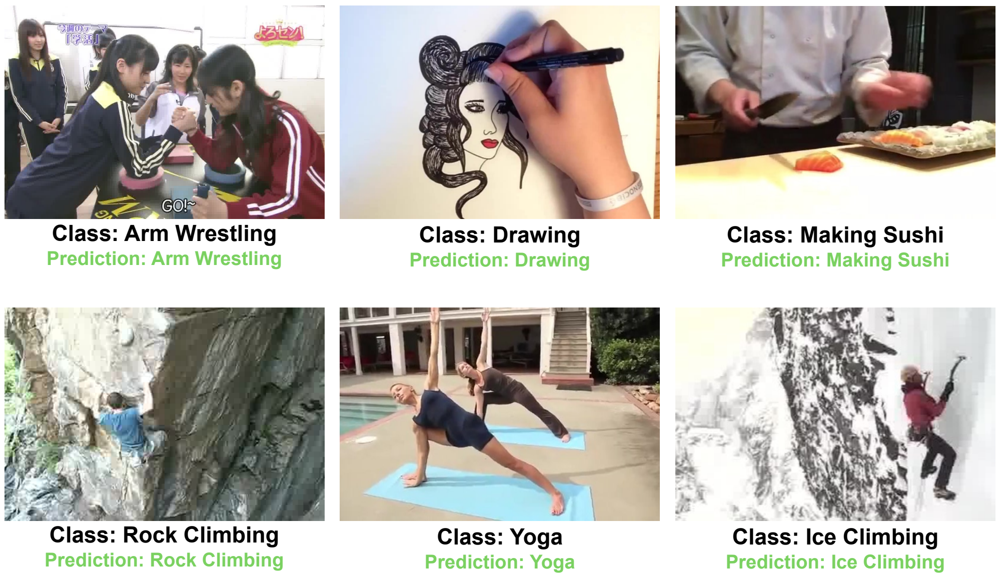


Figure 4.3: Server-based inference - Examples of correct predictions.

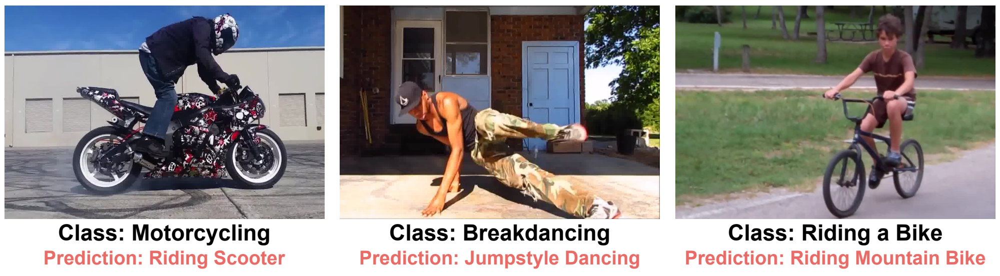

Class: Motorcycling
Prediction: Riding Scooter
Class: Breakdancing
Prediction: Jumpstyle Dancing
Class: Riding a Bike
Prediction: Riding Mountain Bike

Figure 4.4: Server-based inference - Examples of incorrect predictions.

Table 4.5 provides a comprehensive examination of key performance metrics for a specified model within a server context. The results contain critical parameters, namely Preprocess Time, Postprocess Time, Inference Speed, Latency, and RAM Usage, with evaluations conducted across diverse test datasets and camera configurations. Noteworthy findings include variations in preprocessing time, measured at 2.88 ms and 5.63 ms for different datasets, and postprocessing time, exhibiting distinctions at 1.71 ms and 1.89 ms. The model demonstrates commendable real-time processing capabilities, as evidenced by an Inference Speed of 141 FPS and 126 FPS, respectively. Latency values of 33.12 ms and 91.83 ms depict the temporal delay between input and output. The observed RAM usage remains consistent at 2.41 GB and 2.42 GB for the test datasets, indicating stable memory utilization. Furthermore, the model's accuracy is reported as 74.94% for the test dataset, while the accuracy for the camera is 73%. This dataset contributes crucial empirical insights into the real-time computational efficiency of the model within a server environment.

Table 4.5: Real-time performance on server.

Item	**Test - Dataset**	**Camera**
Preprocess Time	2.88 ms	5.63 ms
Postprocess Time	1.71 ms	1.89 ms
Inference Speed	141 FPS	126 FPS
Latency	33.12 ms	91.83 ms
RAM Usage	2.42 GB	2.41 GB
Accuracy	74.94%	73.00%

4.7 Deployment on Edge

Executing the model in real-time on an edge device presents a significant challenge. To achieve optimal performance of our HAR system, we must utilize the full potential of the Jetson Nano. As explained in Methodology 3.12, we used the Torch-TensorRT [54] tool to optimize the model for execution on the edge device. The version parameters of the system dependencies used are shown in Table 4.6. The comprehensive configuration of the Edge device is illustrated in the Appendix A.2.

Table 4.6: Jetson Nano system dependencies.

Python version	**Torch-TensorRT**	**Nvidia Pytorch**
3.8	v1.0.0	Jetpack - 5.0 Pytorch v1.11.0

4.7.1 Inference

The final stage of the thesis involves validating the conceptual framework and implementing the model on an Edge Device, specifically the Jetson Nano. The following section provides a comprehensive analysis of the model's performance on the Jetson Nano platform. Figure 4.5,4.6 shows illustrative examples taken from the Kinetics-400 test dataset.

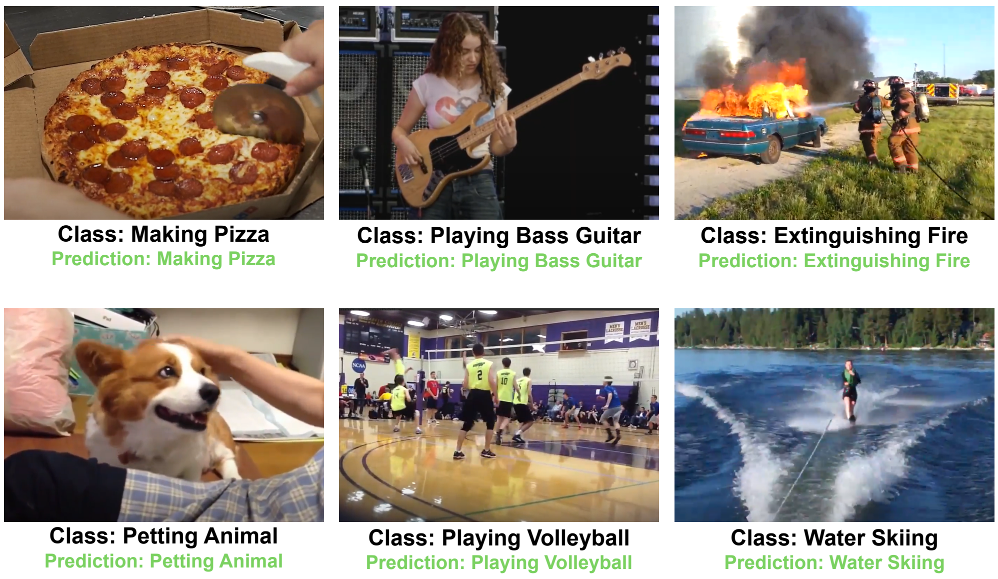

Class: Making Pizza
Prediction: Making Pizza
Class: Playing Bass Guitar
Prediction: Playing Bass Guitar
Class: Extinguishing Fire
Prediction: Extinguishing Fire
Class: Petting Animal
Prediction: Petting Animal
Class: Playing Volleyball
Prediction: Playing Volleyball
Class: Water Skiing
Prediction: Water Skiing

Figure 4.5: Jetson Nano-based inference - Examples of correct predictions.

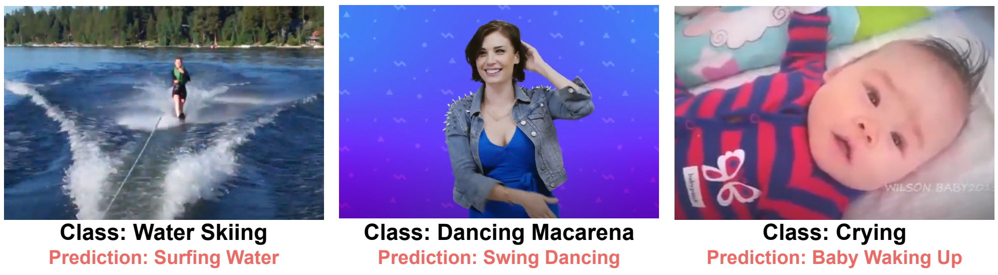

Class: Water Skiing
Prediction: Surfing Water
Class: Dancing Macarena
Prediction: Swing Dancing
Class: Crying
Prediction: Baby Waking Up

Figure 4.6: Jetson Nano-based inference - Examples of incorrect predictions.

The Table 4.7 presents real-time performance metrics for the Jetson Nano platform, providing insights into the computational efficiency of a specified model under various conditions. It outlines key parameters, including Preprocess Time, Postprocess Time, Inference Speed, Latency, and RAM Usage, evaluated across diverse test datasets (Kin-400 test) and camera configurations. The evaluation of model performance on Jetson Nano involves utilizing a test dataset associated with Kinetics-400. The Kinetics dataset is divided into three primary parts: training, validation, and testing. Specifically, our assessment involves the execution of a testing video sourced from the Kinetics-400 dataset on the Jetson Nano. The outcomes obtained with Camera input are depicted in the third column. Jetson Nano has been configured within a real-world setting, utilizing real-time camera data. Figure A.2 illustrates the calibration process.

Latency refers to the time delay between the input provided to the system and the output generated. Preprocessing time is the time spent getting the input data into a format suitable for the model, and any variations can impact the system's overall efficiency. Postprocessing is the time spent after the model has produced its inference and involves any necessary actions or analysis of the output. Similar to preprocessing time, variations in postprocessing time can influence the overall system performance.

The preprocessing time shows negligible variations at 0.03 ms and 0.17 ms for different datasets, while postprocessing time has differences at 0.20 ms and 0.21 ms. The system achieves a real-time processing capability of 16.45 FPS and 14.92 FPS for inference speed, respectively. Latency is reported at 6.50 ms and 8.24 ms, demonstrating the time lapse between input and output. RAM usage remains consistent at 2.58 GB and 2.56 GB for the specified datasets. Additionally, the table shows the model's accuracy as 71.07% for the test dataset, and 70% for the camera. These quantitative findings facilitate a nuanced understanding of the computational efficiency and real-time performance of the model on the Jetson Nano platform, contributing valuable insights to the academic discourse in the domain of embedded systems and edge computing.

Table 4.7: Real-time performance on Jetson Nano.

Item	**Test - Dataset**	**Camera**
Preprocess Time	0.03 ms	0.17 ms
Postprocess Time	0.20 ms	0.21 ms
Inference Speed	16.45 FPS	14.92 FPS
Latency	6.50 ms	8.24 ms
RAM Usage	2.58 GB	2.56 GB
Accuracy	71.07%	70.00%

Chapter 5

Conclusions

In my master's thesis, I propose a trade-off between latency, performance and computation cost. My work is heavily influenced by the MobileViT architecture, which serves as the backbone network. The MobileViT architecture, recently proposed, is a connecting bridge between vanilla Vision Transformers and convolutional neural network designs for mobile devices. Therefore, MobileViT was an ideal choice as the base model for further enhancements. My experiences indicate that Vision Transformers efficiently extract low-frequency global signal features, while CNNs excel at extracting local information in high frequency.

To reduce computational costs, I implemented several innovative solutions. I incorporated image-related inductive bias into the core architecture, utilizing pre-trained weights on ImageNet-1k. Subsequent to this, I devised a plan to integrate ST-Adapters, marginally increasing the number of parameters by 6%, leading to a total number of parameters of 5.3 million.

The suggested architecture shows remarkable performance with minimal training. During Adapter Fine-Tuning only 15% parameters are trainable. It nearly accomplishes state-of-the-art results on the action recognition task using the challenging Kinetic-400 dataset. Overall, I achieve encouraging performance with economical computational cost training. The final model is well-suited for deployment on Jetson Nano. Real-time models running on Jetson Nano display admirable performance and appropriate system responsiveness.

A potential limitation of my architecture is that it was primarily developed for action recognition, and its suitability for other tasks has not been explored. The model and associated methods were designed to address the unique challenges of recognizing actions in videos. It is uncertain whether they can be successfully adapted to tackle alternative tasks. Investigating the generalizability of my approach to other computer vision tasks, such as object detection, could be a promising avenue for future research.

Bibliography

[1] Jetson Nano Developer Kit. Available from: `https://developer.nvidia.com/embedded/jetson-nano-developer-kit`

[2] vaishnav, R. Visualizing Feature Maps using PyTorch. June 2021. Available from: `https://ravivaishnav20.medium.com/visualizing-feature-maps-using-pytorch-12a48cd1e573`

[3] Vaswani, A.; Shazeer, N.; Parmar, N.; et al. Attention is all you need. *Advances in neural information processing systems*, volume 30, 2017.

[4] Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[5] Mehta, S.; Rastegari, M. Mobilevit: light-weight, general-purpose, and mobile-friendly vision transformer. *arXiv preprint arXiv:2110.02178*, 2021.

[6] Tsang, S.-H. Review: MobileNetV2 — Light Weight Model (Image Classification). Aug. 2019. Available from: `https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c`

[7] Süzen, A. A.; Duman, B.; Şen, B. Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn. In *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, IEEE, 2020, pp. 1–5.

[8] Fu, J.; Rui, Y. Advances in deep learning approaches for image tagging. *APSIPA Transactions on Signal and Information Processing*, volume 6, 2017: p. e11.

[9] Zhang, S.; Callaghan, V. Real-time human posture recognition using an adaptive hybrid classifier. *International Journal of Machine Learning and Cybernetics*, volume 12, 2021: pp. 489–499.

[10] Kong, Y.; Fu, Y. Human action recognition and prediction: A survey. *International Journal of Computer Vision*, volume 130, no. 5, 2022: pp. 1366–1401.

[11] Houlsby, N.; Giurgiu, A.; Jastrzebski, S.; et al. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*, PMLR, 2019, pp. 2790–2799.

[12] Krishnasamy, E.; Varrette, S.; Mucciardi, M. Edge Computing: An overview of framework and applications. 2020.

[13] Ullah, S.; Kim, D.-H. Benchmarking Jetson platform for 3D point-cloud and hyperspectral image classification. In *2020 IEEE International conference on big data and smart computing (BigComp)*, IEEE, 2020, pp. 477–482.

[14] Mittal, S. A Survey on optimized implementation of deep learning models on the NVIDIA Jetson platform. *Journal of Systems Architecture*, volume 97, 2019: pp. 428–442.

[15] Pietschmann, C. 'Raspberry Pi 4 vs NVIDIA Jetson Nano Developer Kit. *ht tps://build5nines. com/raspberry-pi-4-vs-nvidia-jetson-nano-dev e loper-kit*, 2019.

[16] Aggarwal, C. C.; et al. Neural networks and deep learning. *Springer*, volume 10, no. 978, 2018: p. 3.

[17] Breiman, L. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical science*, volume 16, no. 3, 2001: pp. 199–231.

[18] Jordan, M. I.; Mitchell, T. M. Machine learning: Trends, perspectives, and prospects. *Science*, volume 349, no. 6245, 2015: pp. 255–260.

[19] LeCun, Y.; Bottou, L.; Bengio, Y.; et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, volume 86, no. 11, 1998: pp. 2278–2324.

[20] Bottou, L.; Bengio, Y.; Le Cun, Y. Global training of document processing systems using graph transformer networks. In *proceedings of IEEE computer society conference on computer vision and pattern recognition*, IEEE, 1997, pp. 489–494.

[21] LeCun, Y.; Bengio, Y.; et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, volume 3361, no. 10, 1995: p. 1995.

[22] LeCun, Y.; Bottou, L.; Bengio, Y.; et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, volume 86, no. 11, 1998: pp. 2278–2324.

[23] McCulloch, W. S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, volume 5, 1943: pp. 115–133.

[24] Gallant, S. I.; et al. Perceptron-based learning algorithms. *IEEE Transactions on neural networks*, volume 1, no. 2, 1990: pp. 179–191.

[25] Dubey, S. R.; Singh, S. K.; Chaudhuri, B. B. A comprehensive survey and performance analysis of activation functions in deep learning. *arXiv preprint arXiv:2109.14545*, 2021.

[26] Kashyap, A. Math behind Perceptrons. Nov. 2019. Available from: `https://medium.com/@iamask09/math-behind-perceptrons-7241d5dadbfc`

[27] Almeida, L. B. Multilayer perceptrons. In *Handbook of Neural Computation*, CRC Press, 2020, pp. C1–2.

[28] Sayad, S. "Artificial Neural Network- Perceptron.

[29] Abadi, M.; Agarwal, A.; Barham, P.; et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[30] Paszke, A.; Gross, S.; Massa, F.; et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, volume 32, 2019.

[31] Harris, C. R.; Millman, K. J.; Van Der Walt, S. J.; et al. Array programming with NumPy. *Nature*, volume 585, no. 7825, 2020: pp. 357–362.

[32] Paszke, A.; Gross, S.; Chintala, S.; et al. Automatic differentiation in PyTorch.(2017). 2017.

[33] O'Shea, K.; Nash, R. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[34] Krizhevsky, A.; Sutskever, I.; Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, volume 25, 2012.

[35] Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[36] Szegedy, C.; Liu, W.; Jia, Y.; et al. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[37] He, K.; Zhang, X.; Ren, S.; et al. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[38] Goodfellow, I.; Bengio, Y.; Courville, A. *Deep learning.* MIT press, 2016.

[39] LeCun, Y.; Boser, B.; Denker, J. S.; et al. Backpropagation applied to handwritten zip code recognition. *Neural computation*, volume 1, no. 4, 1989: pp. 541–551.

[40] Lee, H.; Grosse, R.; Ranganath, R.; et al. Unsupervised learning of hierarchical representations with convolutional deep belief networks. *Communications of the ACM*, volume 54, no. 10, 2011: pp. 95–103.

[41] Deng, J.; Dong, W.; Socher, R.; et al. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[42] Kay, W.; Carreira, J.; Simonyan, K.; et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017.

[43] Google Scholar. Available from: `https://scholar.google.com/schhp?hl=en&as_sdt=0,5`

[44] IEEE Xplore. Available from: `https://ieeexplore.ieee.org/Xplore/home.jsp`

[45] Barhoumi, Y.; Ghulam, R. Scopeformer: n-CNN-ViT hybrid model for intracranial hemorrhage classification. *arXiv preprint arXiv:2107.04575*, 2021.

[46] Wang, M.; Xing, J.; Liu, Y. Actionclip: A new paradigm for video action recognition. *arXiv preprint arXiv:2109.08472*, 2021.

[47] Mehta, S.; Rastegari, M. Mobilevit: light-weight, general-purpose, and mobile-friendly vision transformer. *arXiv preprint arXiv:2110.02178*, 2021.

[48] Choe, J.; Shim, H. Attention-based dropout layer for weakly supervised object localization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2219–2228.

[49] Yun, S.; Oh, S. J.; Heo, B.; et al. Videomix: Rethinking data augmentation for video classification. *arXiv preprint arXiv:2012.03457*, 2020.

[50] Pan, J.; Lin, Z.; Zhu, X.; et al. St-adapter: Parameter-efficient image-to-video transfer learning. *Advances in Neural Information Processing Systems*, volume 35, 2022: pp. 26462–26477.

[51] Sandler, M.; Howard, A.; Zhu, M.; et al. MobileNetV2: Inverted Residuals and Linear Bottlenecks. 2019, `1801.04381`.

[52] Basulto-Lantsova, A.; Padilla-Medina, J. A.; Perez-Pinal, F. J.; et al. Performance comparative of OpenCV Template Matching method on Jetson TX2 and Jetson Nano developer kits. In *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, IEEE, 2020, pp. 0812–0816.

[53] Contreras Paucca, J. R. Diseño de un sistema de localización de un robot móvil basado en mapeo simultáneo.

[54] Zhou, Y.; Yang, K. Exploring TensorRT to Improve Real-Time Inference for Deep Learning. In *2022 IEEE 24th Int Conf on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, IEEE, 2022, pp. 2011–2018.

[55] Houlsby, N.; Giurgiu, A.; Jastrzebski, S.; et al. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*, PMLR, 2019, pp. 2790–2799.

[56] Mehta, S.; Rastegari, M. Mobilevit: light-weight, general-purpose, and mobile-friendly vision transformer. *arXiv preprint arXiv:2110.02178*, 2021.

[57] He, K.; Zhang, X.; Ren, S.; et al. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[58] Loshchilov, I.; Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[59] Goyal, P.; Dollár, P.; Girshick, R.; et al. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[60] Schuhmann, C.; Vencu, R.; Beaumont, R.; et al. LAION-400M: Open Dataset of CLIP-Filtered 400 Million Image-Text Pairs. *CoRR*, volume abs/2111.02114, 2021, 2111.02114. Available from: https://arxiv.org/abs/2111.02114

[61] Yu, J.; Wang, Z.; Vasudevan, V.; et al. Coca: Contrastive captioners are image-text foundation models. *arXiv preprint arXiv:2205.01917*, 2022.

[62] Li, K.; Wang, Y.; He, Y.; et al. Uniformerv2: Spatiotemporal learning by arming image vits with video uniformer. *arXiv preprint arXiv:2211.09552*, 2022.

[63] Liu, Z.; Ning, J.; Cao, Y.; et al. Video swin transformer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 3202–3211.

[64] Li, Y.; Wu, C.-Y.; Fan, H.; et al. Mvitv2: Improved multiscale vision transformers for classification and detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 4804–4814.

[65] Bertasius, G.; Wang, H.; Torresani, L. Is space-time attention all you need for video understanding? In *ICML*, volume 2, 2021, p. 4.

[66] Park, J.; Lee, J.; Sohn, K. Dual-path Adaptation from Image to Video Transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 2203–2213.

[67] Li, K.; Wang, Y.; Li, Y.; et al. Unmasked teacher: Towards training-efficient video foundation models. *arXiv preprint arXiv:2303.16058*, 2023.

[68] Tran, D.; Wang, H.; Torresani, L.; et al. A closer look at spatiotemporal convolutions for action recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2018, pp. 6450–6459.

[69] Feichtenhofer, C.; Fan, H.; Malik, J.; et al. Slowfast networks for video recognition. In *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 6202–6211.

[70] Wang, J.; Hu, X.; Gan, Z.; et al. Ufo: A unified transformer for vision-language representation learning. *arXiv preprint arXiv:2111.10023*, 2021.

[71] Zolfaghari, M.; Singh, K.; Brox, T. Eco: Efficient convolutional network for online video understanding. In *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 695–712.

[72] Li, Y.; Wu, C.-Y.; Fan, H.; et al. Mvitv2: Improved multiscale vision transformers for classification and detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 4804–4814.

[73] Carreira, J.; Zisserman, A. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 6299–6308.

[74] Lin, J.; Gan, C.; Han, S. Tsm: Temporal shift module for efficient video understanding. In *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 7083–7093.

[75] Chen, Y.; Kalantidis, Y.; Li, J.; et al. Multi-fiber networks for video recognition. In *Proceedings of the european conference on computer vision (ECCV)*, 2018, pp. 352–367.

[76] Neimark, D.; Bar, O.; Zohar, M.; et al. Video transformer network. In *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 3163–3172.

[77] Kumawat, S.; Verma, M.; Nakashima, Y.; et al. Depthwise spatio-temporal STFT convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 44, no. 9, 2021: pp. 4839–4851.

[78] Kondratyuk, D.; Yuan, L.; Li, Y.; et al. Movinets: Mobile video networks for efficient video recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 16020–16030.

[79] Zhang, N.; Lei, D.; Zhao, J. An improved Adagrad gradient descent optimization algorithm. In *2018 Chinese Automation Congress (CAC)*, IEEE, 2018, pp. 2359–2362.

[80] Ruder, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[81] 262588213843476. Kinetics Dataset Labels (name to ID). Available from: `https://gist.github.com/willprice/f19da185c9c5f32847134b87c1960769`

Appendix A

Some appendix

Table A.1: Kinetics-400 all classes [81].

Id	Name	Id	Name	Id	Name
0	abseiling	68	climbing tree	136	garbage collecting
1	air drumming	69	contact juggling	137	gargling
2	answering questions	70	cooking chicken	138	getting a haircut
3	applauding	71	cooking egg	139	getting a tattoo
4	applying cream	72	cooking on campfire	140	giving or receiving award
5	archery	73	cooking sausages	141	golf chipping
6	arm wrestling	74	counting money	142	golf driving
7	arranging flowers	75	country line dancing	143	golf putting
8	assembling computer	76	cracking neck	144	grinding meat
9	auctioning	77	crawling baby	145	grooming dog
10	baby waking up	78	crossing river	146	grooming horse
11	baking cookies	79	crying	147	gymnastics tumbling
12	balloon blowing	80	curling hair	148	hammer throw
13	bandaging	81	cutting nails	149	headbanging
14	barbequing	82	cutting pineapple	150	headbutting
15	bartending	83	cutting watermelon	151	high jump
16	beatboxing	84	dancing ballet	152	high kick
17	bee keeping	85	dancing charleston	153	hitting baseball
18	belly dancing	86	dancing gangnam style	154	hockey stop
19	bench pressing	87	dancing macarena	155	holding snake
20	bending back	88	deadlifting	156	hopscotch
21	bending metal	89	decorating the christmas tree	157	hoverboarding
22	biking through snow	90	digging	158	hugging
23	blasting sand	91	dining	159	hula hooping
24	blowing glass	92	disc golfing	160	hurdling
25	blowing leaves	93	diving cliff	161	hurling (sport)
26	blowing nose	94	dodgeball	162	ice climbing
27	blowing out candles	95	doing aerobics	163	ice fishing
28	bobsledding	96	doing laundry	164	ice skating
29	bookbinding	97	doing nails	165	ironing
30	bouncing on trampoline	98	drawing	166	javelin throw
31	bowling	99	dribbling basketball	167	jetskiing
32	braiding hair	100	drinking	168	jogging
33	breading or breadcrumbing	101	drinking beer	169	juggling balls
34	breakdancing	102	drinking shots	170	juggling fire
35	brush painting	103	driving car	171	juggling soccer ball
36	brushing hair	104	driving tractor	172	jumping into pool
37	brushing teeth	105	drop kicking	173	jumpstyle dancing
38	building cabinet	106	drumming fingers	174	kicking field goal
39	building shed	107	dunking basketball	175	kicking soccer ball
40	bungee jumping	108	dying hair	176	kissing
41	busking	109	eating burger	177	kitesurfing
42	canoeing or kayaking	110	eating cake	178	knitting
43	capoeira	111	eating carrots	179	krumping
44	carrying baby	112	eating chips	180	laughing
45	cartwheeling	113	eating doughnuts	181	laying bricks
46	carving pumpkin	114	eating hotdog	182	long jump
47	catching fish	115	eating ice cream	183	lunge
48	catching or throwing baseball	116	eating spaghetti	184	making a cake
49	catching or throwing frisbee	117	eating watermelon	185	making a sandwich
50	catching or throwing softball	118	egg hunting	186	making bed
51	celebrating	119	exercising arm	187	making jewelry
52	changing oil	120	exercising with ball	188	making pizza
53	changing wheel	121	extinguishing fire	189	making snowman
54	checking tires	122	faceplanting	190	making sushi
55	cheerleading	123	feeding birds	191	making tea
56	chopping wood	124	feeding fish	192	marching
57	clapping	125	feeding goats	193	massaging back
58	clay pottery making	126	filling eyebrows	194	massaging feet
59	clean and jerk	127	finger snapping	195	massaging legs
60	cleaning floor	128	fixing hair	196	massaging person's head
61	cleaning gutters	129	flipping pancake	197	milking cow
62	cleaning pool	130	flying kite	198	mopping floor
63	cleaning shoes	131	folding clothes	199	motorcycling
64	cleaning toilet	132	folding napkins	200	moving furniture
65	cleaning windows	133	folding paper	201	mowing lawn
66	climbing a rope	134	front raises	202	news anchoring
67	climbing ladder	135	frying vegetables	203	opening bottle

Table A.2: Kinetics-400 all classes (Continued) [81].

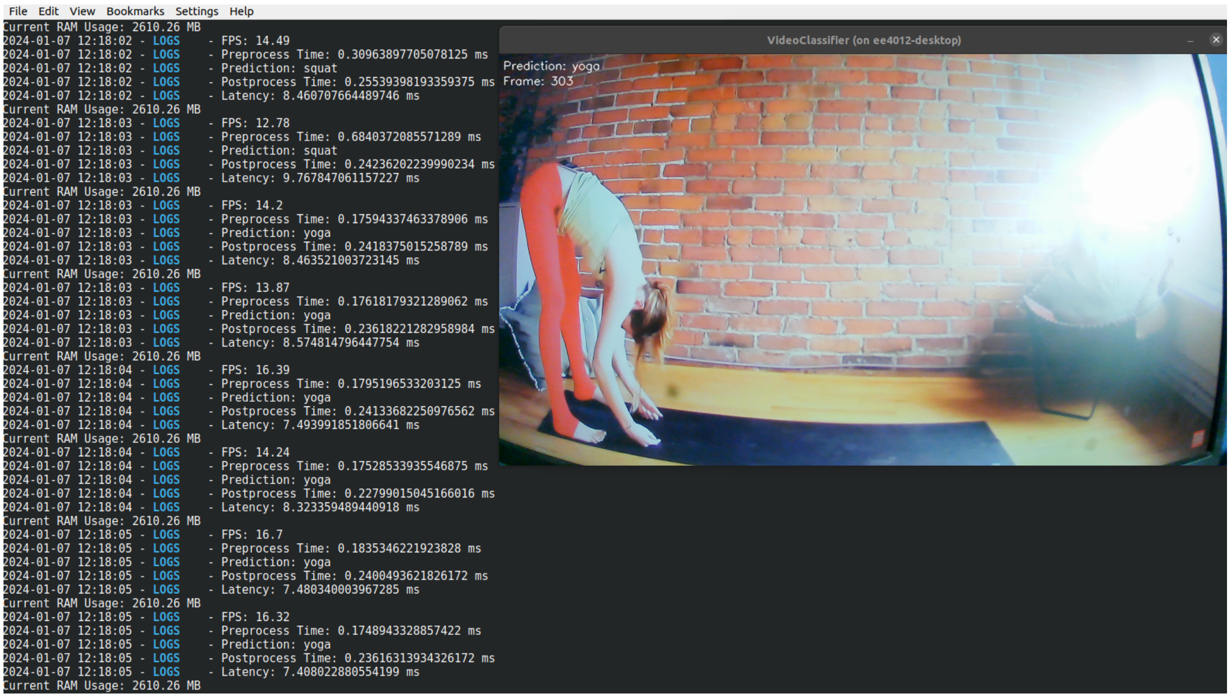
Id	Name	Id	Name	Id	Name
204	opening present	270	riding mechanical bull	336	surfing crowd
205	paragliding	271	riding mountain bike	337	surfing water
206	parasailing	272	riding mule	338	sweeping floor
207	parkour	273	riding or walking with horse	339	swimming backstroke
208	passing American football	274	riding scooter	340	swimming breast stroke
209	passing American football	275	riding unicycle	341	swimming butterfly stroke
210	peeling apples	276	ripping paper	342	swing dancing
211	peeling potatoes	277	robot dancing	343	swinging legs
212	petting animal (not cat)	278	rock climbing	344	swinging on something
213	petting cat	279	rock scissors paper	345	sword fighting
214	picking fruit	280	roller skating	346	tai chi
215	planting trees	281	running on treadmill	347	taking a shower
216	plastering	282	sailing	348	tango dancing
217	playing accordion	283	salsa dancing	349	tap dancing
218	playing badminton	284	sanding floor	350	tapping guitar
219	playing bagpipes	285	scrambling eggs	351	tapping pen
220	playing basketball	286	scuba diving	352	tasting beer
221	playing bass guitar	287	setting table	353	tasting food
222	playing cards	288	shaking hands	354	testifying
223	playing cello	289	shaking head	355	texting
224	playing chess	290	sharpening knives	356	throwing axe
225	playing clarinet	291	sharpening pencil	357	throwing ball
226	playing controller	292	shaving head	358	throwing discus
227	playing cricket	293	shaving legs	359	tickling
228	playing cymbals	294	shearing sheep	360	tobogganing
229	playing didgeridoo	295	shining shoes	361	tossing coin
230	playing drums	296	shooting basketball	362	tossing salad
231	playing flute	297	shooting goal (soccer)	363	training dog
232	playing guitar	298	shot put	364	trapezing
233	playing harmonica	299	shoveling snow	365	trimming or shaving beard
234	playing harp	300	shredding paper	366	trimming trees
235	playing ice hockey	301	shuffling cards	367	triple jump
236	playing keyboard	302	side kick	368	tying bow tie
237	playing kickball	303	sign language interpreting	369	tying knot (not on a tie)
238	playing monopoly	304	singing	370	tying tie
239	playing organ	305	situp	371	unboxing
240	playing paintball	306	skateboarding	372	unloading truck
241	playing piano	307	ski jumping	373	using computer
242	playing poker	308	skiing	374	using remote controller
243	playing recorder	309	skiing crosscountry	375	using segway
244	playing saxophone	310	skiing slalom	376	vault
245	playing squash or racquetball	311	skipping rope	377	waiting in line
246	playing tennis	312	skydiving	378	walking the dog
247	playing trombone	313	slacklining	379	washing dishes
248	playing trumpet	314	slapping	380	washing feet
249	playing ukulele	315	sled dog racing	381	washing hair
250	playing violin	316	smoking	382	washing hands
251	playing volleyball	317	smoking hookah	383	water skiing
252	playing xylophone	318	snatch weight lifting	384	water sliding
253	pole vault	319	sneezing	385	watering plants
254	presenting weather forecast	320	sniffing	386	waxing back
255	pull ups	321	snorkeling	387	waxing chest
256	pumping fist	322	snowboarding	388	waxing eyebrows
257	pumping gas	323	snowkiting	389	waxing legs
258	punching bag	324	snowmobiling	390	weaving basket
259	punching person (boxing)	325	somersaulting	391	welding
260	push up	326	spinning poi	392	whistling
261	pushing car	327	spray painting	393	windsurfing
262	pushing cart	328	spraying	394	wrapping present
263	pushing wheelchair	329	springboard diving	395	wrestling
264	reading book	330	squat	396	writing
265	reading newspaper	331	sticking tongue out	397	yawning
266	recording music	332	stomping grapes	398	yoga
267	riding a bike	333	stretching arm	399	zumba
268	riding camel	334	stretching leg		
269	riding elephant	335	strumming guitar		

A.1 Code

```

    def forward(self, x, T):

        #B= batch size,
        #L=Length of the input sequence,
        #C= Number of Chanels,
        #T= Time

        BT, L, C = x.size()
        B = BT // T

        x = self.fc1(x) # Downsampling to lower dim. space
        x = x.view(B, T, H, W).permute(0, 3, 1, 2).contiguous()

        #cudnn_enabled = torch.backends.cudnn.enabled
        #torch.backends.cudnn.enabled = cudnn_enabled
        #torch.backends.cudnn.enabled = DWCONV3D_DISABLE_CUDNN

        x = self.conv_3d(x)
        #torch.backends.cudnn.enabled = cudnn_enabled

        x = x.permute(0, 2, 3, 4, 1).contiguous().view(BT, L - 1)
        x = self.fc2(x) # Upsampling to original dim space
        return x
```

Listing A.1: The PyTorch-like pseudo-code of our Spatio-Temporal Adapter

A.2 Attachments


File Edit View Bookmarks Settings Help
VideoClassifier (on ee4012-desktop)
Prediction: yoga
Frame: 303

```
Current RAM Usage: 2610.26 MB
2024-01-07 12:18:02 - LOGS    - FPS: 14.49
2024-01-07 12:18:02 - LOGS    - Preprocess Time: 0.30963897705078125 ms
2024-01-07 12:18:02 - LOGS    - Prediction: squat
2024-01-07 12:18:02 - LOGS    - Postprocess Time: 0.25539398193359375 ms
2024-01-07 12:18:02 - LOGS    - Latency: 8.460707664489746 ms
Current RAM Usage: 2610.26 MB
2024-01-07 12:18:03 - LOGS    - FPS: 12.78
2024-01-07 12:18:03 - LOGS    - Preprocess Time: 0.6840372085571289 ms
2024-01-07 12:18:03 - LOGS    - Prediction: squat
2024-01-07 12:18:03 - LOGS    - Postprocess Time: 0.24236202239990234 ms
2024-01-07 12:18:03 - LOGS    - Latency: 9.767847061157227 ms
Current RAM Usage: 2610.26 MB
2024-01-07 12:18:03 - LOGS    - FPS: 14.2
2024-01-07 12:18:03 - LOGS    - Preprocess Time: 0.17594337463378906 ms
2024-01-07 12:18:03 - LOGS    - Prediction: yoga
2024-01-07 12:18:03 - LOGS    - Postprocess Time: 0.2418375015258789 ms
2024-01-07 12:18:03 - LOGS    - Latency: 8.463521003723145 ms
Current RAM Usage: 2610.26 MB
2024-01-07 12:18:03 - LOGS    - FPS: 13.87
2024-01-07 12:18:03 - LOGS    - Preprocess Time: 0.17618179321289062 ms
2024-01-07 12:18:03 - LOGS    - Prediction: yoga
2024-01-07 12:18:03 - LOGS    - Postprocess Time: 0.23618221282958984 ms
2024-01-07 12:18:03 - LOGS    - Latency: 8.574814796447754 ms
Current RAM Usage: 2610.26 MB
2024-01-07 12:18:04 - LOGS    - FPS: 16.39
2024-01-07 12:18:04 - LOGS    - Preprocess Time: 0.1795196533203125 ms
2024-01-07 12:18:04 - LOGS    - Prediction: yoga
2024-01-07 12:18:04 - LOGS    - Postprocess Time: 0.24133682250976562 ms
2024-01-07 12:18:04 - LOGS    - Latency: 7.493991851806641 ms
Current RAM Usage: 2610.26 MB
2024-01-07 12:18:04 - LOGS    - FPS: 14.24
2024-01-07 12:18:04 - LOGS    - Preprocess Time: 0.17528533935546875 ms
2024-01-07 12:18:04 - LOGS    - Prediction: yoga
2024-01-07 12:18:04 - LOGS    - Postprocess Time: 0.22799015045166016 ms
2024-01-07 12:18:04 - LOGS    - Latency: 8.323359489440918 ms
Current RAM Usage: 2610.26 MB
2024-01-07 12:18:05 - LOGS    - FPS: 16.7
2024-01-07 12:18:05 - LOGS    - Preprocess Time: 0.1835346221923828 ms
2024-01-07 12:18:05 - LOGS    - Prediction: yoga
2024-01-07 12:18:05 - LOGS    - Postprocess Time: 0.2400493621826172 ms
2024-01-07 12:18:05 - LOGS    - Latency: 7.480340003967285 ms
Current RAM Usage: 2610.26 MB
2024-01-07 12:18:05 - LOGS    - FPS: 16.32
2024-01-07 12:18:05 - LOGS    - Preprocess Time: 0.1748943328857422 ms
2024-01-07 12:18:05 - LOGS    - Prediction: yoga
2024-01-07 12:18:05 - LOGS    - Postprocess Time: 0.23616313934326172 ms
2024-01-07 12:18:05 - LOGS    - Latency: 7.408022880554199 ms
Current RAM Usage: 2610.26 MB
```

Figure A.1: Capture of live data output from Jetson Nano during the ongoing inference process with camera input. The left side displays the accurate prediction "Yoga," accompanied by details on preprocessing time, postprocessing time, latency, frames per second (FPS), and RAM usage.

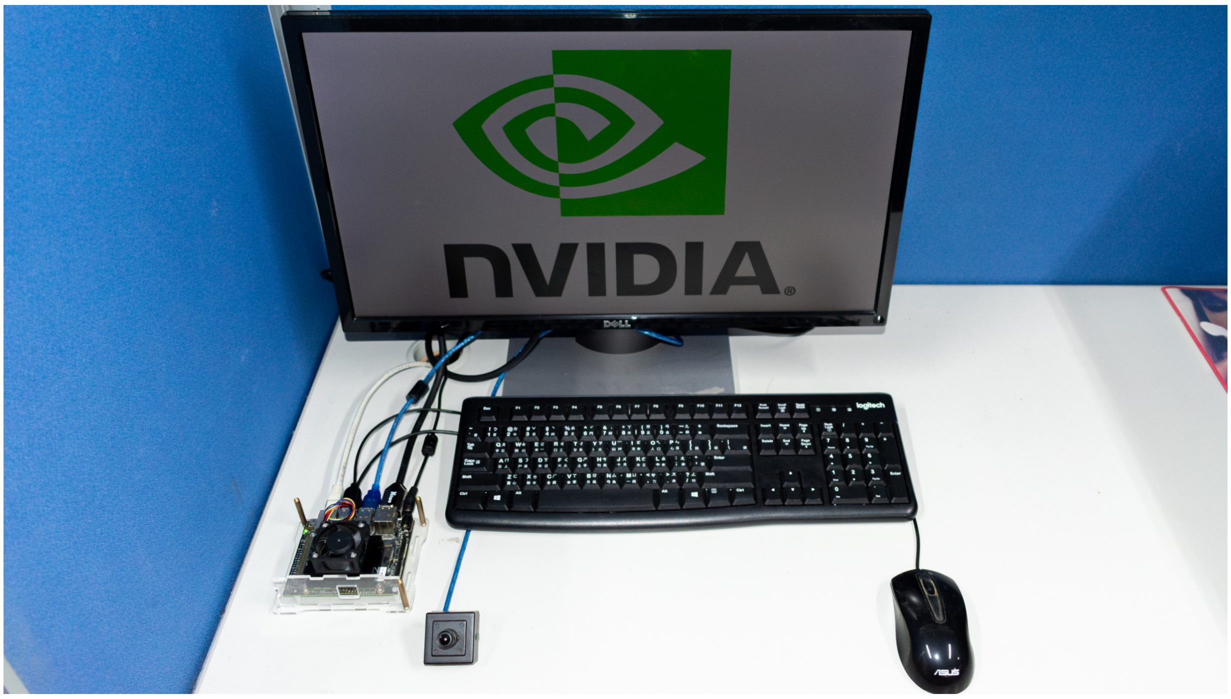


Figure A.2: Actual configuration setup of the Edge Device along with essential accessories, with particular emphasis on the camera utilized for real-time inference.