11-2021

Impact of Image Complexity on Early Exit Neural Networks for Edge Applications

Sharan Vidash Vidya Shanmugham
sv7190@rit.edu

Impact of Image Complexity on Early Exit Neural Networks for Edge Applications

Sharan Vidash Vidya Shanmugham

Impact of Image Complexity on Early Exit Neural Networks for Edge Applications

Sharan Vidash Vidya Shanmugham

November 2021

A Thesis Submitted
in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in
Computer Engineering

RIT | Kate Gleason College of Engineering

Department of Computer Engineering

Impact of Image Complexity on Early Exit Neural Networks for Edge Applications

Sharan Vidash Vidya Shanmugham

Committee Approval:

Prof Dr. Amlan Ganguly *Advisor* Date
Department of Computer Engineering

Prof Dr. Cory Merkel Date
Department of Computer Engineering

Prof Dr. Sai Manoj P D Date
George Mason University, Department of Electrical and Computer Engineering

Acknowledgments

I would like to thank my advisor Prof. Dr. Amlan Ganguly for his constant support and guidance throughout my academic and research work and always keeping me on track. I am thankful to Dr. Sai Manoj P D and Dr. Cory Merkel for being on my thesis committee and providing valuable insights. I am forever indebted to my family and friends who always believed in me and reminded me to hold on.

To amma, appa, mithil and my friends..Thank you for reminding me that there was light at the end of the tunnel. Your support and belief made this work possible.

Abstract

The advancement of deep learning methods has ushered in novel research in the field of computer vision as the success of deep learning methods are irrefutable when it comes to images and video data. However deep learning methods such as convolutional neural networks are computationally heavy and need specialized hardware to give results within a reasonable time. Early-exit neural networks offer a solution to reducing computational complexity by placing exits in traditional networks bypassing the need to compute the output of all convolutional layers. In this thesis, a reinforcement learning-based exit selection algorithm for early-exit neural networks is analyzed. The exit selection algorithm receives information about the previous state of the early-exit network to make decisions during runtime. The state of the early-exit network is determined by the previously achieved accuracy and inference time. A novel feature is proposed to improve the performance of the reinforcement learning network to make better decisions. The feature is based on the input image and attempts to quantify the complexity of every single input. The impacts of adding the new feature and potential performance improvements are documented. The testing is performed on two image classification datasets to record any variance in performance with the dataset. A scenario with the computation of the exit selection algorithm offloaded to a local edge server is also investigated by creating a simple analytical edge computing model. The decision-making rate is varied in this scenario and the potential differences in performance are documented.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Convolutional Neural Networks (CNNs) is a well-known deep learning architecture that is inspired by the natural visual perception process of living creatures. In the past few years, the advancement of convolutional neural networks has ushered in novel research in the field of computer vision as the success of convolutional neural networks is irrefutable when it comes to images and video data. CNNs are instrumental in solving problems of image classification, object detection, image translation, and image segmentation to name a few. Although CNNs have high performance, they also require a lot of computational power to provide results within reasonable times. Early-Exit Deep Neural networks are a type of neural network which have several exits within the network to provide faster inference time and lower computational load.

The objective of this thesis work is to understand Early-Exit Deep Neural Networks and investigate the impact of changes to the exit selection algorithm and decision update rate on an edge computing platform and propose a novel feature for a deep reinforcement learning exit selection algorithm. The decision update rate is the number of subsequent input frames for which an exit selected is updated from the exit selection algorithm. A Deep-Q-Network based reinforcement learning algorithm is used to choose the exits of a Early-Exit convolutional neural network in a dynamic environment. Based on the results observed in the work done by Wilder, John et

al.[1], it is observed that the detection of a contour (an outline especially of a curving or irregular figure) is more difficult to the human eye if there are more turning angles in the said contour. If the results were put in simpler words, the more zigs and zags in the contour, the more difficult the detection is. Since a CNN behaves similar to the human eye in image classification, the complexity of the image based on the curves would be a good estimation. This complexity would be a newly added feature to a Deep-Q-Network reinforcement learning algorithm designed for choosing early exits in a convolutional neural network. This work will be applied on modified convolutional neural networks with early exits and performance would be compared with the plain and feature added approach. The thesis also aims to investigate the impact of decision update rate in an edge computing platform where the Deep-Q-Network reinforcement learning algorithm would be isolated in a separate edge server to avoid the overhead caused by the Deep-Q-Network.

1.1 Motivation

The reinforcement learning network used as an exit selection algorithm for the early-exit neural network is proposed by using information such as the accuracy and inference time of the previous decision to estimate the future decision which would give the optimal performance. Although these features are important in deciding the next exit, in real-time scenarios, the image encountered also contributes to deciding if an earlier exit would be possible. Analyzing the image's features would contribute to making better decisions and lower the computational cost of the early-exit network. Although features could be extracted by running the image through convolutional layers, the approach would add additional computational complexity which would be counterproductive. This thesis proposes a method to estimate the image's features to correlate with the exit selection process by deriving a numerical value for each image which gives an idea of the complexity of a particular image. This value is supplied as

an input to the exit selection algorithm to make more efficient decisions. Although the exit selection algorithm designed in this approach is designed to be processed locally, future work on this approach could introduce additional computational complexity and would not be ideal for local processing. Although the network would perform better with future work, the computation of the exit selection algorithm would be better if offloaded to a dedicated server for computation. The traditional approach to offload computing would be to utilize cloud computing and offload the computation to a cloud server. Since the exit selection algorithm proposed requires lower computational resources and communication latency is required to be minimum, an edge computing platform is a viable option to offload computing in this case. A simple edge computing network is assumed and the effects of offloading computation to the computing server on edge are investigated in this thesis.

1.2 Contributions

The main contributions of this thesis are outlined below:

- Traditional CNNs have been modified and restructured to act as early-exit CNNs
- A lightweight feature is developed for improving the exit selection algorithm and performance improvements are demonstrated
- The impact of decision update rate which is an important aspect of edge computing based early-exit CNNs are investigated and documented
- The ideal decision update rate for the networks constructed are estimated and concluded

1.3 Document Structure

Chapter 2 discusses the background on various works in early-exit neural networks, applications of deep reinforcement learning, recent works in edge computing and an overview of various approaches used to estimate complexity of an image. Chapter 3 discusses the design of early-exit neural networks, the architecture of the exit selection algorithm and a comparison between an exit selection algorithm without the newly added feature and and exit selection algorithm with the newly added feature. Chapter 4 shows the analytical implementation of the exit selection algorithm on an edge computing platform and analyze the relationship between decision update rate and the performance of the exit selection algorithm. Chapter 5 gives the conclusion and directions for future work in the same topic.

Chapter 2

Background

Convolutional Neural Networks have been proven as a very effective approach in several image-related problems such as image segmentation, image classification, object detection, etc. Although CNN's or deep neural networks present a lot of advantages, they are resource hungry and require a lot of computational power with specialized hardware such as GPUs to run the algorithms under a reasonable time. Attempts to reduce the computational load of running deep neural networks have been recorded before. A branch of such attempts is Multi-exit DNNs or Early-Exit networks which design the architecture of the neural network to utilize branches that allow for earlier exit from the network. These approaches are based on the concept that easier to classify images need to go through lesser convolutional layers in order for accurate classification. Multi-exit DNN approaches shown in [2],[3],[4] use static threshold confidence at each exit to make the exit decision where the exit decision is sequential in nature. Work presented in [2],[5] uses similar threshold-based exit criteria where threshold comparison is performed at every exit. Skip gates and reinforcement learning were used in the work presented by [6] where certain groups of layers could be skipped during inference via a policy-based reinforcement learning algorithm. A dynamic approach for the same problem was shown in Epnet[7].

The following sections will explore popular works in the field of early exit convolutional networks, deep reinforcement learning, edge computing and image complexity

analysis.

2.1 Skipnet

Skipnet[6] uses a modified residual network that uses gating layers between sequential layers to make a decision on whether to go through the layer or skip the layer. The problem of dynamically skipping the layer is taken in the context of a sequential decision-making process and combines supervised learning and reinforcement learning to solve the problem. The gating layers are similar to ones used in Recurrent Neural Networks(RNNs). The output of the previous layer or a group of layers is mapped by the gating modules to a single binary decision on whether to skip or go through the subsequent layer or group of layers. Since the decision to skip the layers appears to be a sequential problem as one decision is made at each gating layer, the problem is formulated as a policy optimization through reinforcement learning. The approach allows for a dynamic approach to solving the problem but is limited in a distributed system as the reinforcement learning algorithm is built into the layers of the model itself and doesn't allow for the algorithm to work independently. This could pose an issue in devices where the additional complexity of adding the decision-making process is not feasible.

2.2 Branchynet

Branchynet[2] is based on the observation that earlier layers in convolutional neural networks will still be able to provide the same performance without compromise on the accuracy. The network is designed with branches from the original network with additional layers on them. The usual route for the input is considered to be the original network without the additional branches. These branches act as the early exits of the respective original convolutional neural network. The authors also propose

that the branches could be fitted with branches of themselves and act as a tree-like structure but limit their work within the scope of linear branching. These branches act as the early exits of the network and produce faster inference if the network takes one of the earlier exits. The decision to choose a particular exit is calculated by an entropy calculation or confidence which is checked against a given threshold to infer the decision. The approach is limited in the aspect of taking decisions which is static in nature and don't allow for the state of the system to be included in the decision-making process.

2.3 Epnet

EPNet[7] is another approach that uses a dynamic approach to creating earlier exits and choosing them. The exit branch design is lightweight and made to be compatible in attaching after any convolutional layer in the original network. This differs from other approaches where early exit branches are designed with the network and are not dynamic. The decision of choosing the location of the exit branch is designed as a Markov decision process and the training is formalized by using policy gradient without any sampling. Epnet does not include any state information such as the inference time of the previous run. Such state information monitors the effectiveness of the previous decision in energy conservation and allows for dynamic adaptability during run time. This hinders the Epnet from performing as an energy-effective process compared to the proposed approach.

2.4 Other Approaches

Authors in [4] propose a conditional deep learning network that attempts to place classifiers after every convolutional layer and make decisions on which layer to exit. The approach uses a static confidence measure from a classifier placed at every layer

and chooses to exit or continue the network based on the confidence produced. Work presented in [3] proposes a multi-exit DNN by using the softmax output of intermediate classifiers through the network to choose the exit. The approach uses the softmax output to determine the confidence of classification at that stage and thresholds are dynamically set during inference time. The thresholds for confidence are determined by taking into account the preferred accuracy degradation specified before run time. The approach is only tested on the MNIST dataset which does not allow for complex image inputs. "Early Exit CNNs" is proposed in [5] where the network has both softmax branches and confidence branches which learn independently whose outputs are used to determine the exit point when checking with a threshold value. However, during inference, the network has to flow through all consequent layers until a threshold is met to determine the output. The training approach in this work is used to train the network in our proposed approach using a simple loss function. The performance of this approach is used as a benchmark comparison to test the efficiency of our proposed approach.

2.5 Applications of Deep Reinforcement Learning

Deep reinforcement learning is an unsupervised deep learning algorithm that has proven highly effective in problems with complex state spaces. Deep Reinforcement learning presented in [8] is used to play Atari games and was proven to perform better than all previous approaches to the problem. The approach uses a convolutional neural network trained with a variant of Q-network and uses the raw pixels from the game to predict the rewards for future actions. The proposed method in this proposal uses a lightweight version of the same network without convolutional neural networks and only linear layers to predict the future actions which in this case would be the exit choosing process. Deep reinforcement learning has also been used in the navigation of a pedestrian environment in [9] with human-like actions for an autonomous

vehicle. The network also solves multiagent collision avoidance by predicting actions such as passing, crossing, and overtaking which allows the autonomous vehicle to safely navigate through crowded spaces. The deep reinforcement learning approach allows solving problems without much hyperparameter tuning usually seen in traditional supervised learning approaches. The network learns on its own by training in a simulated environment which allows for the network to accommodate unknown influencing factors by only focusing on maximizing the rewards. The reward formulation also allows for choosing which aspects need to be given more importance as in the proposed approach, importance is varied to accuracy and inference.

2.6 Recent works in Edge Computing

The general architecture of an edge computing network consists of placing edge devices between the end-user devices and the cloud computing layer. There are three major layers in this network architecture namely the terminal layer, boundary layer, and the cloud layer [10]. The terminal layer consists of all the devices connected to the edge network such as mobile terminals and IOT devices. These devices in the terminal layer both access data and upload data to the edge network. The boundary layer consists of all the devices which allow for the implementation of an edge network such as the access points, base stations, gateways, etc. This layer is the additional layer implemented to bring computing power closer to the terminal layer from the cloud computing layer. By placing an additional layer closer to the end-user, the transmission of data is suitable for real-time data analysis and also brings in more efficiency and security. The final layer is the cloud layer where the heavy computation takes place. The cloud layer handles tasks that are not suitable for service on the edge layer and tasks which can afford the extra communication delay incurred by offloading the processing to the cloud layer. One of the most important areas to handle in an edge computing environment is the offloading of tasks. The offloading

problem could be further divided into 2 sub-problems namely the offloading decision problem and the resource allocation problem. The offloading decision consists of choosing the optimal layer to offload the task and resource allocation consists of the required amount of resources in terms of memory and processing nodes. There is a tradeoff between task communication delay and the task processing load at all times which decides the offloading layer and amount of resource allocated for the specific task. Pushing AI applications away from local processing and towards off-device processing is not a task to be taken lightly due to several concerns on performance, cost, and privacy. The traditional method to implement this has been by enabling cloud computing to transfer large amounts of data from the device to the cloud layer for further analytics. However, such large amounts of data transfer incur heavy monetary costs and cause huge amounts of transmission delay which are not suitable for real-time applications. Privacy leakage is also a source of concern in this case. The alternative of local processing is not very appealing due to the high computing power requirements of AI applications which cause poor performance and energy efficiency. Edge computing shows a promising way out since the transmission delays are far lesser and computing servers in the edge layer do not require the high computational power of cloud layer computing servers. When running AI applications, there are various ways to accomplish the same task. An overview presented in [11] shows a model named Edge Intelligence which curates the various implementations of artificial intelligence on an edge computing platform. There are six layers in Edge Intelligence where the dependency on the cloud layer varies. Level 0 consists of both training and inference on the cloud layer with no local processing. On the other side of the spectrum, a level 6 consists of all processing on the device and no offloading is present. While deciding on the training of a network, the training could either be centralized, decentralized, or a hybrid of both methods. A centralized training model consists of all the training being done on the cloud layer and the data being fed from all the

edge nodes on the network. A decentralized model consists of each node training its own deep neural network with local data fetched by that device. The model with all data is achieved by communication between all nodes to share parameters. The hybrid model falls under levels 4 and 5 where model training is co-dependent on both the cloud layer and edge layer or partial reliance on the edge layer with the end node also contributing to training.

2.7 Analysing Complexity in Images

Various methods have been used in recent years to assert an image's perceived complexity. Most of this work has been in the domain of human visual perception and since the workings of a convolutional neural network used in the classification of objects have been inspired from the human visual perception, these works are a useful parallel. The complexity of an image could be understood in various ways and several works define it in their own way. In work done by [12], complexity is understood in programming terms where the shortest possible program to recreate the object from scratch is defined as the complexity of the object in question. Although this approach could be useful in defining the complexity of many objects, defining an image's complexity in that method would not be helpful as any image of the same size could be constructed with the same program. The complexity of an image is defined as the number of visual features or details in the image in [13]. This definition offers a good potential starting point for the method developed in this thesis. In experiments by [14], the difficulty of providing a visual description for the image defines the complexity of the image. When dealing with complexity, notable work is shown in [15] where the complexity of a dataset is assessed. The approach uses a complexity measure labeled as a cumulative spectral gradient(CSG). The probabilistic divergence between different classes of the dataset is analyzed in a clustering framework to derive the CSG. This CSG gives a numerical representation of the separability between

different classes of the dataset. The approach follows the idea that more variance in classes offers a much lower complexity of the dataset. This is due to the nature of classification problems as when a network trains on a dataset with more variance in classes, the network could learn to predict classes with higher confidence. Compared to other works in the same domain, this approach falls under complexity estimation methods designed for neural networks in particular and hence worth mentioning in this work. Although the CSG gives a good estimate of the complexity of a dataset, it is limited to a particular dataset and cannot be used to derive a complexity at a particular point during runtime.

Wilder, John et al. in [1] focuses on the visual difficulty of detecting a closed contour from a noisy background. The contours are assumed to be curves that are generated by a series of turning angles. The curves are represented using a gaussian distribution defining the distribution of the turning angles. The approach focuses on the unpredictability of the curve formed. The higher the unpredictability of the curve formed, the more complex the resulting contour is. This unpredictability is identified by the difference between the expected distribution of the turning angles and the actual observation of turning angles. The work focused on simple closed contours set against a noisy background which would be very difficult to implement for real-world images. Images taken from the real world would contain several contours and it is not easy to assume which would be the main contour to observe. The method also influences heavier computation which would not be a viable approach as the work in this thesis focuses on minimum overhead. Although there are limitations to exactly using the same approach, the approach sets as a good prelude by the way unpredictability in a contour offer information about the complexity. The unpredictability of a contour in this thesis work has been defined by the minimum points it requires to represent the specific contour.

Chapter 3

Exit Selection Algorithm with Deep Reinforcement Learning

This chapter first discusses the construction and modification of early-exit convolutional neural networks. This is followed by the design of the exit selection algorithm and the design of a new feature extracted from the input image to improve the performance of the exit selection algorithm. A comparison is also presented comparing the performance of the exit selection algorithm with and without the new feature.

3.1 Design of the Early-Exit Network

Traditional Mobilenet-v1[16] and Alexnet[17] have been modified to include early exits as additional branches from the original network and will be referred to as Exit-Mobilenet and Exit-Alexnet hereafter in this document. The Mobilenet- v1 architecture was designed to have lower model complexity and offers a significant reduction in the total computational load of the model. The model has also shown impressive performance in object detection problems providing real-time inference times in the COCO dataset[18]. The model's important novelty is in the use of depthwise separable convolutions. The depthwise separable convolution is shown in Fig 3.2. Each depthwise separable convolution consists of one depthwise convolution and one point convolution layer. The replacement of the traditional convolutional layer with a depthwise separable convolution offers 8-9 times lower computation in a 3x3 filter with a very small reduction of accuracy. The Mobilenet-v1 model is ideal in

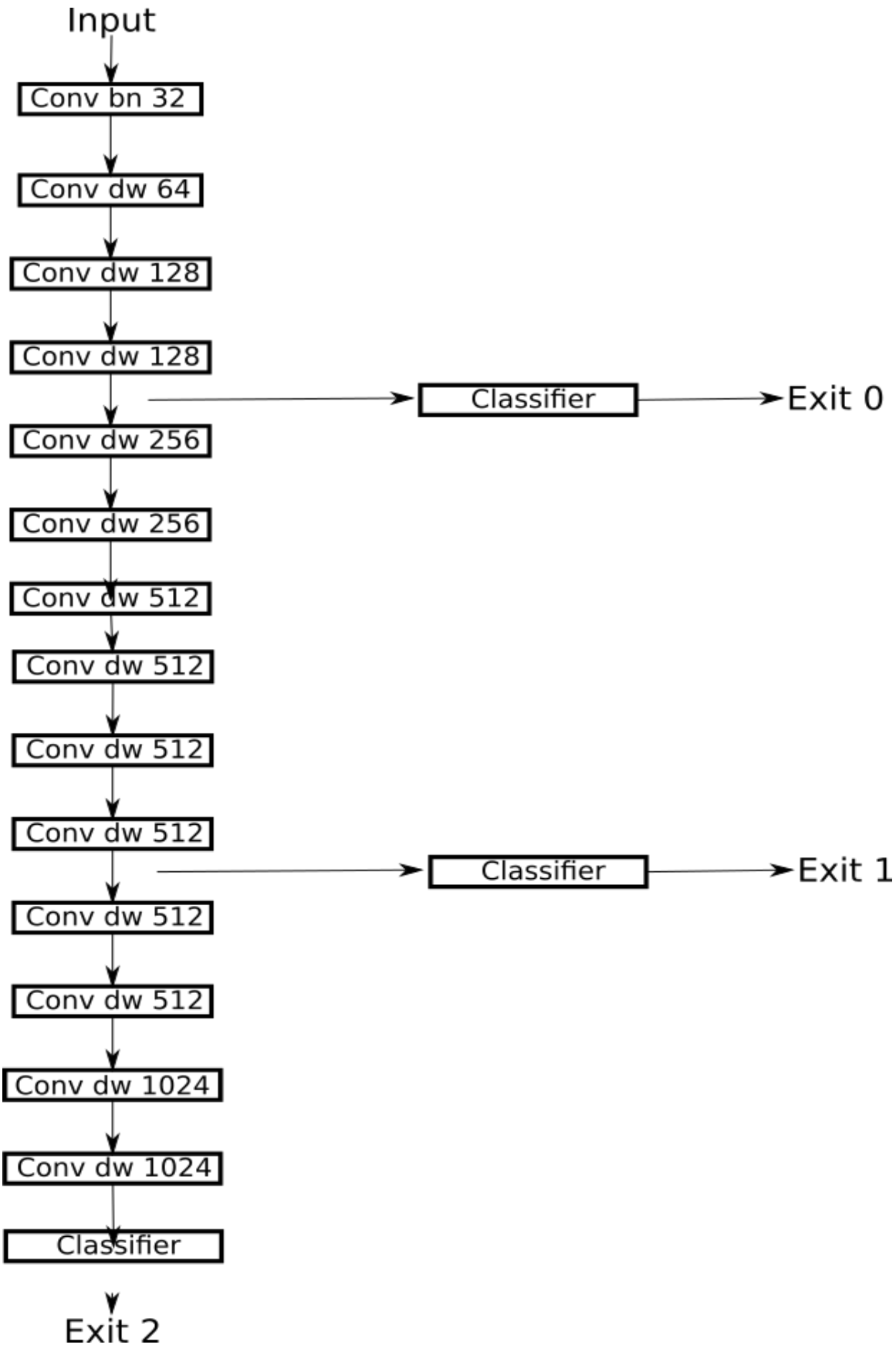

Figure 3.1: Exit-Mobilenet modified from Mobilenet v1

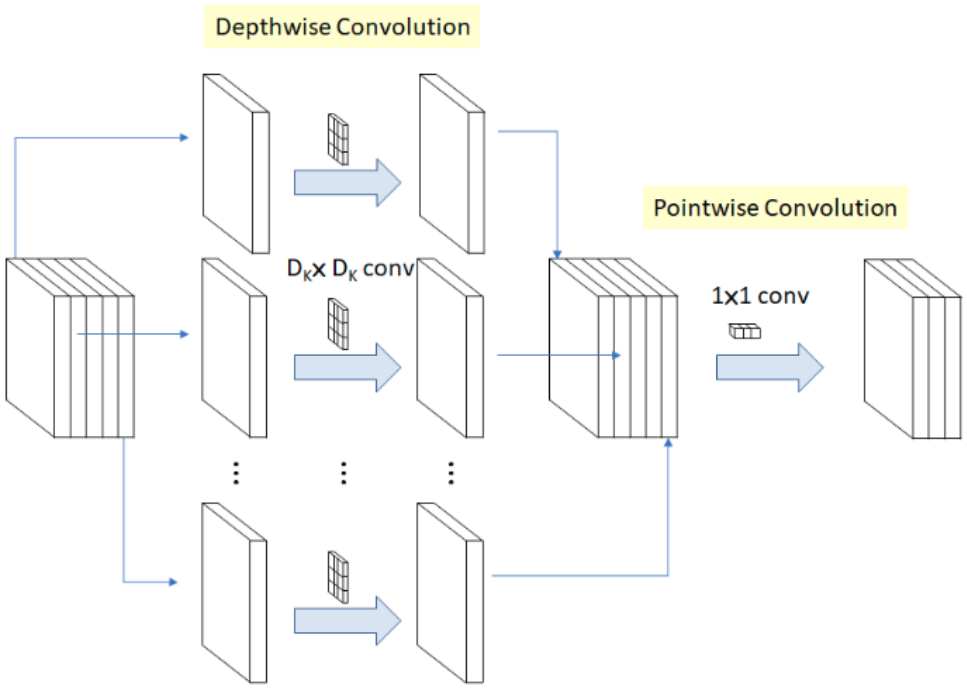

Figure 3.2: Depthwise Seperable Convolution

this work as the model already offers lower complexity than traditional models and offers a challenge to lower the computation in the model. The model has been modified with 2 early exits and is shown in Fig 3.1 where an overview of the depthwise separable convolutional layers are presented. The model has the initial layer as a convolutional layer with batch normalization applied to it to further reduce overfitting.

The Alexnet[17] presented in this work has been modified from the original architecture to be compatible with 32x32 images from the Cifar-10 dataset and referred to as Alexnet-32 in this work hereafter. The filter depths have been reduced for the convolutional layers and 2 additional convolutional layers with an output depth of 96 have been added. The model is also fitted with 2 early exits which are shown in Fig 3.3 where the overview of the convolutional layers in the network are presented. The first exit is added after 4 convolutional layers and the second exit is placed just before the last convolutional layer. The choice of exits ensures that exit selection is difficult to perform due to the tradeoff between inference time and accuracy. A

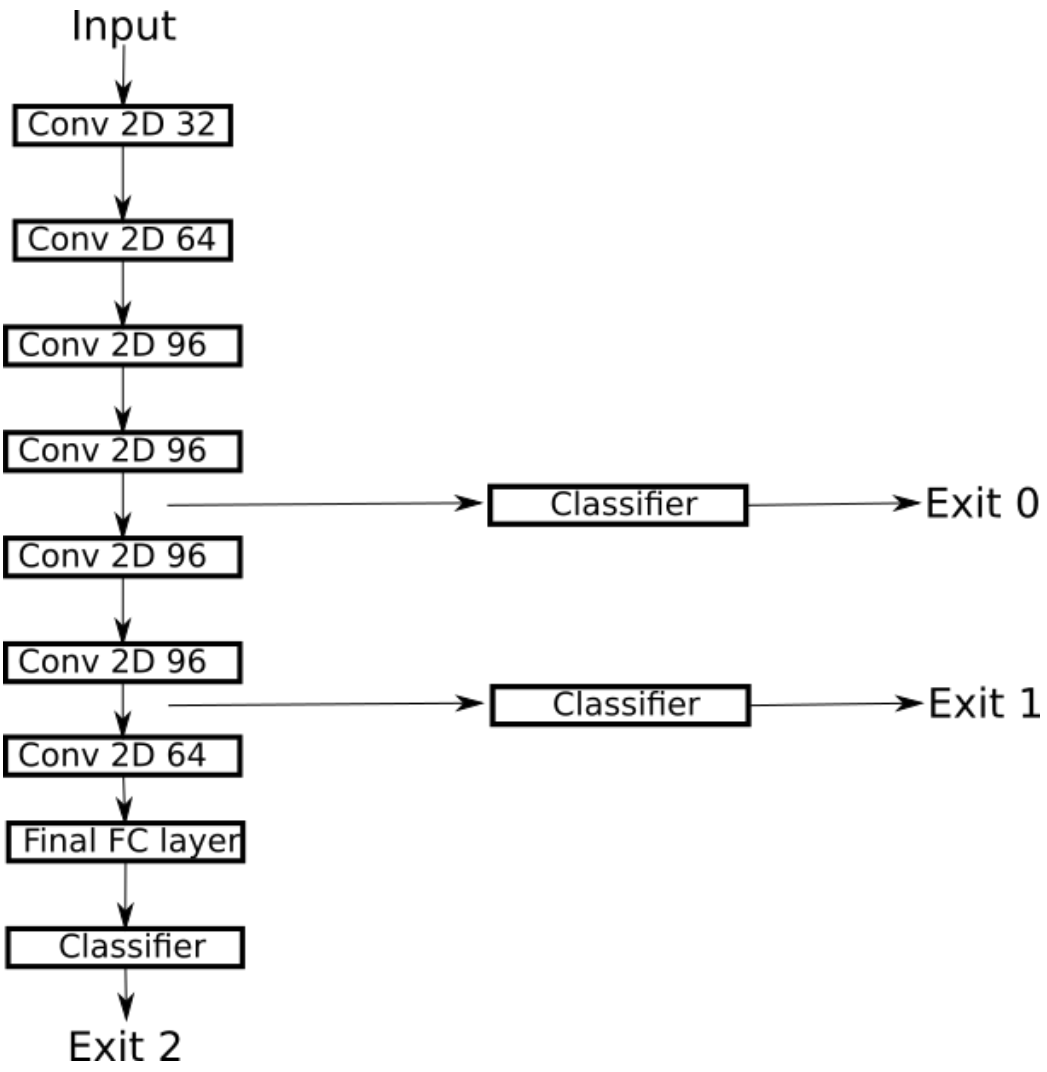

Figure 3.3: Exit-Alexnet modified from a deeper Alexnet

modified Resnet-56 presented in [5] named as Eenet-56 is also used for comparison. Exit branches are added at places in the original network where it would be possible to attain at least a minimum accuracy at the first exit and an intermediate exit which offers better accuracy at most times compared to the first exit while having a significant reduction in computation compared to the final exit. This model is referred to as Exit-EENet56 in this thesis hereafter. All early exit neural networks are designed and subsequently trained by the method observed in the work done by [5]. Each exit branch in the network has a linear layer with output neurons equal to the number of classes in the dataset and a softmax activation function to calculate the probability distribution of choosing each class.

$$p_k = \frac{e^{f_k}}{\sum_j e^{f_j}} \tag{3.1}$$

The softmax activation function is shown in eqn 3.1 and gives an array of values with the length as the total number of classes with each value between 0 and 1. The array is also normalized in a way that the sum of the values equals 1. This layer is the classification layer and is used in every exit branch to arrive at an intermediate classification output without passing through the entire network. The network is trained using a cumulative prediction loss across all exits to enable training of every exit branch. Traditional deep neural networks are trained using a single loss observed at the end of the classification layer placed as the last layer of the network. During the forward pass of the training, each exit has a prediction loss calculated by using the negative log-likelihood loss function.

$$L_i = -\log\left(p_{y_i}\right)$$

(3.2)

The negative log-likelihood loss shown in eqn 3.2 calculates the negative log values of the correct class and is applied to the softmax function output. The loss function gives lower values when the class confidence for the correct class is high and higher values when the class confidence of the correct class is low. This forces the network to make more right choices while also being more confident about the prediction. The prediction losses calculated during the forward pass are cumulated from all the branches and backpropagated. This ensures that all branches of the network are trained and selective branch training is avoided.The training is optimized by an adam optimization algorithm[19] that uses an adaptive learning rate instead of traditionally maintaining a single learning rate. The algorithm is an effective replacement to classic stochastic gradient descent and has been widely used in many problems of computer vision.

The early exit neural network has an input parameter in the forward function to receive the exit. The forward function has conditional statements built in to check against the exit before passing through a block of convolutional layers. This exit is supplied by the exit selection algorithm and the architecture of the exit selection network is discussed in 3.2

3.2 Architecture of the Exit Selection Network

The Deep Q Network based approach for exit selection in early-exit networks which is a reinforcement learning based approach directly builds upon the work presented in [20]. Reinforcement learning offers the advantage of being dynamic and maps the behavior of an environment accurately as it learns in a dynamic fashion. The learning is unsupervised and the model learns from an environment by making actions and

receiving rewards based on the reward function.This ensures that the model learns from an environment and the state of the environment during runtime instead of static values. The environment used for training the reinforcement learning algorithm here is the early exit network during inference. A Deep Q-Network(DQN) works by approximating a state-value function using a neural network in a Q-Learning framework. The state represents the previously acquired accuracy and inference time along with the previously taken decision. The exit selection network designed in this thesis is referred to as the DQN agent and makes decision during runtime. The DQN agent consists of a 2 hidden layer multi-layer perceptron(MLP). The action space for the DQN agent consists of all the possible exits for the early exit neural network and since all models in this work contain 2 early exits, the action space is the 3 possible exits any model contains.

3.2.1 Overview of the Deep-Q Network

The state-action value function known as the Q-function of a learned policy π, estimates the expected sum of rewards when taking action 'a' from state 's' and proceeding to follow the learned policy π from that point. This notion of a Q-function is the basis of Q-learning. The Q-function could also be simply expanded as the expected rewards for a single state-action pair. The Q function is defined by the Bellman equation shown in eqn 3.3.

$$Q^*(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q^*(s',a')\right] \tag{3.3}$$

This equation estimates the maximum return for the state-action pair is calculated as the sum of the immediate reward and the return obtained by following the same policy until the end of the episode. The return is simply the maximum reward from the estimated next state. The expectation calculated is calculated with both the

distribution of immediate rewards and probable next states in consideration. The equation shown in eqn [**?**] depicts the iterative version of the bellman equation where it shows that the optimal Q-function would be converged when the number of iterations tends to infinity.

$$Q_{i+1}(s,a) \leftarrow \mathbb{E}\left[r + \gamma \max_{a'} Q_i\left(s', a'\right)\right] \tag{3.4}$$

Working with the Q-function as a table containing all the values for each state-action pair is not a very efficient method for many problems. This is where a function approximator is used to estimate the Q-values. A neural network is a type of function approximator which when used for approximating the Q-values gives rise to the Deep Q-Learning algorithm. The network is trained by minimizing the loss for each step and the loss equation is shown in eqn 3.5.

$$L_i\left(\theta_i\right) = \mathbb{E}_{s,a,r,s' \sim \rho(.)}\left[\left(y_i - Q\left(s, a; \theta_i\right)\right)^2\right] \text{ where } y_i = r + \gamma \max_{a'} Q\left(s', a'; \theta_{i-1}\right) \tag{3.5}$$

Here, y_i is called the TD (temporal difference) target, and $y_i - Q$ is called the TD error. The distribution over transitions $\{s, a, r, s'\}$ collected from the environment is represented by the behaviour distribution ρ.

The expectation calculated during the loss calculation could be simplified and solved using a stochastic gradient descent algorithm. To improve the performance of the network, the work in [8] introduced a technique called Experience Replay which results in the network updates being more stable. This technique avoids using only the last transition which simplifies the problem to standard Q-Learning. There is a buffer of transitions of fixed size which stores a fixed number of previous transitions at each step of the data collection. This buffer is named the replay buffer and a mini-batch of transitions is sampled from this buffer to calculate the loss and its gradient instead of using just the last transition during training. This helps to improve the data efficiency by reusing each transition for several updates. The selection of transitions

from the buffer also allows for better stability due to using uncorrelated transitions in a batch. Once the DQN agent has been designed, the next step is the design of the reward function to setup training of the DQN agent.

3.2.2 Designing the reward function

The agent's performance during training is determined by the reward function and is the critical part of the agent's learning. The agent uses only the reward to optimize its decision making and hence designing an appropriate reward function holds very high importance. The reward function in this work needs to make sure the agent makes decisions that offer lower inference times while still giving a higher accuracy.

$$Reward = ((InfTime_{base} - InfTime)/InfTime_{base}) * \alpha + ((Acc - Acc_{base})/Acc_{base}) * \beta \tag{3.6}$$

The reward function used in this work is shown in Eqn 3.6. The reward function contains the sum of differences between the accuracy and inference time with their respective base values. The base values included in the reward function for both inference time and accuracy are values against which the current values are compared to. The reward function is set up in such a way that the higher the accuracy value, the higher the reward value, and the lesser the inference time, the higher the reward value. The base values also act as thresholds that determine the negative penalty. When accuracy drops below the accuracy threshold or when inference time is greater than the inference time threshold, the agent is penalized with a negative reward value. The individual rewards from accuracy and inference time are summed together which might skew the values as the accuracy and inference time are calculated in different scales. To avoid this disparity, both the accuracy and inference time are normalized to fall between 0 and 1 by dividing each difference with the base values. The steps till now comprise of building the basic exit selection algorithm with the inference time

and accuracy values determining the state of the early exit network. The next step is designing a new feature from the input image itself and is discussed in 3.3.

3.3 Adding the Image Information to the State

While deciding on the exit to choose for each frame, the accuracy, inference time and the exit chosen for the previous frame offer important information to make an exit selection. However, it is intuitive that the frame itself is also an important factor on which the success of an early exit would depend. We could hypothesize that including information about the image would help in increasing the performance of the exit selection algorithm by reducing inference time. The direct approach would be to build the DQN agent with added convolutional layers to extract image information similar to work in [8]. Although this approach has been tried and tested, the added convolutional layers add additional computational complexity and are not compatible with adding information such as accuracy, inference time, and previous exit. The next best approach would be to extract numerical information about an image which would help in selecting an exit. Work done in [1] offers a prelude to working on a similar approach where a closed contour's complexity is assessed using the unpredictability of the contour from a smooth distribution. The complexity calculated was with respect to human visual perception and since the convolutional neural network used in our work is derived from the workings of human perception, it offers a satisfactory parallel representation. Although the algorithm discussed offers an estimate of the complexity of a contour, when the algorithm needs to be ported for an image, the computation involves detecting contours in an image with very little granularity and computation for several points of each contour to determine the complexity. Therefore, we come up with our own estimate of an image's complexity by determining the amount of unpredictable distribution in all the contours detected in an image. This is performed by using the OpenCV framework to detect all contours in a respective image and count

Table 3.1: Contour complexity scores of images of fishes and birds

Image	Contour Complexity
Simple Bird	689
Complex Bird	1897
Simple Fish	1029
Complex Fish	1990
Silhouette fish	323
Silhouette bird	459

the minimum number of points required to represent each contour. OpenCV allows determining the minimum number of points required to represent a contour by using a simple contour chain approximation. This number allows is very useful as a contour with more unpredictability requires significantly more points to represent the contour. This could be observed in Fig 3.4 where a simple polygon such as a square requires

Figure 3.4: Comparing a square and an irregular shape

only 4 points to represent whereas a much more unpredictable shape requires more points. We extend this idea and sum the total number of points required for each contour as shown in Eqn 3.7.

$$\sum_{i=1}^{numContours} (len(ContourRepresentation_i)) \tag{3.7}$$

This number gives us an estimated complexity of the image and how many features are there in the image. We refer to this as the contour complexity and is included in the state information. Complying with the hypotheses that the contour complexity

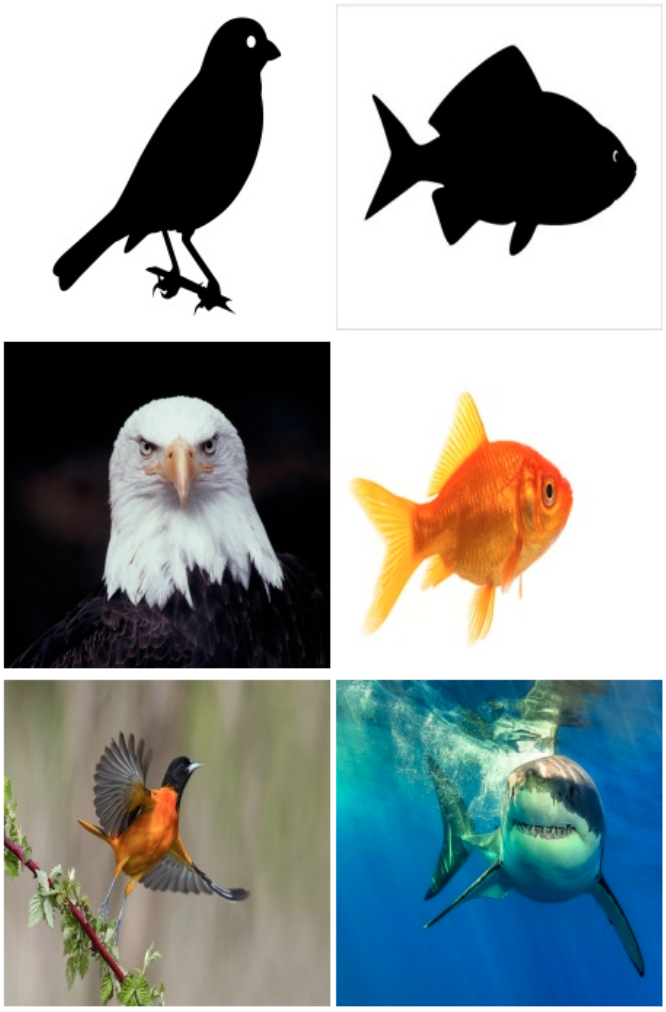

Figure 3.5: Various Images of birds and fish with varying visual complexity. First row: Silhouette Images, Second Row: Simple Images, Third Row: Complex Images

should show higher values with increased visual complexity of the images observed. Looking at Table 3.1 and Fig. 3.5, the silhouette images have the least amount of visual detailing and their complexity scores also concur with the same by showing complexity values between the range of 300-500. The images with the highest visual detail as seen in the complex fish and bird have a contour complexity in the range of 1800-2000. These observations provide a level of confidence to apply this method as an additional feature in the input state allowing for probable improvements in performance.

3.4 Datasets

Cifar-10[21] and Tiny-Imagenet[22] image classification datsets have been used to test the early exit neural networks in this thesis. The Cifar-10 dataset contains 60,000 32x32 colour images in 10 classes, with 6000 images per class. There are 50,000 training images and 10000 test images. The 10,000 images in dataset is split into 2 and the first 5000 are used for training the DQN agent while the second 5000 images are used for testing the DQN agent. The image samples from the Cifar-10 dataset are shown in Fig. 3.6.

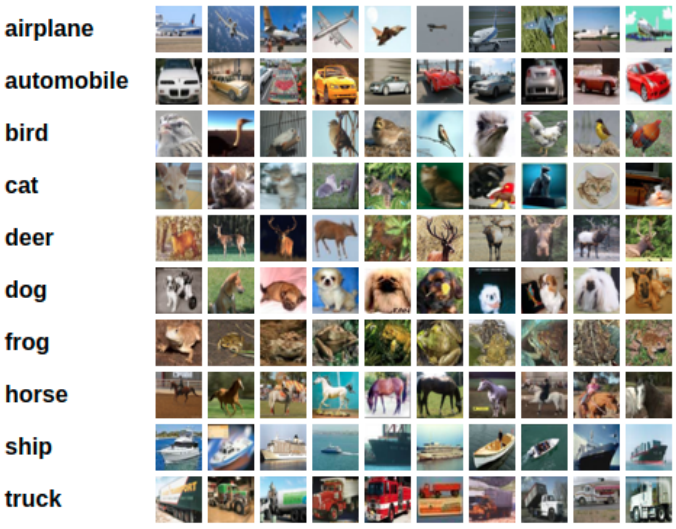

Figure 3.6: Various Images and classes from the Cifar-10 dataset

Tiny-ImageNet[22] is a dataset derived from the ImageNet dataset. The dataset contains 100,000 images of 200 classes (500 for each class) downsized to 64×64 colored images. The images are further downsized to 32x32 images for compatibility with the networks used in this work. Each class has 500 training images, 50 validation images, and 50 test images. The 10,000 validation images in the dataset are split into 2 and the first 5000 are used for training the DQN agent while the second 5000 images are used for testing the DQN agent. The image samples from the tiny-Imagenet dataset are shown in Fig. 3.7.

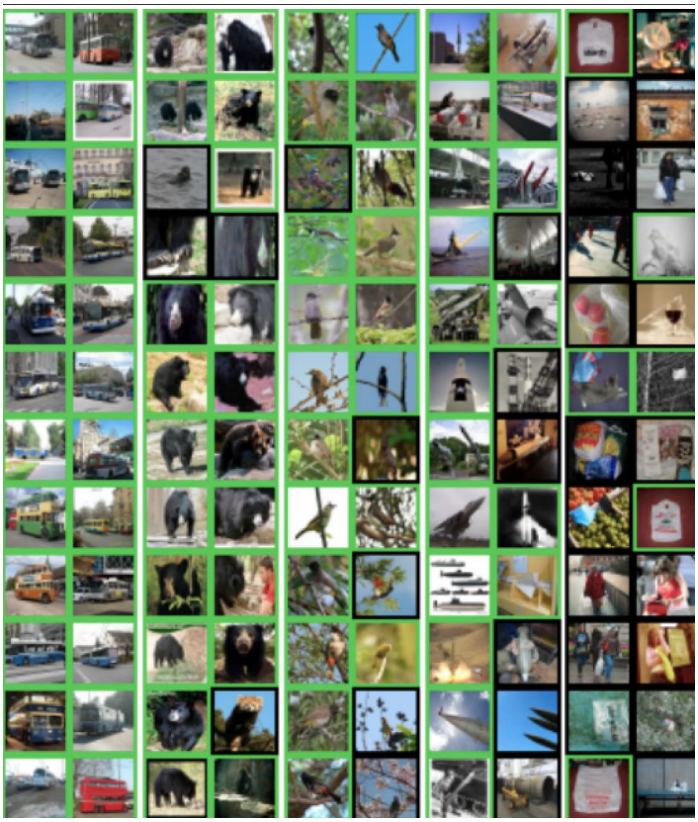

Figure 3.7: Various Images from the Tiny-Imagenet dataset

3.5 Comparison between DQN Agents with complexity and without complexity information

Three models are used for comparison and compared over 2 different datasets. All input images from both datasets have been scaled to 32x32 images. The parameters alpha and beta correspond to the importance given to accuracy and inference time during training of the exit selection algorithm. The parameters have been varied and the resulting performances have been recorded. The reward function used while training determines the behavior of the DQN agent and this reward function is modified with alpha and beta. There have been 3 major conditions where the reward function has been modified and the performance after testing noted down. The first case is where more importance is given to the inference time by altering alpha's value to be larger than beta. In this case, the agent should give more preference to inference time and therefore potentially decrease inference time. In the second case, alpha and beta are given equal values and the model is tested. In the third case, beta is set to be larger than alpha, thereby giving more importance to accuracy and make decisions that do not result in a loss of accuracy at all times. During inference, models have been tested under a batch size of 1 and 10 to investigate any potential improvements in performance. The batch size here represents the number of frames that were run through inference at the same time with the same exit taken for every frame. To avoid any irregularities during inference owing to background system optimizations, the inference times are calculated analytically in proportion to the exit percentages of each exit. To investigate the impacts of adding the contour complexity as a state, the exit selection algorithm is trained and tested with and without the new state. Both the model's performances are compared in terms of accuracy and inference time.

3.5.1 Mobilenet comparison between DQN agents with complexity and without complexity information

The Mobilenet-v1 architecture modified to include early exits has been trained with the method described in [5] and also with the approach in this thesis of using a DQN agent as the exit selection algorithm. The comparison is presented with testing on both Cifar-10 and Tiny-Imagenet dataset on batch sizes of 1 and 10 during inference.

3.5.1.1 Training Response Analysis

In Table 3.2 and Table 3.3, the model with complexity information and without complexity information respond differently to the training parameters set during training. The model without complexity information performs in accordance with the hypothesis that when importance is placed on the accuracy, the model's exit selection percentage of later exits increases and when importance is placed on the inference time, the exit selection percentage of earlier exits is more. The model with complexity information behaves differently from the traditional model where the model has lower inference time when lesser importance is placed on the inference time. Similar behavior is also observed in Table 3.4 where such behavior is observed in both models with and without complexity information. Although this behavior doesn't conform with the hypothesis, the explanation lies in the way the exits have been placed in the model. Exits 1 and 2 are very close in the accuracy achieved with a similarly small difference in the inference time obtained by choosing either exit. This causes rewards obtained from either exit to be close enough and causes decisions during testing which do not conform with the hypothesis. In Table 3.5 however when testing with a more difficult dataset and using a batch size of 10 for testing, both models conform to the hypothesis and show increasing inference time and higher later exit percentage with the importance being placed more on the accuracy.

Table 3.2: Exit Mobilenet comparison on Cifar-10 with batch size 1

Model	Batch Size	Alpha	Beta	Inf Time(ms)	Accuracy %	Exit 1 %	Exit 2 %	Exit 3 %
Mobilenet v1	1	NA	NA	0.1836	84.4	0.0	0.0	100.0
EENet Mobilenet	1	NA	NA	0.1403	84.26	2.3	97.7	0.0
Inf-time imp with complexity	1	2	0.2	0.1762	84	0.5	16.3	83.2
Inf-time imp without complexity	1	2	0.2	0.1421	84.04	0.0	100.0	0.0
Equal imp with complexity	1	1	1	0.1767	84.28	0.0	16.4	83.6
Equal imp without complexity	1	1	1	0.1487	84	0.0	84.0	16.0
Acc imp with complexity	1	0.2	2	0.1431	84.08	0.0	97.4	2.6
Acc imp without complexity	1	0.2	2	0.1836	84.3	0.0	0.0	100.0

Table 3.3: Exit Mobilenet comparison on Cifar-10 with batch size 10

Model	Batch Size	Alpha	Beta	Inf Time(ms)	Accuracy %	Exit 1 %	Exit 2 %	Exit 3 %
Mobilenet v1	1		NA	0.1836	84.4	0.0	0.0	100.0
EENet Mobilenet	1	NA	NA	0.1403	84.26	2.3	97.7	0.0
Inf-time imp with complexity	10	2	0.2	0.1565	84.24	0.0	60.0	40.0
Inf-time imp without complexity	10	2	0.2	0.1422	83.98	0.0	100.0	0.0
Equal imp with complexity	10	1	1	0.1422	83.98	0.0	100.0	0.0
Equal imp without complexity	10	1	1	0.1565	84.12	0.0	60.0	40.0
Acc imp with complexity	10	0.2	2	0.1493	84.24	0.0	80.0	20.0
Acc imp without complexity	10	0.2	2	0.1779	84.38	0.0	0.0	100.0

Table 3.4: Exit Mobilenet comparison on Tiny-Imagenet with batch size 1

Model	Batch Size	Alpha	Beta	Inf Time(ms)	Accuracy %	Exit 1 %	Exit 2 %	Exit 3 %
Mobilenet v1	1	NA	NA	0.1836	35.24	0.0	0.0	100.0
EENet Mobilenet	1	NA	NA	0.1548	34.84	3.5	59.6	36.9
Inf-time imp with complexity	1	2	0.2	0.1719	34.44	10.0	0.0	90.0
Inf-time imp without complexity	1	2	0.2	0.1478	34.76	28.6	6.2	65.2
Equal imp with complexity	1	1	1	0.1836	34.28	0.0	0.0	100.0
Equal imp without complexity	1	1	1	0.1068	33.98	66.0	0.0	33.4
Acc imp with complexity	1	1	3	0.1795	34.62	0.0	10.0	90.0
Acc imp without complexity	1	1	3	0.1253	34.42	49.4	1.7	48.9

Table 3.5: Exit Mobilenet comparison on Tiny-Imagenet with batch size 10

Model	Batch Size	Alpha	Beta	Inf Time(ms)	Accuracy %	Exit 1 %	Exit 2 %	Exit 3 %
Mobilenet v1	1	NA	NA	0.1836	35.24	0.0	0.0	100.0
EENet Mobilenet	1	NA	NA	0.1548	34.84	3.5	59.6	36.8
Inf-time imp with complexity	10	2	0.2	0.0692	34.7	100.0	0.0	0.0
Inf-time imp without complexity	10	2	0.2	0.0713	34.7	98.0	0.2	1.8
Equal imp with complexity	10	1	1	0.0694	34.7	99.8	0.0	0.2
Equal imp without complexity	10	1	1	0.1329	34.68	38.4	9.2	52.4
Acc imp with complexity	10	1	3	0.1422	35.92	0.0	100.0	0.0
Acc imp without complexity	10	1	3	0.1424	34.46	32.6	0.0	67.4

3.5.1.2 Model Performance Analysis

Looking at the results of the models performances on the Cifar-10 dataset, both models with and without complexity information achieve a performance similar to the comparison work in EENet and do not offer a significant performance increase. When observing the model's performance in testing on a Tiny-Imagenet dataset, both exit selection algorithms make better decisions than the comparison EENet Mobilenet model. In testing with a batch size of 1, the best performance is exhibited by the DQN agent without any complexity analysis by having an inference time of 4.52% lesser than the EENet Mobilenet model with no significant loss of accuracy. The best performance in both models is exhibited in a batch size of 10. With an accuracy loss of just 0.4%, the DQN agent has 53.94% reduced inference time without any overhead added. Comparing between the DQN agent with complexity and without complexity analysis, the DQN agent with complexity offers 2.94% lesser inference time than the model without it. The model with complexity performs better on a batch size of 10 where 10 images are run through inference at the same time. During this time, accuracy, inference time and complexity are averaged out over the 10 images when passed as a state for the subsequent 10 images. Averaging out the complexity over a series of images tends to give better results and shows that the model tends to perform better if it has information of the general trend in complexity rather than the complexity of a single frame.

3.5.2 EENet56 comparison between DQN Agent with complexity and without complexity

The EENet56 architecture derived from the EENet work has been trained with the EENet work [5] and also with the approach in this thesis of using a DQN agent as the exit selection algorithm. The comparison is presented with testing on both Cifar-10 and Tiny-Imagenet dataset on batch sizes of 1 and 10 during inference.

3.5.2.1 Training Response Analysis

In Tables 3.6 and 3.7, where the models' have been tested on the Cifar-10 dataset, there is no distribution between the exit percentages of the 3 exits chosen by any exit selection model. This simply means that the DQN agent prefers to choose the same exit irrespective of the input state. The comparison EENet56 model also behaves in a similar fashion by choosing Exit 1 at all times. When importance is placed on the inference time, both models choose Exit 1 as the default exit. This behavior could be explained by the accuracy obtained from the first exit which is lower than the other 2 exits but the inference time increase for the additional accuracy obtained doesn't justify choosing the later exits. When placing more importance on accuracy, the models choose Exit 3 as the default exit. This is due to Exit 2 not having a sufficient increase in accuracy to justify taking Exit 2. In Tables 3.8 and 3.9, there is a slight improvement in the exit selection distribution. The comparison EENet56 model doesn't offer much improvement from the original Resnet56 model and chooses predominantly the last exit. As observed in Tables 3.6 and 3.7, the middle exit is not preferred by any model and the decisions swing between Exit 1 or Exit 3. When importance is placed on inference time during training, the models choose Exit 1 as the default exit as it is the only exit offering a justified decrease in inference time. Although the exit distribution is very low, both models with and without complexity information conform to the training hypothesis and show exit selections in line with the parameters selected during training.

3.5.2.2 Model Performance Analysis

Looking at the results of the models' performances on the Cifar-10 dataset, both models with and without complexity achieve a performance the same as the comparison work in EENet56 and do not offer a significant performance increase. This behavior can be explained by looking at the exits taken by the EENet56 model. The model

Table 3.6: Exit EENet56 comparison on Cifar-10 with batch size 1

Model	Batch Size	Alpha	Beta	Inf Time(ms)	Accuracy %	Exit 1 %	Exit 2 %	Exit 3 %
Resnet56	1	NA	NA	0.4764	85.46	0.0	0.0	100.0
EENet 56	1	NA	NA	0.0632	71.24	100.0	0.0	0.0
Inf-time imp with complexity	1	2	0.2	0.0632	72.14	100.0	0.0	0.0
Inf-time imp without complexity	1	2	0.2	0.0632	72.14	100.0	0.0	0.0
Equal imp with complexity	1	1	1	0.4764	85.46	0.0	0.0	100.0
Equal imp without complexity	1	1	1	0.4764	85.46	0.0	0.0	100.0
Acc imp with complexity	1	0.2	2	0.4764	85.46	0.0	0.0	100.0
Acc imp without complexity	1	0.2	2	0.4764	85.46	0.0	0.0	100.0

Table 3.7: Exit EENet56 comparison on Cifar-10 with batch size 10

Model	Batch Size	Alpha	Beta	Inf Time(ms)	Accuracy %	Exit 1 %	Exit 2 %	Exit 3 %
Resnet56	1	NA	NA	0.4650	85.46	0.0	0.0	100.0
EENet 56	1	NA	NA	0.0632	71.24	100.0	0.0	0.0
Inf-time imp with complexity	10	2	0.2	0.0628	71.14	100.0	0.0	0.0
Inf-time imp without complexity	10	2	0.2	0.0628	71.14	100.0	0.0	0.0
Equal imp with complexity	10	1	1	0.4650	86.2	0.0	0.0	100.0
Equal imp without complexity	10	1	1	0.4650	86.2	0.0	0.0	100.0
Acc imp with complexity	10	0.2	2	0.4650	86.2	0.0	0.0	100.0
Acc imp without complexity	10	0.2	2	0.4650	86.2	0.0	0.0	100.0

Table 3.8: Exit EENet56 comparison on Tiny-Imagenet with batch size 1

Model	Batch Size	Alpha	Beta	Inf Time(ms)	Accuracy %	Exit 1 %	Exit 2 %	Exit 3 %
Resnet56	1	NA	NA	0.4650	40	0.0	0.0	100.0
EENet 56	1	NA	NA	0.4599	39.44	3.9	0.1	96.0
Inf-time imp with complexity	1	2	0.2	0.0880	21.9	21.5	78.5	0.0
Inf-time imp without complexity	1	2	0.2	0.0948	22	0.0	100.0	0.0
Equal imp with complexity	1	1	1	0.4764	39.98	0.0	0.0	100.0
Equal imp without complexity	1	1	1	0.4764	39.98	0.0	0.0	100.0
Acc imp with complexity	1	0.2	2	0.4764	39.98	0.0	0.0	100.0
Acc imp without complexity	1	0.2	2	0.4764	39.98	0.0	0.0	100.0

Table 3.9: Exit EENet56 comparison on Tiny-Imagenet with batch size 10

Model	Batch Size	Alpha	Beta	Inf Time(ms)	Accuracy %	Exit 1 %	Exit 2 %	Exit 3 %
Resnet56	1		NA	0.4650	40	0.0	0.0	100.0
EENet 56	1	NA	NA	0.4599	39.44	3.9	0.1	96.0
Inf-time imp with complexity	10	2	0.2	0.0628	21.4	99.8	0.2	0.0
Inf-time imp without complexity	10	2	0.2	0.0873	21.66	0.0	100.0	0.0
Equal imp with complexity	10	1	1	0.4650	39.86	0.0	0.0	100.0
Equal imp without complexity	10	1	1	0.4650	39.86	0.0	0.0	100.0
Acc imp with complexity	10	0.2	2	0.4650	39.86	0.0	0.0	100.0
Acc imp without complexity	10	0.2	2	0.4650	39.86	0.0	0.0	100.0

makes all decisions as the first exit and doesn't offer any room for improving the performance as the inference time is already very low. When observing the model's performance in testing on a Tiny-Imagenet dataset, both exit selection algorithms make better decisions than the comparison EENet56 model. In testing with a batch size of 1, the best performance is exhibited by the DQN agent including complexity as a state by having an inference time of 80.8% lesser than the EENet56 model. Although the model has a significant decrease in inference time, the model shows a very significant loss of accuracy. The DQN agent's decisions cause an 18.04% decrease in accuracy from the EENet56 model. The best performance in both models is exhibited in a batch size of 10. With almost the same accuracy loss as observed in the batch size of 1, the DQN agent shows an 86.34% reduction in inference time. Comparing between the DQN agent with complexity and without complexity analysis, the DQN agent with complexity offers 28.06% lesser inference time than the model without it.

3.5.3 Exit-Alexnet 32 comparison between DQN Agent with complexity and without complexity

The Alexnet-32 architecture modified to include early exits has been trained with the EENet work [5] and also with the approach in this thesis of using a DQN agent as the exit selection algorithm. The comparison is presented with testing with the Cifar-10 dataset on batch sizes of 1 and 10 during inference. The models have not been tested on the TIny-Imagenet dataset due to very low accuracies being noted during training since the model used here is smaller than traditional Alexnet and does not perform well on complex datasets.

3.5.3.1 Training Response Analysis

In Table 3.10, the model with complexity information takes Exit 1 more frequently than the model without complexity information when both models are trained with

more importance on inference time. This allows the model with complexity information to perform better and achieve lower inference time while maintaining good accuracy. The reduction in inference time is due to Exit 1 having considerably lesser inference time while having a good level of accuracy. When the models are trained with greater importance to accuracy, both models choose more of the later exits with an increase in inference time observed.

Table 3.10: Exit Alexnet-32 comparison on Cifar-10 with batch size 1

Model	Batch Size	Alpha	Beta	Inf Time(ms)	Accuracy %	Exit 1 %	Exit 2 %	Exit 3 %
Alexnet-32	1	NA	NA	0.1147	82.6	0.0	0.0	100.0
EENet Alexnet	1	NA	NA	0.0869	76.82	75.6	14.2	10.2
Inf-time imp with complexity	1	2	0.2	0.0805	76.18	99.7	0.1	0.2
Inf-time imp without complexity	1	2	0.2	0.0879	77.56	77.5	0.0	22.5
Equal imp with complexity	1	1	1	0.1141	82.6	0.0	0.0	100.0
Equal imp without complexity	1	1	1	0.1142	82.6	0.0	0.0	100.0
Acc imp with complexity	1	0.2	2	0.1043	82.26	0.0	82.1	17.9
Acc imp without complexity	1	0.2	2	0.1142	82.6	0.0	0.0	100.0

3.5.3.2 Model Performance Analysis

When observing the model's performance in testing on a Cifar-10 dataset, the exit selection algorithm including complexity analysis as a state makes better decisions than the comparison EENet Alexnet model and the DQN agent without complexity analysis when the best performance of both batch sizes is considered. In testing with a batch size of 1, the best performance is exhibited by the DQN agent with complexity analysis by having an inference time of 7.36% lesser than the EENet Alexnet model

Table 3.11: Exit Alexnet-32 comparison on Cifar-10 with batch size 10

Model	Batch Size	Alpha	Beta	Inf Time(ms)	Accuracy %	Exit 1 %	Exit 2 %	Exit 3 %
Alexnet-32	1	NA	NA	0.1147	82.6	0.0	0.0	100.0
EENet Alexnet	1	NA	NA	0.0869	76.82	75.6	14.2	10.2
Inf-time imp with complexity	10	2	0.2	0.0969	82.48	0.0	100.0	0.0
Inf-time imp without complexity	10	2	0.2	0.1147	82.94	0.0	0.0	100.0
Equal imp with complexity	10	1	1	0.1005	82.56	0.0	80.0	20.0
Equal imp without complexity	10	1	1	0.1147	82.94	0.0	0.0	100.0
Acc imp with complexity	10	0.2	2	0.1147	82.94	0.0	0.0	100.0
Acc imp without complexity	10	0.2	2	0.1147	82.94	0.0	0.0	100.0

with only a 0.83% loss of accuracy. The best performance in both models is exhibited in a batch size of 1. Comparing between the DQN agent with complexity and without complexity analysis, the DQN agent with complexity offers 8.4% lesser inference time than the model without it. Although the inference time is higher on the model with complexity, the model offers higher accuracy than the other 2 models. Although the best performance is offered in a batch size of 1, when comparing the difference in performance between the DQN agent with and without complexity, the model with complexity performs comparatively better in a batch size of 10. This behavior tends to concur with the behavior observed in Exit-Mobilenet where the model with complexity performs better comparatively in a batch size of 10. This further validates that averaging out the complexity over a series of images tends to give better results as the model tends to perform better if it has information of the general trend in complexity rather than the complexity of a single frame.

3.6 Exit Distribution Analysis

This section looks at the exit selections made by the DQN agent with complexity information in different datasets and different models. Certain scenarios shows different exit percentages for different exits when the DQN agent makes exit decisions for different early exit neural networks. Looking at the results in Table 3.12, the DQN agent shows more variety in the selection of exits when taking decisions for the Mobilenet and Alexnet models. There is distribution between exits 2 and 3 in mobilenet and similarly in alexnet where exit 2 has been given more importance in both models. This is due to exit 2 having sufficient accuracy while still having lower inference time than the last exit. There is a percentage of last exits being chosen only for estimated complex images. However for the EENet56 model, the DQN agent chooses the first exit irrespective of the input state. This behavior is caused due to EENet56 being a more computationally intensive model and the first exit offers enough accuracy for the Cifar-10 dataset which has only 10 classes. The behavior is not observed when the same model is tested on a Tiny-Imagenet dataset as it offers significantly more complexity due to a large number of classes(200) and inherently complex images. In this case the DQN agent chooses between Exits 1 and 2 as the accuracy observed is low in Exit 1 and Exit 2 is chosen more to offer lower compromise on accuracy. In the Tiny-Imagenet dataset, the DQN agent chooses 90% of the last exit in Mobilenet as a considerable accuracy is achieved only in the last exit. This does not occur for the EENet56 model even though the model could offer more accuracy in the last exit as the last exit has a very high inference time which does not justify the increase in accuracy. To conclude, the placement of exits in the network is a viable topic for future work as the DQN agent could potentially make better decisions with better exit placement.

Table 3.12: Different models showing different exit distributions with different datasets

Model	Dataset	Exit 1 %	Exit 2 %	Exit 3 %
Mobilenet Inf-time imp with complexity	Cifar10	0.0	60.0	40.0
EENet56 Inf-time imp with complexity	Cifar10	100.0	0.0	0.0
Mobilenet Inf-time imp with complexity	TIN	10.0	0.0	90.0
EENet56 Inf-time imp with complexity	TIN	21.5	78.5	0.0
Alexnet Acc imp with complexity	Cifar10	0.0	82.1	17.9

3.7 Overhead Analysis

The overhead of using the DQN agent to select exits for the early exit neural network is estimated in terms of the computational complexity represented as the total number of floating point operations(FLOPs) required to run inference of the DQN agent. The floating point operations used here are the total number of floating operations required to run one instance and not to be confused with floating point operations per second(FLOPS). The DQN agent requires a total of 90 FLOPs to run inference which is insignificant in comparison to the inference of the early exit networks such as Exit Mobilenet which requires up to 12,026,274 FLOPs. The average inference time for the DQN agent was 9.417 x 10^{-3} ms which is very low compared to the inference time of the early exit network and could further be offset by running exit selection on the edge which is discussed in detail in 4. The overhead for estimating the contour complexity involves detection of the contours in the image and counting the total number of points in each contour. When dealing with images of size 32x32, the average time taken to process the contour complexity of a single image was found to be 8.7022 x 10^{-3} ms. This number is very small in comparison to the inference time taken for each image. This further advocates the use of contour complexity as an efficient method to estimate the complexity of a given image without adding much overhead.

Overall the DQN agent with complexity information performs similar or better than

the DQN agent without complexity information by producing reduced inference times in select cases. This reduction in inference time is delivered without a significant reduction in accuracy. The DQN agents also show preference in maintaining the accuracy which is shown in the cases where equal importance was placed on inference time and accuracy. In these cases, the agents took decisions which gave greater accuracy and did not offer a significant reduction in inference time. The next chapter shows the implementation of the DQN agent and the early exit network in an analytical edge computing platform.

Chapter 4

Early-Exit Neural Network with Exit Selection on the Edge

This chapter discusses the implementation of the early-exit neural network and the exit selection algorithm in an edge computing environment. The chapter first discusses the overview of the network model and the information flow. Further the decision update interval is varied and the impacts on performance are investigated in an Exit-Alexnet network. An estimation of the overheads present in this approach are also calculated analytically and included in this chapter.

4.1 Early-Exit Neural Network for Edge Computing

An early-exit neural network is used in an image classification problem running on a moving edge node. The edge computing network consists of a moving edge node assumed to be an UAV in this thesis work, a computing server, and an access point to facilitate the communication between the edge node and the server. Fig 4.1 shows the various components of the edge computing network and the flow of decision-making through the network. The early-exit network placed within the edge node takes an input image and classifies it into one of the several classes depending on the dataset. All the early-exit neural networks used in this thesis work contain 2 intermediate exits and a final exit that goes through all the layers in the network. The objective of the early-exit neural network is to perform image classification with lesser computational load by taking early exits. The optimal exit is chosen by taking into consideration

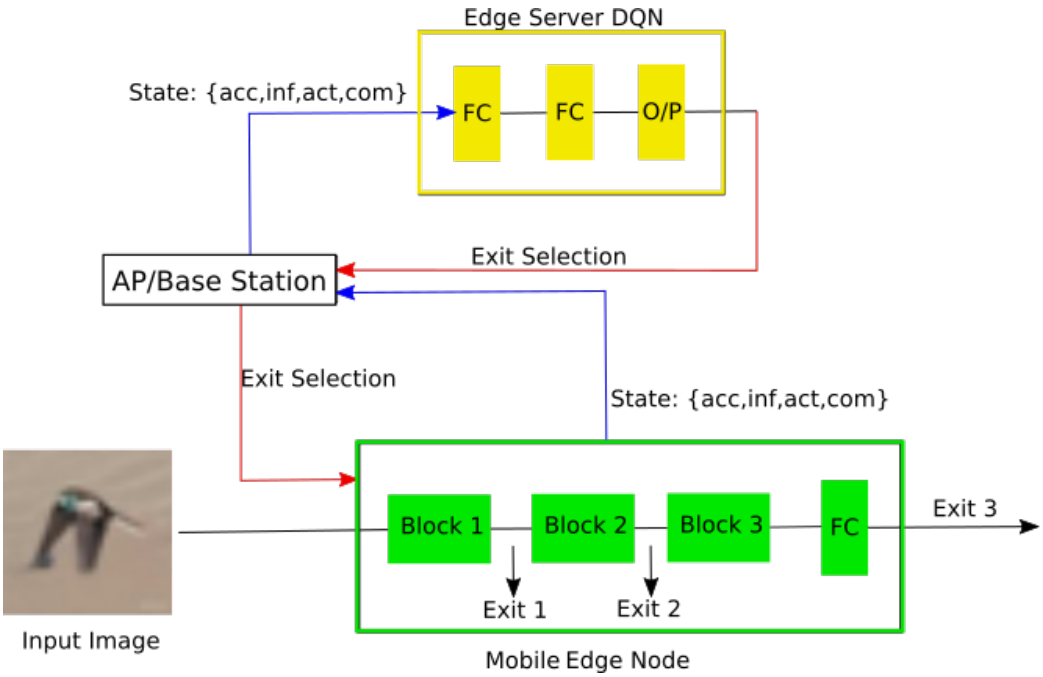

Figure 4.1: Overview diagram of the early-exit neural network on edge

observing the state of the early-exit neural network in terms of accuracy and inference time of the previous decision along with the previous exit taken. The approximate numerical complexity of the current frame is also taken into consideration as a novel addition to this thesis work. The reinforcement learning algorithm is placed in a separate server to reduce the computational load on the edge node. The algorithm used in this work is a small network and does not add heavy computation. However, the work could be expanded to use deeper networks and increase the performance of the target neural network. In such cases, the deployment of the setup in an edge computing environment offers the edge node much-needed respite from the excess computation. The information required to make an exit selection is observed from the edge node as a NumPy array and compressed into 128 bytes and communicated to the access point from where it is sent to the destination computing server. The reinforcement learning algorithm in the computing server utilizes this information and a feed-forward operation gives the required exit to be taken. This information consists of a single numerical value which has a size of 28 bytes. This decision is

once again routed through the access point and transferred back to the edge node to use that exit for the particular frame or for a series of subsequent frames. The flow of information through the network could be observed in Fig4.1. However, in an edge computing environment, communication delays play a crucial part in the performance of the neural network and have to be taken into consideration. Since the early-exit neural networks used in this work take decisions as an external input, it is not necessary to make decisions for every frame encountered. This could be used to make decisions for a batch of frames instead of every frame. We try to vary the decision update interval and study the impacts on the network in the following sections.

4.2 Relationship between Decision Update Interval and Inference Time

The graph in Fig. 4.2 shows the relationship between increasing decision update interval and the corresponding analytical inference time in Exit-Alexnet with exit selection by the trained DQN agent. The decision update interval is defined as the interval of images between which decisions are received from the DQN agent to the early exit neural network. The images are inferred at a batch size of 1 and the same exit is allocated for subsequent images until the new exit is updated depending on the decision update interval. For example, a decision update interval of 20 conveys that decisions are made only between every 20 images or frames. The Exit-Alexnet is tested on a Cifar-10 dataset with an image input size 32x32. There is a significant correlation between the decision update interval and inference time in both cases, with and without complexity information, when using the DQN agent to choose the exit. We observe an increase in inference time with an increase in the decision update interval. The increase in inference time is due to the DQN agent choosing the last

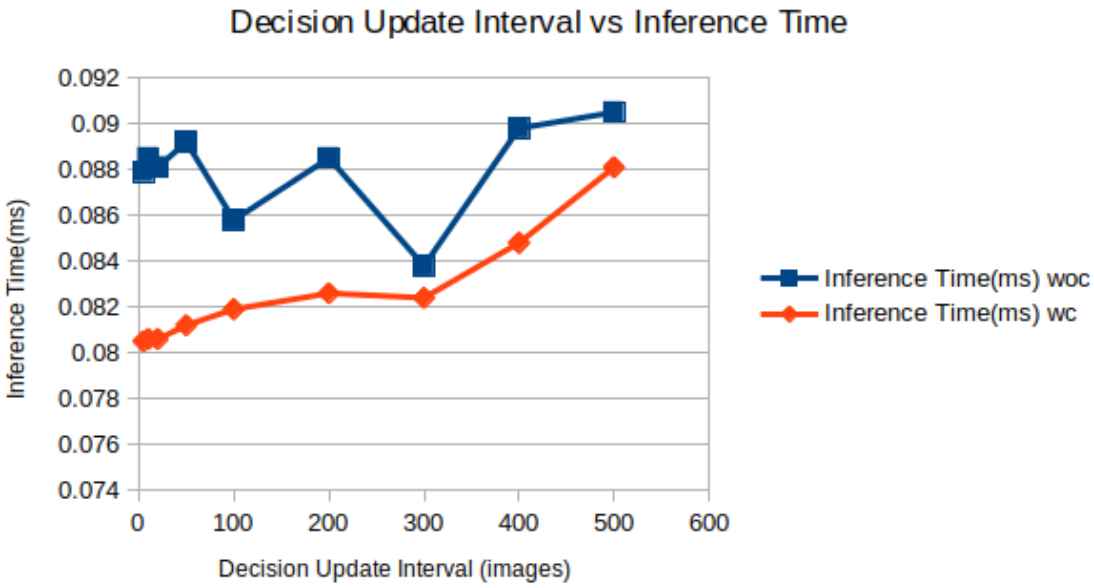

Figure 4.2: Decision Update Interval Vs Inference Time with and without complexity information

exit in a high percentage. The later exits cause added computation and cause an increase in average inference time. This could be observed from Table 4.1 where, when the network is forced to make the same decisions for all images, the network has an increase in inference time with later exits.

Table 4.1: Single Exit Selection Accuracy and Inference Time

Exit	Accuracy	Inference Time(ms)
0	76.16	0.0804
1	81.89	0.1022
2	82.6	0.1142

$$S_{input} = [Accuracy_{prev}, InfTime_{prev}, Exit_{prev}, Complexity_{prev}] \tag{4.1}$$

Equation 4.1 shows the constituents of each input state used to make decisions by the DQN agent. The input which contains the information of the previous decision as previous accuracy, previous inference time, previous Action Taken, and complexity of the Previous Image. These input parameters allow the DQN agent to make the

decision for a new batch of images. During inference, the DQN agent is supplied with a starting state which describes the last taken action as the last exit and quotes previous accuracy and previous inference to be both 0.

$$Reward = ((InfTime_{base} - InfTime)/InfTime_{base}) * \alpha + ((Acc - Acc_{base})/Acc_{base}) * \beta \tag{4.2}$$

Analyzing the reward function used to train the DQN agent in equation 4.2, we could observe that the accuracy and inference time are the only factors that give an increase in reward obtained for the decision. The inference time has an inference time threshold value of 3 ms and causes a reward penalty for greater inference time. Since the inference time is initialized with 0, the DQN agent does not give importance to inference time for the first decision. When looking at the accuracy of the previous image, the accuracy base is set to 50% accuracy and a previous accuracy of 0 gives a large penalty to the reward. This causes the DQN agent to take the next decision with more importance to accuracy and from the observation in Table 4.1, it could be observed that the last exit tends to give greater average accuracy. Setting this state as the starting allows for the DQN agent to make safer decisions and not compromise on the accuracy of the network. Although the DQN agent makes safer decisions, it comes with an increase in inference time and allows for lesser performance. With subsequent decisions, the DQN agent tends to make more decisions based on reducing inference time and therefore gives better performance. To further validate the hypothesis, the DQN agent including the complexity information of the image as an input state was tested on making decisions for a varying number of states, and the corresponding accuracy and analytical inference time were recorded. In all cases, the starting state was the same([0,0,2,0]).

Observing the graph in 4.3, we notice that the average inference time of the network decreases when the DQN agent makes more decisions while also maintaining

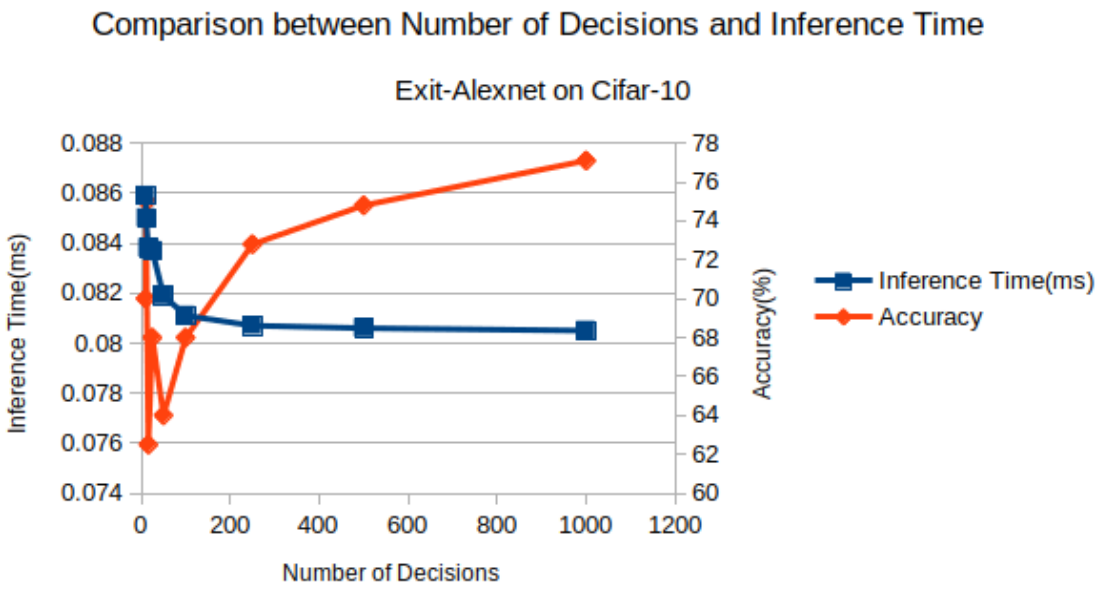

Figure 4.3: Number of Decisions Vs Inference Time and Accuracy

high accuracy. This further validates the hypothesis that the DQN agent offers a better performance with a higher number of decisions. The same behavior is exhibited in the chart in Fig.4.2 where more frequent decision making offers lower inference time with minimal reduction of accuracy. The analysis shows that there is a possibility to reduce the overhead incurred by choosing a higher decision update interval and prevent frequent communication between the DQN agent and the early exit network. However choosing a higher decision update interval causes reduction in performance of the DQN agent as the DQN agent makes better decisions with a higher number of decisions. The impact of communication delay is shown in section 4.3 which would help to balance the trade-off between performance and overhead.

4.3 Overhead Analysis

The overhead for this approach predominantly stems from the communication delay incurred due to communicating state information and exit information through the network.The calculation of communication delay begins with calculating the node

delay of the edge server. The node delay consists of processing delay, propagation delay, transmission delay, and queueing delay as shown in eqn 4.3.

$$N_d = Pc + Qd + Td + Pg \tag{4.3}$$

4.3.1 Processing Delay

The processing delay is the time taken for the access point to process each packet of data transferred. The average processing delays for different modes of communication is shown in Table 4.2. The processing delay in an LTE network is caused by the mobile substation delay which would be the access point in that network. The processing delays are derived from [23].

Table 4.2: Processing delays of different networks

Network Type	Uplink delay	Downlink delay	Total Delay
IP Router	0.0979 ms	0.0979 ms	0.1958 ms
LTE Network	21 ms	8 ms	29 ms

4.3.2 Transmission Delay

Transmission delay is calculated using the packet length and divided by the bit rate of the network as shown in eqn 4.4.

$$TD = L/R \tag{4.4}$$

The packet length during transmission from the edge node to the server is 128 bytes or 1024 bits. The packet length during transmission of action from the server to the edge node is 28 bytes or 224 bits. The total transmission delay from both ends is calculated and shown in Table 4.3.

Table 4.3: Transmission Delays

Network	Network bitrate	Uplink delay	Downlink delay	Total delay
Wifi 802.11g (2.4 Ghz)	54 Mbit/s	0.0189 ms	0.0042 ms	0.0231 ms
Wifi 802.11n (2.4 Ghz)	600 Mbit/s	0.00171 ms	0.00037 ms	0.0021 ms
Wifi 802.11ac (5 Ghz)	1.3 Gbit/s	0.0000984 ms	0.000022 ms	0.00012 ms
Lte (4g)	100 Mbit/s	0.01024 ms	0.00224 ms	0.01248 ms

4.3.3 Propoagation Delay

The propagation delay is calculated using the approximate distance between the edge node and the server and divided by the speed of propagation as shown in eqn 4.5

$$PD = \frac{B - A}{S} \tag{4.5}$$

With an average distance between the edge node and server as 100 meters and speed of propagation as 3 x 10^8 m/s which would give a propagation delay of 0.0003333 ms. the propagation delay is negligible in comparison to the transmission delay.

4.3.4 Queueing Delay

Average queuing delay in an m/m/1 server system could be estimated using the following equation in 4.6.

$$1/(\mu - \lambda) \tag{4.6}$$

The queuing delay at the access point is considered to be negligible since the load of the network is very low.

4.3.5 Total Delay and Effect on various Decision Update Intervals

The total delay in the network is calculated by adding up the transmission delay between an edge node and access point, and between the access point and edge server. The processing delay of the access point is also taken into account. Other delays are

negligible. In a 5g network with peak speeds of up to 20Gbps, the transmission delay is negligible and proposed 5g models have a total delay of about 2 ms including all other delays [24]. The total calculated delays are shown in Table 4.4.

Table 4.4: Total Communication Delays

Network	Total Delay
Wifi 802.11g (2.4 Ghz)	0.2882 ms
Wifi 802.11n (2.4 Ghz)	0.2042 ms
Wifi 802.11ac (5 Ghz)	0.1963 ms
Lte (4G)	29.049 ms
3GPP standard 5G	2 ms

Data is transferred between intervals and regulated by the decision update interval. Communication delay is incurred only at points where the exit decision is required. The maximum delay for any image would be the total communication delay incurred. However, since the inference time is averaged out over the length of all images, the communication delay is also averaged out over the length of all samples. The average communication delay for different decision update intervals is shown in Table 4.5. The delays are calculated by assuming a 5Ghz Wifi access point since the delays were found to be minimum for a 5Ghz Wifi access point. The relationship between decision update interval and different networks are shown in 4.4. The communication delay difference between the different networks is similar to the total delay difference observed in 4.4.

Overall the analysis shows that the offloading of the DQN agent to an edge computing network is viable due to the lower communication delays observed in higher decision update intervals. The trade-off between communication delay and average inference time could be achieved by choosing a decision update interval lesser than 300. The communication delay could also be potentially mitigated by using the previously received exit while waiting for the newer exit. This is essentially masking the latency by performing computation and is commonly used in message passing interface models for parallel computing.

Table 4.5: Communication Delays incurred for different Decision Update Intervals

Decision Update Interval	Average Communication Delay
1	0.1963
5	0.03926
10	0.01963
20	0.00982
50	0.00393
100	0.00196
200	0.00098
300	0.00065
400	0.00049
500	0.00039

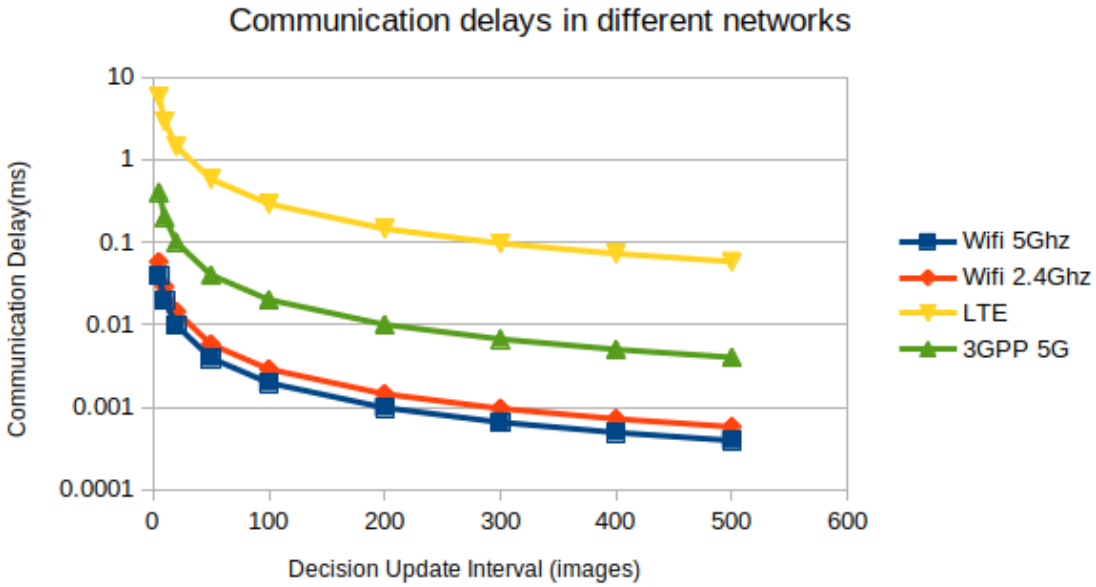

Figure 4.4: Relationship between decision update interval and communication delays in different networks

Chapter 5

Conclusions

5.1 Conclusion

Image features have been captured by a novel approach involving counting the representation of contours in an image and added as an input feature to a Deep Q-Network based exit selection algorithm. The newly added feature reduces inference time in certain cases when tested with 2 different datasets and over 3 different multi-exit convolutional neural networks. The new added feature offers up to 28% decrease in inference time when compared to the DQN agent without the new information. The experiments conclude that the newly added feature could be a valuable addition with negligible overhead when used under right circumstances. The investigation into varying decision update interval when implemented in an edge computing network shows that a lower decision update interval offers better performance and ideal performance is observed around rates of 100 to 300 and higher intervals are not recommended.

5.2 Future Work

The datasets used in this work do not have images that do not have a temporal correlation. When using the exit selection algorithm proposed, the complexity calculated for a previous state would prove more efficient in scenarios where the previous image has a close correlation with the current image. This would enable the net-

work to make fewer decisions as the decisions made for a single frame would be more applicable to subsequent frames. This work would benefit from using a temporally correlated dataset in the future. The edge computing framework used in this work has been purely analytical in nature and might not be the ideal representation of a real edge computing network. The experiments related to the edge computing platform implementation of the exit selection algorithm could give additional insights when performed on an actual edge computing network. The reinforcement learning based exit selection model proposed in this work is trained once before applying the model during run-time. The model's robustness could be improved by making the model adaptive to new changes in the environment by using online training similar to the model implemented in [25]. This could be achieved by continuously training the model with new data observed by the mobile edge node and use the features to update the exit selection model.

Bibliography

[1] J. Wilder, J. Feldman, and M. Singh, "Contour complexity and contour detection," *Journal of vision (Charlottesville, Va.)*, vol. 15, no. 6, pp. 6–6, 2015.

[2] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks." IEEE, 2016, pp. 2464–2469.

[3] K. Berestizshevsky and G. Even, *Dynamically Sacrificing Accuracy for Reduced Computation: Cascaded Inference Based on Softmax Confidence*. Cham: Springer International Publishing, 2019, pp. 306–320.

[4] P. Panda, A. Sengupta, and K. Roy, "Conditional deep learning for energy-efficient and enhanced pattern recognition," *CoRR*, vol. abs/1509.08971, 2015. [Online]. Available: http://arxiv.org/abs/1509.08971

[5] E. Demir, "Early-exit convolutional neural networks," Master's thesis, Middle East Technical University, 2019.

[6] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez, *SkipNet: Learning Dynamic Routing in Convolutional Networks*. Cham: Springer International Publishing, 2018, pp. 420–436.

[7] X. Dai, X. Kong, and T. Guo, "Epnet: Learning to exit with flexible multi-branch network," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, ser. CIKM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 235–244. [Online]. Available: https://doi.org/10.1145/3340531.3411973

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[9] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," *CoRR*, vol. abs/1703.08862, 2017. [Online]. Available: http://arxiv.org/abs/1703.08862

[10] K. Cao, Y. Liu, G. Meng, and Q. Sun, "An overview on edge computing research," *IEEE access*, vol. 8, pp. 85 714–85 728, 2020.

[11] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[12] A. N. Kolmogorov, "Three approaches to the quantitative definition of information," *International Journal of Computer Mathematics*, vol. 2, no. 1-4, pp. 157–168, 1968. [Online]. Available: https://doi.org/10.1080/00207166808803030

[13] J. G. Snodgrass and M. Vanderwart, "A standardized set of 260 pictures: norms for name agreement, image agreement, familiarity, and visual complexity." *Journal of experimental psychology. Human learning and memory*, vol. 6 2, pp. 174–215, 1980.

[14] C. Heaps and S. Handel, "Similarity and features of natural textures," *Journal of Experimental Psychology: Human Perception and Performance*, vol. 25, pp. 299–320, 04 1999.

[15] F. Branchaud-Charron, A. Achkar, and P.-M. Jodoin, "Spectral metric for dataset complexity assessment," 2019.

[16] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[18] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," 2015.

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[20] A. Vashist, S. V. Vidya Shanmugham, S. M. P D, and A. Ganguly, "Dqn based exit selection in multi-exit deep neural networks for applications targeting situation awareness," in *IEEE International Conference on Consumer Electronics (ICCE)*, 2022.

[21] A. Krizhevsky, "Learning multiple layers of features from tiny images." [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[22] Y. Le and X. Yang, "Tiny imagenet visual recognition challenge," 2015.

[23] A. Kurian, "Latency analysis and reduction in a 4g network," Master's thesis, TU Delft, 2018.

[24] S. Jun, Y. Kang, J. Kim, and C. Kim, "Ultra-low-latency services in 5g systems: A perspective from 3gpp standards," *ETRI journal*, vol. 42, no. 5, pp. 724–736, 2020.

[25] S. Shukla, P. D. Sai Manoj, G. Kolhe, and S. Rafatirad, "On-device malware detection using performance-aware and robust collaborative learning," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 967–972.