NETWORK ATTACK CLASSIFICATION WITH FEW-SHOT LEARNING METHODS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

İSMAİL TÜZÜN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2022

Approval of the thesis:

NETWORK ATTACK CLASSIFICATION WITH FEW-SHOT LEARNING METHODS

submitted by **İSMAİL TÜZÜN** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** _____

Assoc. Prof. Dr. Pelin Angın
Supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Pınar Karagöz
Computer Engineering, METU _____

Assoc. Prof. Dr. Pelin Angın
Computer Engineering, METU _____

Assoc. Prof. Dr. Ahmet Burak Can
Computer Engineering, Hacettepe University _____

Date: 14.09.2022

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: İsmail Tüzün

Signature :

ABSTRACT

NETWORK ATTACK CLASSIFICATION WITH FEW-SHOT LEARNING METHODS

Tüzün, İsmail
M.S., Department of Computer Engineering
Supervisor: Assoc. Prof. Dr. Pelin Angın

September 2022, 74 pages

As the number of devices using the Internet increases, the network attacks that these devices are exposed to also diversify. Identifying network attack types from network packets is important to prevent the damage of the attack and to minimize it in cases where it cannot be prevented. Classical machine learning methods and deep learning methods need a lot of data to get successful results. Unfortunately, preparing and labeling large amounts of data is costly in current conditions. This cost is mostly due to the training of the experts who will do the labeling process, the difficulty of generating attack environments, and the complexity of attacks. This study examines the problem of classifying network attacks with limited data in the learning process by applying few-shot learning methods. To investigate the problem, we generate three different datasets using previously labeled large datasets including CIC-IDS2017 and UNSW-NB15. We apply three promising approaches, where two of them are based on Prototypical Networks, and one of them is based on Relation Networks.

Keywords: Cybersecurity, Network Attack Classification, Few-shot Learning, Network Intrusion Detection, Meta-Learning Framework

ÖZ

AZ ATIŞLI ÖĞRENME YÖNTEMLERİ KULLANARAK AĞ SALDIRI SINIFLANDIRILMASI

Tüzün, İsmail
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Pelin Angın

Eylül 2022 , 74 sayfa

İnternet kullanan araçların sayısı arttıkça, bu araçların maruz kalmış oldukları ağ saldırıları da çeşitlenmektedir. Ağ saldırı tipinin ağ paketlerinin incelenerek anlaşılması, saldırının zararını önlemek, önlenemediği durumlarda da en aza indirmek için önem sarfetmektedir. Klasik makine öğrenme yöntemleri ve derin öğrenme yöntemleri, başarılı sonuçlar alabilmek için çok miktarda veriye ihtiyaç duymaktadırlar. Maalesef, çok miktarda verinin hazırlanarak etiketlenmesi günümüz şartlarında maliyetli olmaktadır. Bu maliyet çok büyük oranda bahsedilen etiketleme işlemini yapacak olan bilir kişilerin eğitilmesi, saldırı oluşturma ortamlarının zor oluşu ve saldırıların karmaşık oluşundan kaynaklanmaktadır. Bu çalışmada az atışlı öğrenme yöntemleri ile az verinin bulunduğu veri kümesindeki sınıfları, eğitim sürecinde kullanmadan ağ saldırı sınıflandırma yapılması problemi incelenmiştir. Problemin incelenmesi sırasında, daha önceden hazırlanmış büyük veri kümelerinden CIC-IDS2017 ve UNSW-NB15 kullanılarak üç farklı küçük veri seti hazırlanmıştır. Problemin çözümü için iki tanesi Prototipik Ağ tabanlı, bir tanesi İlişkisel Ağ tabanlı olmak üzere üç farklı yöntem denenmiştir.

Anahtar Kelimeler: Sibergüvenlik, Ağ Saldırı Sınıflandırılması, Az Atışlı Öğrenme, Ağ Saldırı Tespiti, Meta-Öğrenme Altyapısı

To my lovely precious wife Münteha Nur

ACKNOWLEDGMENTS

I would like to express my gratitude to my beloved wife, who supported me in every way during my thesis work and discussed special situations when necessary. If it wasn't for her help, I would have had really a hard time. Additionally, I would like to thank my family for their unwavering support throughout the thesis.

I would like to thank my advisor Assoc. Prof. Dr. Pelin Angın, who guided me during the studies and had a significant impact on the emergence of this study. Her ideas and support during difficult times helped me to move forward in this work. Also, I would like to thank the jury members, Pınar Karagöz and Ahmet Burak Can, for their contributions.

The numerical calculations reported in this paper were partially performed using TÜBİTAK ULAKBİM, High Performance and Grid Computing Center (TRUBA) resources. I would like to thank the TRUBA project for making the computational resources available for the realization of this project and the officials at TÜBİTAK ULAKBİM involved in its maintenance.

This research has been supported by TÜBİTAK under grant number 120E537. The entire responsibility of the thesis belongs to the author of the thesis. The support received from TÜBİTAK does not mean that the content of the publication is approved in a scientific sense by TÜBİTAK.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vii

ACKNOWLEDGMENTS x

TABLE OF CONTENTS xi

LIST OF TABLES xiii

LIST OF FIGURES xv

LIST OF ABBREVIATIONS xviii

CHAPTERS

1 INTRODUCTION 1

1.1 Problem Definition 1

1.2 Contributions 3

1.3 Notation and Terminology 3

1.4 Organization of the Thesis 3

2 BACKGROUND AND RELATED WORK 5

2.1 Network-based Intrusion Detection Systems 5

2.2 Artificial Neural Networks 5

2.3 Convolutional Neural Networks 7

2.4 Few-shot Learning 8

3 METHODOLOGY 17
3.1 Dataset Generation 17
3.2 ANN 18
3.3 FSL Methods 19
3.3.1 Prototypical Networks for Network Attack Classification . . . 20
3.3.1.1 CNN Encoder 23
3.3.1.2 ANN Encoder 23
3.3.2 Relation Networks for Network Attack Classification 24
4 EXPERIMENTS AND EVALUATION 29
4.1 Datasets 29
4.1.1 CIC-IDS2017 29
4.1.1.1 CIC-IDS2017 FSL-1 Dataset 30
4.1.1.2 CIC-IDS2017 FSL-2 Dataset 31
4.1.2 UNSW-NB15 31
4.1.2.1 UNSW-NB15 FSL Dataset 32
4.2 Evaluation Metrics 33
4.3 Experiments 34
4.3.1 Experiments on CIC-IDS2017 FSL1 Dataset 36
4.3.2 Experiments on CIC-IDS2017 FSL2 Dataset 47
4.3.3 Experiments on UNSW-NB15 FSL Dataset 58
5 DISCUSSION AND CONCLUSION 69
REFERENCES 71

LIST OF TABLES

TABLES

Table 2.1 Sample Machine Learning Task, Performance Metric, Experience . . 8

Table 2.2 Sample Few-shot Learning Tasks, Performance Metric, and Experience 9

Table 4.1 CIC-IDS2017 FSL-1 Dataset 30

Table 4.2 CIC-IDS2017 FSL-2 Dataset 31

Table 4.3 UNSW-NB15 FSL Dataset 32

Table 4.4 ANN Performance on CIC-IDS2017 FSL1 meta test set trained only using meta train set classes 34

Table 4.5 *ANN* method Performances on CIC-IDS2017 FSL1 meta test set . . 37

Table 4.6 *PN-CNN* method Performances on CIC-IDS2017 FSL1 meta test set 37

Table 4.7 *PN-ANN* method Performances on CIC-IDS2017 FSL1 meta test set 38

Table 4.8 *RN-CNN* method Performances on CIC-IDS2017 FSL1 meta test set 38

Table 4.9 *ANN* method Performances on CIC-IDS2017 FSL1 meta validation set 42

Table 4.10 *PN-CNN* method Performances on CIC-IDS2017 FSL1 meta validation set 42

Table 4.11 *PN-ANN* method Performances on CIC-IDS2017 FSL1 meta validation set 43

Table 4.12 *RN-CNN* method Performances on CIC-IDS2017 FSL1 meta validation set 43

Table 4.13 *ANN* method Performances on CIC-IDS2017 FSL2 meta test set . . 48

Table 4.14 *PN-CNN* method Performances on CIC-IDS2017 FSL2 meta test set 48

Table 4.15 *PN-ANN* method Performances on CIC-IDS2017 FSL2 meta test set 49

Table 4.16 *RN-CNN* method Performances on CIC-IDS2017 FSL2 meta test set 49

Table 4.17 *ANN* method Performances on CIC-IDS2017 FSL2 meta validation set 52

Table 4.18 *PN-CNN* method Performances on CIC-IDS2017 FSL2 meta validation set 53

Table 4.19 *PN-ANN* method Performances on CIC-IDS2017 FSL2 meta validation set 53

Table 4.20 *RN-CNN* method Performances on CIC-IDS2017 FSL2 meta validation set 54

Table 4.21 *ANN* method Performances on UNSW-NB15 FSL meta test set . . . 58

Table 4.22 *PN-CNN* method Performances on UNSW-NB15 FSL meta test set . 59

Table 4.23 *PN-ANN* method Performances on UNSW-NB15 FSL meta test set . 59

Table 4.24 *RN-CNN* method Performances on UNSW-NB15 FSL meta test set . 60

Table 4.25 *ANN* method Performances on UNSW-NB15 FSL meta validation set 64

Table 4.26 *PN-CNN* method Performances on UNSW-NB15 FSL meta validation set 64

Table 4.27 *PN-ANN* method Performances on UNSW-NB15 FSL meta validation set 64

Table 4.28 *RN-CNN* method Performances on UNSW-NB15 FSL meta validation set 65

LIST OF FIGURES

FIGURES

Figure 1.1 Number of Internet Connected Devices 2

Figure 2.1 Sample Neuron 6

Figure 2.2 Sample ANN 7

Figure 2.3 Comparison of Convolutional Layer with Fully Connected layer . 7

Figure 2.4 Sample meta test task 10

Figure 2.5 Sample Meta Training Phase 11

Figure 2.6 Sample Meta Testing Phase 12

Figure 2.7 Siamese Convolution Network Architecture 13

Figure 2.8 Matching Network Architecture 14

Figure 2.9 Prototypical Networks Sample Embeddings 15

Figure 2.10 Relation Network Architecture 16

Figure 3.1 ANN Architecture 19

Figure 3.2 Prototypical Network Meta Training Phase 21

Figure 3.3 Prototypical Network Meta Testing Phase 22

Figure 3.4 Prototypical Network CNN Encoder Architecture 23

Figure 3.5 Prototypical Network ANN Encoder Architecture 24

Figure 3.6 Relation Network Meta Training 25

Figure 3.7 Relation Network Meta Testing 26

Figure 3.8 Relation Network Encoder Architecture 27

Figure 3.9 Relation Network Architecture 28

Figure 4.1 Example Multiclass Confusion Matrix 33

Figure 4.2 CIC-IDS2017 FSL1 Meta test results when $k = 5$ 39

Figure 4.3 CIC-IDS2017 FSL1 Meta test results when $k = 10$ 39

Figure 4.4 CIC-IDS2017 FSL1 Meta test results when $k = 15$ 40

Figure 4.5 CIC-IDS2017 FSL1 Meta test results when $k = 20$ 40

Figure 4.6 CIC-IDS2017 FSL1 Meta test results when $k = 25$ 41

Figure 4.7 CIC-IDS2017 FSL1 Meta test results when $k = 50$ 41

Figure 4.8 CIC-IDS2017 FSL1 Meta validation results when $k = 5$ 44

Figure 4.9 CIC-IDS2017 FSL1 Meta validation results when $k = 10$ 45

Figure 4.10 CIC-IDS2017 FSL1 Meta validation results when $k = 15$ 45

Figure 4.11 CIC-IDS2017 FSL1 Meta validation results when $k = 20$ 46

Figure 4.12 CIC-IDS2017 FSL1 Meta validation results when $k = 25$ 46

Figure 4.13 CIC-IDS2017 FSL1 Meta validation results when $k = 50$ 47

Figure 4.14 CIC-IDS2017 FSL2 Meta test results when $k = 5$ 50

Figure 4.15 CIC-IDS2017 FSL2 Meta test results when $k = 10$ 50

Figure 4.16 CIC-IDS2017 FSL2 Meta test results when $k = 15$ 51

Figure 4.17 CIC-IDS2017 FSL2 Meta test results when $k = 20$ 51

Figure 4.18 CIC-IDS2017 FSL2 Meta test results when $k = 25$ 51

Figure 4.19 CIC-IDS2017 FSL2 Meta test results when $k = 50$ 52

Figure 4.20 CIC-IDS2017 FSL2 Meta validation results when $k = 5$ 55

Figure 4.21 CIC-IDS2017 FSL2 Meta validation results when $k = 10$ 55

Figure 4.22 CIC-IDS2017 FSL2 Meta validation results when $k = 15$ 56

Figure 4.23 CIC-IDS2017 FSL2 Meta validation results when $k = 20$ 56

Figure 4.24 CIC-IDS2017 FSL2 Meta validation results when $k = 25$ 57

Figure 4.25 CIC-IDS2017 FSL2 Meta validation results when $k = 50$ 57

Figure 4.26 UNSW-NB15 FSL Meta test results when $k = 5$ 61

Figure 4.27 UNSW-NB15 FSL Meta test results when $k = 10$ 61

Figure 4.28 UNSW-NB15 FSL Meta test results when $k = 15$ 62

Figure 4.29 UNSW-NB15 FSL Meta test results when $k = 20$ 62

Figure 4.30 UNSW-NB15 FSL Meta test results when $k = 25$ 62

Figure 4.31 UNSW-NB15 FSL Meta test results when $k = 50$ 63

Figure 4.32 UNSW-NB15 FSL Meta validation results when $k = 5$ 65

Figure 4.33 UNSW-NB15 FSL Meta validation results when $k = 10$ 66

Figure 4.34 UNSW-NB15 FSL Meta validation results when $k = 15$ 66

Figure 4.35 UNSW-NB15 FSL Meta validation results when $k = 20$ 67

Figure 4.36 UNSW-NB15 FSL Meta validation results when $k = 25$ 67

Figure 4.37 UNSW-NB15 FSL Meta validation results when $k = 50$ 68

LIST OF ABBREVIATIONS

ANN	Artificial Neural Network
CNN	Convolutional Neural Network
FSL	Few-Shot Learning
IDS	Intrusion Detection System
NIDS	Network-based Intrusion Detection System
NIPS	Network-based Intrusion Prevention System
LSTM	Long Short-Term Memory
ReLU	Rectified Linear Units
NaN	Not a Number
inf	Infinity

CHAPTER 1

INTRODUCTION

1.1 Problem Definition

With the increase in the use of the Internet, almost all technological devices that we use in our daily lives have started to access the Internet [1]. Due to the increase in devices using the Internet as shown in the *Figure 1.1*, both the number of types and the complexity of network attacks on these devices are increasing. At this point, Network Intrusion Detection Systems (NIDS) and Network Intrusion Prevention Systems (NIPS) are being developed to be not affected by the attacks. These systems can be divided into two categories based on how they detect attacks [2]. The first one is called Misuse Detection, where the system tries to detect attacks using previously analyzed attack signatures using monitored resources. The second one is called Anomaly Detection. In this type of method, the system tries to learn the normal behavior of the network traffic pattern. Misuse Detection systems fail to distinguish new attack types compared to Anomaly Detection systems.

To effectively eliminate network attack damages, it is necessary to classify the types of attacks. Classifying the types of attacks is just as important as recognizing the occurrence of an attack during a network flow since different prevention techniques are used against each type of attack. To give an example, if your system is under a DDoS attack and, as a solution, you stop providing service to avoid damaging your system, the attack will reach its goal. On the other hand, if you have been exposed to a Worm attack that spreads as the service continues, it would be a good starting point for you to stop your service for a while and try to neutralize the related worm. If we compare the two cases, stopping providing service means that the attack in the first

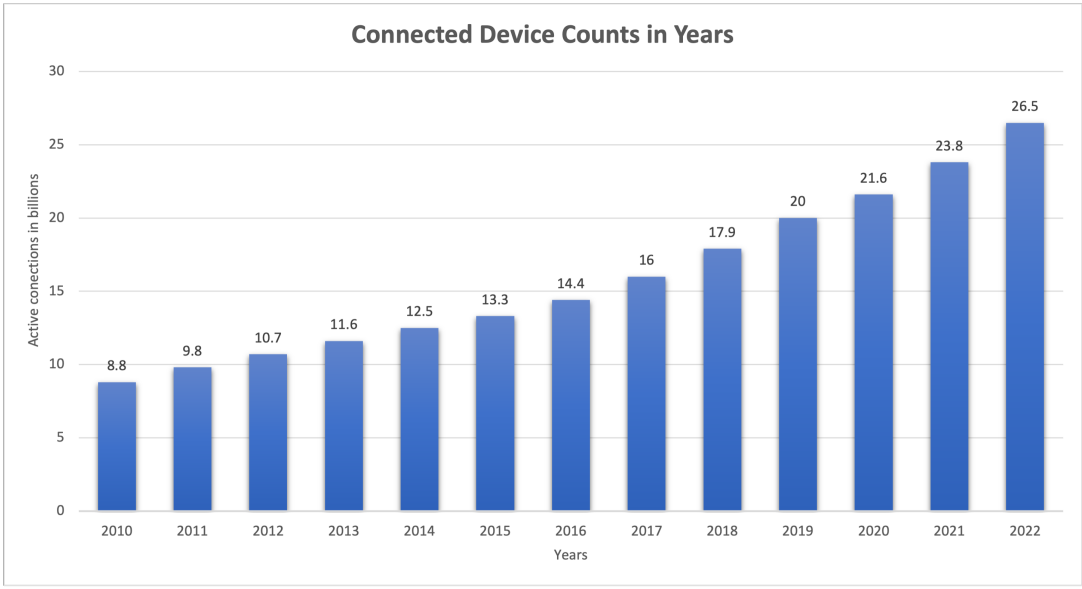


Figure 1.1: Number of Internet Connected Devices

one has achieved its purpose, while in the second it is a good starting point against the attack. It is not always optimal to apply the same method for different types of attacks. Therefore, understanding what type of attack the system is dealing with is important for deciding the appropriate defensive action to take.

In the case of diversification of attacks, previously developed network attack classification systems fail, since the system is unaware of unseen classes. To make them aware, it should be retrained using a new dataset that contains new classes. Creating sufficient datasets to adapt old systems to new attack types requires both time and the work of expert personnel which is rare. In most cases, it is not possible to collect data as much as the number of samples in the datasets created for previous attacks. Furthermore, the datasets that are commonly used for network intrusion detection problems contain unbalanced data. While there are classes with a lot of data, there are also classes with very little data. The reasons behind having few data on some attack classes include the high complexity of the attack type, the difficulties in generating the attack environment, and the fact that the attack is recent. Adding unseen attack types to these datasets will increase the unbalance of a dataset, which will end up in bias at the learning stage. The new attack types will share the same destiny with the ones that are a small number of samples in datasets. On the other hand, previously developed methods like artificial neural networks (ANN), convolutional neural net-

works (CNN), etc require a large dataset to learn from for the classification problem. The problem we are examining is the case where there is little data in the network attack classification problem. Few-shot learning (FSL) methods take a role here to overcome the problem of learning to classify network attack types using few training data instances.

In this work, to investigate the defined problem above, we prepared three small datasets using CICIDS2017 and UNSW-NB15 datasets to generate the problem settings. Then we evaluate some of the FSL methods on the problem of network attack classification. These methods do not require retraining for classifying unseen classes.

1.2 Contributions

Our contributions are as follows:

- Prepared FSL Network Attack Classification datasets using existing datasets
- Applied two FSL methods to Network Attack Classification problem
- Compared applied FSL methods' results with both FSL methods and the ANN method.

1.3 Notation and Terminology

A few-shot learning task *T* uses a dataset $D = \{D_{Train}, D_{Test}\}$. The training set $D_{Train} = \{x_i, y_i\}_{i=1}^{m}$ where m is small, contains a small number of samples. If the testing set D_{Test} contains n classes and each class has k samples, then the problem is called ***n-way k-shot*** learning.

1.4 Organization of the Thesis

The remainder of the thesis is organized as follows: Chapter 2 provides an overview of the background information and related work done by others. Chapter 3 describes

the methods developed in this study. This section details three methods for solving the few-shot intrusion classification problem. Chapter 4 provides detailed information about the experiments performed and their results. Finally, Chapter 5 concludes the study by summarizing the findings and providing future work.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Network-based Intrusion Detection Systems

An intrusion [3], which is an event that consists of one or more security events, and is performed by an attacker with an aim of gaining access to a system or system resource without having permission. Intrusion Detection Systems are tools to understand whether there is an active intrusion attempt or not. These systems can be developed based on three different approaches [4]. The first one is signature based, which analyzes attacks and generates a signature for them to compare later on. The second approach is anomaly based, which tries to learn the normal behavior of the system and tries to detect anomalies. The last approach is the hybrid approach; as the name implies, it uses both signatures and anomalies.

IDSs are divided into two categories with respect to how they are deployed [4]. The first category is Host-based Intrusion Detection Systems (HIDS), which is deployed on the target device and analyzes the system to detect intrusions. Antivirus systems can be given as an example of HIDS systems. The second category is Network-based Intrusion Detection Systems (NIDS), which aims to detect attacks using network flows via network connections. NIDSs can be deployed on both a computer and a network device.

2.2 Artificial Neural Networks

Artificial Neural Networks (ANN) is a method that mimics human brain activity to solve computer science problems. This idea was first written by McCulloch et al. [5].

ANN consists of neurons named perceptrons [6] just like brain structure. These neurons are connected to each other and transfer data to output neurons after processing input data. *Figure 2.1* represents a simple neuron structure. The left tentacles are connected to the input neurons and data flows from those neurons into the neuron. Then, the information is processed inside this neuron with the neuron's parameters. After calculating the output, it transfers the value to the output neurons, which are connected to the neuron via the right tentacles.



Figure 2.1: Sample Neuron

A set of connected neurons construct an artificial neural network. The sample structure of the ANN is given at *Figure 2.2*. The first group of neurons is called the input layer, and the last group of neurons is called the output layer. In our example, the output layer contains only one neuron, which is most probably the problem that the ANN tries to solve, and it requires only the value calculated at the last neuron. The layers between the input and output layers are called hidden layers.

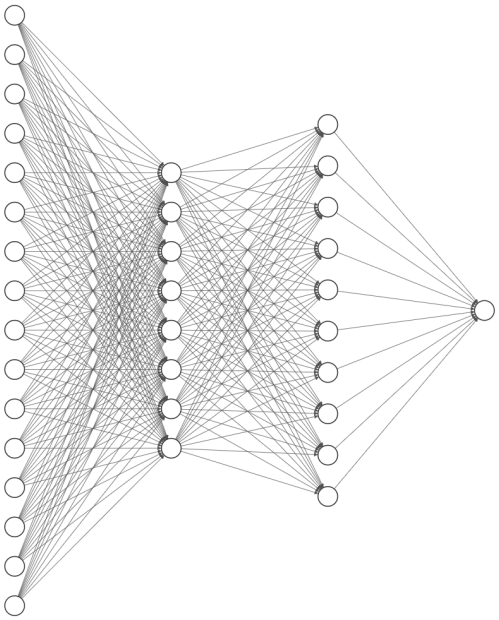


Figure 2.2: Sample ANN

2.3 Convolutional Neural Networks

Convolutional Neural Networks which are first mentioned in the study [7] by Fukushima et al. are a type of neural network. The main difference between CNN from any other neural network is that CNN contains a convolutional layer. This layer aims to extract features using only neurons close to each other. As *Figure 2.3* shows, when we investigate one of the output neurons of the convolutional layer, its value depends on only three input neurons, not all of them.

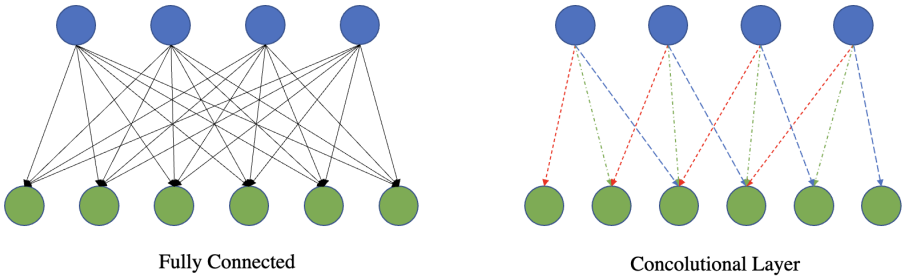


Figure 2.3: Comparison of Convolutional Layer with Fully Connected layer

CNNs are efficiently used in extracting features from image inputs, since, pixels near each other construct information rather than pixels far from each other.

2.4 Few-shot Learning

Few-shot learning is one of the subfields of machine learning. In order to increase understanding of the FSL terminology used in this study, we need to define both machine learning and few-shot learning. Machine learning can be defined as an algorithm or an application that can learn to increase performance metric P on some task T using experience E [8] [9]. The following case is a good example: consider an image classification task T. This task aims to classify any query image to its correct class. The accuracy of the correctly classified queries can be defined as performance metric P. A machine learning application tries to increase P using previously labeled images. The large dataset containing labelled images can be considered as experience E [10]. Another example is spam mail detection [11]. In this problem, the task is to discriminate spam mails from useful mails. Performance metric can be defined as the percentage of correctly labeled spam mails, whereas the experience is the previously labeled large number of spam and non-spam emails dataset. Task, performance metric and experience for given examples can be found at *Table 2.1*.

Table 2.1: Sample Machine Learning Task, Performance Metric, Experience

Task	Performance Metric	Experience
Image classification	Classification accuracy	Large image dataset that contains images for each classes
Spam mail detection	Detection accuracy	Large mail dataset that contains spam and normal mails

Most of the machine learning applications, like the given examples above, need a large dataset with supervised information to reach high performance metrics. But, it may be impossible or costly to get a large supervised dataset for few-shot learning problems. FSL is a specific area of machine learning, that also aims high performance metric P on the task T like the other machine learning areas. Different from other techniques, it requires experience E which contains a small number of supervised samples [12]. Current FSL problems in the literature are mostly supervised learning problems. In other words, few-shot classification tries to classify each class using

only a small number of samples per class. For example, image classification [13] and object recognition [14] are problems where few-shot classification can be applied. *Table 2.2* summarizes the details of the tasks.

Table 2.2: Sample Few-shot Learning Tasks, Performance Metric, and Experience

Task	Performance	Experience	
		Supervised info	Prior knowledge
Object recognition	Intersection over Union	A few labelled images for each object	Images of other classes or previously trained models on different dataset
Image classification	Classification accuracy	Small number of labelled samples for each classes	Images of other classes or previously trained models on different dataset

Wang et al. [12] discussed four learning problems relevant to FSL. Some of the methods compared with FSL can be summarized as follows:

- *Transfer learning [15]* is a method that tries to transfer knowledge from a domain that has a large amount of data or a model that works well on the source domain, while there is scarce data for the target domain. As an example, the study [16] done by Wu et al. tries to detect network intrusions using transfer learning method.
- *Weakly supervised learning [17]* aims to learn from a dataset that contains noisy, limited, or imprecise labeled information. Wang et al. [12] realized that the most similar problem to few-shot learning is *weakly supervised learning with incomplete supervision* which is a subarea of *weakly supervised learning*. In this problem, there is only a little data with supervision.
- *Meta learning [18]* is a process to improve the performance of a task by training a set of tasks different than the target task. Although meta learning and normal learning methods are very much alike, there is a separate dataset for each task in meta learning [19].

The formal definition of few-shot classification is that given task T and performance

metric P, improve P on T using an experience E where E contains n classes and each one of the classes contains k samples, where k is a relatively small number with respect to similar tasks in the same domain. If $k = 1$ then the problem is called *one-shot learning*.

Some of the leading studies in the field of few-shot learning used the meta learning paradigm as a baseline for their work. A sample meta learning task is illustrated at *Figure 2.4*. The support set can be considered as a training dataset for the normal learning procedure. On the other hand, the query set can be correlated with the testing dataset in the normal learning process. The main difference between meta learning and the classical learning procedure is that meta learning tries to train a model that solves the meta test task via learning by tasks and those tasks have their own training set (that is support set) and their own test set (that is query set). In other words, meta learning uses a set of tasks for the learning stage and tries to generalize a model that has high performance on the meta test task.

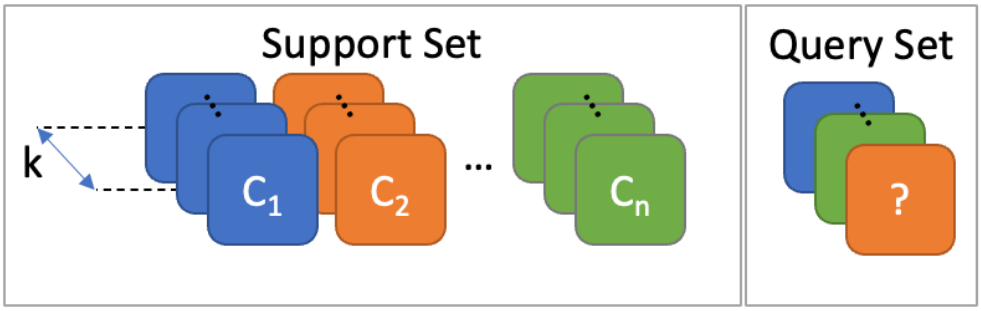


Figure 2.4: Sample meta test task

To explain the essence of the meta learning phases, let's investigate over figures. The *Figure 2.5* summarizes the training phase of meta learning, whereas *Figure 2.6* shows meta testing phase. The following conditions must be satisfied for meta learning:

1. The meta train phase contains l tasks, and each task has a support set and a query set. At each learning iteration a model tries to learn query set samples by using supervised support set information.

2. Support set of T_x where $1 \leq x \leq l$, and query set of T_x must have the same classes.

3. Support set of T_y where $1 \leq y \leq l$ and $y \neq x$, and support set of T_x may or may not have disjoint classes

4. Support set of $T_{MetaTest}$ and query set of $T_{MetaTest}$ must have the same classes.

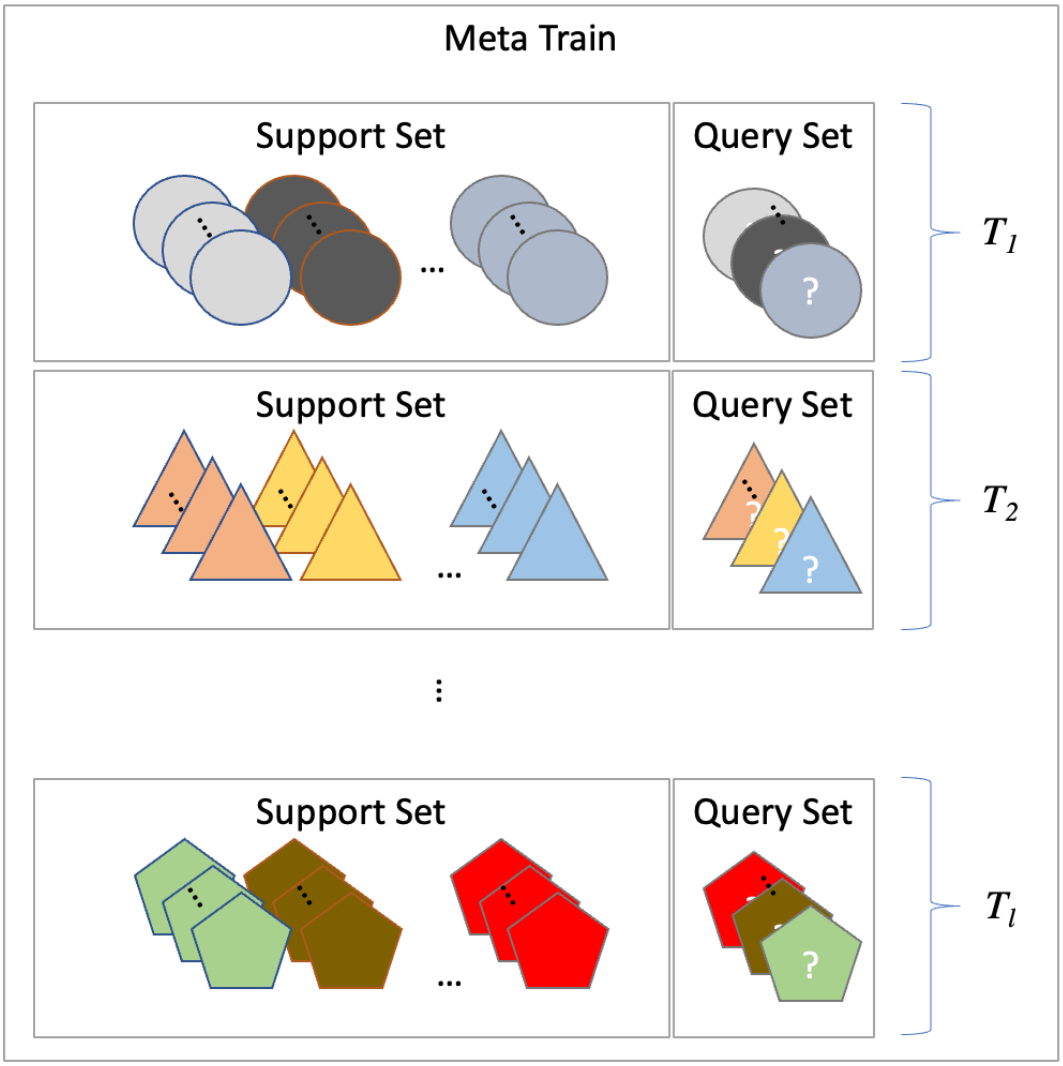


Figure 2.5: Sample Meta Training Phase

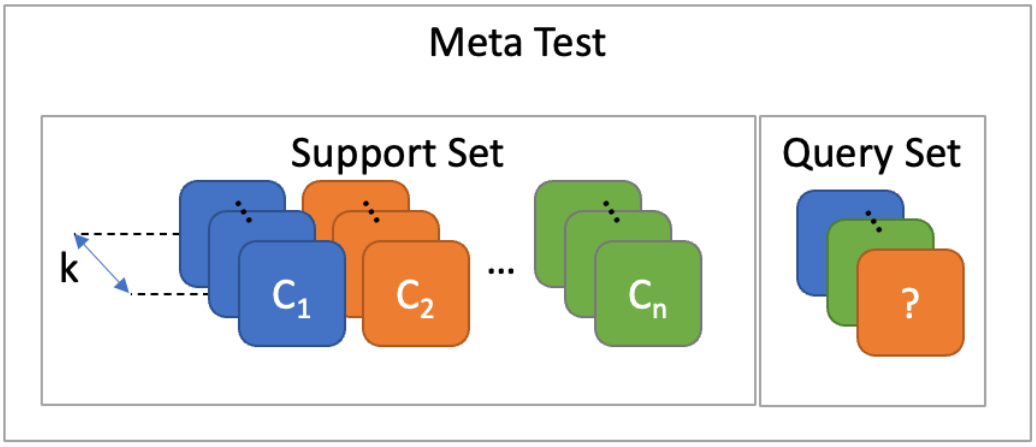


Figure 2.6: Sample Meta Testing Phase

Koch et al. [20] proposed an approach to solve one-shot image classification using the Siamese neural network. The architecture is illustrated in *Figure 2.7*. In the model training section, they feed two images to the model as input and hope that the model would learn whether the images are from the same class or not. For the test time, they select only one sample per class and form a support set. In order to classify an image, they feed the model with the query image along with an image from the support set, until the model finds a match.

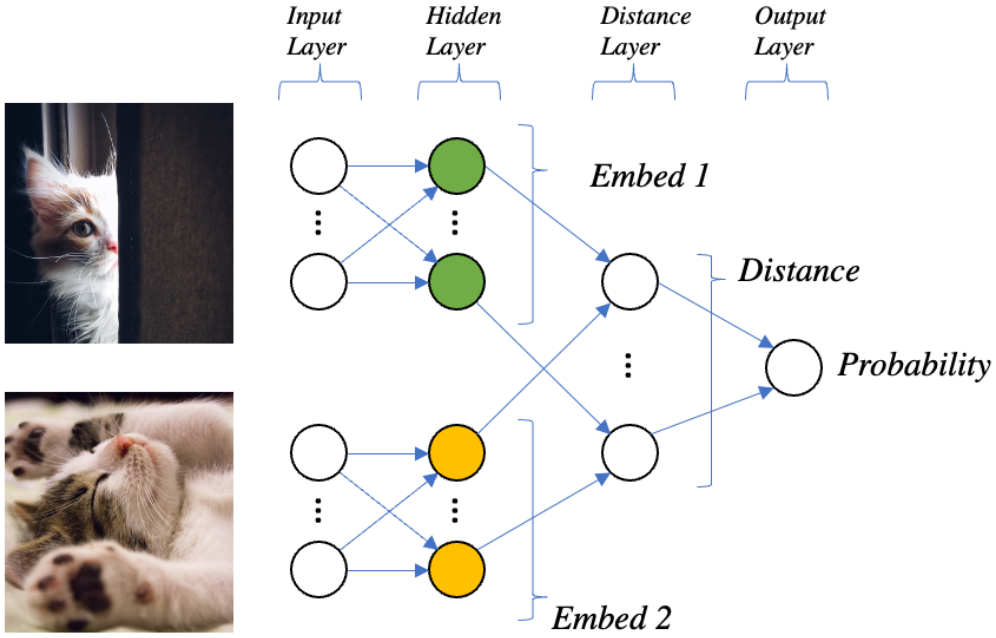


Figure 2.7: Siamese Convolution Network Architecture

Xu et al. used a meta learning framework in their study [21], in order to detect anomalies in network traffic. In their work, they tried to discriminate benign network flows from attacks. They defined their problem as detecting the attack using a small dataset containing benign traffic and one of the attack types for testing. During the training phase, they generated tasks using benign traffic and three different attack types. They use a convolutional network called F-Net to extract a feature map of raw network traffic. To compare two feature maps, first, they combine them and feed them to another convolutional network called C-Net which aims to calculate the delta score. The main difference between our study and this work is that they tried to differentiate benign classes from one of the attack classes that was not included at training phase. On the other hand, our study tries to distinguish multiple classes from each other for which no data was included in the training phase.

Vinyals et al. [22] used the meta learning paradigm to train a metric based one shot image classifier. The proposed method contains two LSTM networks as shown at the *Figure 2.8*, g_θ is the first one to generate embeddings for the support set, and f_θ is the second one to find probability distributions over the classes of the query image. The

proposed method feeds all support set samples to the embedding network to achieve more distinguishable embeddings. They used cosine distance to calculate the distance between query image embedding and each class's embeddings.

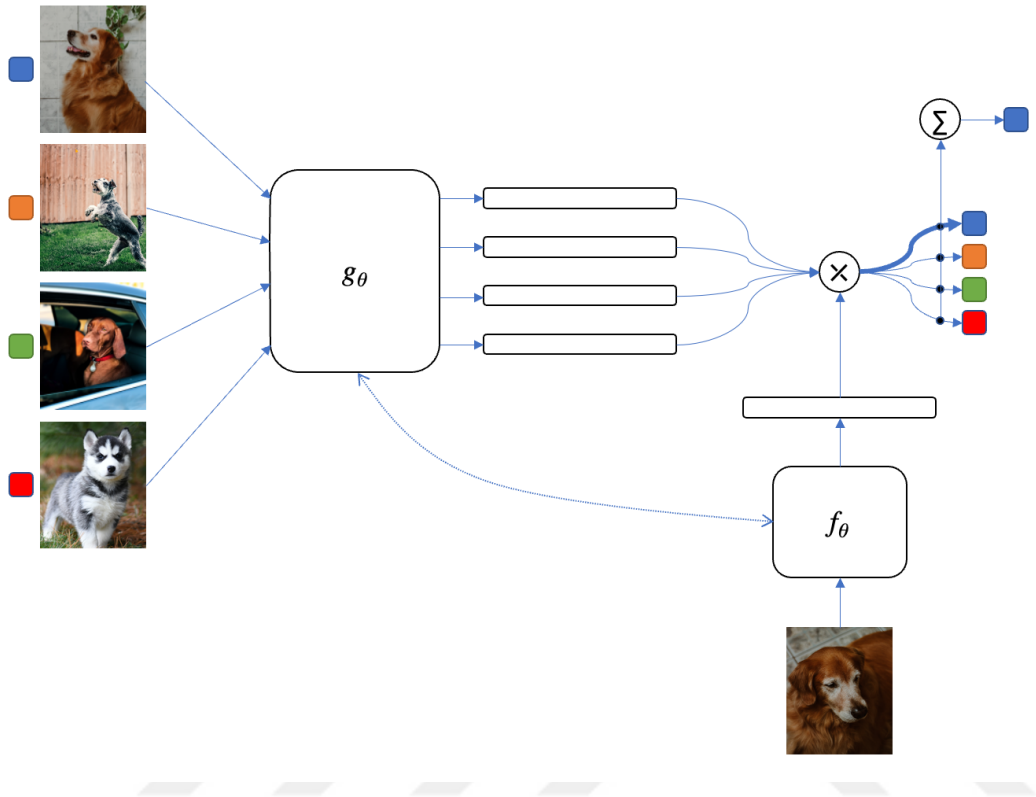


Figure 2.8: Matching Network Architecture

Prototypical Network [23] which is proposed by Snell et al. defines few-shot classification as a task for classifying new classes that are not seen at the training phase, where the new classes have only a few samples per each class. The study suggests calculating the prototype of each class in the support set at the embedding space. In order to predict the class of any image, first, they calculate its embedding with the same network and they calculate the distance to each prototype. This distance is calculated by using squared Euclidean distance. Their approach is similar to *k-means* algorithm. The embedding space and prototypes are illustrated at the *Figure 2.9*. The embedding network will be trained during meta training. They used episodic composition which is a naive way to create episodes. At each episode they choose random N classes with k samples per class for the support set, then they choose random q samples per class for the query set at train time. In the *Figure 2.9*, they represent

3-way 5-shot learning problem. After training the encoder, they calculate each class's prototype. The black circles are class prototypes.

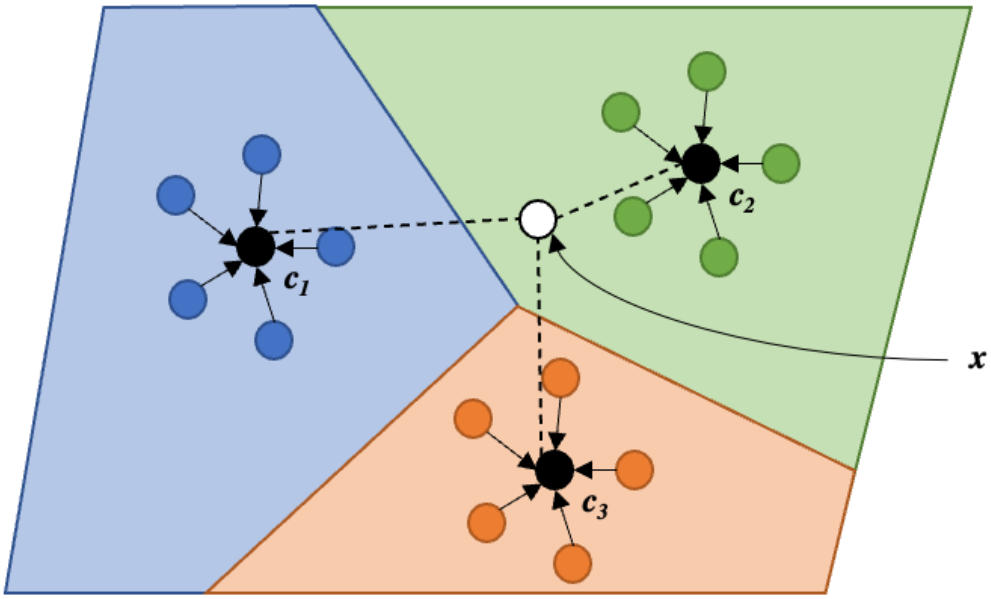


Figure 2.9: Prototypical Networks Sample Embeddings

Another work on few-shot image classification is *Relation Network* by Sung et al [24]. In this method, there are two networks, one for encoding inputs to the embedding area and the second for comparing feature maps. The difference from FC-Nets is that they train end-to-end from scratch. Unlike Prototypical Networks, in this study, they trained Relation Module to make the comparison. Their architecture is given at *Figure 2.10*. As seen in the figure, there is an embedding module f_Ψ that calculates feature maps. Then they concatenate query embedding with each of the support set embeddings to generate pairs of feature maps. These pairs are fed to the relation network g_ϕ to calculate relation scores. In the end, they convert the relation score to the one hot vector.

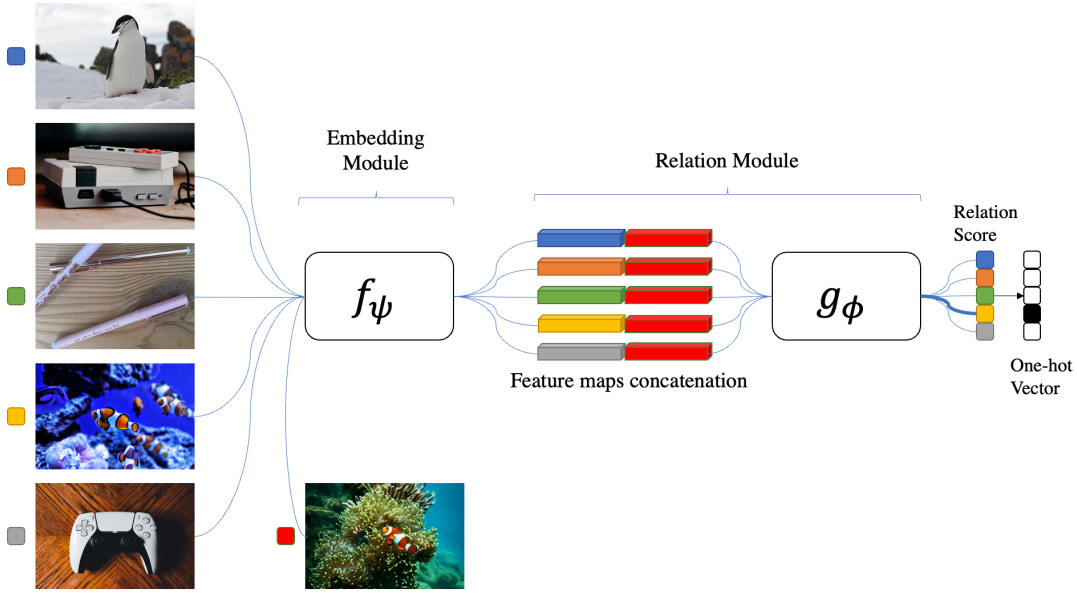


Figure 2.10: Relation Network Architecture

CHAPTER 3

METHODOLOGY

In this section, details of the proposed approaches and the way that we follow for dataset generation will be explained. More details for the dataset content will be provided in the experimentation section. This study investigates two main methods and their variants. The first method is called Prototypical Networks [23], which tries to learn an encoder that aims to place the same class samples close to each other in the embedding space. Relation Networks [24] which is the second method, aims to learn both the encoder and the comparison network as a whole. According to the research [25] done by Gamage et al., a feed-forward neural network (ANN) works better than the other three popular methods such as autoencoders, LSTMs, and deep belief networks on the classical network attack classification task. The study conducted experiments using four different datasets including KDD99, NSL-KDD, CICIDS2017, and CICIDS2018. The results show that the other methods did not perform as well as ANN. It is important to remark that in their study they do not investigate few-shot network attack classification problem, they use the full dataset. As they prove that ANN works better than any other method on a large dataset, we choose it to compare our results. The ANN implementation and the model architecture are highly influenced by their research.

3.1 Dataset Generation

In this work, we generate three different datasets for network attack classification problem using previously generated large datasets. Our datasets should be small since we are working on the few-shot learning problem. Here comes the question of how

many data samples a small dataset should contain to be a small dataset. When we investigate this question, we found that there is no well-defined definition for a dataset to be small. Researchers intuitively agree that a dataset is a small dataset, or that a few-shot learning problem is a few-shot problem. This intuition emerges from the ratio between the size of the dataset used for the non-few-shot version of the same problem and the size of the dataset used for the few-shot problem.

For our datasets, we choose 600 samples per class. In order to make this decision, first, we made simple experimentation, and we try to find an acceptable number of samples per class for the meta training dataset. Additionally, we investigate few-shot image classification datasets like Mini-ImageNet [22] and CIFAR-FS [26] . These datasets contain 100 classes and each one of these classes has 600 samples. Also, these datasets have about 10 times more classes than our dataset.

For the CICIDS2017 FSL1 dataset and CICIDS2017 FSL2 dataset, we use approximately 0.25% of the large dataset. For the last FSL dataset, which is UNSW-NB15 FSL dataset, we use approximately 0.28% of the large dataset. The details of these datasets are provided in the experimentation section.

3.2 ANN

In the study [25], they found that for network attack classification problem, artificial neural network with the following configuration works well. Their simple architecture is given in *Figure 3.1*.

We used the ANN classifier explained below in our study to compare the proposed FSL methods.

- Input layer with node number equals input features and ReLU [27] activation
- Batch normalization after input layer
- Dropout with 0.2
- Output layer with node number equals to number of classes to be classified with the softmax [28] activation

- Using binary cross entropy loss
- Using Adam optimizer

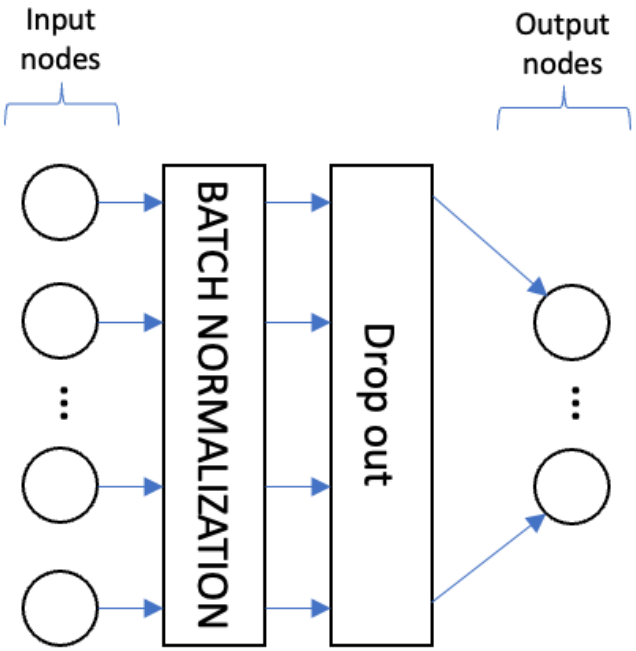


Figure 3.1: ANN Architecture

The implementation is done using Tensorflow [29] framework. During the training phase, ANN needs all classes that are needed to be classified. Therefore, we have to make small changes to the dataset used by FSL methods. The changes will be explained in detail in the Experiments and Evaluation chapter.

3.3 FSL Methods

To solve the few-shot network attack classification problem, we applied two main approaches. These approaches try to generalize a model using the learning framework. They need prior knowledge from a different dataset that is not included in the problem's main dataset. In order to create prior knowledge, we prepared datasets using existing ones. The meta training dataset contains different classes from the meta test

dataset, where the meta test dataset consists of the main problem's classes. Dataset creation and their properties will be explained in the Experiments and Evaluation chapter. We implement these methods using Pytorch [30] framework.

3.3.1 Prototypical Networks for Network Attack Classification

Prototypical networks [23] are proposed by Snell et al aims to encode inputs into an embedding space, then use a $k\text{-}means$ like algorithm on the embedding space. The same methodology was used in this study with some differences. Firstly, their problem contains inputs with 2-dimensional images, on the other hand, our problem's input is 1-dimensional feature vectors. Therefore, the used encoders are different. In this work, we applied two different encoders to map inputs to the embedding space, the first one is the Convolutional Neural Network encoder, and the second one is an Artificial Neural Network encoder which is very similar to the ANN that we compare as the base model.

In order to understand how this approach works, we should understand two phases: *Meta Training* and *Meta Testing*. During *Meta Training* phase, we use the meta training dataset, which contains classes other than meta test dataset. Meta training process is explained in the *Figure 2.5*. The model uses episodic training where for each episode the model tries to learn one task. That task is similar to the task that we try to solve at the end. For each task, there is a support set and a query set. At each episode, one task is constructed by sampling its support and query sets from the meta training dataset. The model uses samples from the support set and encodes them into the embedding space. After encoding all samples from the support set, it calculates the prototype for each class, by simply taking the mean of the all embedding vectors of the same class element-wise. Then the model also calculates embedding vectors of all query set samples. After that, the model calculates the loss value using *Euclidean Distance* between query samples with their ground truth class prototypes. Then the encoder updates its weights with respect to calculated gradients using the Adam optimizer. Learning rate decreases between each episode with the parametric decay rate. The algorithm decides when to stop by looking at a fixed size window of the last training accuracy and calculating the mean of the accuracy change. If the change in the

mean is less than a specified threshold and the window has reached its fixed size, the learning process is stopped. In addition, if learning accuracy is reached to a specified threshold, the learning process is completed.

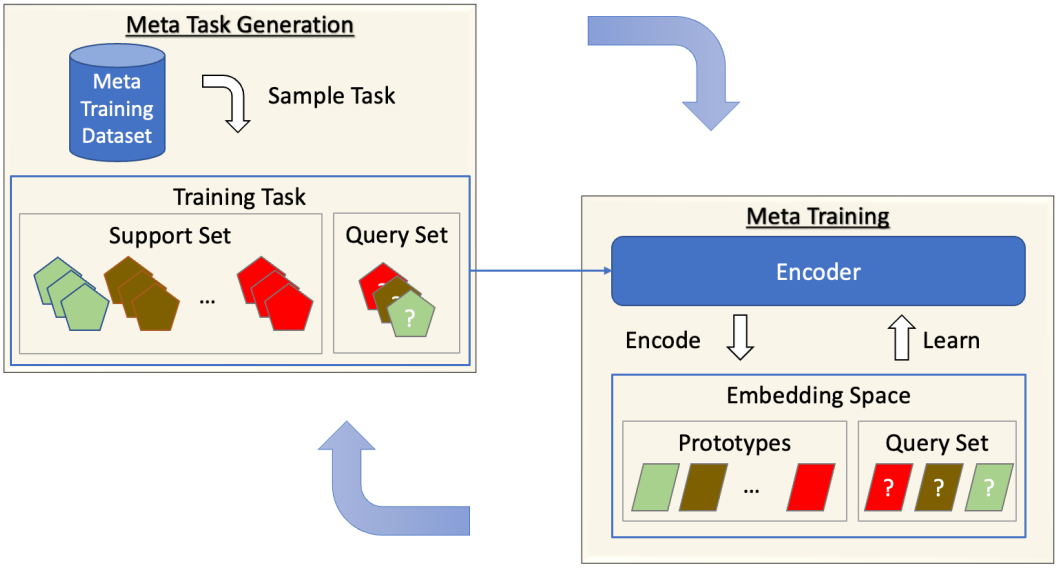


Figure 3.2: Prototypical Network Meta Training Phase

At *Meta Testing* time, the model uses the meta test dataset, which consists of the support set that has n classes, with k samples per class, and the query set that has q samples per class. Recall that k is small, which means the meta test dataset is not a large dataset, and the goal to be achieved is to classify these query samples using the support set. In addition to that, the meta test dataset consists of classes that are not used in the meta training phase. To achieve this purpose, the model takes the meta test support set as input and encodes them into the embedded space. As done in the meta training phase, it calculates prototypes for each class. When predicting the class label for a new data sample from the query set, the model also encodes the query sample into the embedded space and then calculates *Euclidean Distance* between the encoded query sample with all prototypes and labels the query sample with the label of the nearest prototype's class. Illustration of the meta testing phase can be found at *Figure 3.3*. In order to get more accurate results, during the meta testing phase, we sampled 1000 different meta testing tasks, with different support and query sets. Then we take the average of the results.

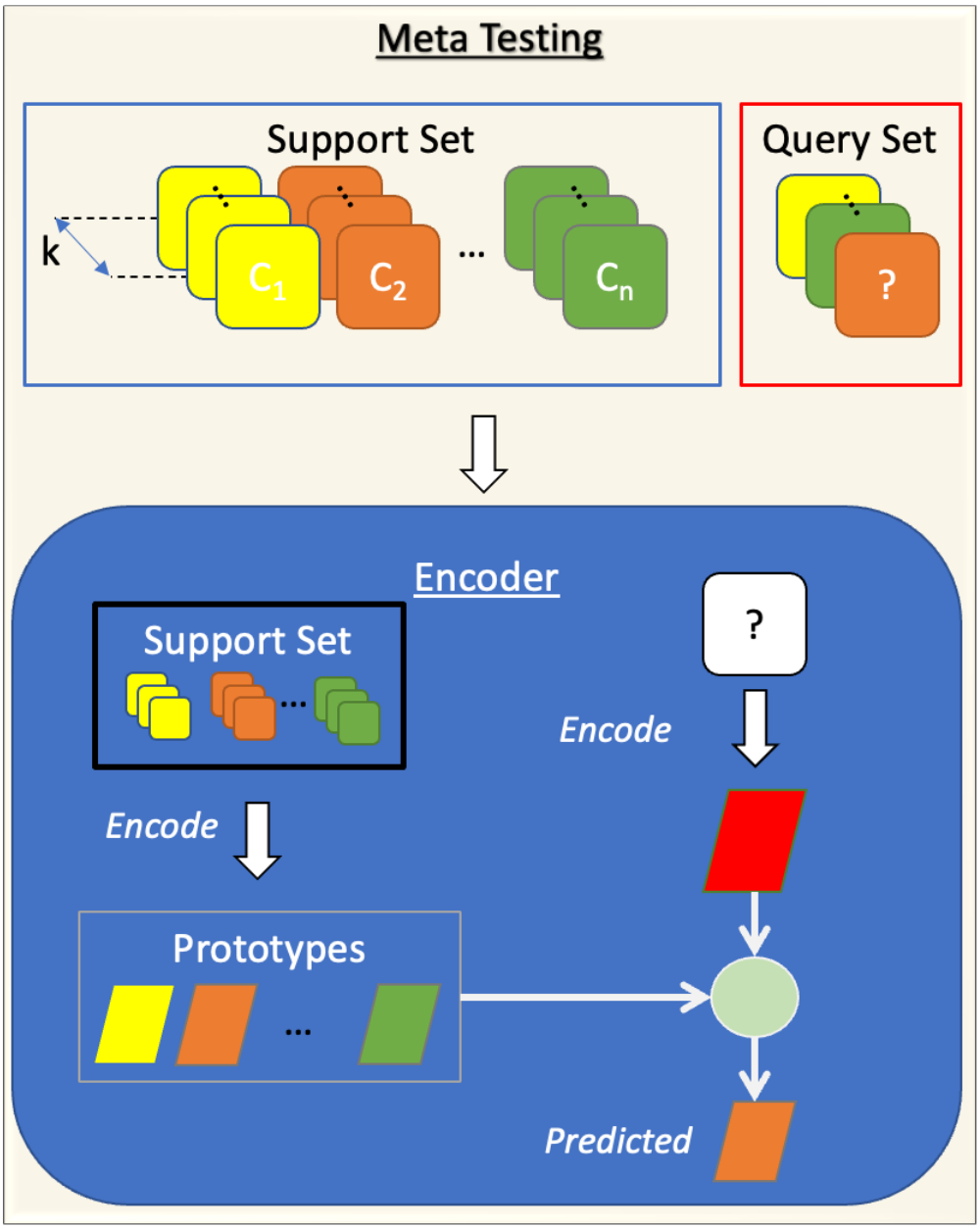


Figure 3.3: Prototypical Network Meta Testing Phase

The following two sections will explain the encoder used in the Prototypical Network approach.

3.3.1.1 CNN Encoder

In this study, one of the encoders used with the prototypical network approach is the CNN encoder, whose architecture is shown in *Figure 3.4*. We used four convolutional blocks, each one of them consists of a one-dimensional convolutional block with kernel size equal to 3 and padding equal to 1, a one-dimensional batch normalization, ReLU activation, and a one-dimensional max pool layer with kernel size equal to 2. At the end of the convolutional blocks, we used a fully connected layer which flattens all the inputs and generates a one-dimensional vector. The input channel size of the first convolutional block equals the feature size of the input. The rest of the convolutional blocks' input and output channel sizes are left as parametric, at the experimentation time we provide that information. From this point, we will call this method as *PN-CNN* method.

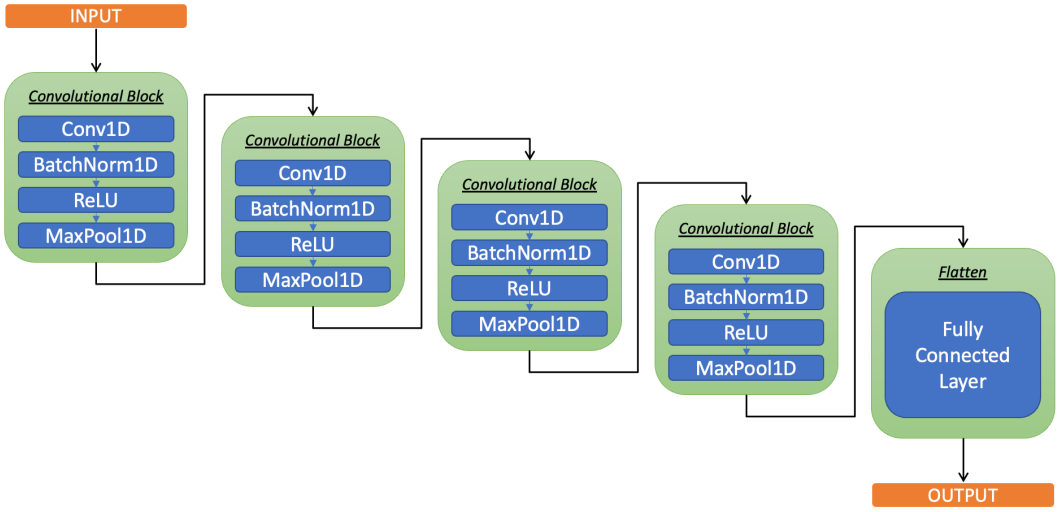


Figure 3.4: Prototypical Network CNN Encoder Architecture

3.3.1.2 ANN Encoder

Another encoder used with prototypical networks in this study is the artificial neural network encoder. Architecture of the ANN encoder is illustrated at the *Figure 3.5*. The encoder has three ANN blocks and at the end a fully connected layer to generate a one-dimensional vector. The ANN blocks are influenced by the work done by

Gamage et al. [25], even though they did not use the ANN as an encoder, we choose this architecture because it was successful at a similar task. The ANN block used in this encoder consists of a fully connected layer, a batch normalization layer, a ReLU activation layer, and a dropout layer with the probability of 0.5. At the end of the three ANN blocks, we place a fully connected layer to generate a one-dimensional vector. From this point, we will call this method as *PN-ANN* method.

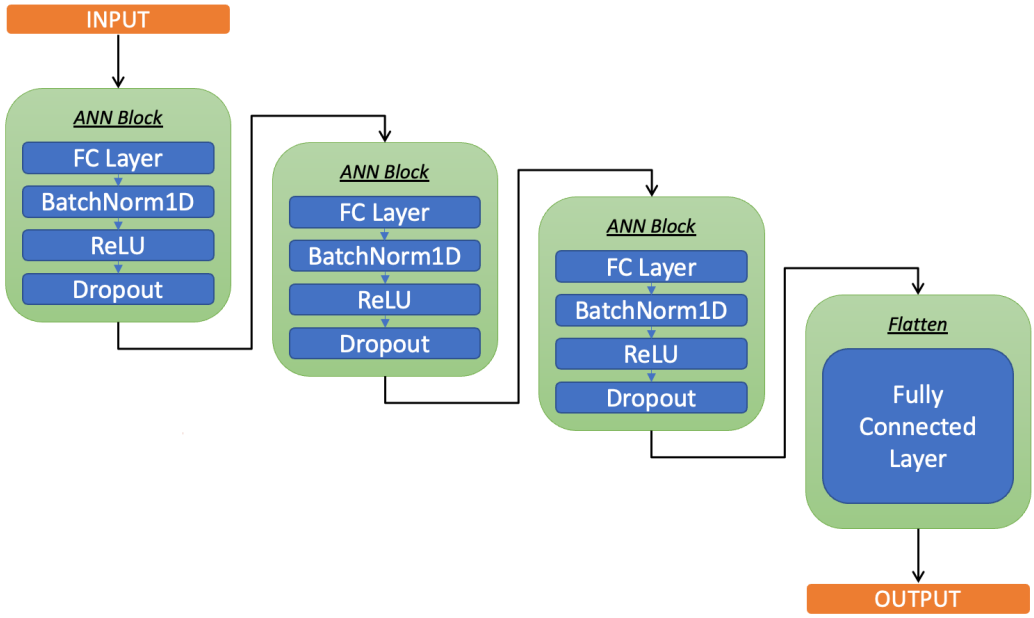


Figure 3.5: Prototypical Network ANN Encoder Architecture

3.3.2 Relation Networks for Network Attack Classification

Relation network [24] which is proposed by Sung et al., is similar to prototypical networks, additionally, it tries to learn the comparison method also. The approach first calculates mappings to an embedding space. Then, it generates pairs of prototype and query embeddings. These pairs feed into a network called *Relation Network*, which aims to calculate relation score of the pairs. The pairs that contain instances from the same class get higher relation scores, whereas, the pairs that consist of instances from different classes get low relation scores. This method also uses the meta learning framework. It has two phases, *Meta Learning* and *Meta Testing*. Meta learning phase is shown in the *Figure 3.6*.

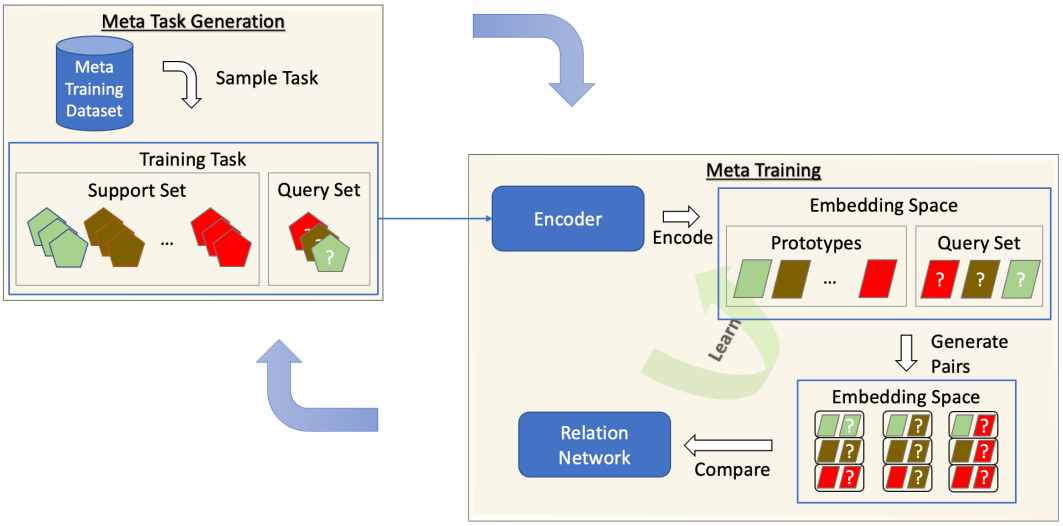


Figure 3.6: Relation Network Meta Training

During the meta learning phase, this method uses episodic training. That is, taking tasks relatively similar to the main goal at each episode and trying to learn the episode task first, hoping to be successful at the meta test task. Firstly, we generate a meta train task using samples from the meta training dataset. Each meta training task has a support set and a query set. Secondly, the encoder calculates embedding space vectors of all samples in the support and query sets. Then, we generate pairs of embedding vectors, one of the pairs is from the support set and the other one is from the query set. These pairs are fed into the relation network, and it tries to maximize the relation score of the pairs with the same class and minimize the relation score of the other pairs. During the training phase, we used the Adam optimizer with the parametric learning rate. The learning rate is decreased at each episode by using the decay parameter at the test time. We also use the early stopping method proposed in the previous section. If not decided to early stop or the batch size is not reached, this process continues from the first step.

At the meta test time of this approach, like the meta training phase, the encoder first encodes the support set and the query set into the embedding space. Then we construct pairs of embedding vectors. Next, the relation network predicts the query label by selecting the highest relation score of the pairs. As done at the prototypical approach, we test 1000 meta test tasks by sampling different support and query sets and

taking the average of them in order to get accurate metrics.

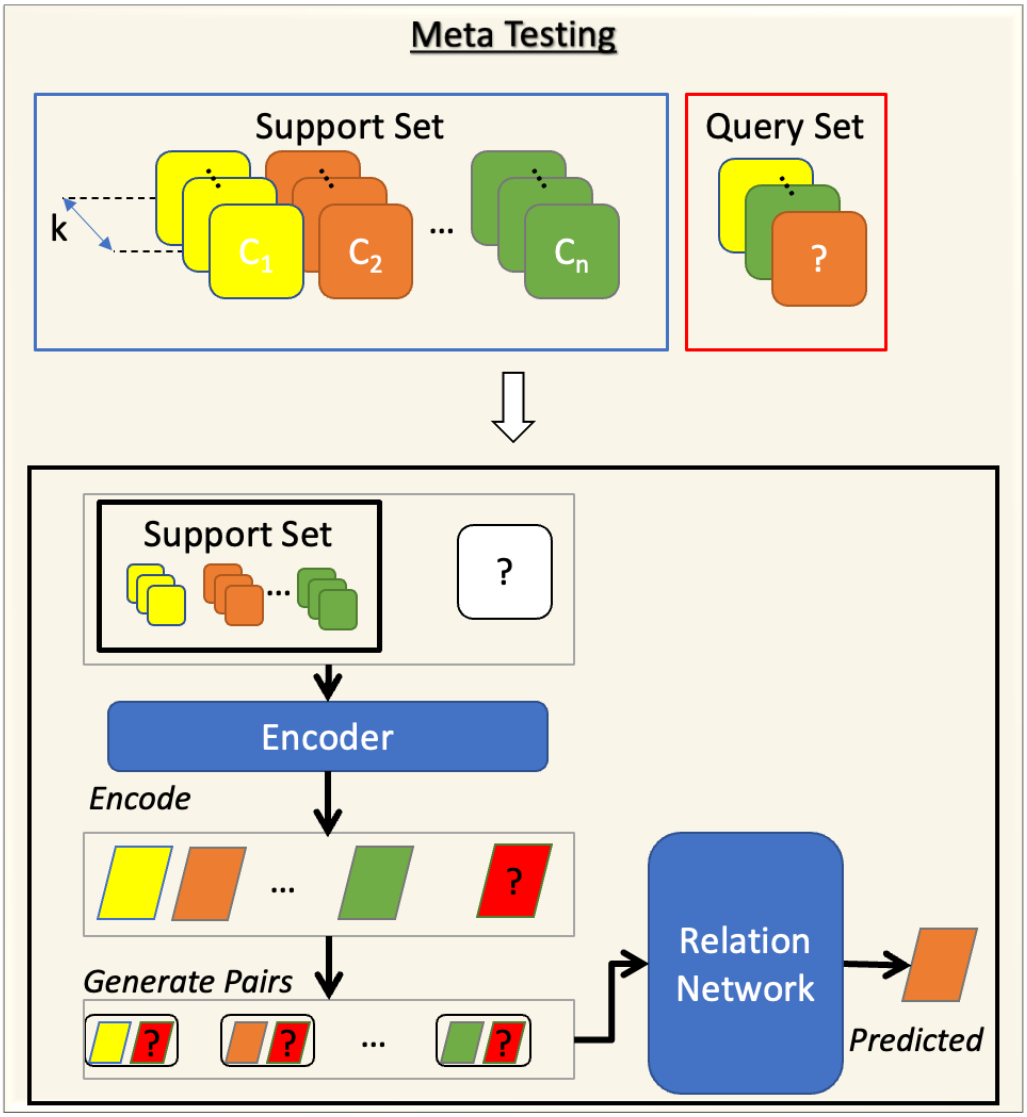


Figure 3.7: Relation Network Meta Testing

As mentioned earlier, this approach needs an encoder in order to map inputs to embedding space vectors. *Figure 3.8* shows architecture of encoder that we use for this method. The encoder consists of four convolutional blocks. The first two convolutional blocks consist of a one-dimensional convolutional layer, a batch normalization layer, a ReLU activation layer, and a max pool layer, respectively. The one-dimensional convolutional layers used in these blocks have a kernel of 3 and padding of 1, and input and output channels are left parametric so that we can change them

at experimentation. The max pool layers have kernel size 2. The remaining convolutional blocks are similar to the first two of them except they do not utilize the ReLU layer, since, we will use ReLU layers also in the Relation Network architecture.

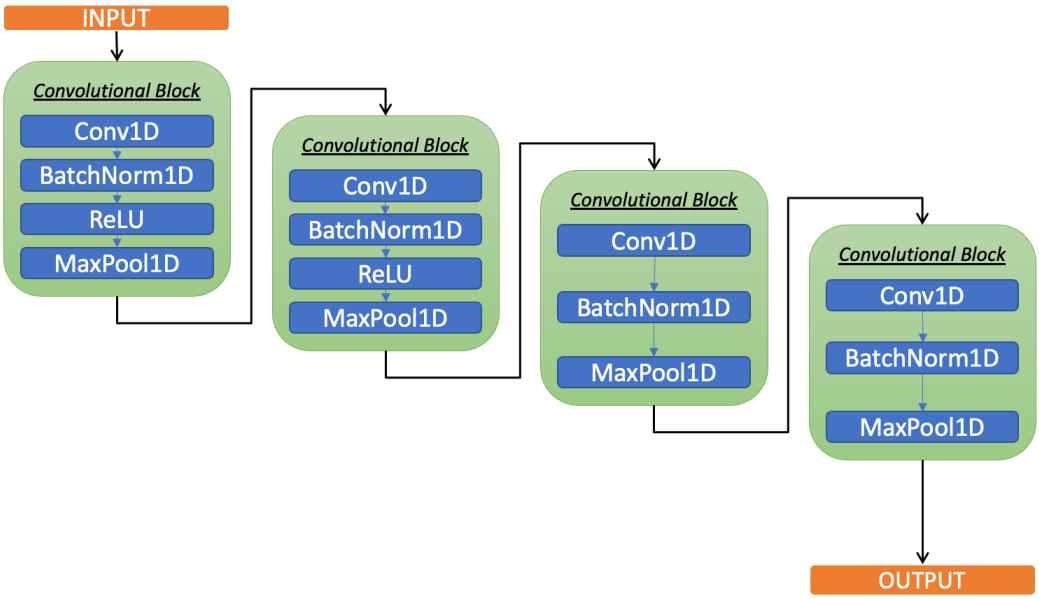


Figure 3.8: Relation Network Encoder Architecture

Learning to compare is done by the relation network. *Figure 3.9* illustrates the relation network architecture used in this study. This network aims to calculate a number that represents the relation score of the provided input, which consists of two embedded vectors. The network architecture contains two convolutional blocks, a flatten block, and two fully connected layers respectively. The convolutional blocks are the same as the first two convolutional blocks used at the encoder. In the middle, the flatten block resides, and it aims to create a one-dimensional vector. This block is required since we used different datasets, that have different sizes of feature vectors. In order to continue after convolutional blocks, we need to arrange output vectors. Then, a fully connected layer with a ReLU activation layer takes that output as input. In the end, another fully connected layer resides. The difference between this block and the previous one is that it utilizes a sigmoid function instead of ReLU since we need to end with a number that shows the relation score. From this point, we will call this method as *RN-CNN* method.

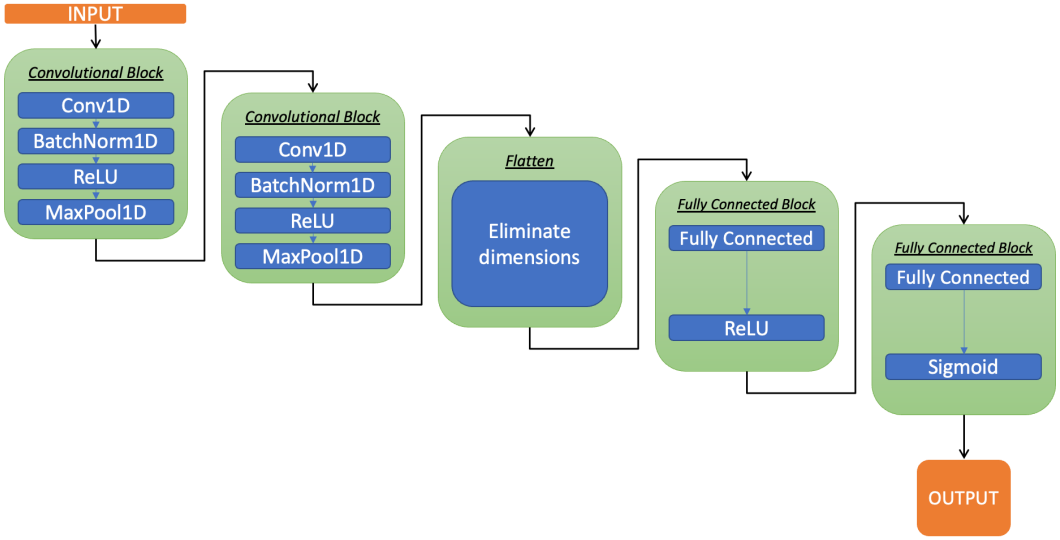


Figure 3.9: Relation Network Architecture

CHAPTER 4

EXPERIMENTS AND EVALUATION

4.1 Datasets

The methods used in this study are based on a meta learning framework that has two main phases. The first one is called the meta-training phase and the second one is called the meta testing phase. As the name implies, in the meta training phase, model training occurs. In this phase, the model tries to learn different tasks at each episode. We use two IDS datasets in order to generate a new dataset for few-shot network intrusion classification problem. Therefore, we need to prepare appropriate meta-training and meta-testing datasets. Additionally, in order to compare results with the ANN method, we also prepare a meta-validation set that contains classes from both meta-training and meta-testing datasets. After this point, we will explain the details of the mentioned datasets.

4.1.1 CIC-IDS2017

The first large intrusion dataset is CIC-IDS2017 [31] prepared by Sharafaldin et al., which contains five days of network flows. After collecting raw network packets, they extract flows and label them. Then they calculate feature vectors of flows which consists of 78 numeric features and an attack label. The dataset contains 15 different classes. We chose this dataset since it has more attack types than other datasets. After eliminating invalid values such as *NaN* and *inf*, we randomly choose 600 samples from 12 classes, since some of the classes have less instances than others. Then we split this into three datasets which are meta train, meta test, and meta validation. The meta train dataset contains 8 classes and each class has 470 samples. The meta

test data set consists of 4 attack types, 470 samples per class. Meta train and meta test datasets have different classes. The remaining samples form the meta validation dataset which contains samples from all 12 classes.

4.1.1.1 CIC-IDS2017 FSL-1 Dataset

The first collection of FSL datasets prepared using CIC-IDS2017 is named *CIC-IDS2017 FSL-1 Dataset*. The *Table 4.1* shows classes and their distributions over datasets. As presented in the table, meta training dataset classes are Benign, DDos, Dos Goldeneye, Dos Hulk, Dos Slowhttptest, Dos Slowloris, FTP-Patrator, Port Scan, SSH-Patrator. Each of the classes has 470 samples, in total, the meta training dataset contains 3760 samples. The meta testing dataset consists of the following classes: Bot, DoS Slowhttptest, Web Attack Brute Force, Web Attack XSS. Each class has 470 samples as in the case of the meta train dataset. The meta test dataset contains 1880 samples in total. The meta validation set contains all classes from the meta test and meta train datasets, and each class has 130 samples. In total, the meta validation dataset contains 1560 samples.

Table 4.1: CIC-IDS2017 FSL-1 Dataset

Meta Train Dataset		Meta Test Dataset		Meta Validation Dataset	
Label	Sample Count	Label	Sample Count	Label	Sample Count
BENIGN	470	Bot	470	BENIGN	130
DDoS	470	DoS Slowhttptest	470	DDoS	130
DoS GoldenEye	470	Web Attack Brute Force	470	DoS GoldenEye	130
DoS Hulk	470	Web Attack XSS	470	DoS Hulk	130
DoS slowloris	470			DoS slowloris	130
FTP-Patator	470			FTP-Patator	130
PortScan	470			PortScan	130
SSH-Patator	470			SSH-Patator	130
				Bot	130
				DoS Slowhttptest	130
				Web Attack Brute Force	130
				Web Attack XSS	130

4.1.1.2 CIC-IDS2017 FSL-2 Dataset

The second FSL dataset collection created using CIC-IDS2017 is called *CIC-IDS2017 FSL-2 Dataset*. The *Table 4.2* summarizes distribution of classes over datasets. As shown in the table, meta training dataset classes are Benign, DDos, Dos Goldeneye, Dos Hulk, Dos Slowhttptest, Dos Slowloris, Web Attack Brute Force, and Web Attack XSS. Each one of the classes has 470 samples, in total the meta learning dataset contains 3760 samples. The meta testing dataset consists of the following classes: Bot, FTP-Patrator, PortScan, SSH-Patrator. Each class has 470 samples like the case in the meta train dataset. The meta test dataset consists of 1880 samples in total. The meta validation set contains all classes from meta test and meta train datasets and each class has 130 samples. In total, the meta validation dataset contains 1560 samples.

Table 4.2: CIC-IDS2017 FSL-2 Dataset

Meta Train Dataset		Meta Test Dataset		Meta Validation Dataset	
Label	Sample Count	Label	Sample Count	Label	Sample Count
BENIGN	470	Bot	470	BENIGN	130
DDoS	470	FTP-Patator	470	Bot	130
DoS GoldenEye	470	PortScan	470	DDoS	130
DoS Hulk	470	SSH-Patator	470	DoS GoldenEye	130
DoS Slowhttptest	470			DoS Hulk	130
DoS slowloris	470			DoS Slowhttptest	130
Web Attack Brute Force	470			DoS slowloris	130
Web Attack XSS	470			FTP-Patator	130
				PortScan	130
				SSH-Patator	130
				Web Attack Brute Force	130
				Web Attack XSS	130

4.1.2 UNSW-NB15

In order to prepare the FSL dataset, the second large IDS dataset used in this study is UNSW-NB15 [32]. It was developed by Moustafa et al. from UNSW Sydney

University. This dataset contains 9 attack types in total, which is less than the previous dataset but still above the literature. Therefore, we choose this data set as the second choice. As we did in the previous dataset generation process, first we drop samples that contain invalid values like *NaN* and *inf*. Then we select 600 samples per class.

4.1.2.1 UNSW-NB15 FSL Dataset

The last FSL dataset used in this study is generated using UNSW-NB15 dataset. After selecting 600 samples per classes, we split them as shown in the *Table 4.3*. The meta train dataset contains 6 classes which consist of DoS, Exploits, Fuzzers, Generic, Normal, Reconnaissance. The meta test dataset has 3 classes which are Analysis, Backdoor, and Shellcode. Each class at meta train and meta test dataset has 470 samples. Like the other generated FSL datasets, the meta validation dataset contains both classes from the meta train and meta test dataset, and each class has 130 samples in the meta validation dataset.

Table 4.3: UNSW-NB15 FSL Dataset

Meta Train Dataset		Meta Test Dataset		Meta Validation Dataset	
Label	Sample Count	Label	Sample Count	Label	Sample Count
DoS	470	Analysis	470	Normal	130
Exploits	470	Backdoor	470	Analysis	130
Fuzzers	470	Shellcode	470	Backdoor	130
Generic	470			DoS	130
Normal	470			Exploits	130
Reconnaissance	470			Fuzzers	130
				Generic	130
				Reconnaissance	130
				Shellcode	130

4.2 Evaluation Metrics

In this study, we used the following evaluation metrics to compare the results of the methods. These metrics are well-defined, de facto metrics in machine learning literature. *True Positive (TP)* is the number of correct predictions made by the classifier. *True Negative (TN)* is the number of negative correct predictions. *False Positive (FP)* is the number of predictions in which the classifier labels negative classes as positive. *False Negative (FN)* is the number of predictions in which the classifier labels positive classes as negative. In order to understand more clearly, we can investigate the example multiclass confusion matrix shown in *Figure 4.1*.



Figure 4.1: Example Multiclass Confusion Matrix

For the class *DOG* the metrics *TP*, *TN*, *FP*, *FN* are calculated as follows:

$$TP = 1$$

$$TN = 5 + 6 + 8 + 9$$

$$FP = 2 + 3$$

$$FN = 4 + 7$$

Using the definition above, the metrics *Accuracy*, *Precision*, *Recall* and *F1-Score* are

calculated.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1\ score = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN}$$

4.3 Experiments

As a starting point, we need to show that the ANN method cannot learn classes that are unseen during training time. To prove that ANN requires all classes at training time, the following experimentation is made. The number of nodes at ANN's output layer is set to the number of meta validation classes, and the ANN model is trained using only the meta train dataset. Note that the meta validation dataset contains all classes from both meta train and meta test datasets where the meta train and meta test datasets have disjoint classes. The results are shown in *Table 4.4*. As the table implies, the ANN method cannot predict correctly any of the unseen classes. The F1 Score of the unseen classes is equal to zero as shown in red marked cells.

Table 4.4: ANN Performance on CIC-IDS2017 FSL1 meta test set trained only using meta train set classes

k	Accuracy	F1 Score	Benign	DDoS	DoS Gold-enEye	DoS Hulk	DoS slowloris	FTP-Patator	PortScan	SSH-Patator	Bot	DoS Slowhttptest	Web Attack Brute Force	Web Attack XSS
5	0.637	0.553	0.426	0.974	0.949	0.977	0.563	0.989	0.851	0.908	0	0	0	0
10	0.653	0.568	0.382	0.970	0.952	0.962	0.684	0.989	0.977	0.905	0	0	0	0
15	0.649	0.567	0.389	0.970	0.952	0.966	0.694	0.953	0.974	0.904	0	0	0	0
20	0.651	0.568	0.394	0.977	0.952	0.981	0.686	0.964	0.956	0.908	0	0	0	0
25	0.654	0.569	0.392	0.963	0.948	0.966	0.705	0.992	0.955	0.902	0	0	0	0
50	0.656	0.573	0.386	1.000	0.952	0.981	0.686	0.996	0.974	0.901	0	0	0	0

For this study, we prepared the following test setups to investigate the few-shot network attack classification problem.

- **Experiment 1** is a setting that uses the CIC-IDS2017 FSL-1 dataset explained previously.

- **Experiment 2** is a setting that uses the CIC-IDS2017 FSL-2 dataset explained previously.

- **Experiment 3** is a setting that uses the UNSW-NB15 FSL dataset explained previously.

For all experiment settings mentioned above, we performed the actions below:

- Extracted k samples for each class from the meta test set to train the ANN model. Trained ANN model using those samples. After training the ANN model, we test it using the remaining meta test set and collect performance metrics. We take an average of the performance metrics by repeating this process at least 10 times.

- Trained *PN-CNN* method using meta train dataset, evaluated using meta test dataset, and collected performance metrics.

- Trained *RN-CNN* method using meta train dataset. Collected performance metrics on meta test set using different training parameters.

- Trained *PN-ANN* method using meta train dataset. Collected performance metrics on meta test set using different training parameters.

- Extracted k samples for each class from the meta test set and combined with the meta train set for only using this step. Trained ANN model using extended meta train dataset. After training the ANN model, we test it using the meta validation set and collect performance metrics. We take the average of the performance metrics by repeating this process at least 10 times.

- Trained *PN-CNN* method using meta train dataset, evaluated using meta validation dataset, and collect performance metrics.

- Trained *RN-CNN* method using meta train dataset. Collected performance metrics on meta validation set using different training parameters.

- Trained *PN-ANN* method using meta train dataset. Collected performance metrics on meta validation set using different training parameters.

For the steps where we trained *PN-CNN*, *RN-CNN* and *PN-ANN*, we changed model parameters like CNN's hidden layer's neuron counts, k used at meta training phase, learning rate decay rate used at training phase, meta train epoch size and epoch count, ANN hidden layer's neuron counts, etc. to search for best performance of the method that we are working on. These steps are executed for all three FSL datasets generated by this study, and the results achieved are explained in the next sections.

As a remark, for the actions for training the ANN method, we enlarge the training dataset of ANN in order to make ANN learn test classes. To explain more specifically, we used a train set by adding classes from the meta test dataset by randomly selecting k samples per class from the meta test dataset and dropping them from the meta test dataset. In other words, the training dataset of ANN contains the meta train dataset and k examples per class from the meta test dataset. As an important remark, different from ANN, FSL methods used in this study do not need to train with meta test classes during the training phase. They aim to generalize a model that can classify even if it does not see classes at the train set, and just use unseen class samples as input at prediction time.

Tables from 4.5 to 4.28 show experimentation results. The first three columns show k which is the number of samples per class on the evaluation dataset, the accuracy of the method, and the F1 score of the method respectively. The remaining columns represent the F1 scores of the classes given in the column name. For the performance tables on meta validation datasets, yellow marked columns represent classes that are not used at the meta training phase of the three methods which are *PN-CNN*, *RN-CNN* and *PN-ANN*.

4.3.1 Experiments on CIC-IDS2017 FSL1 Dataset

This section details the results of experiments on the CIC-IDS2017 FSL1 Dataset. *Table 4.5* shows the results capturing performances of ANN trained using the extended meta train dataset. As the table shows, the accuracy increases as k increases, which means ANN gives better results when the number of samples in the training dataset increases. As in the case of accuracy, the f1 score of ANN increases as k increases. There is a 10.2% difference between $k = 5$ and $k = 50$ which is a huge improvement.

When we investigate individual class f1 scores, they seem to increase as k increases. This implies that ANN is a data-hungry method.

Table 4.5: *ANN* method Performances on CIC-IDS2017 FSL1 meta test set

k	Avg Accuracy	Avg f1_score	Bot	DoS Slowhttptest	Web Attack Brute Force	Web Attack XSS
5	0.638	0.625	0.810	0.806	0.418	0.465
10	0.689	0.668	0.884	0.868	0.412	0.508
15	0.709	0.683	0.914	0.899	0.399	0.521
20	0.727	0.701	0.930	0.917	0.411	0.546
25	0.732	0.707	0.949	0.930	0.441	0.508
50	0.758	0.727	0.969	0.962	0.464	0.514

Table 4.6 shows the results of the *PN-CNN* method. Unlike in the case of ANN, increasing k did not affect accuracy and f1 score much. Individual class f1 scores except Web Attack XSS are also not affected proportionally when k is increased.

Table 4.6: *PN-CNN* method Performances on CIC-IDS2017 FSL1 meta test set

k	Best Accuracy	Best f1_score	Bot	DoS Slowhttptest	Web Attack Brute Force	Web Attack XSS
5	0.680	0.625	0.893	0.867	0.340	0.534
10	0.692	0.650	0.898	0.889	0.326	0.572
15	0.696	0.657	0.886	0.878	0.321	0.588
20	0.702	0.656	0.897	0.895	0.301	0.599
25	0.693	0.661	0.891	0.885	0.323	0.623
50	0.702	0.646	0.878	0.887	0.267	0.653

Table 4.7 gives the results of the *PN-ANN* method. Different from the case of ANN, increasing k did not affect much accuracy and f1 score much. F1 scores of the meta test dataset classes are also not affected by the increase of k, except for the Web Attack XSS class.

Table 4.7: *PN-ANN* method Performances on CIC-IDS2017 FSL1 meta test set

k	Best Accuracy	Best f1_score	Bot	DoS Slowhttptest	Web Attack Brute Force	Web Attack XSS
5	0.677	0.633	0.898	0.857	0.312	0.478
10	0.647	0.586	0.866	0.779	0.239	0.473
15	0.668	0.605	0.888	0.851	0.248	0.498
20	0.674	0.608	0.869	0.861	0.216	0.523
25	0.694	0.640	0.903	0.904	0.257	0.532
50	0.689	0.624	0.874	0.903	0.171	0.592

In the *Table 4.8*, the results of the *RN-CNN* method are shown. Like the previous FSL techniques, increasing k did not affect accuracy much. However, the f1 score is increased with the increase in k. Additionally, different from previous FSL methods, as k increases, f1 scores of Web Attack Brute Force and Web Attack XSS mostly increase.

Table 4.8: *RN-CNN* method Performances on CIC-IDS2017 FSL1 meta test set

k	Best Accuracy	Best f1_score	Bot	DoS Slowhttptest	Web Attack Brute Force	Web Attack XSS
5	0.643	0.564	0.827	0.935	0.444	0.456
10	0.656	0.599	0.837	0.948	0.491	0.470
15	0.687	0.630	0.858	0.949	0.506	0.584
20	0.697	0.637	0.888	0.955	0.557	0.573
25	0.679	0.638	0.893	0.931	0.515	0.577
50	0.694	0.635	0.882	0.953	0.572	0.634

Figure 4.2 gives a comparison of meta test performance of the methods, when k is equal to 5. As represented in the figure, *PN-ANN* and *PN-CNN* are leading in accuracy, but at the f1 score *PN-ANN* gives a slightly better result. The highest Bot f1 score is achieved by *PN-ANN* again. For DoS Slowhttptest and Web Attack Brute Force classes, *RN-CNN* outperforms other methods in terms of class f1 scores. On the other hand, *PN-CNN* performs better than other methods on Web Attack XSS. For k equals 5, *PN-ANN* gives better performance on the CICIDS2017 FSL1 meta test dataset.

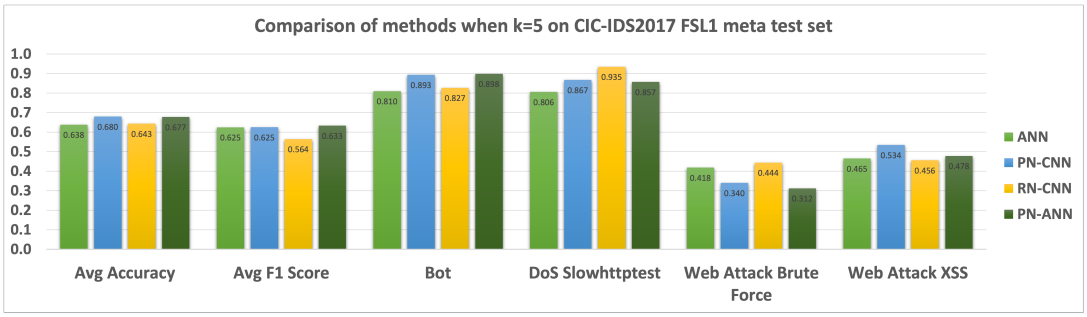


Figure 4.2: CIC-IDS2017 FSL1 Meta test results when $k = 5$

Figure 4.3 shows performance of methods on the CICIDS2017 FSL1 meta test dataset. *PN-CNN* gives the highest accuracy over the methods. However, it lost the first place to ANN at the f1 score. *PN-CNN* gives better results for Bot and Web Attack XSS classes. *RN-CNN* method performs well on the remaining classes.

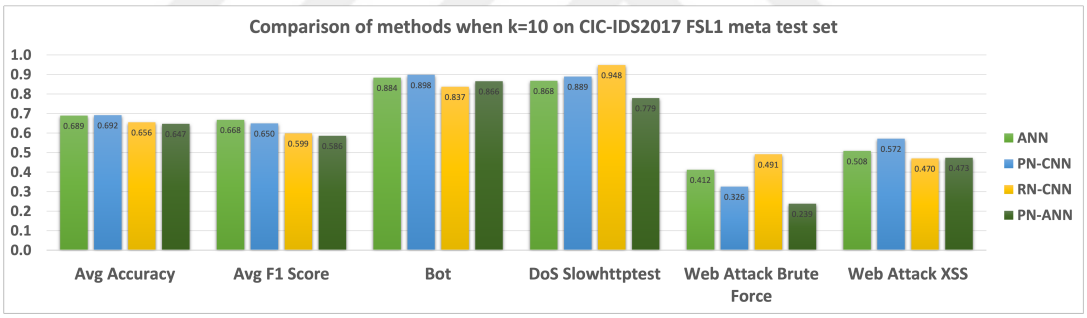


Figure 4.3: CIC-IDS2017 FSL1 Meta test results when $k = 10$

Figure 4.4 illustrates the captured performance on the CICIDS2017 FSL1 meta test dataset, when k is increased to 15. At this configuration, the order of the performance of methods on both accuracy and f1 score is the following respectively: ANN, *PN-CNN*, *RN-CNN*, and *PN-ANN*. Even the good performance on DoS Slowhttptest and Web Attack Brute Force classes individually did not rescue *RN-CNN* from fitting in third place overall. ANN method performed well on Bot class, whereas *PN-CNN* was the best at classifying Web Attack XSS class correctly.

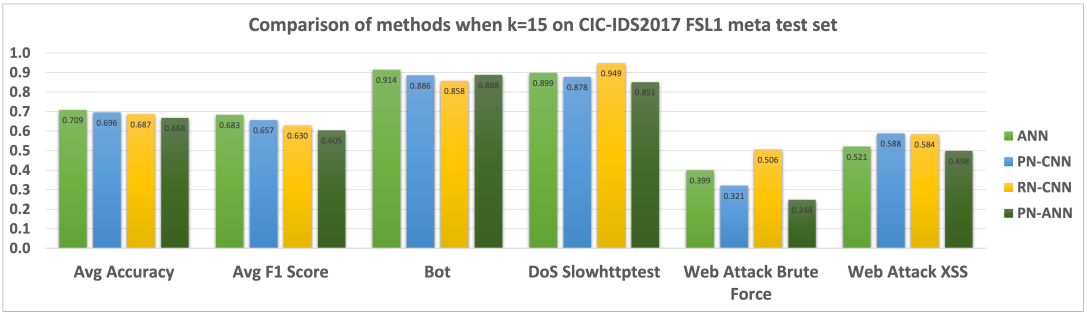


Figure 4.4: CIC-IDS2017 FSL1 Meta test results when $k = 15$

Figure 4.5 shows performances of methods on CICIDS2017 FSL1 meta test dataset for $k = 20$. As k increases, the performance of ANN is improved. On both accuracy and f1 score, ANN gets a higher score. The other methods are ordered as *PN-CNN*, *RN-CNN*, and *PN-ANN* at overall performance like the case where k is equal to 15. *RN-CNN* performs well on DoS Slowhttptest and Web Attack Brute Force classes interestingly.

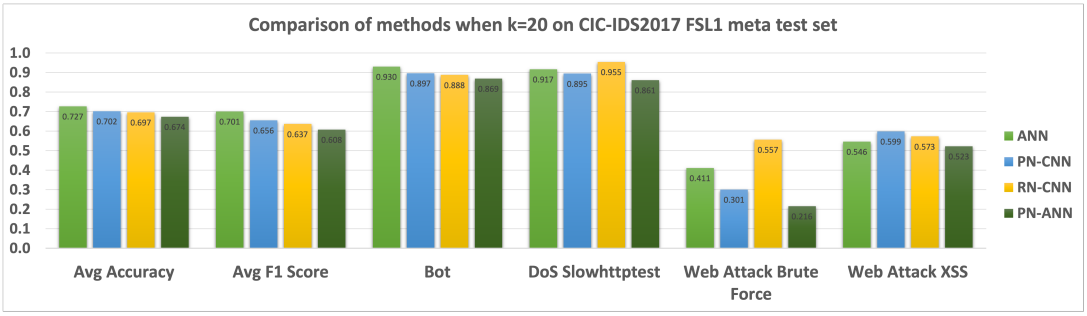


Figure 4.5: CIC-IDS2017 FSL1 Meta test results when $k = 20$

Figure 4.6 shows performances of methods on the CICIDS2017 FSL1 meta test dataset for k equals 25. On both accuracy and f1 score, ANN gets a higher score. ANN also gets the highest f1 score on Bot class.

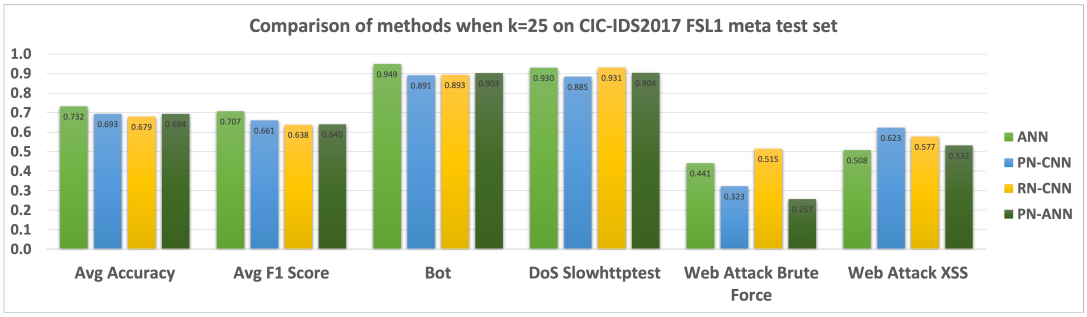


Figure 4.6: CIC-IDS2017 FSL1 Meta test results when $k = 25$

Figure 4.7 shows performances of methods on CICIDS2017 FSL1 meta test dataset for k equals 50. On both accuracy and f1 score, ANN gets the highest score. Additionally, ANN beats the other methods on the Bot and DoS Slowhttptest classes.

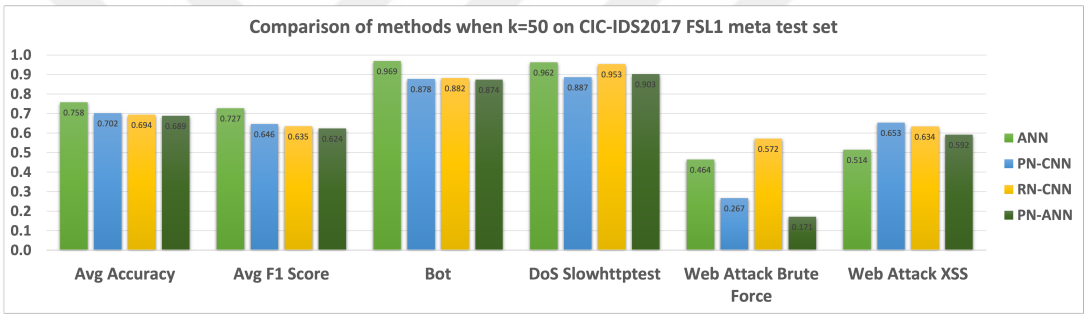


Figure 4.7: CIC-IDS2017 FSL1 Meta test results when $k = 50$

For the cases where the value of the variable k is bigger than 15, ANN works better on the Bot class. Except k is equal to 50, *RN-CNN* performs best on the Dos Slowhttptest class. Additionally, for the Web Attack Brute Force class, *RN-CNN* achieves the highest f1 score. For all k values that we examine, *PN-CNN* acheieves the highest f1 score on the Web Attack XSS class.

Table 4.9, *Table 4.10*, *Table 4.11* and *Table 4.12* show performances on the CICIDS2017 FSL1 meta validation dataset for the methods ANN, *PN-CNN*, *PN-ANN* and *RN-CNN* respectively. The yellow mark shows the classes that the meta validation dataset contains but the meta train dataset does not contain.

As in the case of meta test dataset performance, ANN performance on meta validation

dataset increases with the increase of k, as shown in the *Table 4.9*.

Table 4.9: *ANN* method Performances on CIC-IDS2017 FSL1 meta validation set

k	accuracy	f1_score	BENIGN	Bot	DDoS	DoS Golden-Eye	DoS Hulk	DoS Slowhttp test	DoS slowloris	FTP-Patator	PortScan	SSH-Patator	Web Attack Brute Force	Web Attack XSS
5	0.769	0.725	0.588	0.174	0.975	0.955	0.971	0.689	0.846	0.976	0.962	0.924	0.197	0.440
10	0.798	0.762	0.648	0.407	0.967	0.962	0.972	0.739	0.855	0.983	0.965	0.922	0.215	0.506
15	0.808	0.776	0.691	0.469	0.978	0.965	0.975	0.741	0.860	0.975	0.969	0.937	0.490	0.265
20	0.834	0.807	0.756	0.701	0.977	0.965	0.974	0.783	0.881	0.986	0.963	0.928	0.257	0.510
25	0.806	0.776	0.658	0.498	0.965	0.962	0.954	0.762	0.873	0.971	0.958	0.935	0.214	0.567
50	0.852	0.831	0.776	0.816	0.976	0.974	0.973	0.862	0.912	0.978	0.952	0.946	0.311	0.493

Table 4.10 shows the performance on the meta validation dataset of *PN-CNN* method. An increase in k seems to increase the overall performance of the method unlike the performance of the meta test dataset.

Table 4.10: *PN-CNN* method Performances on CIC-IDS2017 FSL1 meta validation set

k	accuracy	f1_score	BENIGN	Bot	DDoS	DoS Golden-Eye	DoS Hulk	DoS Slowhttp test	DoS slowloris	FTP-Patator	PortScan	SSH-Patator	Web Attack Brute Force	Web Attack XSS
5	0.756	0.752	0.602	0.744	0.945	0.965	0.915	0.815	0.834	0.911	0.757	0.866	0.600	0.661
10	0.778	0.751	0.700	0.736	0.970	0.967	0.918	0.836	0.864	0.928	0.794	0.864	0.601	0.669
15	0.791	0.785	0.700	0.736	0.961	0.974	0.917	0.838	0.860	0.925	0.834	0.894	0.580	0.667
20	0.783	0.770	0.738	0.752	0.969	0.973	0.919	0.853	0.866	0.954	0.821	0.879	0.583	0.669
25	0.802	0.798	0.749	0.755	0.981	0.972	0.931	0.850	0.856	0.933	0.855	0.885	0.592	0.677
50	0.818	0.815	0.771	0.807	0.975	0.976	0.947	0.870	0.880	0.950	0.898	0.884	0.570	0.673

Table 4.11 shows the performance on the meta validation dataset of *PN-ANN* method. Overall performance of the method is not affected by the change in k, just like the performance at meta test dataset.

Table 4.11: *PN-ANN* method Performances on CIC-IDS2017 FSL1 meta validation set

k	accuracy	f1_score	BENIGN	Bot	DDoS	DoS Golden-Eye	DoS Hulk	DoS Slowhttp test	DoS slowloris	FTP-Patator	PortScan	SSH-Patator	Web Attack Brute Force	Web Attack XSS
5	0.557	0.550	0.162	0.664	0.780	0.964	0.792	0.778	0.802	0.755	0.649	0.615	0.556	0.643
10	0.519	0.502	0.122	0.388	0.747	0.955	0.717	0.674	0.578	0.715	0.445	0.631	0.504	0.655
15	0.532	0.510	0.124	0.370	0.762	0.945	0.652	0.669	0.602	0.760	0.452	0.567	0.571	0.648
20	0.553	0.522	0.115	0.374	0.748	0.952	0.718	0.789	0.613	0.760	0.459	0.615	0.501	0.614
25	0.556	0.532	0.105	0.400	0.759	0.949	0.729	0.696	0.610	0.793	0.470	0.613	0.140	0.639
50	0.555	0.537	0.128	0.382	0.795	0.952	0.690	0.692	0.585	0.782	0.451	0.607	0.531	0.661

Table 4.12 shows the performance on the meta validation dataset of *RN-CNN* method. Overall performance of the method is not directly related to k.

Table 4.12: *RN-CNN* method Performances on CIC-IDS2017 FSL1 meta validation set

k	accuracy	f1_score	BENIGN	Bot	DDoS	DoS Golden-Eye	DoS Hulk	DoS Slowhttp test	DoS slowloris	FTP-Patator	PortScan	SSH-Patator	Web Attack Brute Force	Web Attack XSS
5	0.721	0.679	0.789	0.742	0.988	0.973	0.973	0.811	0.827	1.000	0.930	0.951	0.604	0.665
10	0.778	0.736	0.780	0.479	0.988	0.964	0.976	0.819	0.848	1.000	0.944	0.952	0.610	0.661
15	0.783	0.753	0.750	0.623	0.987	0.970	0.983	0.817	0.892	1.000	0.979	0.958	0.614	0.673
20	0.783	0.745	0.808	0.682	0.987	0.973	0.978	0.766	0.829	1.000	0.956	0.960	0.608	0.662
25	0.759	0.712	0.788	0.671	0.995	0.972	0.977	0.794	0.842	1.000	0.967	0.963	0.619	0.675
50	0.763	0.729	0.772	0.639	0.994	0.988	0.988	0.779	0.855	1.000	0.952	0.970	0.613	0.667

Figure 4.8 shows performance metrics of the methods using CICIDS2017 FSL1 meta validation dataset, where k is equal to 5. Red marked columns show the classes that are not used in the training phase for FSL methods. Since ANN cannot learn unseen classes, those classes are added to ANN's training dataset. As seen from the figure, the ANN method achieves the highest accuracy, but the second-best accuracy is not so far from it. When we look at the best f1 score, we see that *PN-CNN* reaches the highest score. For the unseen classes which are Bot, DoS Slowhttptest, Web Attack Brute Force, and Web Attack XSS, the ANN approach gives the worst results among all four approaches. On the other hand, *PN-CNN* and *RN-CNN* methods both worked well on all unseen classes. Because of the small k value, we expect that ANN should

have worse results. Interestingly, for this experimentation setup, the ANN method worked well overall.

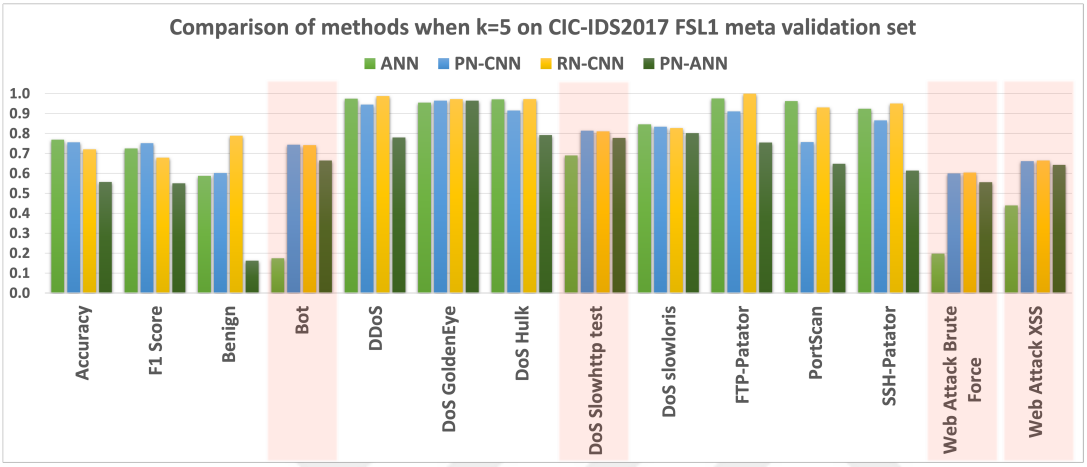


Figure 4.8: CIC-IDS2017 FSL1 Meta validation results when $k = 5$

Figure 4.9 gives detailed information about the performances of the methods using CICIDS2017 FSL1 where k is equal to 10. From the figure, it can be easily understood that *PN-CNN* and *RN-CNN* give better accuracy and f1 score than the ANN method. *PN-CNN* achieves a slightly better f1 score than *RN-CNN*. For unseen classes, the ANN approach's performance is the worst except for the DoS Slowhttptest class. For this class, ANN performance passes *PN-ANN* performance with a small distance. *PN-CNN* nearly achieves the best results for the unseen classes. Only for the Web Attack Brute Force class, *PN-CNN* lost first place to the *RN-CNN*. On the other hand, *RN-CNN* gets the second position for the remaining unseen classes with a small difference from first place.

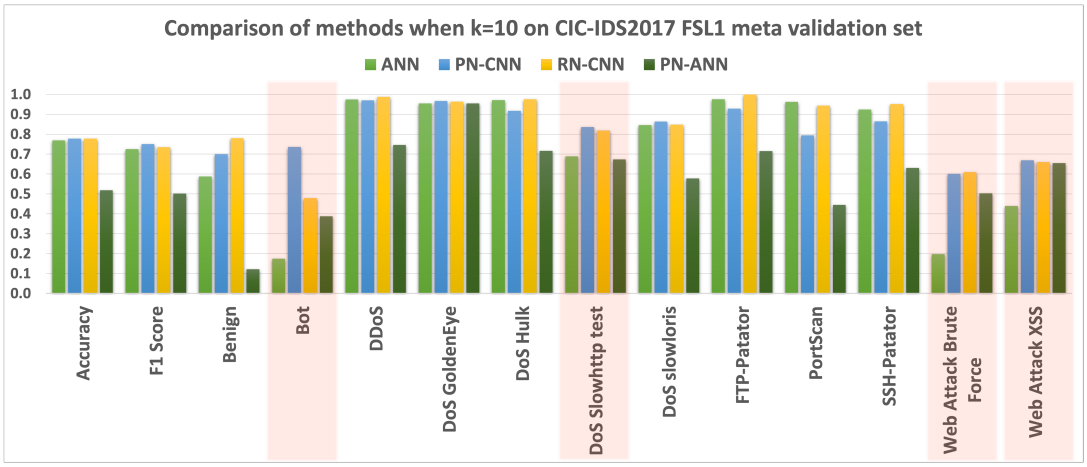


Figure 4.9: CIC-IDS2017 FSL1 Meta validation results when $k = 10$

Figure 4.10 explains the performances of the methods using CICIDS2017 FSL1 meta validation dataset, for $k = 15$. In this experimentation setup, ANN reaches the best accuracy performance over the four methods. However, it cannot perform better than *PN-CNN* in terms of the f1 score. For the Bot and DoS Slowhttptest classes, the ANN approach only beats the *PN-ANN* approach. The worst results for remaining unseen classes are performed by ANN. *PN-ANN* gave the best f1 scores for Bot and DoS Slowhttptest classes. On the other hand, for Web Attack Brute Force and Web Attack XSS classes *RN-CNN* method performed best.

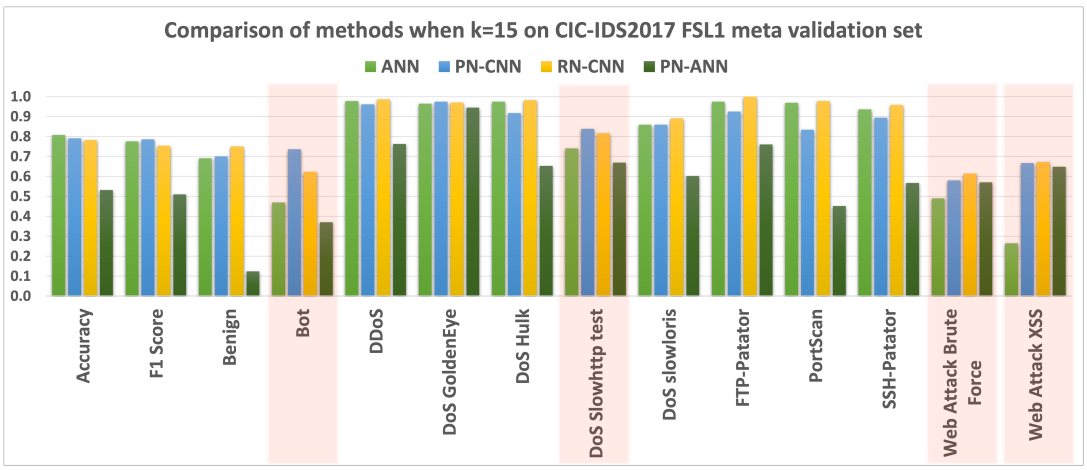


Figure 4.10: CIC-IDS2017 FSL1 Meta validation results when $k = 15$

ANN method finally performs better in terms of accuracy and f1 score than FSL

methods when k reaches 20 as shown in the *Figure 4.11*. However, it gives the worst results for the Web Attack Brute Force and Web Attack XSS classes. On the other hand, *PN-CNN* gives the best results for the unseen classes except for Web Attack Brute Force.

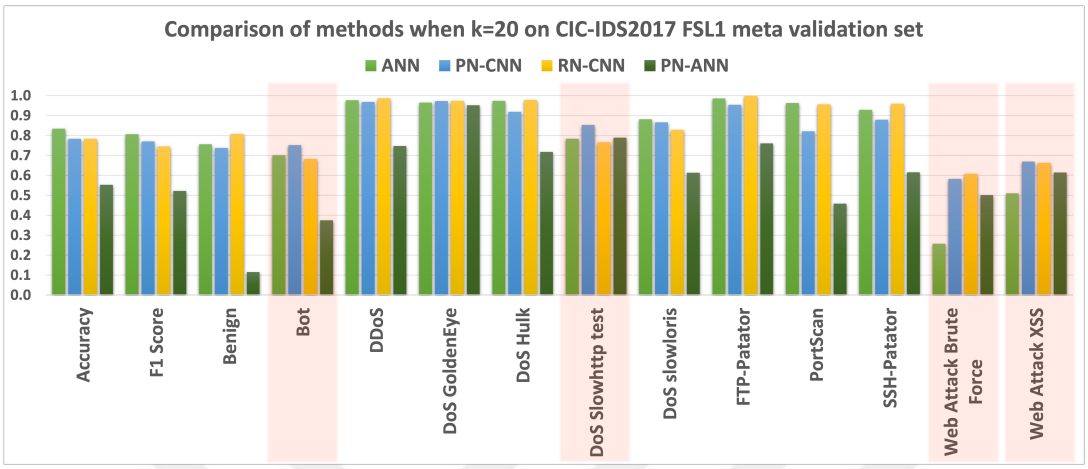


Figure 4.11: CIC-IDS2017 FSL1 Meta validation results when $k = 20$

When we increase k to the 25, *PN-CNN* passes ANN method in terms of f1 score as shown in the *Figure 4.12*. Starting from $k = 20$, ANN achieves the best accuracy on the CICIDS2017 FSL1 meta validation dataset. For the unseen classes except for Web Attack Brute Force, *PN-CNN* gives better results.

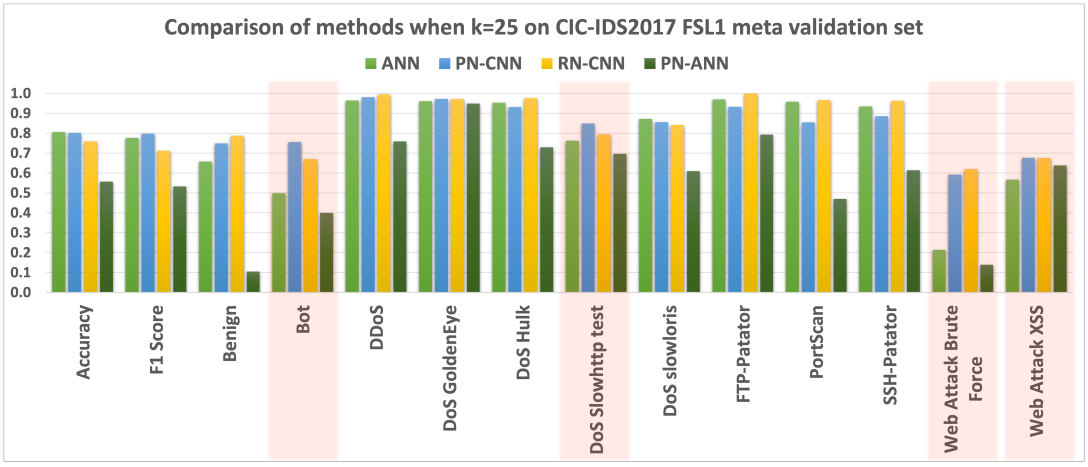


Figure 4.12: CIC-IDS2017 FSL1 Meta validation results when $k = 25$

Figure 4.13 shows the performances of all four methods using CICIDS2017 FSL1

meta validation dataset, where k is equal to 50. The main different outcome of this setup is that ANN achieves the best f1 score on Bot class.

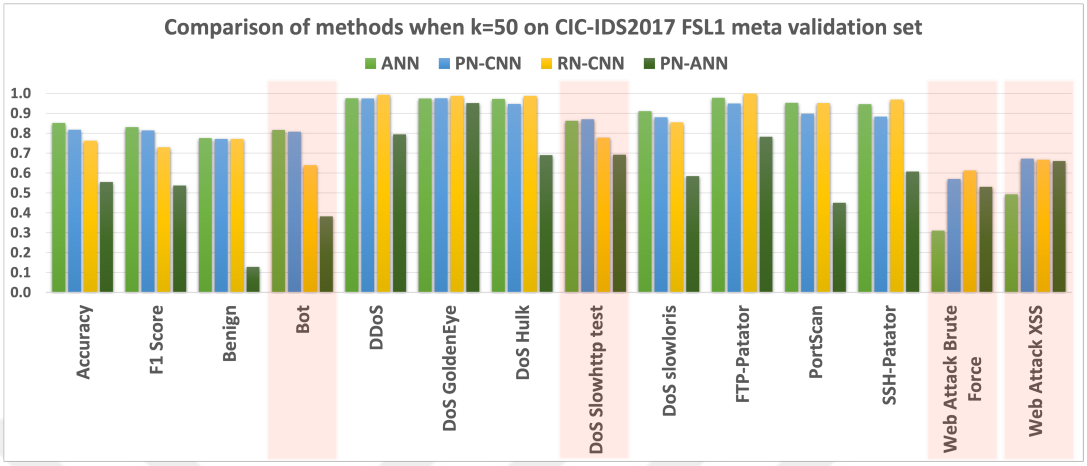


Figure 4.13: CIC-IDS2017 FSL1 Meta validation results when $k = 50$

PN-ANN approach achieved the worst accuracy and f1 score on the CICIDS2017 FSL1 meta validation dataset over four methods for all k values.

4.3.2 Experiments on CIC-IDS2017 FSL2 Dataset

This section provides experimentation results of four approaches using the CIC-IDS2017 FSL2 Dataset. As *Table 4.13* shows, ANN method performance on CICIDS2017 FSL2 meta test increased proportionally when k is increased. Even for the small training dataset, this ANN architecture gives good results on the CICIDS2017 FSL2 dataset. Remark that this architecture is proposed by study [25] done by Gamage et al., and they also find interesting that simple ANN architecture's performance on the CICIDS2017 dataset.

Table 4.13: *ANN* method Performances on CIC-IDS2017 FSL2 meta test set

k	Avg Accuracy	Avg f1_score	Bot	FTP-Patator	PortScan	SSH-Patator
5	0.867	0.863	0.792	0.873	0.855	0.932
10	0.925	0.925	0.883	0.927	0.933	0.956
15	0.960	0.960	0.937	0.972	0.949	0.980
20	0.964	0.964	0.952	0.970	0.957	0.978
25	0.978	0.978	0.969	0.983	0.976	0.986
50	0.982	0.982	0.976	0.987	0.981	0.984

PN-CNN performance on CICIDS2017 FSL2 meta test dataset is shown in the *Table 4.14*. The performance of the method is nearly proportional to k.

Table 4.14: *PN-CNN* method Performances on CIC-IDS2017 FSL2 meta test set

k	Best Accuracy	Best f1_score	Bot	FTP-Patator	PortScan	SSH-Patator
5	0.591	0.590	0.613	0.602	0.627	0.657
10	0.623	0.620	0.626	0.683	0.648	0.653
15	0.607	0.610	0.683	0.720	0.627	0.652
20	0.663	0.672	0.712	0.684	0.640	0.706
25	0.674	0.683	0.635	0.741	0.606	0.779
50	0.672	0.687	0.743	0.762	0.676	0.666

Performance of *PN-ANN* on the CICIDS2017 FSL2 meta test dataset is shown in the *Table 4.15*. As the table implies, chancing at k did not affect directly the performance of the method.

Table 4.15: *PN-ANN* method Performances on CIC-IDS2017 FSL2 meta test set

k	Best Accuracy	Best f1_score	Bot	FTP-Patator	PortScan	SSH-Patator
5	0.471	0.405	0.450	0.332	0.547	0.344
10	0.474	0.415	0.463	0.313	0.554	0.350
15	0.470	0.409	0.508	0.312	0.540	0.405
20	0.466	0.403	0.484	0.296	0.546	0.454
25	0.469	0.403	0.505	0.300	0.567	0.438
50	0.471	0.411	0.479	0.338	0.564	0.453

Performance of *RN-CNN* on the CICIDS2017 FSL2 meta test dataset is shown in the *Table 4.16*. Like in the case of *PN-ANN*, chancing at k did not affect directly the performance of the method.

Table 4.16: *RN-CNN* method Performances on CIC-IDS2017 FSL2 meta test set

k	Best Accuracy	Best f1_score	Bot	FTP-Patator	PortScan	SSH-Patator
5	0.638	0.617	0.570	0.529	0.628	0.739
10	0.442	0.398	0.538	0.491	0.547	0.559
15	0.660	0.627	0.695	0.484	0.692	0.795
20	0.515	0.486	0.592	0.583	0.598	0.661
25	0.440	0.376	0.555	0.592	0.612	0.652
50	0.577	0.574	0.660	0.621	0.587	0.785

From *Figure 4.14* to *Figure 4.19*, we compare the performances of the four methods on CICIDS2017 FSL2 meta test dataset. For all performance metrics, ANN achieves the highest score. However, the ANN method needs to train using all classes, whereas FSL techniques did not require to use those classes during the training phase.

As *Figure 4.14* shows, *RN-CNN* gives best performance among FSL techniques on both accuracy and f1 score. It also beats other FSL methods on SSH-Patator and

PortScan classes. For the classes, FTP-Patator and Bot, *PN-CNN* is the best method among FSL approaches.

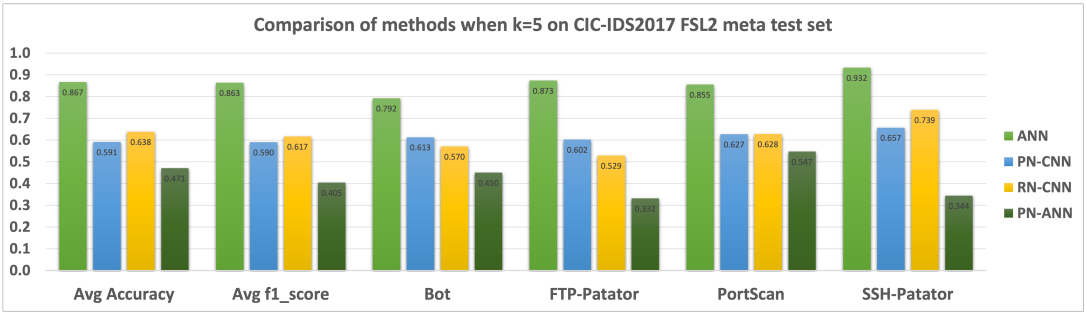


Figure 4.14: CIC-IDS2017 FSL2 Meta test results when $k = 5$

For k equals 10, among FSL approaches, *PN-CNN* gives best performance on both accuracy and f1 score using the CICIDS2017 FSL2 meta test dataset, as shown in the *Figure 4.15*. Additionally, it achieves the best f1 scores on unseen classes among FSL approaches.

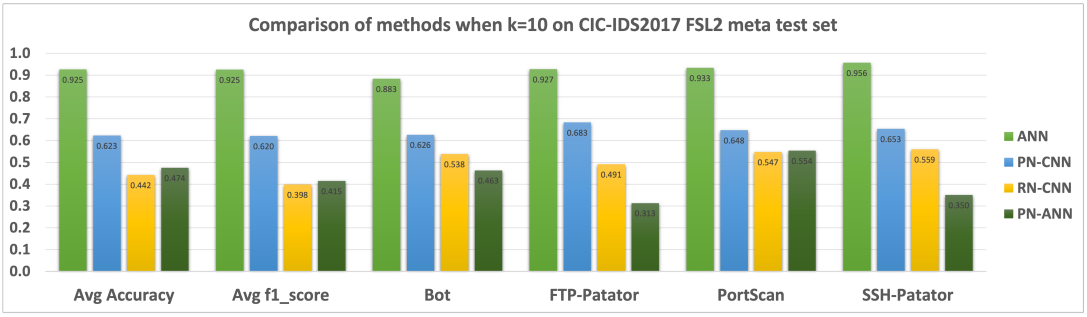


Figure 4.15: CIC-IDS2017 FSL2 Meta test results when $k = 10$

Figure 4.16 shows performances of the methods using the CICIDS2017 FLS2 meta test dataset, when k is equal to 15. As shown in the figure, *RN-CNN* beats the other two FSL methods on all metrics except FTP-Patator class. For the FTP-Patator class, *PN-CNN* takes the lead over the FSL methods.

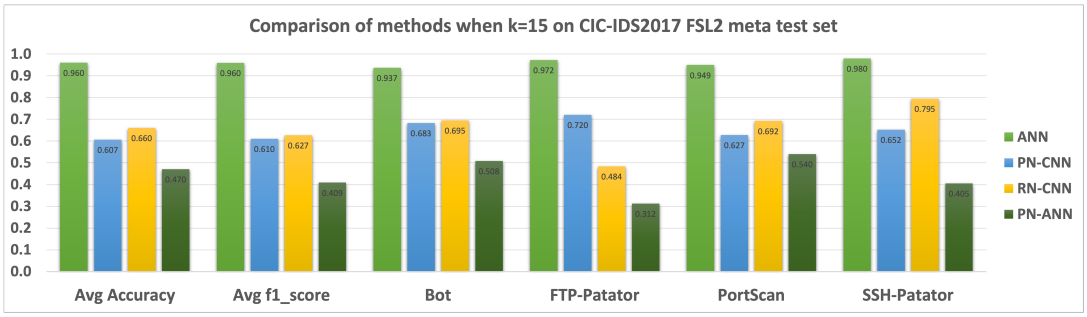


Figure 4.16: CIC-IDS2017 FSL2 Meta test results when $k = 15$

PN-CNN scores higher than three FSL methods using the CICIDS2017 FSL2 meta test dataset, when k equals 20, as shown in *Figure 4.17*.

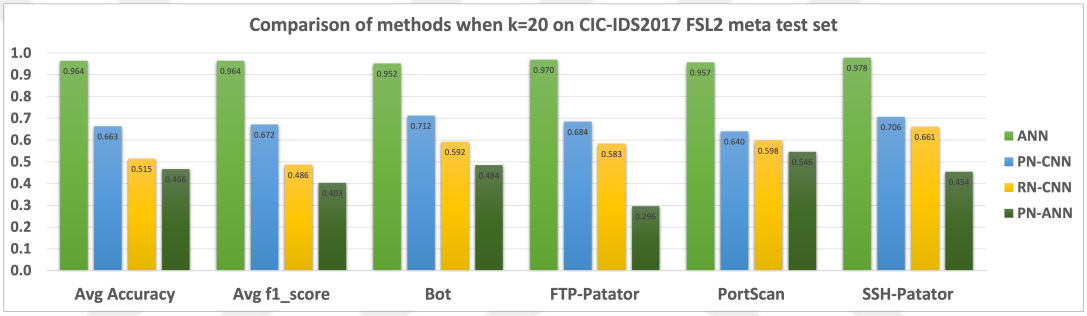


Figure 4.17: CIC-IDS2017 FSL2 Meta test results when $k = 20$

Figure 4.18 shows performances of the methods using CICIDS2017 FLS2 meta test dataset, when k is equal to 25. As shown in the figure, *PN-CNN* performs better other two FSL methods on all metrics except the PortScan class. For this class, *RN-CNN* gives a slightly better f1 score than other FSL methods.

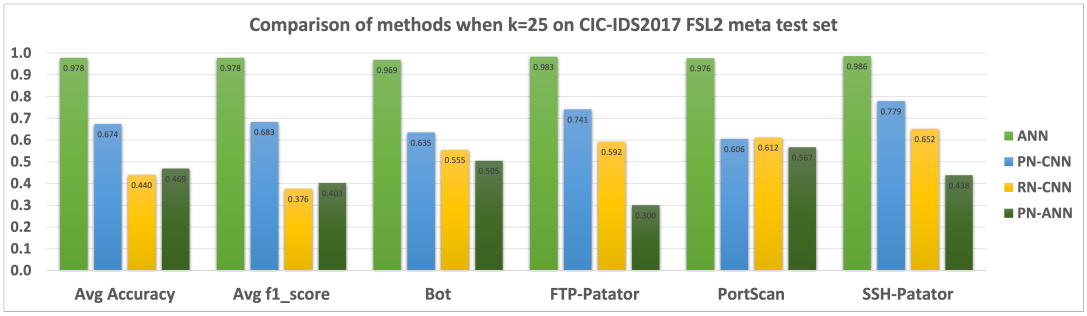


Figure 4.18: CIC-IDS2017 FSL2 Meta test results when $k = 25$

Figure 4.19 compares performances of the methods using CICIDS2017 FLS2 meta test dataset, when k is equal to 50. As shown in the figure, *PN-CNN* performs better other two FSL methods on all metrics except SSH-Patator class. For this class, *RN-CNN* achieves a better f1 score than other FSL techniques.

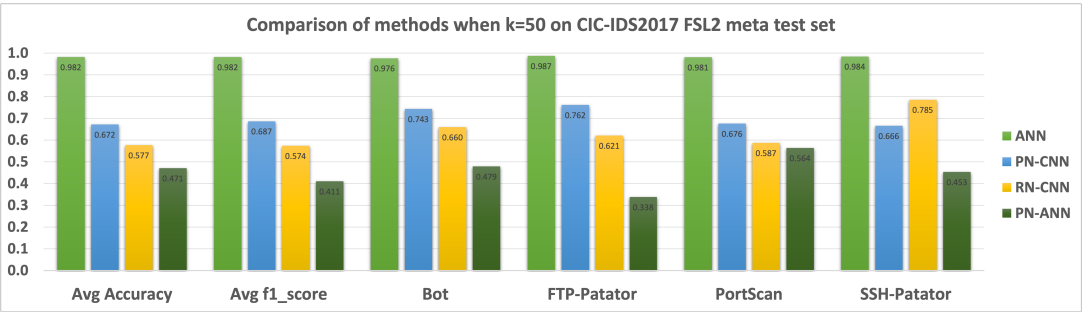


Figure 4.19: CIC-IDS2017 FSL2 Meta test results when $k = 50$

Performances of all four methods using CICIDS2017 meta validation set are given tables from *Table 4.17* to *Table 4.20*. Unseen classes during the meta training phase of the FSL methods are marked as yellow at these tables. Only the ANN method used those classes during the training phase, since it cannot learn classes without having them in the training dataset.

Table 4.17 shows the performance of the ANN method using the CICIDS2017 FSL2 meta validation dataset. As k increases, the performance of ANN increases, in general. Web Attack Brute Force class cannot be recognized by ANN well even if k is equal 50.

Table 4.17: *ANN* method Performances on CIC-IDS2017 FSL2 meta validation set

k	accuracy	f1_score	BENIGN	Bot	DDoS	DoS Golden-Eye	DoS Hulk	DoS Slowhttp test	DoS slowloris	FTP-Patator	PortScan	SSH-Patator	Web Attack Brute Force	Web Attack XSS
5	0.703	0.684	0.509	0.261	0.992	0.983	0.966	0.939	0.867	0.666	0.515	0.557	0.272	0.686
10	0.726	0.717	0.542	0.408	0.992	0.982	0.970	0.934	0.872	0.704	0.628	0.646	0.240	0.689
15	0.756	0.753	0.588	0.626	0.991	0.983	0.975	0.931	0.896	0.785	0.665	0.668	0.253	0.671
20	0.760	0.755	0.601	0.656	0.990	0.984	0.968	0.926	0.897	0.763	0.722	0.664	0.234	0.656
25	0.752	0.748	0.575	0.603	0.980	0.981	0.957	0.920	0.900	0.775	0.732	0.668	0.218	0.661
50	0.815	0.812	0.696	0.831	0.990	0.982	0.974	0.930	0.932	0.876	0.854	0.727	0.278	0.680

Results of *PN-CNN* captured from the experiment made for CICIDS2017 FSL2 meta

validation dataset is shown in *Table 4.18*. Overall f1 score is increased proportionally to increase at k. When k is small, *PN-CNN* suffers from classifying the Benign traffic. However, an increase at k results in better performance in the Benign traffic classification.

Table 4.18: *PN-CNN* method Performances on CIC-IDS2017 FSL2 meta validation set

k	accuracy	f1_score	BENIGN	Bot	DDoS	DoS Golden-Eye	DoS Hulk	DoS Slowhttp test	DoS slowloris	FTP-Patator	PortScan	SSH-Patator	Web Attack Brute Force	Web Attack XSS
5	0.647	0.639	0.337	0.636	0.949	0.980	0.892	0.945	0.893	0.693	0.717	0.635	0.639	0.693
10	0.715	0.717	0.387	0.734	0.959	0.983	0.908	0.963	0.897	0.687	0.700	0.652	0.629	0.708
15	0.718	0.724	0.411	0.731	0.970	0.983	0.924	0.970	0.907	0.729	0.736	0.659	0.630	0.708
20	0.717	0.728	0.432	0.730	0.964	0.991	0.915	0.973	0.901	0.751	0.648	0.659	0.622	0.703
25	0.716	0.728	0.405	0.743	0.967	0.990	0.899	0.972	0.904	0.722	0.686	0.654	0.622	0.720
50	0.740	0.750	0.568	0.765	0.969	0.994	0.927	0.976	0.926	0.737	0.625	0.672	0.620	0.724

Table 4.19 shows the performance of *PN-ANN* approach on the CICIDS2017 FSL2 meta validation dataset. As shown in the table, the increase at k is not directly correlated with the performance of the overall method. Like the *PN-CNN* method, the *PN-ANN* method also suffers from classifying the Benign classes for small values k. Unlike in the *PN-CNN* case, the performance of the Benign class does not improve as k increases. Furthermore, PortScan performance of the method, which is one of the unseen classes, decreases as k increases.

Table 4.19: *PN-ANN* method Performances on CIC-IDS2017 FSL2 meta validation set

k	accuracy	f1_score	BENIGN	Bot	DDoS	DoS Golden-Eye	DoS Hulk	DoS Slowhttp test	DoS slowloris	FTP-Patator	PortScan	SSH-Patator	Web Attack Brute Force	Web Attack XSS
5	0.543	0.548	0.339	0.594	0.764	0.957	0.768	0.873	0.788	0.574	0.546	0.532	0.551	0.661
10	0.552	0.515	0.110	0.737	0.770	0.954	0.782	0.872	0.568	0.450	0.440	0.473	0.593	0.663
15	0.568	0.530	0.151	0.524	0.792	0.957	0.792	0.869	0.800	0.497	0.440	0.432	0.562	0.660
20	0.568	0.524	0.111	0.541	0.800	0.955	0.794	0.886	0.814	0.564	0.431	0.608	0.593	0.662
25	0.564	0.521	0.117	0.557	0.787	0.967	0.806	0.876	0.806	0.507	0.429	0.461	0.591	0.664
50	0.525	0.494	0.077	0.429	0.775	0.969	0.731	0.888	0.571	0.593	0.434	0.639	0.558	0.670

CICIDS2017 FSL2 meta validation performances of the *RN-CNN* method are shown

in the *Table 4.20*. At the overall performance in both f1 score and total accuracy, *RN-CNN* gives better results for the minimum value of k in this dataset.

Table 4.20: *RN-CNN* method Performances on CIC-IDS2017 FSL2 meta validation set

k	accuracy	f1_score	BENIGN	Bot	DDoS	DoS Golden-Eye	DoS Hulk	DoS Slowhttp test	DoS slowloris	FTP-Patator	PortScan	SSH-Patator	Web Attack Brute Force	Web Attack XSS
5	0.643	0.618	0.553	0.631	0.850	0.980	0.835	0.925	0.845	0.301	0.640	0.588	0.561	0.667
10	0.613	0.565	0.483	0.497	0.954	0.987	0.885	0.938	0.885	0.286	0.478	0.483	0.635	0.706
15	0.617	0.581	0.425	0.733	0.965	0.991	0.966	0.958	0.891	0.556	0.583	0.640	0.577	0.679
20	0.564	0.505	0.677	0.540	0.916	0.991	0.845	0.947	0.860	0.651	0.598	0.533	0.417	0.682
25	0.521	0.498	0.403	0.512	0.824	0.990	0.822	0.939	0.854	0.481	0.577	0.622	0.598	0.649
50	0.635	0.593	0.600	0.592	0.988	0.994	0.963	0.957	0.942	0.630	0.609	0.595	0.640	0.670

From figures 4.20 to 4.25, comparisons of the performance of all four methods are represented for different values of k. The red marked classes on these figures represent classes which did not seen during the meta training phase of the FSL methods. On the other hand, k samples per the red marked classes are used for ANN's training dataset.

Figure 4.20 shows the performance of the four approaches in the CICIDS2017 FSL2 meta validation dataset when the variable k is 5. In terms of overall f1 score and accuracy, ANN performs the best. ANN is beaten by FSL methods on the following classes' f1 scores: Benign, Bot, DoS Slowloris, FTP-Patator, PortScan, SSH-Patator, Web Attack Brute Force, and Web Attack XSS. In other words, for twelve classes, FSL methods perform better than ANN for eight classes. Those eight classes contain all four unseen classes. Additionally, for all unseen classes, *PN-CNN* gives the best score.

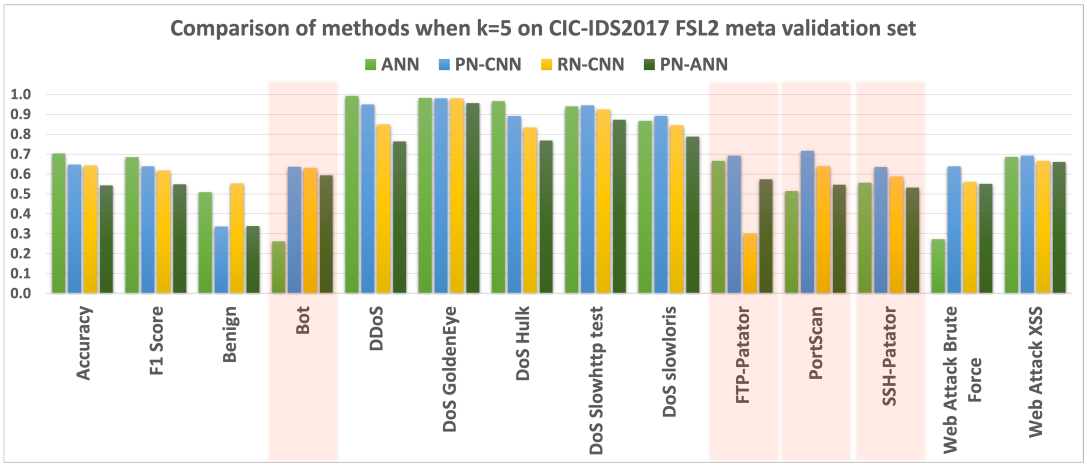


Figure 4.20: CIC-IDS2017 FSL2 Meta validation results when $k = 5$

CICIDS2017 FSL2 meta validation dataset performances when the variable k is equal to 10 is shown in *Figure 4.21*. *PN-CNN* method reaches the highest score in terms of both accuracy and f1 score. Additionally, it performs better than the other three methods for all four unseen classes. ANN method scores best in three classes. *PN-CNN* achieves the highest score in seven classes. In the remaining two classes, the highest score was achieved by the *RN-CNN* method.

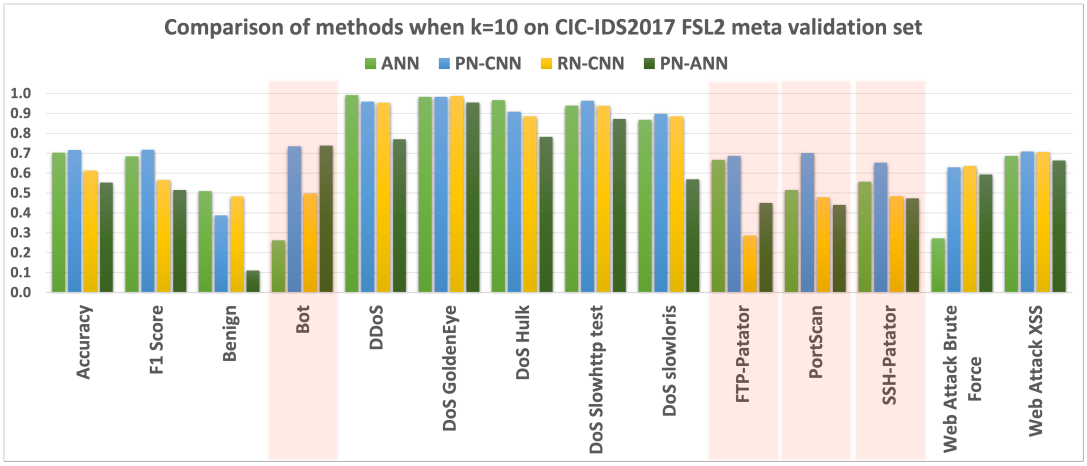


Figure 4.21: CIC-IDS2017 FSL2 Meta validation results when $k = 10$

For k equals 15, performance of investigated methods on the CICIDS2017 FSL2 meta validation dataset is given at *Figure 4.16*. In terms of the f1 score and accuracy, ANN method performs better than others. In addition, in two of the unseen classes, ANN

scores the best. For the Bot class, which is also one of the unseen classes, *RN-CNN* performs best. The last remaining unseen class is best classified by *PN-CNN*.

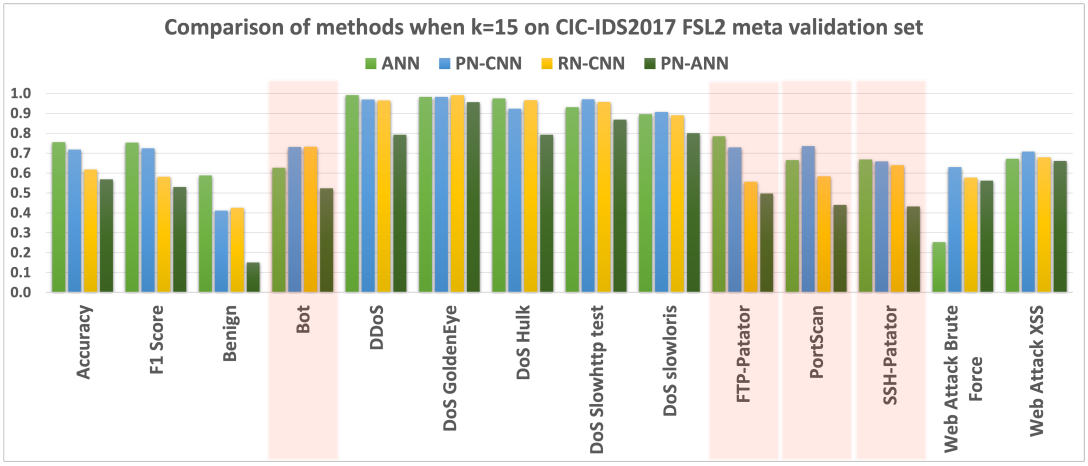


Figure 4.22: CIC-IDS2017 FSL2 Meta validation results when $k = 15$

The experimentation performed for variable k equals 20 using the CICIDS2017 FSL2 meta validation dataset is shown in *Figure 4.23*. As expected, as the value k increases, the performance of ANN increases. It performs best in six classes which also contain three of the unseen classes. The last unseen class, which is the Bot class, was classified best by *PN-CNN*.

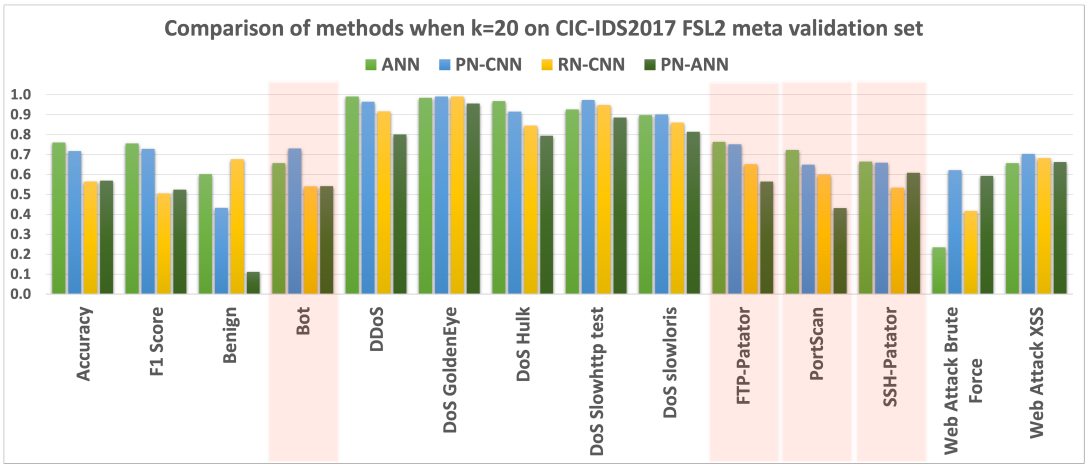


Figure 4.23: CIC-IDS2017 FSL2 Meta validation results when $k = 20$

Figure 4.24 shows the performance of methods using the CICIDS2017 FSL2 meta validation dataset where the variable k is equal to 25. ANN achieves the best accuracy

and the best f1 score in all four approaches. Like the case where the variable k is 20, ANN method reaches the highest f1 score on three of the unseen classes. *PN-CNN* scores best in the Bot class, which is the last unseen class. FSL methods score better than ANN method in the Web Attack Brute Force class.

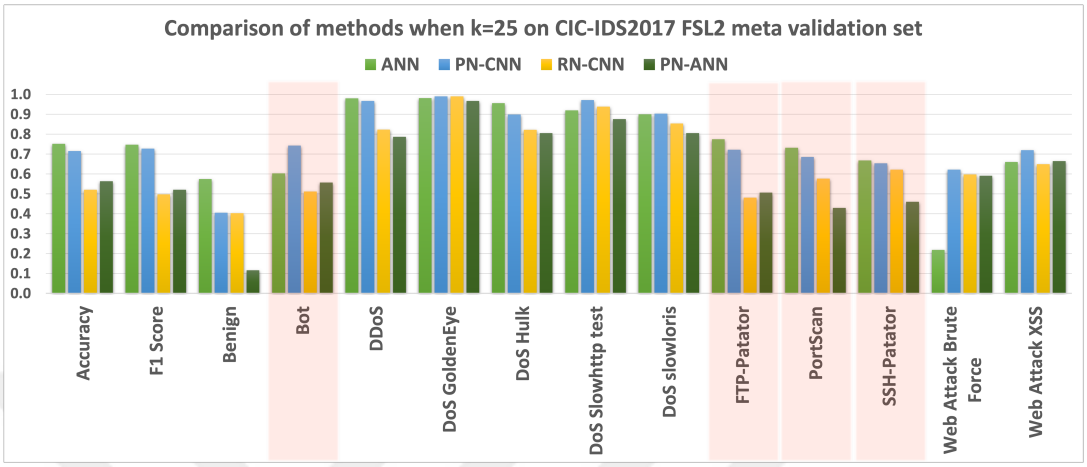


Figure 4.24: CIC-IDS2017 FSL2 Meta validation results when $k = 25$

The last figure in this section is *Figure 4.25* which shows the performance of methods using the CICIDS FSL2 meta validation dataset where k equals 50. As expected, the increase at k results in a performance improvement in ANN method. It performed best in eight classes. ANN approach performs the best f1 score for all unseen classes. Only the Web Attack Brute Force class cannot be classified well using ANN method.

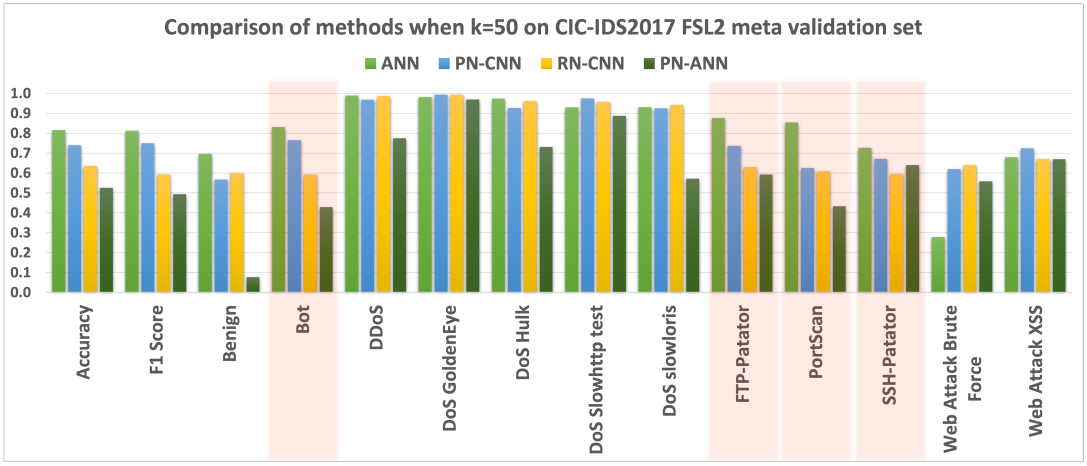


Figure 4.25: CIC-IDS2017 FSL2 Meta validation results when $k = 50$

4.3.3 Experiments on UNSW-NB15 FSL Dataset

Experimentation results for the UNSW-NB15 FSL dataset are provided in this section. Tables from 4.21 to 4.24 contains performances of four methods using meta test dataset. The meta test dataset's classes were not used in the meta training phase of FSL methods. On the other hand, ANN method used these classes during the training time. All four methods cannot reach good performance on this dataset. The maximum overall accuracy or f1 score is around 0.5.

UNSW-NB15 meta test dataset performance of the ANN method is shown in *Table 4.21*. It can be easily realized that the overall performance is not as well as previously tested datasets. Using a much larger portion of the original dataset did not change the method's performance too much. Therefore, we do not expect higher accuracy and f1 score from the four methods using this dataset. Like previously used dataset experiments, ANN performance increased with the increase in variable k.

Table 4.21: *ANN* method Performances on UNSW-NB15 FSL meta test set

k	Avg Accuracy	Avg f1_score	Analysis	Backdoor	Shellcode
5	0.404	0.381	0.323	0.391	0.428
10	0.408	0.399	0.330	0.417	0.452
15	0.441	0.425	0.319	0.428	0.529
20	0.452	0.436	0.383	0.361	0.563
25	0.462	0.442	0.325	0.419	0.582
50	0.504	0.491	0.423	0.387	0.664

Table 4.22 shows the performance of the *PN-CNN* approach using the UNSW-NB15 FSL meta test dataset. The performance of this approach also improved as the value of k increased.

Table 4.22: *PN-CNN* method Performances on UNSW-NB15 FSL meta test set

k	Avg Accuracy	Avg f1_score	Analysis	Backdoor	Shellcode
5	0.414	0.392	0.344	0.383	0.477
10	0.440	0.417	0.330	0.413	0.530
15	0.443	0.420	0.324	0.435	0.547
20	0.449	0.426	0.330	0.409	0.554
25	0.457	0.431	0.323	0.428	0.577
50	0.469	0.437	0.339	0.446	0.593

The performance of the *PN-ANN* method on the UNSW-NB15 FSL meta test dataset is shown in *Table 4.23*. Its accuracy also increases with the increase in k. However, the overall f1 score reached its maximum value when k is equal to 20, not 50.

Table 4.23: *PN-ANN* method Performances on UNSW-NB15 FSL meta test set

k	Avg Accuracy	Avg f1_score	Analysis	Backdoor	Shellcode
5	0.422	0.375	0.280	0.390	0.468
10	0.438	0.376	0.262	0.407	0.515
15	0.444	0.381	0.241	0.418	0.534
20	0.448	0.389	0.238	0.426	0.542
25	0.449	0.387	0.174	0.443	0.548
50	0.453	0.384	0.181	0.478	0.550

The last table, which is *Table 4.23*, shows the performance of *RN-CNN* on the UNSW-NB15 FSL meta test dataset. The performance of this method seems not to be related

to the value of k. But it reaches its best performance when k equals 50.

Table 4.24: *RN-CNN* method Performances on UNSW-NB15 FSL meta test set

k	Avg Accuracy	Avg f1_score	Analysis	Backdoor	Shellcode
5	0.410	0.383	0.515	0.366	0.481
10	0.395	0.339	0.500	0.406	0.472
15	0.391	0.353	0.501	0.407	0.506
20	0.464	0.429	0.519	0.424	0.601
25	0.406	0.382	0.500	0.373	0.532
50	0.481	0.463	0.500	0.463	0.634

Figures from 4.26 to 4.31 show the comparisons of method performances on the UNSW-NB15 FSL meta test dataset. Different from FSL methods, ANN uses k sample per class from the meta test dataset during training time. FSL methods use only the meta test dataset, which does not contain samples from the meta test dataset's classes.

The comparison of UNSW-NB15 FSL meta test performances of all four approaches when k equals 5 is shown in *Figure 4.26*. *PN-ANN* reached the best accuracy, but, it has the worst f1 score overall. On the other hand, *PN-CNN* achieves the best f1 score and the second-best accuracy. For the Analysis class, *RN-CNN* scores the best, with huge differences from its opponents.

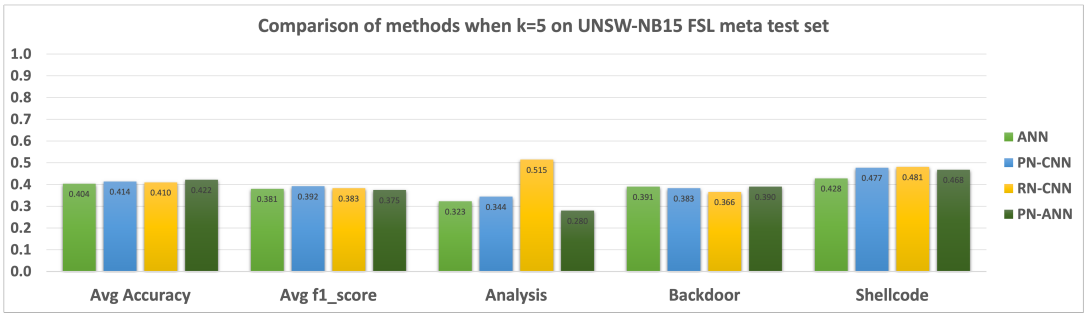


Figure 4.26: UNSW-NB15 FSL Meta test results when $k = 5$

When we increase the variable k to 10, on both accuracy and f1 score *PN-CNN* performs the highest score, as shown in the *Figure 4.27*. Like the case where k equals 5, *RN-CNN* achieves the best f1 score for the Analysis class.

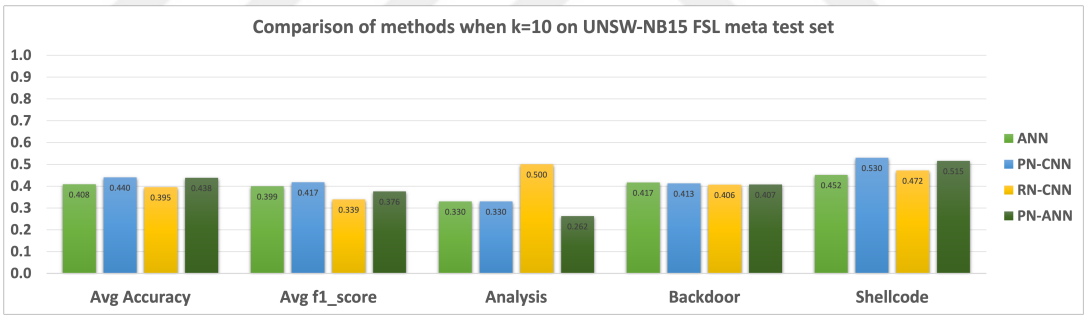


Figure 4.27: UNSW-NB15 FSL Meta test results when $k = 10$

Comparison of the methods for k equals 15 using UNSW-NB15 FSL meta test dataset is shown in the *Figure 4.28*. The best accuracy is achieved by *PN-ANN*. However, the highest f1 score is reached by the ANN method. One of the important points in this comparison is that *PN-CNN* places a second position on both accuracy and the f1 score.

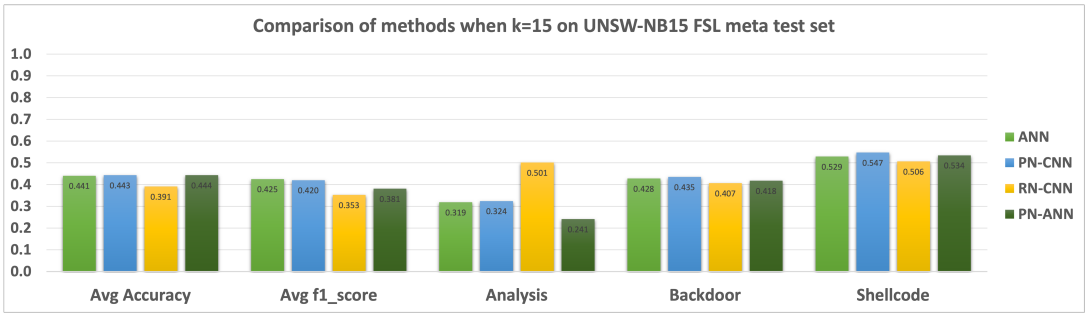


Figure 4.28: UNSW-NB15 FSL Meta test results when $k = 15$

Figure 4.30 shows the results for k is 20 using UNSW-NB15 FSL meta test dataset. In terms of accuracy, *RN-CNN* is placed the first and ANN is placed the second. When we look at the f1 score, the order of these two methods swaps.

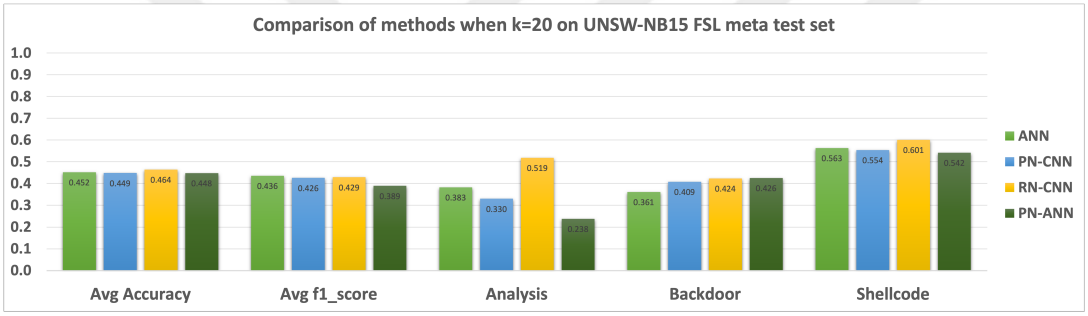


Figure 4.29: UNSW-NB15 FSL Meta test results when $k = 20$

When we increase k to 25, ANN approach performs better than the FSL methods in terms of both the f1 score and the accuracy. *PN-ANN* is the second best in terms of accuracy and *PN-CNN* is the third. For the f1 score, *PN-CNN* is the second best.

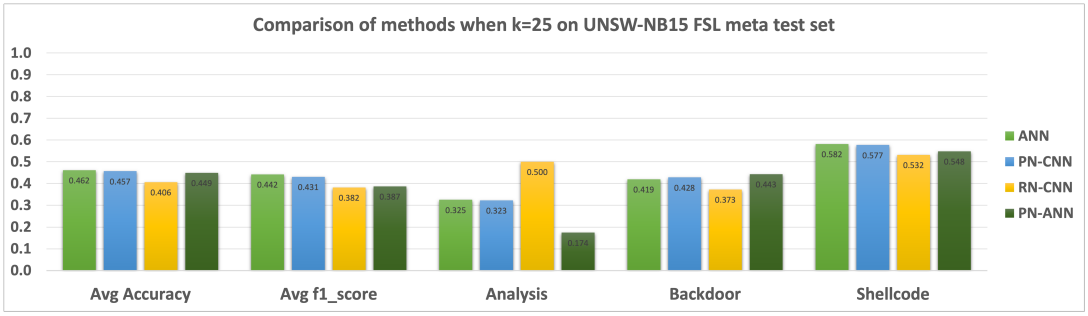


Figure 4.30: UNSW-NB15 FSL Meta test results when $k = 25$

Figure 4.31 shows the performance of methods using the UNSW-NB15 FSL meta test dataset, for the variable k is 50. ANN approach is the best on both the f1 score and accuracy. The second best method is *RN-CNN*. It achieves the best f1 score in both Analysis and Backdoor classes, but its performance in the Shellcode class places it second in overall performance.

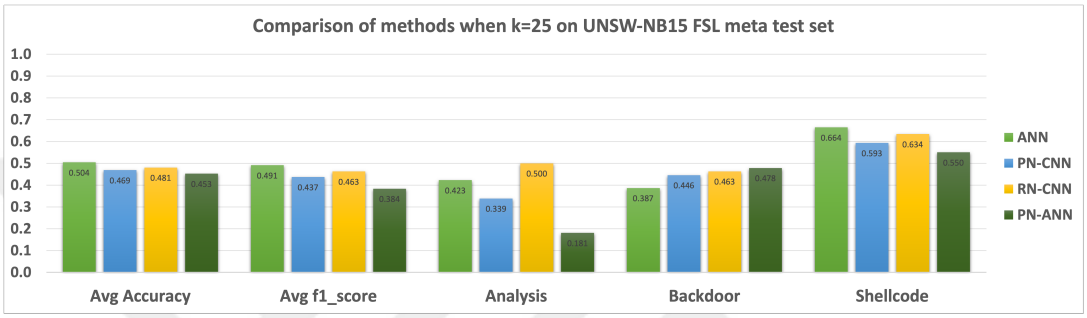


Figure 4.31: UNSW-NB15 FSL Meta test results when $k = 50$

Tables from 4.25 to 4.28 show the individual performance of the methods using the UNSW-NB15 FSL meta validation dataset for different values of k. The yellow marked columns represent the classes that are not used during the meta training phase of the FSL methods. Remark that those classes are used in the training phase of the ANN method, since, this method requires all classes during training time.

Table 4.25 shows performance of ANN method using UNSW-NB15 FSL meta validation dataset. As the table illustrates, the ANN method cannot learn yellow-marked classes. These classes cannot be learned due to unbalance of the classes. The yellow columns have k samples per class; on the other hand, the other classes have 470 samples per class. This is also valid for the FSL methods except that those classes are not used in the meta training phase, only used as the support set during the meta test phase. ANN tests are done at least 10 times and we get an average of them.

Table 4.25: *ANN* method Performances on UNSW-NB15 FSL meta validation set

k	Avg Accuracy	Avg F1 Score	Analysis	Backdoor	DoS	Exploits	Fuzzers	Generic	Normal	Reconnaissance	Shellcode
5	0.318	0.250	0.000	0.000	0.313	0.128	0.294	0.852	0.553	0.108	0.000
10	0.319	0.248	0.000	0.000	0.313	0.181	0.280	0.853	0.556	0.052	0.000
15	0.319	0.251	0.000	0.000	0.312	0.187	0.298	0.861	0.572	0.027	0.000
20	0.330	0.259	0.000	0.000	0.322	0.147	0.297	0.861	0.600	0.104	0.000
25	0.327	0.254	0.000	0.000	0.316	0.182	0.292	0.861	0.585	0.053	0.000
50	0.357	0.297	0.000	0.000	0.337	0.174	0.289	0.868	0.611	0.389	0.000

Table 4.26 shows performance of *PN-CNN* method using UNSW-NB15 FSL meta validation dataset. It gives its highest accuracy for k equals 50 and its best f1 score occurs for k is 10.

Table 4.26: *PN-CNN* method Performances on UNSW-NB15 FSL meta validation set

k	Avg Accuracy	Avg f1_score	Analysis	Backdoor	DoS	Exploits	Fuzzers	Generic	Normal	Reconnaissance	Shellcode
5	0.256	0.236	0.307	0.248	0.281	0.270	0.287	0.584	0.532	0.211	0.233
10	0.318	0.296	0.323	0.263	0.296	0.272	0.268	0.776	0.586	0.237	0.228
15	0.284	0.261	0.304	0.268	0.142	0.281	0.303	0.687	0.547	0.179	0.218
20	0.286	0.258	0.311	0.298	0.289	0.250	0.298	0.660	0.628	0.179	0.199
25	0.291	0.258	0.309	0.258	0.274	0.286	0.296	0.619	0.573	0.203	0.210
50	0.306	0.288	0.344	0.269	0.264	0.281	0.321	0.708	0.598	0.186	0.203

Table 4.27 shows the performance of the *PN-ANN* method using the UNSW-NB15 FSL meta validation dataset. It gives its highest accuracy for k equal to 15 and its best f1 score occurs for k is 15.

Table 4.27: *PN-ANN* method Performances on UNSW-NB15 FSL meta validation set

k	Avg Accuracy	Avg f1_score	Analysis	Backdoor	DoS	Exploits	Fuzzers	Generic	Normal	Reconnaissance	Shellcode
5	0.205	0.178	0.251	0.085	0.253	0.275	0.238	0.516	0.265	0.131	0.150
10	0.207	0.131	0.082	0.241	0.141	0.219	0.226	0.397	0.281	0.000	0.081
15	0.220	0.152	0.186	0.275	0.096	0.164	0.273	0.389	0.269	0.152	0.033
20	0.208	0.121	0.170	0.230	0.111	0.266	0.263	0.384	0.279	0.000	0.134
25	0.207	0.121	0.179	0.165	0.166	0.257	0.266	0.379	0.273	0.067	0.033
50	0.217	0.129	0.162	0.222	0.020	0.263	0.270	0.395	0.297	0.085	0.023

RN-CNN performance using the UNSW-NB15 FSL meta validation dataset is shown in the *Table 4.28*. *RN-CNN* gives its highest accuracy and its best f1 score for k equals 50.

Table 4.28: *RN-CNN* method Performances on UNSW-NB15 FSL meta validation set

k	Avg Accuracy	Avg f1_score	Analysis	Backdoor	DoS	Exploits	Fuzzers	Generic	Normal	Reconnaissance	Shellcode
5	0.331	0.283	0.242	0.308	0.235	0.192	0.244	0.934	0.611	0.384	0.201
10	0.260	0.204	0.255	0.176	0.227	0.187	0.153	0.809	0.462	0.197	0.244
15	0.297	0.248	0.291	0.227	0.160	0.249	0.280	0.835	0.655	0.219	0.192
20	0.324	0.291	0.368	0.210	0.177	0.287	0.317	0.964	0.454	0.510	0.185
25	0.367	0.337	0.287	0.179	0.231	0.339	0.282	0.953	0.533	0.479	0.174
50	0.415	0.396	0.300	0.236	0.358	0.359	0.353	0.981	0.624	0.506	0.271

Figures from 4.32 to 4.37 represents comparison of the methods using UNSW-NB15 meta validation dataset. The red marked columns show the classes which are not seen at the meta training phase of the FSL methods.

Figure 4.32 shows comparison of performances of the methods using UNSW-NB15 FSL meta validation dataset, for the variable k is equal to 5. As the figure illustrates, *RN-CNN* achieves the best f1 score and accuracy. For Analysis and Shellcode classes, which are two of the unseen classes, the best performance is achieved by *PN-CNN* method. For the Backdoor class, which is the remaining unseen class, *RN-CNN* scores the best performance.

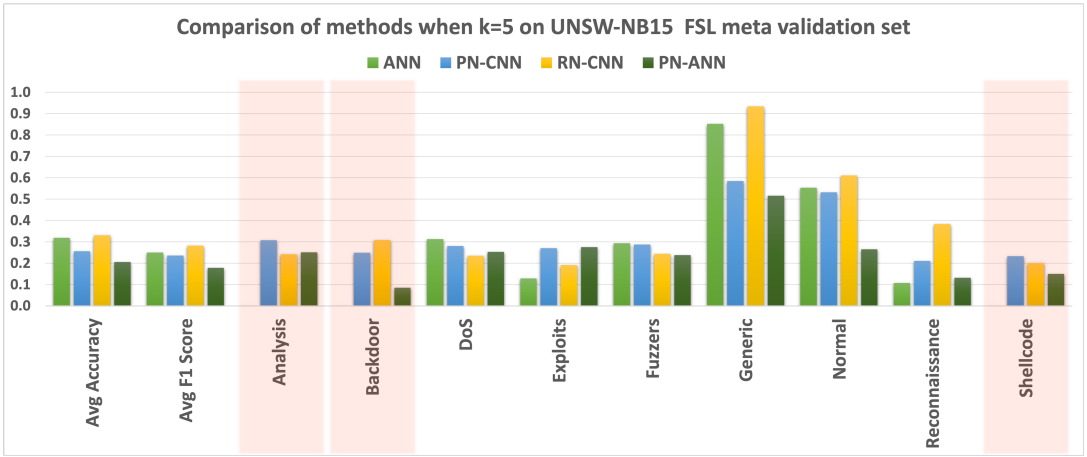


Figure 4.32: UNSW-NB15 FSL Meta validation results when $k = 5$

Figure 4.33 shows the performance of all the methods for the variable k is equal to 10. The two methods, ANN and *PN-CNN* give almost the same result in accuracy score. However, *PN-CNN* achieves the highest score on the f1 score. Furthermore, *PN-CNN* has the highest score in Analysis and Backdoor classes, which are unseen in the meta training phase. For the remaining unseen class, *RN-CNN* scores the highest.

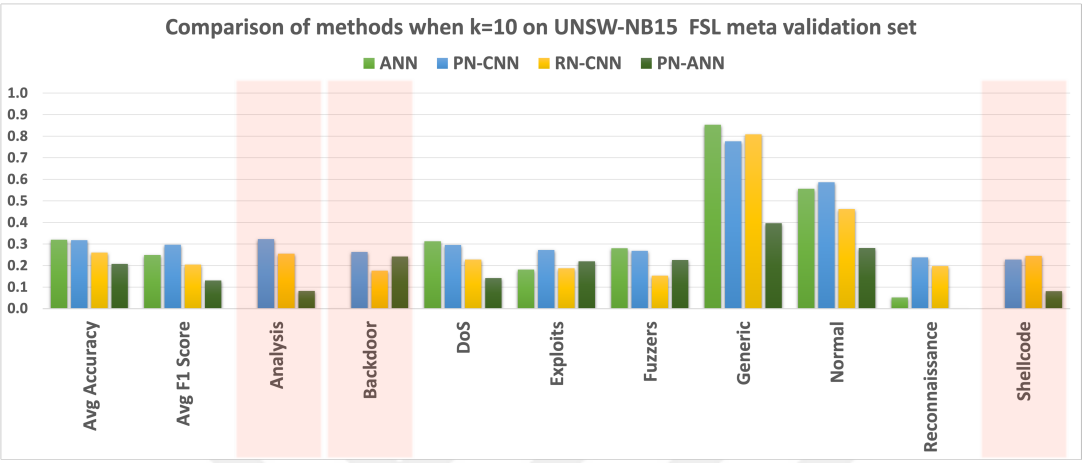


Figure 4.33: UNSW-NB15 FSL Meta validation results when $k = 10$

Performance comparison of the methods on UNSW-NB15 FSL meta validation dataset for k is 15, is shown in the *Figure 4.34*. The accuracy of ANN beats FSL methods for this experimentation setup. But, the f1 score of both *PN-ANN* and *RN-CNN* is higher than ANN's f1 score. *PN-ANN* reaches the best f1 score for Backdoor class. The remaining unseen classes' best f1 score was reached by *PN-CNN* method.

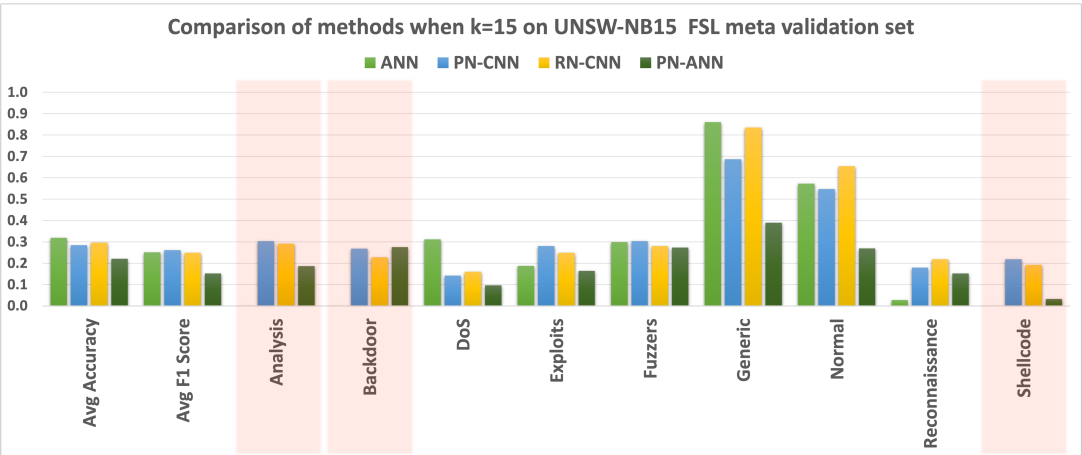


Figure 4.34: UNSW-NB15 FSL Meta validation results when $k = 15$

Figure 4.35 shows performance of the investigated approaches using UNSW-NB15 FSL meta validation dataset for k is 20. ANN gets the highest accuracy, like the case where k is 15. Also, like the case k is 15, ANN's f1 score is beaten by both *PN-CNN* and *RN-CNN* methods. For two unseen classes, *PN-CNN* performs the best f1 score. *RN-CNN* scores best for the remaining unseen classes.

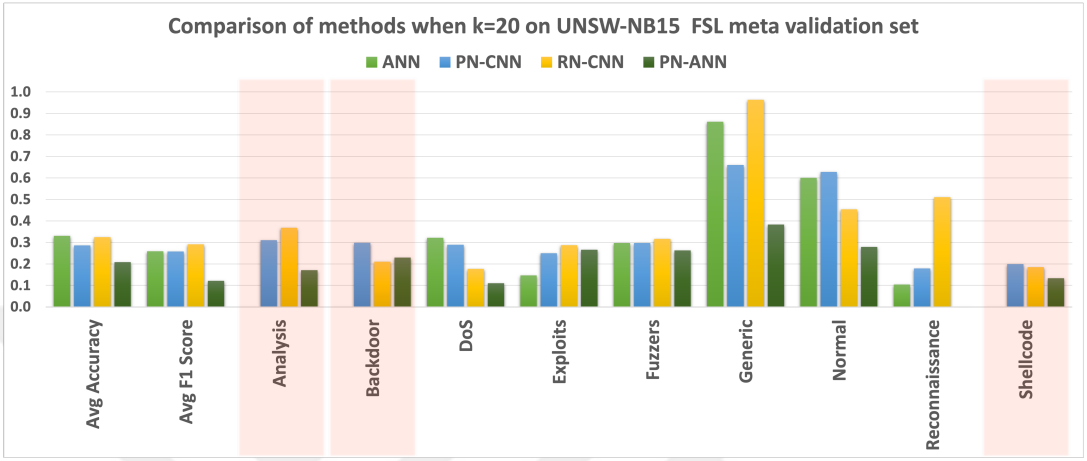


Figure 4.35: UNSW-NB15 FSL Meta validation results when $k = 20$

UNSW-NB15 meta validation performances of the all methods is shown in *Figure 4.36*, where k equals 25. *RN-CNN* performs best in terms of both accuracy and f1 score. On the other hand, *PN-CNN* achieves the best f1 score on all three unseen classes.

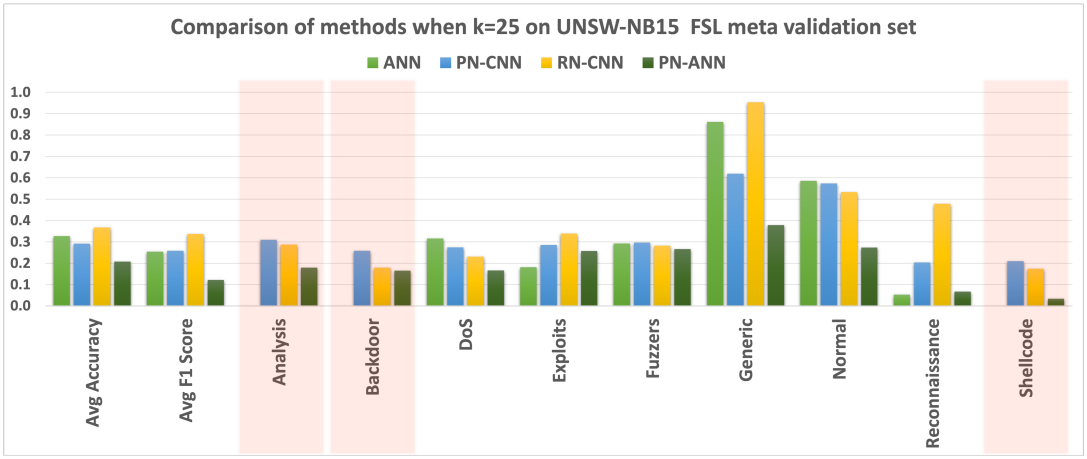


Figure 4.36: UNSW-NB15 FSL Meta validation results when $k = 25$

Figure 4.37 shows the comparison of methods using the UNSW-NB15 meta validation dataset, where the variable k is 50. Like in the case where k is 25, *RN-CNN* reaches the best accuracy and the best f1 score. Also, the Backdoor performance of *RN-CNN* is the best. However, *PN-CNN* scores the highest f1 score for the remaining two unseen classes.

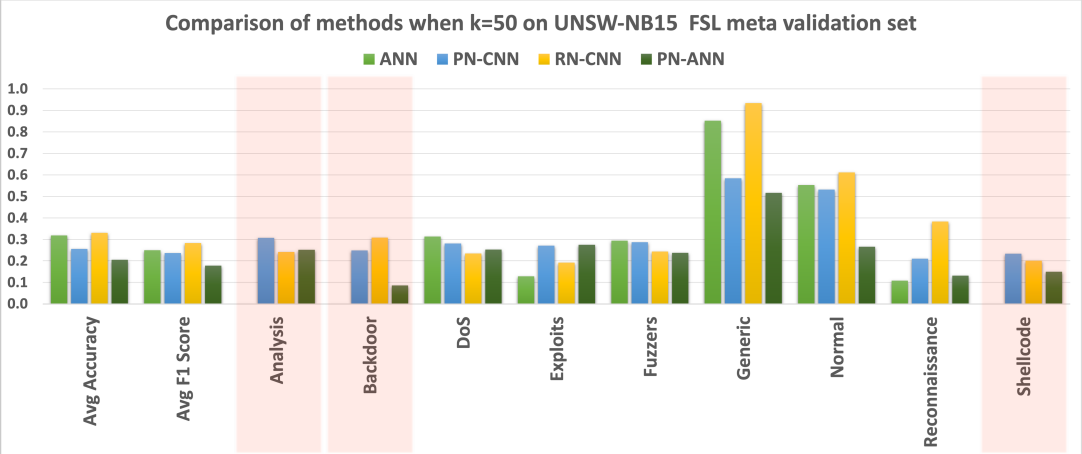


Figure 4.37: UNSW-NB15 FSL Meta validation results when $k = 50$

CHAPTER 5

DISCUSSION AND CONCLUSION

With the increase in smart devices and the use of the Internet, new types of network attacks have emerged. Existing NIDSs based on deep learning techniques suffer from adapting to these new attack types, due to lack of data and the difficulties of generating a new dataset. This brings to mind that the problem of classifying network attacks can be solved with few-shot learning approaches.

In this study, we investigated three different few-shot learning methods and compared them with both ANN and each other. To make this evaluation, we generate three few-shot network attack classification datasets using two modern network intrusion datasets. The novelty of this study is the classification of network attacks using few data without using test class samples in the training phase, contrary to previous research to our knowledge. As a common approach, our methods try to match an input vector to a vector in embedding space, then predict its class by comparing it with the embeddings of the support set. Additionally, in order to work on this problem, we also generate three different few-shot network attack classification datasets by using previously developed two modern intrusion datasets. These generated datasets are another contribution that this study made.

This study shows that FSL methods achieved better performance for small datasets. As the number of samples in the dataset grows, the ANN method's performance increases more than the FSL methods' performance. Furthermore, we found that for low values of k, good results are obtained in unseen classes using the methods *PN-CNN* and *RN-CNN*. On the other hand, even if there are some cases that *PN-ANN* works better, its performance is not good as the other FSL methods.

When we compare the three FSL methods, *PN-CNN* and *RN-CNN* methods performed well on overall. But *PN-ANN* did not perform as we expected. Since the dataset that we used is small, this method cannot achieve good performance. Maybe, we should eliminate some of the ANN blocks used on its encoder, in order to improve its performance. Whereas, the other two methods perform good results, especially for the classes which are not used during the training phase. Also, these methods performed good classification results overall. This proves that the meta learning paradigm can be used for the few-shot learning problems.

We realize that support set quality for the investigated problem is an essential point to get better performance. As future work, we propose to choose representative support set samples to get better classification results using active learning methods. Additionally, the encoders for both prototypical network based and relation network based methods can be replaced with different neural network architectures, such as more simple ANN architecture, recurrent neural network architecture, etc. Also, different neural network architectures can be tested for the relation network on the RN-CNN method.

REFERENCES

[1] L. S. Vailshery, "IoT and non-IoT connections worldwide 2010-2025," *Statista, March*, vol. 8, 2021.

[2] M. M. U. Chowdhury, F. Hammond, G. Konowicz, C. Xin, H. Wu, and J. Li, "A few-shot deep learning approach for improved intrusion detection," *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference, UEMCON 2017*, vol. 2018-Janua, pp. 1–7, 2018.

[3] R. Shirey, "Internet security glossary, version 2," tech. rep., 2007.

[4] W. Stallings and L. Brown, *Computer security : principles and practice*. Pearson Education, Inc, 2015.

[5] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[6] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[7] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," in *Competition and cooperation in neural nets*, pp. 267–285, Springer, 1982.

[8] T. M. Mitchell and T. M. Mitchell, *Machine learning*, vol. 1. McGraw-hill New York, 1997.

[9] M. Mohri, A. Rostamizadeh, and A. Talwalkar, "Foundations of machine learning.[sl]," 2012.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[11] E. G. Dada, J. S. Bassi, H. Chiroma, A. O. Adetunmbi, O. E. Ajibuwa, *et al.*, "Machine learning for email spam filtering: review, approaches and open research problems," *Heliyon*, vol. 5, no. 6, p. e01802, 2019.

[12] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, "Generalizing from a few examples: A survey on few-shot learning," *ACM computing surveys (csur)*, vol. 53, no. 3, pp. 1–34, 2020.

[13] Y. Liu, H. Zhang, W. Zhang, G. Lu, Q. Tian, and N. Ling, "Few-shot image classification: Current status and research trends," *Electronics*, vol. 11, no. 11, p. 1752, 2022.

[14] M. Köhler, M. Eisenbach, and H.-M. Gross, "Few-shot object detection: A survey," *arXiv preprint arXiv:2112.11699*, 2021.

[15] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.

[16] P. Wu, H. Guo, and R. Buckland, "A transfer learning approach for network intrusion detection," in *2019 IEEE 4th international conference on big data analytics (ICBDA)*, pp. 281–285, IEEE, 2019.

[17] Z.-H. Zhou, "A brief introduction to weakly supervised learning," *National science review*, vol. 5, no. 1, pp. 44–53, 2018.

[18] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, pp. 1126–1135, PMLR, 06–11 Aug 2017.

[19] L. Weng, "Meta-learning: Learning to learn fast," *lilianweng.github.io*, 2018.

[20] G. Koch, R. Zemel, R. Salakhutdinov, *et al.*, "Siamese neural networks for one-shot image recognition," in *ICML deep learning workshop*, vol. 2, p. 0, Lille, 2015.

[21] C. Xu, J. Shen, and X. Du, "A method of few-shot network intrusion detection

based on meta-learning framework," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3540–3552, 2020.

[22] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, *et al.*, "Matching networks for one shot learning," *Advances in neural information processing systems*, vol. 29, 2016.

[23] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," *Advances in neural information processing systems*, vol. 30, 2017.

[24] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales, "Learning to compare: Relation network for few-shot learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1199–1208, 2018.

[25] S. Gamage and J. Samarabandu, "Deep learning methods in network intrusion detection: A survey and an objective comparison," *Journal of Network and Computer Applications*, vol. 169, p. 102767, 2020.

[26] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[27] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[28] J. Bridle, "Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters," *Advances in neural information processing systems*, vol. 2, 1989.

[29] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[30] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[31] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization.," *ICISSp*, vol. 1, pp. 108–116, 2018.

[32] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *2015 military communications and information systems conference (MilCIS)*, pp. 1–6, IEEE, 2015.