

# Deep Learning Based Radio Signal Classification and Detection of Modulation Schemes.

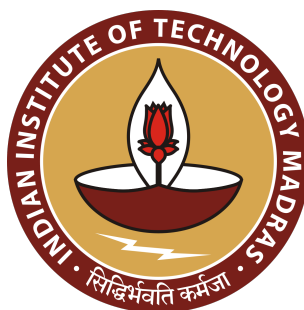
## PROJECT REPORT

Submitted by

**RASHMI PRAJAPATI**

*in partial fulfillment of the requirements  
for the award of the degree of*

**MASTER OF TECHNOLOGY**



Department Of Electrical Engineering

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

JUNE 2022

DEPARTMENT OF ELECTRICAL ENGINEERING  
INDIAN INSTITUTE OF TECHNOLOGY MADRAS

2022



**CERTIFICATE**

This is to certify that this thesis (or project report) entitled “*Deep Learning Based Radio Signal Classification and Detection of Modulation Scheme used At the Transmitter*” submitted by **RASHMI PRAJAPATI** to the Indian Institute of Technology Madras, for the award of the degree of **Masters of Technology** is a bona fide record of the research work done by him under my supervision. The contents of this thesis (or project report), in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma..

**Dr. T. G. Venkatesh**

*Research Guide*

*Associate Professor*

*Department of Electrical Engineering*

*IIT Madras 600036*

# Acknowledgment

First and foremost, I would like to express my deepest gratitude to my guide, **Dr. T G Venkatesh**, Associate Professor, Department of Electrical Engineering, IIT Madras, for providing me an opportunity to work under him. I would like to express my deepest appreciation for his patience, valuable feedback, suggestions and motivations.

I convey my sincere gratitude to **Rohan Desai**, MS Scholar, IIT Madras, for all his suggestions and support during the entire course of the project. Throughout the course of the project he offered immense help and provided valuable suggestions which helped me in completing this project.

I would like to extend my appreciation to all my friends and for their help and support in completing my project successfully.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Introduction to Automatic Modulation Classification . . . . .	3
1.2	Aim and Motivation . . . . .	4
1.3	Outline Of Report . . . . .	5
<b>2</b>	<b>Literature survey</b>	<b>7</b>
<b>3</b>	<b>Dataset and Preprocessing</b>	<b>9</b>
3.1	Dataset Information . . . . .	9
3.2	Data Preprocessing . . . . .	10
3.2.1	Packages and Libraries . . . . .	10
3.2.2	Loading the dataset . . . . .	12
3.2.3	Conversion into Numpy array . . . . .	12
3.2.4	Encoding into onehot vector . . . . .	12
3.2.5	Train test split . . . . .	13
3.2.6	Compiling the Model . . . . .	13
3.2.7	Training the Model . . . . .	14
3.2.8	Testing/Evaluating the Model . . . . .	14
3.2.9	Training Parameters . . . . .	14

3.2.10	Performance Metrics . . . . .	15
<b>4</b>	<b>Convolutional Neural Network</b>	<b>16</b>
4.1	Convolutional Layer . . . . .	16
4.1.1	The 2D Convolution layer . . . . .	19
4.1.2	The Dilated Convolution . . . . .	21
4.1.3	Asymmetric Convolution . . . . .	22
4.1.4	Transposed Convolution . . . . .	23
4.2	Pooling Layer . . . . .	25
4.2.1	<b>Max Pooling :</b> . . . . .	25
4.2.2	<b>Average pooling :</b> . . . . .	27
4.2.3	<b>Global Average pooling :</b> . . . . .	28
4.3	Padding Layer . . . . .	29
4.4	Dropout Layer . . . . .	31
4.5	Fully connected Layer . . . . .	32
4.6	Activation function . . . . .	32
4.7	Batch normalization . . . . .	33
<b>5</b>	<b>Benchmark Models</b>	<b>35</b>
5.1	VGG Model Architecture . . . . .	35
5.2	RanNet Architecture . . . . .	36
<b>6</b>	<b>Hybrid Model Architectures and Results</b>	<b>40</b>
6.1	ENet Architecture: . . . . .	40
6.1.1	Architecture . . . . .	44
6.1.2	Result . . . . .	47
6.2	Hybrid1 Architecture . . . . .	51
6.2.1	RanNet with VGG . . . . .	51
6.2.2	Architecture . . . . .	52
6.2.3	Results . . . . .	53

6.3	Hybrid2 architecture . . . . .	57
6.3.1	Architecture . . . . .	59
6.3.2	Results . . . . .	60
6.4	Hybrid3 architecture . . . . .	64
6.4.1	Architecture . . . . .	65
6.4.2	Results . . . . .	66

# List of Figures

3.1	Packages and Libraries . . . . .	11
4.1	Convolution operation[1] . . . . .	17
4.2	Example of kernel filter[1] . . . . .	18
4.3	3x3 Convolution kernels with dilation rate as 1,2 and 3[2] . . . . .	21
4.4	Asymmetric Convolution[3] . . . . .	22
4.5	Transposed Convolution . . . . .	24
4.6	Maxpooling operation . . . . .	26
4.7	Averagepooling operation . . . . .	27
4.8	Global Averagepooling layer[4] . . . . .	28
4.9	ReLU and PReLU Activation function[[5]] . . . . .	33
5.1	Skip Connection . . . . .	38
6.1	Enet Architecture Structure Base[6] . . . . .	41
6.2	Dilation convolution filter . . . . .	42
6.3	Confusion matrix for SNR=-6db . . . . .	47
6.4	Confusion matrix for SNR=6db . . . . .	48
6.5	Confusion matrix for SNR=18db . . . . .	49
6.6	RanNet,VGG,ENet architecture Accuracy v/s SNR . . . . .	50
6.7	Confusion matrix for SNR=-6db(Hybrid1) . . . . .	53
6.8	Confusion matrix for SNR=6db(Hybrid1) . . . . .	54

6.9	Confusion matrix for SNR=18db(Hybrid1) . . . . .	55
6.10	Architecture accuracy v/s SNR(Hybrid1) . . . . .	56
6.11	X Block[[7]] . . . . .	58
6.12	Confusion matrix for SNR=-6db(Hybrid2) . . . . .	60
6.13	Confusion matrix for SNR=6db(Hybrid2) . . . . .	61
6.14	Confusion matrix for SNR=18db(Hybrid2) . . . . .	62
6.15	Accuracy v/s SNR(Hybrid2) . . . . .	63
6.16	Confusion matrix for SNR=-6db(Hybrid3) . . . . .	66
6.17	Confusion matrix for SNR=6db(Hybrid3) . . . . .	67
6.18	Confusion matrix for SNR=18db(Hybrid3) . . . . .	68
6.19	Accuracy v/s SNR(Hybrid3) . . . . .	69
6.20	Accuracy v/s SNR for all proposed models . . . . .	70



# List of Tables

3.1	Onehot Vector Example . . . . .	13
3.2	Confusion Matrix . . . . .	15
4.1	Hyperparameters used in our model . . . . .	34
6.1	Comparison Matrix of RanNet,VGGNet and ENet . . . . .	43
6.2	ENet Model Architecture . . . . .	46
6.3	Comparison Matrix of RanNet,VGGNet,Hybrid1 Architecture . . . .	51
6.4	Hybrid 1 Model Architecture . . . . .	52
6.5	Comparison Matrix of RanNet,VGGNet,Hybrid2 Architecture . . . .	57
6.6	Hybrid 2 Model Architecture . . . . .	59
6.7	Comparison Matrix of RanNet,VGGNet,Hybrid3 Architecture . . . .	64
6.8	Hybrid 3 Model Architecture . . . . .	65



# Abstract

Automatic Modulation Classification(AMC) is a technique used in the physical layer of wireless communication systems to identify the modulation coding schemes at the receiver end. In this case, the receiver need not know what modulation scheme the transmitter uses. Deep learning algorithms like Convolution Neural Networks(CNN), Residual Neural Networks, and Attention mechanism based implementations are currently the most popular solutions to AMC. This work presents three hybrid deep learning architectures that combine the benchmark architectures like VGG [8] and RanNet [9]. The new hybrid model combines Residual blocks with cascaded Chained Residual Pooling blocks and serial Convolution and pooling blocks. The models were trained on the RadioML 2016 dataset, which has ten modulation schemes. We compared the hybrid models with the baseline architectures, where the new hybrid architecture outperformed VGG. We found the hybrid model's overall accuracy at 18dB of SNR to be 87.6%.

# Chapter 1

## Introduction

### 1.1 Introduction to Automatic Modulation Classification

Deep Learning is a subdivision of Machine Learning that deals with the study of neural networks, that comes under computer vision. Computer vision is a domain where the computer learns to perform as efficient as human vision. Deep learning are widely used in application in communication system. One of the Application of deep learning in communication system is Automatic Modulation Classification. Automatic Modulation classification (AMC) which detects/identifies the modulation signal coming at the receiver from transmitter in wireless communication system. The Automatic modulation classification (AMC) is a fundamental signal processing technique that eventually improves the spectrum utilization efficiency by identifying modulation signals. Deep learning algorithms like convolution neural network, Residual neural network overcomes the drawback of traditional approach of AMC.

Even Under the presence of channel fading and channel noises deep learning is able to learn characteristic of radio signals and do modulation recognition which improves the classification performance. Over the years technology has been enhanced and more approaches have been made in this field. Automatic modulation classification incorporate Deep learning algorithms because it is a progressive way to detect and extract rich features from modulation signals that in-turns increases classification accuracy.

The AMC Automatic modulation classification use the radio signal having modulation information for classification. In this work RadioML dataset with 10 different modulation signals along with various SNR vlues is used and More Automatic modulation classification algorithms have been proposed.

A convolution neural network in deep learning is center of attraction in this field because they were specially designed for image data processing. Many model architectures have been introduced ResNet, MobileNet, RanNet, VGGNet and many more for modulation classification of radio signals to realise a close proximity to real time data. With increment in number of layers it makes a model more efficient and accurate but on the other hand it increases the size of the model. The struggle of making deep network to extract more diverse features while keeping the model compact in size, many architectures have been proposed and we are also trying to do the same.

## 1.2 Aim and Motivation

Aim of the project is to find appropriate deep neural network architecture for automatic modulation classification with advanced designs of convolution neural network for different data types of incoming radio signals from transmitter.

Find out the performance of the proposed architectures and compare their performance with existing ones.

- Check the existing architecture's accuracy and tabulate it.
- Build a new hybrid model based architecture which can outperform the individual ones.
- Refer a block from RefineNet architecture combined it with RanNet architecture. try out variations and got the results.
- Tabulate the results of all the hybrid model combinations and conclude their performance based on accuracy

The interest of this project lied to build a different and new architecture that can outperform the existing architectures in terms of modulation classification accuracy for radio signals. So we have use two model architectures which is our benchmark model architectures. The two benchmark models are VGGNet[8], RanNet[9]. Used chained residual convolution block from RefineNet model architecture and tried something new to attain better performance in terms of accuracy.

## 1.3 Outline Of Report

**Chapter 1** contains the introduction to the topic and the motivation for the project. **Chapter 2** contains the information about the Dataset that we have used and the procedure to get the data ready for applying to the model. In **Chapter 3** we have provided the brief introduction about convolution neural network and the layers used in the network and in the architectures that we are trying to build.

**Chapter 4** includes the literature survey regarding the existing research in this topic and the novel ideas that have been put forward in the field of classification with machine learning algorithms and Deep learning algorithms. **Chapter 5** has a detailed description of two benchmark models that we have used which are VGGNet[8] and RaNNet[9]. This chapter also includes ENet and all the proposed new Hybrid

model architecture's explanation and their results. The report in the end is summarised by conclusion and results of this work along with the future scope to extend the domain of this work.

## Chapter 2

### Literature survey

This section provides a quick overview of approaches has been made on the Automatic Modulation Classification of modulation signals over the years on RML2016 dataset starting from developing algorithms for machine learning and deep learning in the field of classification.

Jungmin Kwon has explained the automatic network data classification based on several machine learning algorithms that are deployed to classify real network traffic data. Since data contains the part which needs to be deliver to receiver and the part for network maintenance in real network. Thus several machine learning algorithms have been adopted to identify actual real network traffic data and analyse it in two different target network scenarios[10]. Machine learning algorithm has its limitation. Hence in his next paper **Jungmin Kwon** has proposed DNN Deep neural network for automatic network data classification and ensure that it has better performance than the machine learning algorithms as it can extract features based on artificial neural networks from data[11].

With improvment in technologies over the years image classfication Vibhakar Mansotra has proposed Deep learning model that do FUNDUS image classification and



detection easily with greater accuracy using transfer learning[12].

Jung Ho Lee is discussing automatic modulation classification in various SNR environment using convolution neural network. This research shows performance of the proposed feature image-based method is better than the constellation image-based method[13].

A new feature set is proposed by Jie Li and , Qingda Meng which combines statistical and spectral feature uses support vector machines (SVMs) and error correcting output codes (ECOC), for automatic digital modulation classification (MC). Method proposed is efficient in low Signal-to-noise ratio and needs fewer training data[14].

Thien Huynh-The proposed a cost efficient convolution neural network MC-Net architecture for automatic modulation classification having skip connection to preserve more initially residual information. In the experiments, MCNet reaches the overall classification accuracy of over 93% at 20 dB SNR for RML2018 dataset[15]. The paper by SEUNG-HWAN KIM and others proposes a novel convolutional neural network architecture for AMC. A bottleneck and asymmetric convolution structure are employed in the proposed model, which can reduce the computational complexity. the proposed model not only saves the trainable parameters by more than 67% but also reduces the prediction time for a signal by more than 54.4% compared with those of MCNet[14]. Duona Zhang proposes a heterogeneous deep model fusion (HDMF) method to solve the problem of automatic modulation classification. (1) a convolutional neural network (CNN) and long short-term memory (LSTM) are combined by two different ways ; (2) a large database, including eleven types of single-carrier modulation signal with various SNR values with noises and fading channel. As experiment results modified classifiers based on the fusion model in serial and parallel modes are of great benefit to improving classification accuracy when the SNR is from 0 dB to 20 dB[16].

RanNet neural network architecture incorporated multiple advanced blocks such as attention module and skip connection that has been used to extract diversified features. Role of these block is to strengthen relevant features and weakens irrelevant features[9].

# Chapter 3

## Dataset and Preprocessing

### 3.1 Dataset Information

The Dataset that have been used for processing is 'RML2016.10b.dat' RadioML dataset. It contains sample values of 10 different modulation schemes ('PAM4', 'QAM64', 'CPFSK', 'AM-DSB', 'BPSK', '8PSK', 'GFSK', 'QPSK', 'WBFM', 'QAM16') and 20 different SNR values which varies from -20db to +18 db (in even numbers). The signal transmitted from the transmitter having continuos waveforms of different modulation schemes has been transformed into discrete signals and and matrix have been created. The modulation signal is encoded into labels and then converted into onehot vector that is either 0 or 1. It will make the columns same as number of modulation and for each raw one element will be 1 coresspondingto that modulation signal. These 200 examples, each is 6000 samples long and each sample includes 128 real and imaginary sample points.

We divide or seperate the data into X and Y matrix. Where Y matrix includes all modulation signals along with SNR values. Y matrix will have the shape (200 x 2).

X matrix includes sample points corresponding to the pair of modulation signal and SNR value. Shape of X matrix will be = (200 x 6000 x 2 x 128). So the Data used for processing has shape (200 x 6000 x 2 x 128).

In our project, we use free GPUs from Google Colab (colaboratory). The deep learning frameworks that we use are from Tensorflow and Keras API. The graphs are plotted with the help of Matplotlib and seaborn heatmaps.

## 3.2 Data Preprocessing

Data preprocessing is the zeroth step in the deep learning workflow : to customize raw data in a way the network can accept and operate on. This includes resizing of image to match the input size of an input layer, enhance or diminish certain features to avoid bias etc. The following steps explain the process of data preprocessing.

### 3.2.1 Packages and Libraries

Various packages and libraries are available to aid different operations. These packages/libraries are simply a set of dedicated instructions that help achieve a certain objective. The following libraries have been imported and used in our case.

Import imports the whole library. from import only imports specific members of the library. Numpy, Pandas and Theano are common libraries that have been imported for this project. Pandas allows data manipulation and analysis. Numpy provides the ability to handle lists in an efficient way apart from its typical use of efficient multi-dimensional container of generic data. Theano helps to define, optimize, and evaluate mathematical expressions, especially with multidimensional arrays.

Matplotlib and Seaborn are libraries that help in data visualisation and graphical representation of results. Seaborn is based on the matplotlib framework, better suited to handle Panda dataframes. Scikitlearn is a machine learning library fea-

```
%matplotlib inline
import os, random
os.environ["KERAS_BACKEND"] = "theano"
os.environ["THEANO_FLAGS"] = "device=cpu"
import numpy as np
import pandas as pd
import theano as th
import theano.tensor as T
from tensorflow import keras
from keras.utils import np_utils
import keras.models as models
from tensorflow.keras import layers, activations
from keras.layers.core import Reshape, Dense, Dropout, Activation, Flatten, BatchNormalization
from keras.layers.convolutional import Conv2D, MaxPooling2D, ZeroPadding2D, Conv2DTranspose, UpSampling2D
from keras.regularizers import *
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical, plot_model
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
import sklearn.preprocessing
from sklearn import preprocessing
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
import random, sys, keras
from sklearn.metrics import accuracy_score, confusion_matrix
```

Figure 3.1: Packages and Libraries

turing classification, regression and clustering algorithms including support-vector machines, support-vector classifier etc. We here import the SVC member from the library. Keras and Tensorflow are most widely used open source libraries providing Python interfaces for training and testing artificial neural networks. Although their scope and usage is wide, their particular focus is on training deep neural networks. Importing activations includes all activation functions that are used in our architecture. `accuracy_score` and `confusion_matrix` is to evaluate accuracy and to plot matrix of true and predicted class that helps recognizing true and false prediction percentage for each class. Library `pickle` is used to load the .dat file dataset. Library `plot_model` is to plot the block diagram of model.

We use these frameworks to import basic amenities of a convolutional neural network such as convolutional layers (`conv2D`, `Maxpooling2D`, `Zeropadding2D`, `Conv2DTranspose`, `UpSampling2D`), optimiziers (Adam), regularizers etc. These li-

braries coupled with basic Python programming give shape to our Convolutional Neural Network.

### 3.2.2 Loading the dataset

Our dataset is in a .dat file called RML2016.10b.dat. We use the following syntax to load our dataset: **`data = pickle.load(f,encoding='latin-1')`**

### 3.2.3 Conversion into Numpy array

Numpy arrays are compact and faster than Python lists. An array consumes lesser memory and is easier to operate on. NumPy uses much less memory to store the data and provides the freedom of specifying data types. Example: **`X = np.array(X)`** converts a variable X into a numpy array. Also, the NumPy library has many built in algebraic and other functions that make it easier to manipulate numpy arrays compared to lists for which things might have to be written from scratch.

### 3.2.4 Encoding into onehot vector

Most Deep learning algorithms or machine learning tools require preparing the data before it can fit into the model. It will require categorical data values to be converted into binary values. Each categorical value will be converted into categorical columns and the entries in those columns will be 0 and 1. For example we have data 'red', 'green', 'blue' that will be first encoded into categorical values as 1, 2, 3, that will be converted into a 3 digit vector each as [1,0,0],[0,1,0],[0,0,1] respectively.

So this one hot encoding is used to do better classification prediction. For our dataset the modulation signals will be converted into categorical values first then one hot encoding. So the Y Matrix for our dataset will have a matrix with 10 columns (number of classes=10)

Type	Red_onehot	Blue_onehot	Green_onehot
Red	1	0	0
Blue	0	1	0
Green	0	0	1

Table 3.1: Onehot Vector Example

After one hot encoding the matrix we got will be a diagonal matrix.

### 3.2.5 Train test split

A dataset has to be split into train and test sets to evaluate how accurate our model is performing. The train set fits the model and statistics of the train set are known. The test set is used solely for prediction purposes. For the purpose of this project, we have used 30% data for test purposes while 70% data is used for train purposes. The following is a syntax to obtain the train test split using scikitlearn library:

```
sklearn.model_selection.train_test_split(*arrays, test_size=None,  
train_size=None, random_state=None, shuffle=True, stratify=None)
```

### 3.2.6 Compiling the Model

Compiling a model defines loss function, optimizer and metrics. For our model compilation these hyper parameters have following specifications. It does not effect the weights the model learns so we can compile a model as many times as we want.

```
model.compile(loss='categorical_crossentropy',  
optimizer='adam',metrics=['accuracy'])
```

Optimization is an important process in machine learning that compares the prediction and the loss function to optimise the input weights. In our project, the model is using Adam Optimizer from keras.

After compilation model needs to be trained that is done by `model.fit`.

**metrics:** A list of metrics that the model will evaluate during training and testing.

### 3.2.7 Training the Model

Model is trained using `model.fit` for a fixed number of epochs. Epochs = iteration on the dataset. For our model we took number of epochs as 30. Model training API used here is `model.fit`.

### 3.2.8 Testing/Evaluating the Model

For testing the designed deep learning model, `model.evaluate` is used. The API call returns the values of Categorical Cross-entropy Loss and the Accuracy of the model.

### 3.2.9 Training Parameters

**Batch Size** - The number of training samples used in a single iteration is referred to as batch size. The batch size in our case is 128.

**Epoch:** - An epoch is a term used to refer to one passing of the entire training dataset by the algorithm. Datasets are grouped in number of batches (see above point) especially when the amount of data is very large. The term iteration is also used sometimes. It is referred to the passing of one batch of the dataset.

### 3.2.10 Performance Metrics

#### Confusion matrix

Confusion matrix give the idea about how good the model is at predicting the classes. The matrix shows when the model is confused and when it is able to make prediction. When the model has good classification accuracy ,confusion matrix will have elements mostly in diagonal position depicts that classes are being classified correctly.

The importance of confusion matrix is that we are able to identify which class are misclassified and which one are correctly classified with a number when we have a overall model classification accuracy.

It can be applied ti binary classification as well as multiclass-classification. For binary classification confusion matrix is :

In this confusion matrix receives values counts from true and predicted values. The

		Predicted class			
		Negative		Positive	
Actual	Negative		TN	FP	
	Positive		FN	TP	

Table 3.2: Confusion Matrix

terms are:

TN = True Negative, negative examples classified accurately.

TP = True Positive, positive examples classified accurately.

FN = False Negative, Actual positive examples that are classified as negative.

FP = False Positive, Actual negative examples classified as positive.

#### Accuracy

Classification Accuracy of a model is given as:

$$Accuracy = \frac{TN + TP}{TN + TP + FN + FP} \quad (3.2.1)$$



## Chapter 4

# Convolutional Neural Network

Convolutional neural network also known as CNN or ConvNet is a class of artificial neural network used for image processing and visualizing. It is the most common deep learning architecture for image recognition tasks. CNN is neural network designed to learn spatial features through backpropagation. The network uses multiple building blocks, such as convolution layers, pooling layers, and fully connected layers.

### 4.1 Convolutional Layer

In convolution layer, convolution operation is applied to the input and the result is passed to the next layer. Convolution combines all pixels in a receptive field into one value. The receptive field is the area of our filter. Convolution will reduce the image size and combine all the information in the field into one pixel when applied to an image. The convolution operation is done with the help of filters. Filters are used to analyze the impact of nearby pixels.

**Convolution operation :** We take a filter of specified size and move that filter

across the image from top left to bottom right and it will do the operation that includes matrix multiplication and addition of the results onto the feature map. Using a convolution operation, the filter is used to calculate a value for each point on the image.

On our input, we perform numerous convolutions, each using a different filter. Various feature maps are created as a result. In the end, all the feature maps are put together as the final output of convolution layer.

Translation invariance is introduced as well as parameter sharing in these filters.

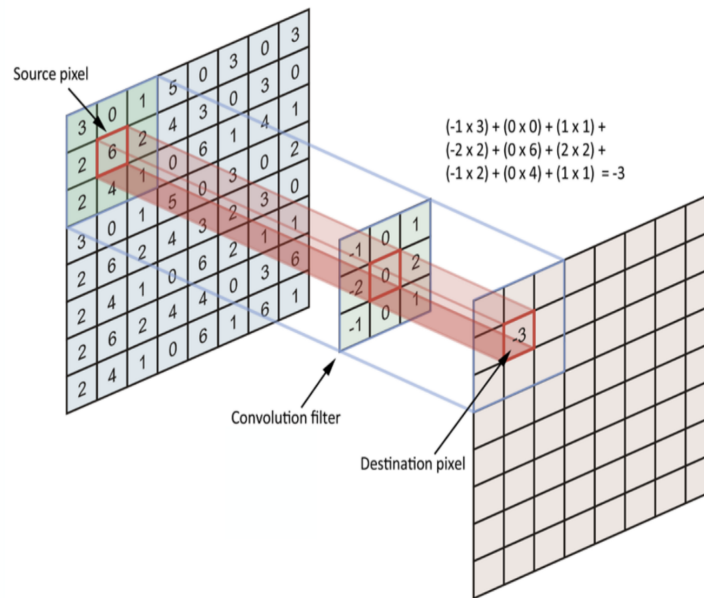


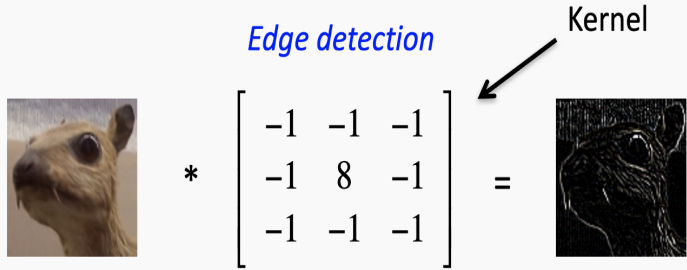
Figure 4.1: Convolution operation[1]

**Filter** : A network will learn different types of features and its very unlikely for a network to learn same features. We randomly specify filter values when we build the network, which continuously update themselves as it is trained.

These filters are used for extracting different kind of features such as edge detection, sharpening of an image, brightening of an image, contrast of an image etc.

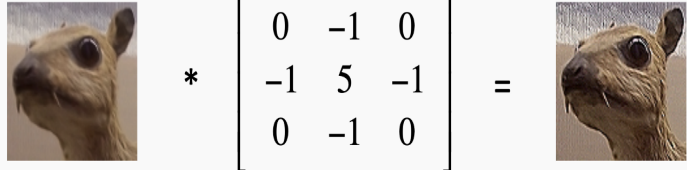
some example of filters or kernels are given below:

*Edge detection*



$$\begin{matrix}
 & & \text{Kernel} \\
 & \swarrow & \\
 \text{Input Image} & * & \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} & = & \text{Edge-detected Image}
 \end{matrix}$$

*Sharpen*



$$\text{Input Image} * \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} = \text{Sharpened Image}$$

Figure 4.2: Example of kernel filter[1]

The feature maps generated for each filter are created after the filters have passed over the image. Then, the image is processed through an activation function, which determines whether or not certain features are present at a given location in image. As network goes deeper and deeper we can do a lot of things such as add more filtering layers and creating more feature maps.

It is possible to use different kinds of convolutions based on the type of problem we are trying to solve and the kind of feature we are looking to learn.

**Parameter Calculation for Convolution layer :** The learnable parameters for a convolution layer can be calculated as  $= [(\text{shape of width of kernel} * \text{shape of height of the kernel} * \text{number of input channels to the layer}) + 1] * \text{Output channels}$ .

#### 4.1.1 The 2D Convolution layer

The most common type of convolution is 2D convolution, generally abbreviated as conv2d. A filter will be slid over the input image with a certain 2D shape to get the convoluted output. For each point it slides over, the kernel will do the same procedure, changing a 2D matrix of features into a different 2D matrix of features. The format of conv2D layer from keras (Keras is a Python-based deep learning API that runs on top of TensorFlow, a machine learning platform) is as follows:

**Conv2D(filters, kernel\_size, strides=(1,1), padding="same")**

The parameters used in the convolution layer(conv2D layer) are as follows[17]:

**Filter:** An integer, the output space's dimensions (i.e. the number of output filters in the convolution).

**Kernel size:** It includes the height and width of the 2D convolution window that is specified by an integer or a tuple/list of two numbers. To express the same value for all spatial dimensions, a single integer can be used.

**Strides:** The steps of the convolution along the height and width are specified by an integer or tuple/list of two numbers. It's possible to express one value for all spatial dimensions, and it can be a single integer.

Any stride value not equal to one is incompatible with any dilation rate value not equal to one.

**Padding:** There are two options "valid" or "same". "Valid" denotes NO padding. "same" padding means row of zeors and column of zeros will be added to spatial dimensions of input image such that input shape will be equal to output shape. It can be left/right or up/down. The output is in same shape as input when padding="same" and strides=1.

**Data fromat:** It's a string, have two options, "channels\_last"(default) or channels\_first. Inputs with shape (batch size, height, width, channels) go to channels\_last, while inputs with shape (batch size, channels, height, width) go to channels\_first.

**Dilation rate:** The dilation rate to utilise for dilated convolution is specified by an integer or a tuple/list of two numbers. It's possible to express one value for all spatial dimensions, and it can be a single integer. Currently, specifying any dilation rate value not equal to with specifying any stride not equal to is incompatible.

**Activation:** To use the activation function. If you do not specify anything, no activation is used. This activation function can be Sigmoid, tanh, ReLU, Leaky ReLU, PReLU and Softmax.

**Kernel initializer:** Initializer for the kernel weights matrix. Defaults to 'glorot\_uniform'.

### 4.1.2 The Dilated Convolution

Dilation convolution simply means inserting zero-values into convolution kernels and it will increase window size of the filter used for convolution operation without changing number of weights. Dilated convolution is used to retain large receptive field and used so that lowering of resolution of input image can be avoided. The convolution operator is changed to use the filter parameters differently. The dilated convolution operator can apply the same filter at different ranges using different dilation factors. This is also known as `dilation_rate` as a hyperparameter of convolution layer. Dilated convolution operation does not involve construction of dilated filters.

It can be understood with following example:

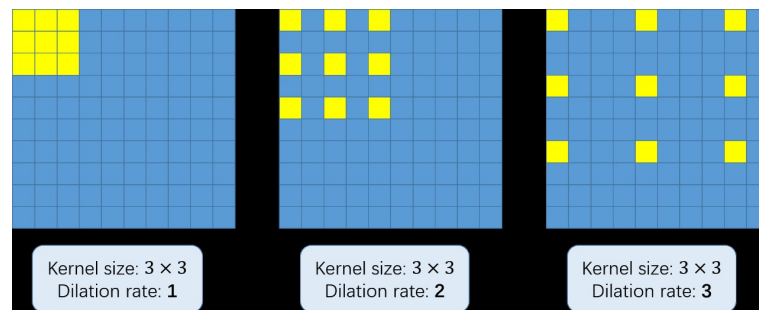


Figure 4.3: 3x3 Convolution kernels with dilation rate as 1,2 and 3[2]

### 4.1.3 Asymmetric Convolution

Asymmetric convolutions are classified into two types: spatial asymmetric convolutions and depthwise asymmetric convolutions.

In the Enet[6] architecture that we have tried on RML2016 dataset includes spatial asymmetric convolution, that is primarily concerned with an image's and kernel's spatial dimensions: width and height. A asymmetric spatial convolution operation divides a kernel into two smaller kernels. For example a 3x3 kernel would be divided into a 1x3 and 3x1 kernel. So for this convolution multiplication operation will be less than the original convolution that reduces computational complexity . For this example total multiplication involved are 6 (3 multiplication each) from as compared to conventional convolution that has number of multiplication as 9[3].

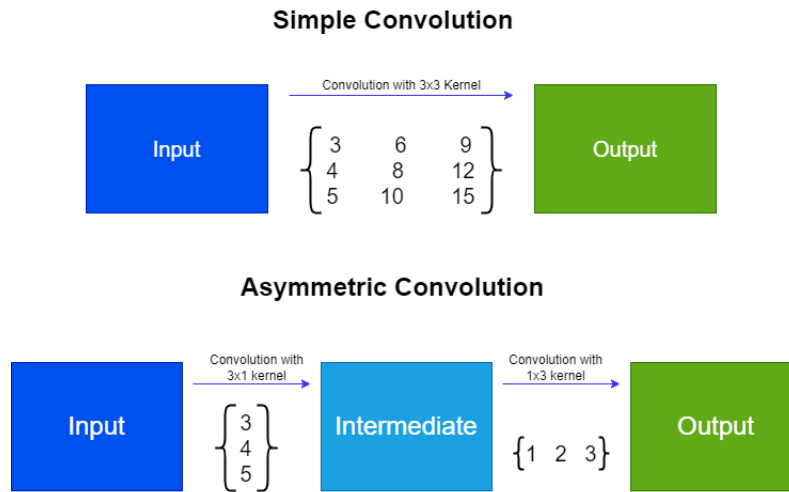


Figure 4.4: Asymmetric Convolution[3]

Number of multiplication involved:

In conventional convolution 64, 3x3 kernel that will move across the input image having size as 4x4x1 will have total no. of multiplication involved is  $64 \times 3 \times 3 \times 4 \times 4 = 9216$ . Number of kernel are same as number of channels of input image. Considering no padding and stride length as 1. The 3x3 kernel will have 9 multiplication everytime and will give out 1 number that results in output shape as  $(4-3+1=2) \times 2$ . Whereas in spatial asymmetrix convolution kernel is divided into two small kernel as 3x1 and 1x3. Each involve number of multiplication;  $64 \times 3 \times 1 \times 4 \times 4 = 3072$ . So total number of multiplication involved is 6144, that is less than conventional convolution operation multiplication.

So copmlexity has been reduced, system will run faster. The difference will be significant for a large size input image or kenel size.

#### 4.1.4 Transposed Convolution

These convolutions are also referred to as deconvolutions or fractionally strided convolutions. A transposed convolutional layer performs a regular convolution while reversing the spatial transformation. It is opposite of simple convolution. Basic operation in transposed convolution can be explained with the following example.

It is also known as upsampled convolution, which refers to the task that it is used to perform, which is to upsample the input feature map[18].

Example with input as a image : Converting a 2x2 feature map/input image into a 3x3 image. A 2x2 feature map needs to be upsampled to a 3x3 feature map. 2x2 feature map has entries as 1,2,3,4. A kernel of size 2x2 which take entries as 5,6,7,8. Now every element of input feature map will be multiplied with each value of the kernel. Gives out 4 feature maps some of the elements of the resulting upsampled feature maps are over-lapping. We simply add the elements of the overlapping positions to solve this problem.



The final upsampled feature map with the required spatial dimensions of 3x3 will be the output.

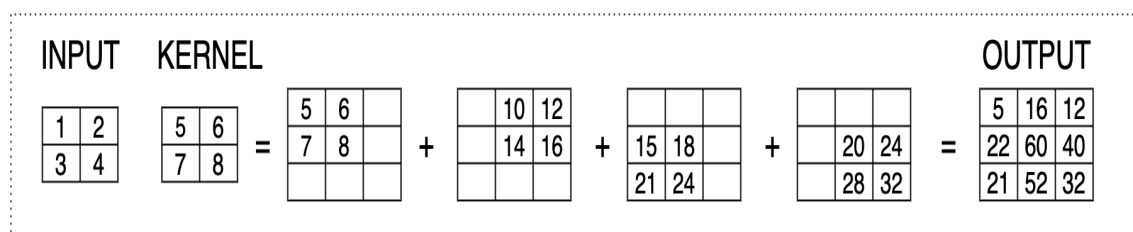


Figure 4.5: Transposed Convolution

## 4.2 Pooling Layer

Pooling layer performs pooling operation; which is sliding a kernel/filter of a specified shape known as pool size over the input image and getting all the features that comes under the region of coverage of filter.

Input shape of the image is  $= h * w * c$

Output shape after a pooling layer will be  $= (h - f + 1)/s * (w - f + 1)/s * c$

$h$  = Height dimension of input

$w$  = Width dimension of input

$c$  = Number of input channels

$f$  = Size of the filter

$s$  = Stride length

There are three different types of pooling layers like Max Pooling, Average Pooling and Global Average Pooling. We discuss them in detail in the following section.

### 4.2.1 Max Pooling :

In Max Pooling operation, given a input feature map (could be a input image), a kernel with a specified shape ('pool size' in the case of pooling layer), will be sliding over the each channel of the input and getting all the features which are lying in that particular region of the input feature map. Output of this layer will have most salient features of the input feature map.

In the Architecture that we have used, Maxpooling2D layer from keras has been used. The kernel window will be shifted along the input image with stride length and maximum value from that window of the input image will be picked. That's how

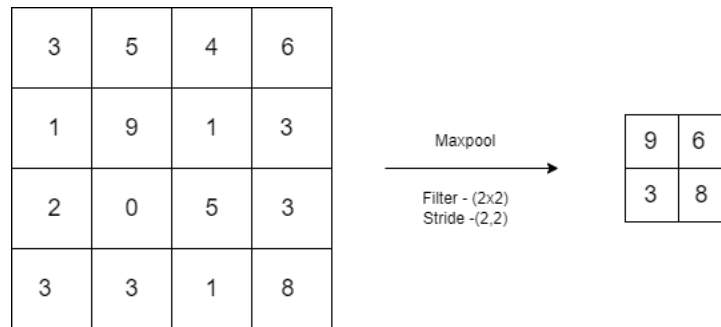


Figure 4.6: Maxpooling operation

entries of the output feature map will be filled. The syntax of the Maxpooling2D layer is as follows:

**MaxPooling2D(pool\_size=(2,2), strides=2, padding='same')**

- **Pool size:** It can be a integer or can be a tuple of two integers. When it takes a single integer, that will be consider for both dimensions. For example if pool\_size is (2,2) the maximum will be taken from the input of this window shape.
- **Stride:** Has three possibilities; can be a single integer, tuple of two integers or None. Stride value defines how pool window moves during the pooling operation in both dimension for each pooling step. If strides are not defined, it by default takes the pool size.
- **Padding:** It can be either 'valid' or 'same'. 'valid' means there is no padding. 'same' results in the padding which gives the output that has same dimension as input, and 'same' will do padding evenly in each side of the input image/tensor.

If its a 'valid' padding, output shape will be =  $((\text{input\_shape} - \text{pool\_size})/s + 1)$

If its a 'same' padding, output shape will be =  $((\text{input\_shape} - 1)/s + 1)$ , it keeps the output shape same as the input shape. Here  $s$  = stride length.

- **Data format:** Its a string; can be either 'channels\_first' or 'channels\_last'. It depicts the arrangement of dimensions of the input. input = (number of rows, number of columns, number of channels)  
output = (number of pooled rows, number of pooled columns, number of channels)  
When data\_format is 'channels\_last';  
input = (number of channels, number of rows, number of columns)  
output = (number of channels, number of pooled rows, number of pooled columns)  
pooled rows/columns means the shape we are getting as a output from the pooling layer.

### 4.2.2 Average pooling :

In Average pooling it will give the average of the elements that are present in the window of the input feature map covered by the pool. Like maxpooling gives maximum number of the particular window of the input image, Average pooling operation will give the average of that window.

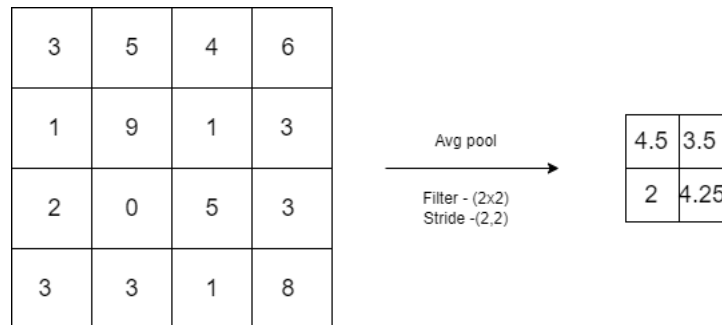


Figure 4.7: Averagepooling operation

Avgpooling2D layer from keras has been used. The window of the kernel/filter shifted along every dimension of the input image by strides. The syntax of the Averagepooling2D layer is as follows:

**AveragePooling2D(pool\_size=(2, 2), strides=None, padding="valid", data\_format=None)**

The Argument explanation of average pooling layer is same as maxpooling layer.

### 4.2.3 Global Average pooling :

Global average pooling layer takes average of entire input window of size  $h \times w$  (height and width of input image) for each channel, where input image shape is  $h \times w \times c$ , and gives output as  $1 \times 1 \times c$ . This is equal to do the average pooling with a filter of size  $h \times w$ .

A fully connected layer can be replaced with the global average pooling layer. Likewise, a Global Maxpooling can also be done.

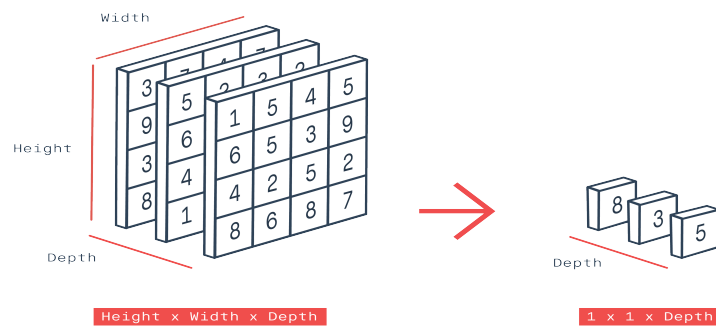


Figure 4.8: Global Averagepooling layer[4]

Syntax for Global Average Pooling layer is as follows:

**GlobalAveragePooling2D(data\_format=None, keepdims=False)**

- **Data format:** Its a string; can be either 'channels\_first' or 'channels\_last'. It depicts the arrangement of dimensions of the input. If its a channels\_first, it corresponds to input shape of (channels, height, width) whereas for channels\_last the input shape is (height, width, channels). By default it takes 'channels\_last'.
- **Keepdims:** It is boolean, will have either 'True' or 'False'. If it is 'False' (default), spatial dimension will be eliminated. And if it is 'True' spatial dimension will be retained with length 1.

If the input image has shape = (batch size, number of rows, number of columns, number of channels). Now if

Keepdims = 'True', output shape = (batch size, number of channels)

Keepdims = 'False':

- if data\_format = 'channels\_last': output shape = (batch size, 1, 1, number of channels)
- if data\_format = 'channels\_first': output shape = (batch size, number of channels, 1, 1).

The pooling layer has no learnable parameters.

## 4.3 Padding Layer

Padding in convolution neural network is adding number of pixels in input image when it is being convolved with the kernel.

**Need of padding :** While doing convolution operation input image shape shrinks by a factor that depends on the filter size. For example a image of 6x6 being convolved with filter size of 3x3, with stride=1 gives a output shape 4x4.

- This shrinking of the input shape might effect the network when use a multiple convolution operation.
- When the filter size increases, then there will be restriction on using the number of convolution layers, If input shape is 6x6 and kernel size 5x5, only two convolution layers can be applied.
- For the edge pixels, full filter cannot be slided, hence full convolution can not be done. So the information at edges is being lost.

That's why is Padding is required.

**Convolution operation :**

$$(n * n) * (f * f) = (n - f + 1) * (n - f + 1) \quad (4.3.1)$$

So the Padding used to keep the output shape same as input shape. The padding will increase the length and width of the input. If we add one row at top and one at bottom, similarly adding columns at left and right side of the input image tensor, that will make the input shape as 8x8 and now if we convolve it with the same kernel/filter size of 3x3, will give the output shape as 6x6. So essentially it is keeping the output shape same as input shape.

There are two types padding can be done in convolution neural network:

- **Same:** Same padding in convolution layer will add enough number of pixels to the input image tensor at the edges so that the shape of output will be equal to the input shape.

By what factor padding needs to be done each side of image entirely depends on the filter size. When filter size is odd, number of pixel needs be added to each side can be calculated as:

$$P = (F - 1)/2 \quad (4.3.2)$$

Because we want to distribute the pixels on each side of input image that's why divided by 2 is done.

- **Valid:** Valid padding means while convolution, filter is applied only to valid pixels of the image. So it is eventually original image. Thus Valid padding means no padding.

## 4.4 Dropout Layer

Dropout layer is also one of the most important characteristics of convolution neural network. This layer act as a mask by which some neurons from the incoming layer towards the next layer has been dropped out and other are free to pass to the next layer.

The Dropout layer randomly sets input units to 0 with a dropout rate between 0 to 1 at each step during training time, which prevent overfitting. Other inputs are scaled up by  $1/(1 - \text{rate})$  such that the sum over all inputs is unchanged. Without presence of these layers, first batch of training sample will effect the learning. So this layer prevent learning of features that appear only in later samples.

**Dropout(rate, noise\_shape=None, seed=None)**

**Rate:** Fraction of the input units to drop.

**Noise\_shape:** 1D integer tensor representing the shape of the binary dropout mask that will be multiplied with the input. For instance, if your inputs have shape (batch size, timesteps, features) and you want the dropout mask to be the same for all timesteps, you can use noise shape=(batch size, 1, features).

**seed:** A Python integer to use as random seed.



## 4.5 Fully connected Layer

In a convolution neural network, the fully connected layer is the last layer after which an activation is used. The Flatten layer gives an output as a 1-D vector which is being fed to the fully connected layer, also called the "Hidden Layer."

Fully connected layers in neural networks are those in which all of the inputs from one layer are connected to every activation unit in the following layer. The final few layers of most popular machine learning models are full connected layers that compile the data extracted by previous layers to form the final output. It takes the second most time, after the Convolution Layer.

## 4.6 Activation function

Acts as a transfer function that maps the output value in between 0 to 1 or -1 to 1. There are two type of activation function:

1. Linear Activation function
2. Non-linear Activation function

In neural network architecture that we are using in our project are using non-linear activation functions. It makes it easy for the model to generalize or adapt with variety of data and to differentiate between the output.

- **Sigmoid activation function** : This function is also called the logistic function. Regardless of the input, the function always outputs a value between 0 and 1. This function passes very large values of the input to be output as 1 and very small or negative values to be output as 0. The sigmoid activation is an ideal activation function or suitable activation function for binary classification but inappropriate for multiclass classification because it requires multinomial probability distribution.
- **ReLU activation function** : Rectified linear activation unit is a piece-wise linear activation unit that passes through the input to the output when its

positive and otherwise output is zero.

Since all the negative values are becoming zero, it decreases the ability of model to train data properly. The function and its derivative both are monotonic.

- **PReLU activation function :** Parameterized ReLU modifies the traditional rectified unit by introducing a parameter by which input gets multiplied for negative values of the input and gives the output, positive values remain the same.

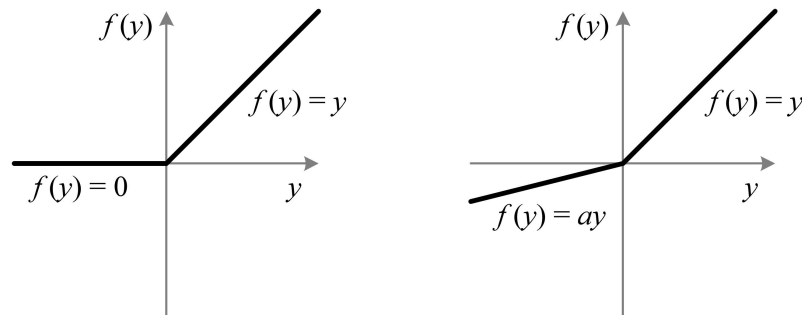


Figure 4.9: ReLU and PReLU Activation function[[5]]

## 4.7 Batch normalization

It is the layer used in convolution neural network that normalizes its input. Batch-normalization is done between the layers. Batch normalisation employs a transformation that keeps the mean output close to 0 and the standard deviation of the output close to 1[19].

**Hyperparameters used for our models:** While building the architecture and training the model, the hyperparameters are used in our project are as follows:

Serial Number	Hyperparameters	Specification
1	Optimizer used	Adam
2	loss	categorical_crossentropy
3	Number of epochs	30
4	Steps per epochs	1000
5	Batch size	128
6	X_train data size	(630000, 2, 128)
7	Y_train data size	(630000, 10)
8	Dropout rates	0.1,0.01,0.3
9	Activation function	ReLU,PReLU

Table 4.1: Hyperparameters used in our model

In our project we are working on RML2016 RadioML data-set that has 10 modulation signals and have a data shape of (200x6000x2x128). Since for our model architecture we are training the dataset from -10db to +18db, having shape = (150x6000x2x128), and has been reshaped to (90000,128,2) according to the model requirements.

By applying train and test split with a factor of 0.3. Hence train data shape = (63000,128,2).

# Chapter 5

## Benchmark Models

### 5.1 VGG Model Architecture

The VGGNet Architecture[8] contains convolutional layers (Conv2D layers), A max-pooling layer and a zeropadding layer is added with the specifications given in the architecture below.

- **Input:** For 'RML2016.10b.dat' RadioML dataset, input shape to the architecture is (4x128x1).We have included different waveform components (e.g., inphase, quadrature, amplitude, phase) as the input data of network by decomposing complex envelope samples of modulation signals.(RANNET,ref)
- **Convolution layers:** Convolution layer in VGG uses filter/kernel size of (1,5) that still captures up/down and left/right. The convolution stride is fixed at 1 pixel to preserve the spatial resolution after convolution. No. of filters used in each convolution layer is 64.

- **Hidden layers:** In the VGG network all the hidden layers use ReLU activation function, which reduces training time. ReLU stands for rectified linear unit activation function, it is a piecewise linear function that passes the input to output when positive; otherwise, the output is zero.
- **Fully Connected layers:** The VGG architecture has two fully connected layers out of which one has 128 channels and the last one has 10 channels, one for each class of the modulation signal.

Classification Accuracy = 84% for 18db SNR. Total number of parameters used = 186,570

## 5.2 RanNet Architecture

The RanNet architecture is an efficient deep network with a novel CNN architecture, has been designed to learn modulation pattern in the training stage automatically and to classify the modulation schemes of incoming signals in the prediction stage.

- **Input:** The architecture starts with the input layer of size (2,256,1). We reshape the inputs to add more number of Ranblocks in the architecture. The original input layer size is (4,128,1), where 128 is the number of discrete samples of signal frames in the RadiomL 2016 dataset.

The Network includes two kind of processing blocks, FeaBlock and RanBlock. The role of these Blocks are Extraction of coarse and fine radio signals from the network body.

- **PreBlock :** The block is structured as follows:  
Cascade connection of two convolution units, where each unit comprises of a convolution layer, a batch normalization (bn) layer and a rectified linear unit (relu) layer. To Computes coarse features from input signals based on their

local correlations, convolution layers of this block has a specific kernel size of (4,7) and (1,7) with stride size of 2.

In order to reduce network sensitivity and accelerate network training convergence, the bn layer is adopted.

A dropout layer prevents overfitting in the network.

- **RanBlock :** This Block consists of two sub-blocks, FeaBlock and Attention module. In RanBlock, attention connection and skip connection are performed using element-wise multiplication and element-wise addition, respectively.

**FeaBlock :** Multiple convolutional units are arranged in parallel structure to extract fine features. Each unit is specified by a different 1-D convolution layer using small filter sizes which are 1x1, 1x3, 1x5 to enhance feature diversity. A depth-wise concatenation (concat) layer is used to gather outcomes of convolution layer to concatenate feature maps along the channel dimension.

A sophisticated structural connection incorporating attention connection and skip connection is proposed to improve the learning efficiency of fine features in Ran-Blocks.

**Attention module :** To strengthen relevant features and deteriorate irrelevant features simultaneously, we build an attention module that calculates attention values. The module has global average pooling (GAP) layer, convolution layers, activation layer with sigmoid function and element-wise multiplication (mult) to perform attention connection. Using the gap layer, we calculate the global features of FeaBlock, which are then processed by 1x1 convolutional layers and a mult layer to compute channel attention values. Multiplicity of chain rule derivatives can cause the informative gradient to decrease as the network deepens, leading to insignificant updates to the initial layers. To solve this issue a Skip connection inspired by residual block in ResNet is adopted so that it will not degrade the performance of classification module.

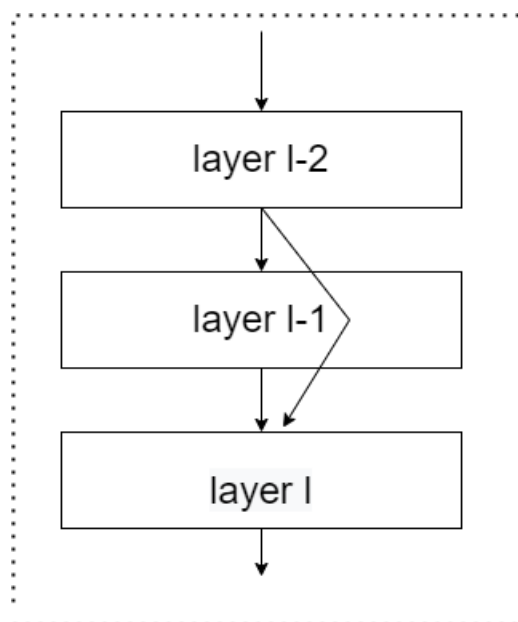


Figure 5.1: Skip Connection

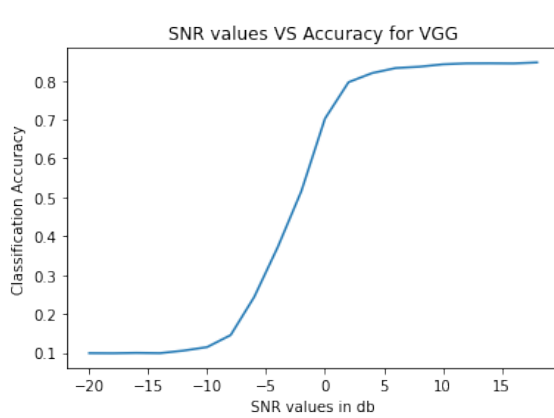
**Skip connection :** While training deep neural networks, the performance of the model drops down with the increase in depth of the architecture. This is known as the degradation problem. This can be reduced with skip connection as feature loss can be . The neural network with skip connections has a smoother loss surface, resulting in faster convergence. An example of Skip connection used in residual block of ResNet architecture is as follows:

There are eight convolutional layers, after each is a max pooling layer added, and a dropout layers in the end with the dropout rate of 0.3 added to prevent overfitting. After that, there are two dense layers, with 128 and 10 units respectively (10 is the number of classes for classification). The batch size is 128

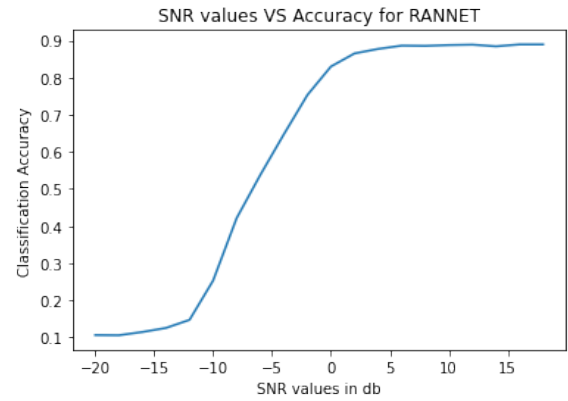
No. of epochs is 35 and the Optimizer is Adam. The Model is trained for -10db to 18db and tested on each SNR values

Classification Accuracy = 84% for 18db SNR.

Total number of parameters used = 186,570



(a) VGG Architecture Accuracy v/s SNR



(b) RANNET Architecture Accuracy v/s SNR



## Chapter 6

# Hybrid Model Architectures and Results

### 6.1 ENet Architecture:

The ENet (Efficient Neural Network) Architecture is a Deep neural network architecture, that is created for applications where it requires minimal delay-time while processing a computer data in presence of a network connection.

We have tried to train the ENet Architecture model on the RML2016 RadioML dataset, that contains 10 modulation schemes and 20 different values of SNR's (-20db to +18db, only even values).

The data shape in the dataset = (200x6000x2x128)

We have trained the model for SNR values from -10db to +18db, (including all modulation signals). So it takes only those value from the dataset to train the model. That is (150x6000x2x128). After pre processing of the the dataset, it has been reshaped to (900000, 64, 8).

#### **The Structure of the architecture :**

The network is divided into many stages. A view of ResNet is adopted, which has one single main branch and a side branch with combination of convolution layers. Both main and side branch will be merged with an element-wise addition.

- **Initial:** Initial block of the architecture contains one maxpooling layer and one convolution layer. That will do downsampling of the input, reduce the dimension by a factor of 2 using specified size of kernel and pool size. Input to this layer is (64, 8, 1). Output of initial block = (32, 4, 1)

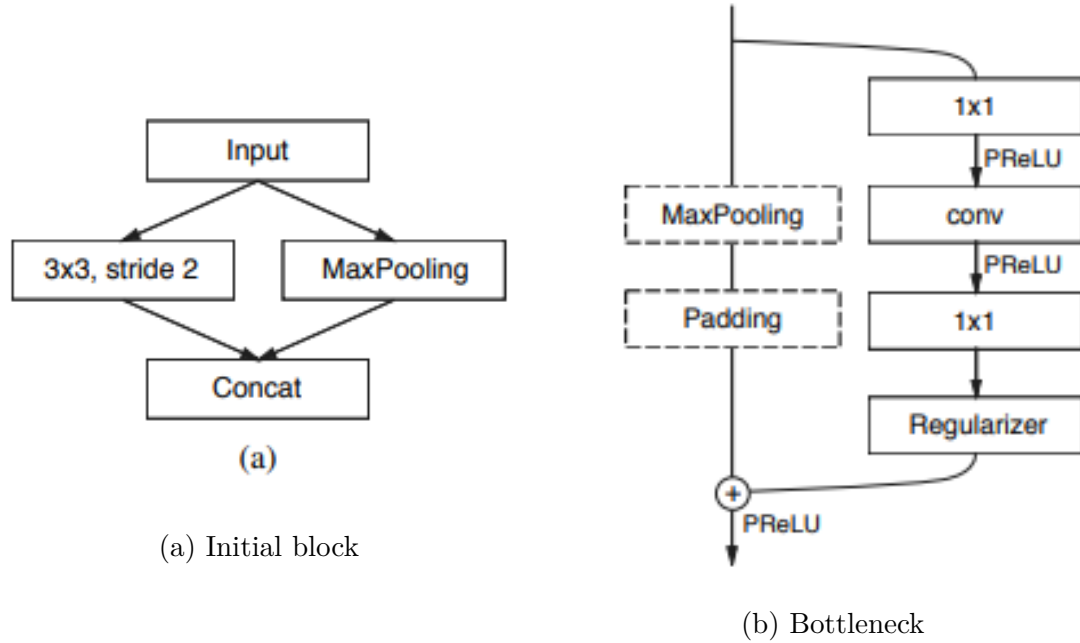


Figure 6.1: Enet Architecture Structure Base[6]

- **Downsampling Bottleneck:** In this bottleneck side branch have three convolution layers. Two with kernel size of (1x1) and one main convolution layer with kernel size (3x3). In downsampling the main convolution layer has a stride length of 2. And in main branch a maxpooling layer with pool size 2x2 is added to do the work. Reduction of full feature map resolution will have loss of spatial information. So every downsampling should have equally strong upsampling.

For every downsampling block input tensor dimensions will go down by a factor of 2.

- **Asymmetric Bottleneck:** Asymmetric block means the main convolution layer in side branch is using asymmetric convolution. A (5,5) filter is decomposed to two small filters with kernel size of (1,5) and (5,1) one after another to reduce number of parameters and computation required. To balance out the dimensions a Zeropadding layer (padding=(0,16)) is added before the asymmetric convolution layer of filter 1x5

- **Upsampling Bottleneck:** This block is having a upsampling layer in the main branch to gain the spatial dimensions that reduced because of the down-sampling. For this purpose we are using UpSampling2D layer from Keras that will upsample the dimension of input.
- **Dilated Bottleneck:** Dilated convolution used to capture or extract more information, after output from each convolution operation, without increasing the number of parameters of kernel. The architecture is using dilation rate for this block is 2,4,8,16. That will be applied to only main convolution layer having kernel size as 3x3 and stride=1. Because stride value =1 will only be compatible with dilation rate not equal to 1.

```

Original Matrix Shape : (9, 9)
[[0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 1. 1. 0. 0.]
 [0. 0. 0. 0. 1. 1. 1. 0. 0.]
 [0. 0. 0. 0. 1. 1. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]]

Original kernel Shape : (3, 3)
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]

Dilated kernel Shape : (7, 7)
[[1 0 0 2 0 0 3]
 [0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]
 [4 0 0 5 0 0 6]
 [0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]
 [7 0 0 8 0 0 9]]

DILATED CONVOLUTION RESULTS [Dilation Factor = 3]
Numpy Results Shape: (3, 3)
[[4. 5. 5.]
 [4. 5. 5.]
 [4. 5. 5.]]

```

Figure 6.2: Dilation convolution filter

- **Fully connected layer:** This part of the architecture is called as classification part of the model, where classification happens. A transpose convolution operation is performed to make the shape same as input tensor dimension, with number of output filter equal to the number of classes in dataset. A Globalaveragepooling and Softmax activation function all together will complete the classification operation.

The final output have shape = (None,10) = (batch size, number of channels)

Tabulating the results from ENet with comparison with Benchmark models:

Model	Accuracy at 18db(%)	Learnable parameter
RanNet	88.96	114746
VGG	84.7	186570
ENet	67.73	136148

Table 6.1: Comparison Matrix of RanNet,VGGNet and ENet

### 6.1.1 Architecture

The Enet architecture description including all significant layers with output shape is given below:

Name	Type	Description	Output Shape	Parameters used
Initial		<b>Main branch</b> 1)maxpool2D(pool=(2,2) stride=2) <b>Side branch</b> 2)conv2D(15,3,stride=1) a)concatenate(1,2)	(None,32,4,1) (None, 32, 4, 15) (None, 32, 4, 16)	210
Bottleneck1.0	Downsampling	<b>Main branch</b> 1)maxpool2D(pool=(2,2) stride=2) 2)Zeropadding2D((0,16), data_format='channels_first') a)concatenate(1,2) <b>Side branch</b> conv2D(4,(1,1),stride=1,p='valid') conv2D(4,(3,3),stride=2,p='same') b)conv2D(64,(1,1),stride=1,p='valid') Add(a,b)	(None,16, 2, 16) (None,16, 2, 48) (None, 16, 2, 64) (None, 32, 4, 4) (None, 16, 2, 4) (None, 16, 2, 64) (None, 16, 2, 64)	3512
Bottleneck1.1		<b>Side branch</b> conv2D(16,(1,1),stride=1,p='valid') conv2D(16,(3,3),stride=2,p='same') conv2D(64,(1,1),stride=1,p='valid')	(None, 32, 4, 4) (None, 16, 2, 4) (None, 16, 2, 64)	7904
Bottleneck1.2		<b>Side branch</b> conv2D(16,(1,1),stride=1,p='valid') conv2D(16,(3,3),stride=2,p='same') conv2D(64,(1,1),stride=1,p='valid')	(None, 32, 4, 4) (None, 16, 2, 4) (None, 16, 2, 64)	7904
Bottleneck1.3		<b>Side branch</b> conv2D(16,(1,1),stride=1,p='valid') conv2D(16,(3,3),stride=2,p='same') conv2D(64,(1,1),stride=1,p='valid')	(None, 32, 4, 4) (None, 16, 2, 4) (None, 16, 2, 64)	7904
Bottleneck1.4		<b>Side branch</b> conv2D(16,(1,1),stride=1,p='valid') conv2D(16,(3,3),stride=2,p='same') conv2D(64,(1,1),stride=1,p='valid')	(None, 32, 4, 4) (None, 16, 2, 4) (None, 16, 2, 64)	7904
Bottleneck2.0	Downsampling	<b>Main branch</b> 1)maxpool2D(pool=(2,2) stride=2) 2)Zeropadding2D((0,0), data_format='channels_first') a)concatenate(1,2) <b>Side branch</b> conv2D(4,(1,1),stride=1,p='valid') conv2D(4,(3,3),stride=2,p='same') b)conv2D(128,(1,1),stride=1,p='valid') Add(a,b)	(None,8, 1, 64) (None,8, 1, 64) (None, 8, 1, 128) (None, 16, 2, 4) (None, 8, 1, 4) (None, 8, 1, 128) (None, 8, 1, 128)	2776 2776
Bottleneck2.1		<b>Side branch</b> conv2D(16,(1,1),stride=1,p='valid') conv2D(16,(3,3),stride=2,p='same') conv2D(128,(1,1),stride=1,p='valid')	(None, 8, 1, 16) (None, 8, 1, 16) (None, 8, 1, 128)	8480

Name	Type	Description	Output Shape	Parameters used
Bottleneck2.2	Dilated 2	<b>Side branch</b> conv2D(16,(1,1),stride=1,p='valid') conv2D(16,(3,3),stride=2,p='same',dilation=2) conv2D(128,(1,1),stride=1,p='valid')	(None, 8, 1, 16) (None, 8, 1, 16) (None, 8, 1, 128)	8480
Bottleneck2.3	Asymmetric	<b>Side branch</b> conv2D(16,(1,1),stride=1,p='valid') ZeroPadding2D(padding=(0,2)) conv2D(16,(1,5),stride=1,p='valid') ZeroPadding2D(padding=(2,0)) conv2D(16,(5,1),stride=1,p='valid') conv2D(128,(1,1),stride=1,p='valid')	(None, 8, 1, 16) (None, 8, 5, 16) (None, 8, 1, 16) (None, 12, 1, 16) (None, 8, 1, 16) (None, 8, 1, 128)	8752
Bottleneck2.4	Dilated 4	<b>Main branch</b> <b>Side branch</b> conv2D(16,(1,1),stride=1,p='valid') conv2D(16,(3,3),stride=2,p='same',dilation=4) conv2D(128,(1,1),stride=1,p='valid')	(None, 8, 1, 16) (None, 8, 1, 16) (None, 8, 1, 128)	8480
Bottleneck2.5		<b>Side branch</b> conv2D(16,(1,1),stride=1,p='valid') conv2D(16,(3,3),stride=2,p='same') conv2D(128,(1,1),stride=1,p='valid')	(None, 8, 1, 16) (None, 8, 1, 16) (None, 8, 1, 128)	8480
Bottleneck2.6	Dilated 8	<b>Side branch</b> conv2D(16,(1,1),stride=1,p='valid') conv2D(16,(3,3),stride=2,p='same',dilation=8) conv2D(128,(1,1),stride=1,p='valid')	(None, 8, 1, 16) (None, 8, 1, 16) (None, 8, 1, 128)	8480
Bottleneck2.7	Asymmetric	<b>Side branch</b> conv2D(16,(1,1),stride=1,p='valid') ZeroPadding2D(padding=(0,2)) conv2D(16,(1,5),stride=1,p='valid') ZeroPadding2D(padding=(2,0)) conv2D(16,(5,1),stride=1,p='valid') conv2D(128,(1,1),stride=1,p='valid')	(None, 8, 1, 16) (None, 8, 5, 16) (None, 8, 1, 16) (None, 12, 1, 16) (None, 8, 1, 16) (None, 8, 1, 128)	8752
Bottleneck2.8	Dilated 8	<b>Side branch</b> conv2D(16,(1,1),stride=1,p='valid') conv2D(16,(3,3),stride=2,p='same',dilation=16) conv2D(128,(1,1),stride=1,p='valid')	(None, 8, 1, 16) (None, 8, 1, 16) (None, 8, 1, 128)	8480
Bottleneck3.0	Upsampling	<b>Main branch</b> conv2D(16,(1,1),stride=1,p='valid') a)UpSampling2D(size=(2, 2)) <b>Side branch</b> Conv2DTranspose(4,1,stride=1,p='valid') Conv2DTranspose(4,3,stride=2,p='same',outpadding=1) b)Conv2DTranspose(64,1,stride=1,p='valid') Add(a,b)	(None, 8, 1, 64) (None, 16, 2, 64) (None, 8, 1, 4) (None, 16, 2, 4) (None, 16, 2, 64) (None, 16, 2, 64)	11736
Bottleneck3.1		<b>Side branch</b> conv2D(4,(1,1),stride=1,p='valid') conv2D(4,(3,3),stride=2,p='same') conv2D(64,(1,1),stride=1,p='valid')	(None, 16, 2, 4) (None, 16, 2, 4) (None, 16, 2, 64)	3320

Name	Type	Description	Output Shape	Parameters used
Bottleneck3.2		<b>Side branch</b> conv2D(4,(1,1),stride=1,p='valid') conv2D(4,(3,3),stride=2,p='same') conv2D(64,(1,1),stride=1,p='valid')	(None, 16, 2, 4)) (None, 16, 2, 4)) (None, 16, 2, 64))	3320
Bottleneck4.0	Upsampling	<b>Main branch</b> conv2D(16,(1,1),stride=1,p='valid') a)UpSampling2D(size=(2, 2)) <b>Side branch</b> Conv2DTranspose(4,1,stride=1,p='valid') Conv2DTranspose(4,3,stride=2,p='same',outpadding=1) b)Conv2DTranspose(16,1,stride=1,p='valid') Add(a,b)	(None, 16, 2, 16) (None, 32, 4, 16)  (None, 16, 2, 4) (None, 32, 4, 4) (None, 32, 4, 16) (None, 32, 4, 16)	4312
Bottleneck4.1		<b>Side branch</b> conv2D(4,(1,1),stride=1,p='valid') conv2D(4,(3,3),stride=2,p='same') conv2D(16,(1,1),stride=1,p='valid')	(None, 32, 4, 4) (None, 32, 4, 4) (None, 32, 4, 16)	8480
FC		Conv2DTranspose(4,1,stride=1,p='valid') GlobalAveragePooling2D Softmax	(None, 64, 8, 10) (None, 10) (None, 10)	1450

Table 6.2: ENet Model Architecture

### 6.1.2 Result

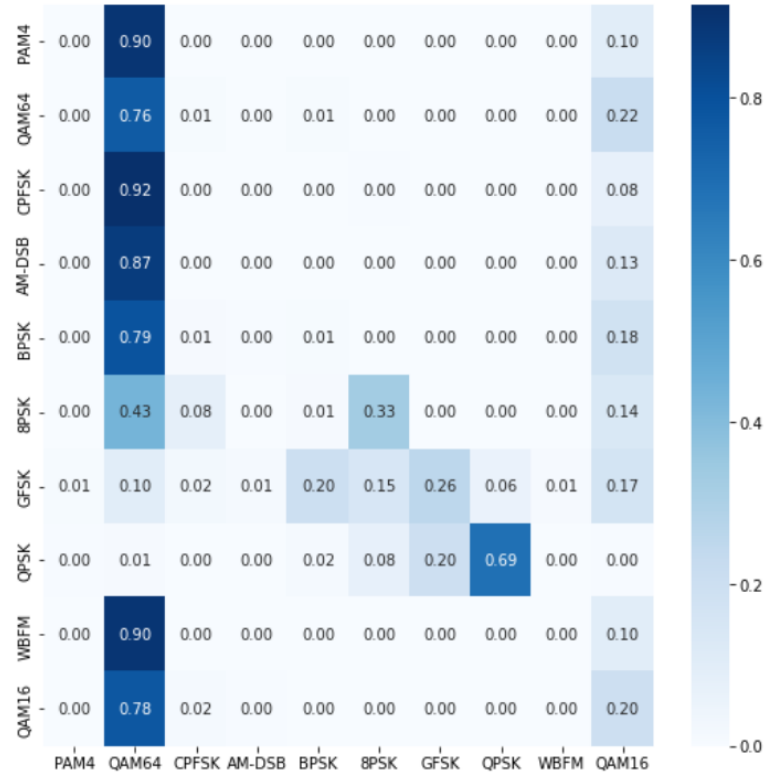


Figure 6.3: Confusion matrix for SNR=-6db

Accuracy at -6db = 22.46%

In confusion matrix most of the classes are predicted false at lower SNR value -6db. A big part of classes are predicted as QAM64. The Enet model is not functioning accurately at low SNR value. 8PSK signal has classification accuracy 33%, which is highest at -6db.



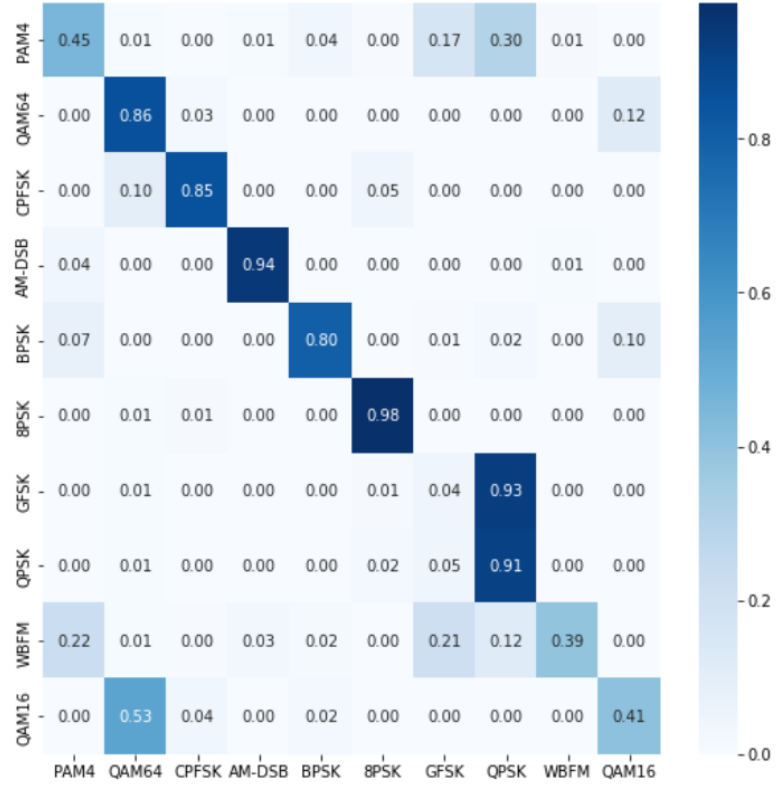


Figure 6.4: Confusion matrix for SNR=6db

Accuracy at 6db = 66.38%

In above results we can see that since classification accuracy is not high but ratio of predicting wrong classes has been reduced at 6db SNR vlue. Although the model is not sufficient to predict QAM64 and QAM16 accurately.

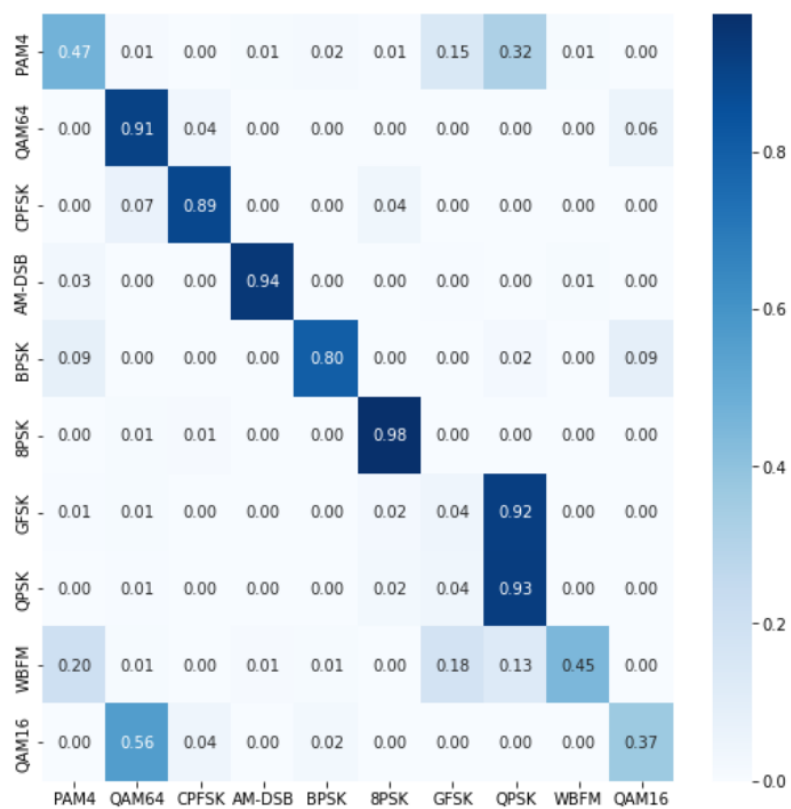


Figure 6.5: Confusion matrix for SNR=18db

Accuracy at 18db = 67.73%

The false prediction accuracy at high SNR value has decreased while for some cases it increased. Prediction accuracy for QAM64 is also increased.

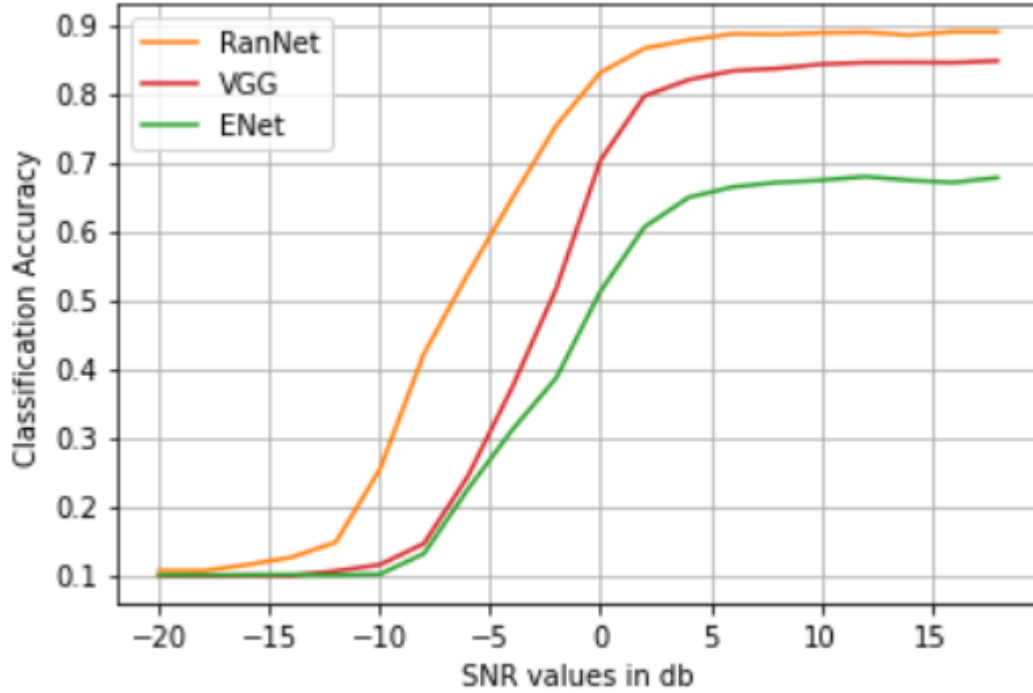


Figure 6.6: RanNet,VGG,ENet architecture Accuracy v/s SNR

The curves shown in figure 6.6 will show the Comparison between accuracy of the benchmark models and Enet model.

The Enet model following performance curve of VGG below -5db SNR vlaue where accuracy is around 30%, after that it started deviating. And reached to a maximum classification accuracy of 67%. It is clearly visible that the experiment of our Enet architecture on RML2016 dataset is not satisfactory, it is not able to perform as required. That leads us to try some new model building methods that can perform nearly accurate as these benchmarks models.

## 6.2 Hybrid1 Architecture

### 6.2.1 RanNet with VGG

We have tried a new and different approach to merge two different models. The RanNet neural network architecture had been modified by adding VGG architecture layers. RanNet architecture is having Ranblocks, we have add convolution layer along with the maxpooling layer incorporating by a skip connection so that to retain all the lost features, in between two Ranblock from VGG architecture.

A second convolution layer along with maxpooling with pool size (2,2) is added at the end of the model. The output shape after all basic layers before flatten layer is (1,32,64).

The number of parameters for this hybrid arctecture is = 145322

While number of parameters for RanNet model = 114746

Adding convolution layer will increase the learnable parameters and accuracy also has been sacrificed a little bit.

Model	Accuracy at 18db(%)	Learnable parameter
RanNet	88.96	114746
VGG	84.7	186570
Hybrid 1	87.89	145322

Table 6.3: Comparison Matrix of RanNet,VGGNet,Hybrid1 Architecture

Got accuracy for hybrid architecture at 18db= 87.89%

## 6.2.2 Architecture

Layers		Description	Output Shape	Parameters used
Preblock		conv(64,(4,7),stride=2) conv(64,(1,7),stride=2)	(None, 1, 128, 64) (None, 1, 64, 64)	30848
Ranblock1	Feablock	m)pool((1,2),stride=2) a)conv(64,(1,3),stride=1) conv(32,(1,3),stride=1) b)conv(16,(1,1),stride=1) c)conv(16,(1,5),stride=1) X)concatenate(a,b,c)	(None, 1, 32, 64) (None, 1, 32, 64) (None, 1, 32, 32) (None, 1, 32, 16) (None, 1, 32, 16) (None, 1, 32, 64)	25472
	Attention	GAP(keepdims=True) d)conv(16,(1,1),stride=1) e)Sigmoid activation function Multiply(d,e) f)conv(64,(1,1),stride=1) g)Sigmoid activation function h)Multiply(g,X) i)Add(h,m)	(None,1,1,64) (None,1,1,16) (None,1,1,16) (None,1,1,16) (None,1,1,64) (None,1,1,64) (None,1,32,64) (None,1,32,64)	2448
VGG Block		j)conv(64,(1,2),stride=1) k)Zeropadding((0,16)) l)maxpool((2,2),stride=2) p)Add(i,l)	(None,1,1,16) (None,1,1,16) (None,1,1,16) (None,1,32,64)	8256
Ranblock2		„	(None,1,16,64)	25216
Attention		„	(None,1,16,64)	2448
Output/FC		Reshape((4,4,64), input=(1,16,64)) conv(64,(1,2),stride=1) maxpool((2,2),stride=2) Flatten Dense(128) Dropout Dense(10) Softmax activation function	(None,4,4,64) (None,3,3,64) (None,2,2,64) (None,256) (None,128) (None,128) (None,10) (None,10)	50634

Table 6.4: Hybrid 1 Model Architecture

Total number of parameters used = 145322

With increment in number of Ranblock model will learn more number of features but after a certain point it will saturate but complexity of model will increase. Here we are using only only two Ranblocks.

### 6.2.3 Results

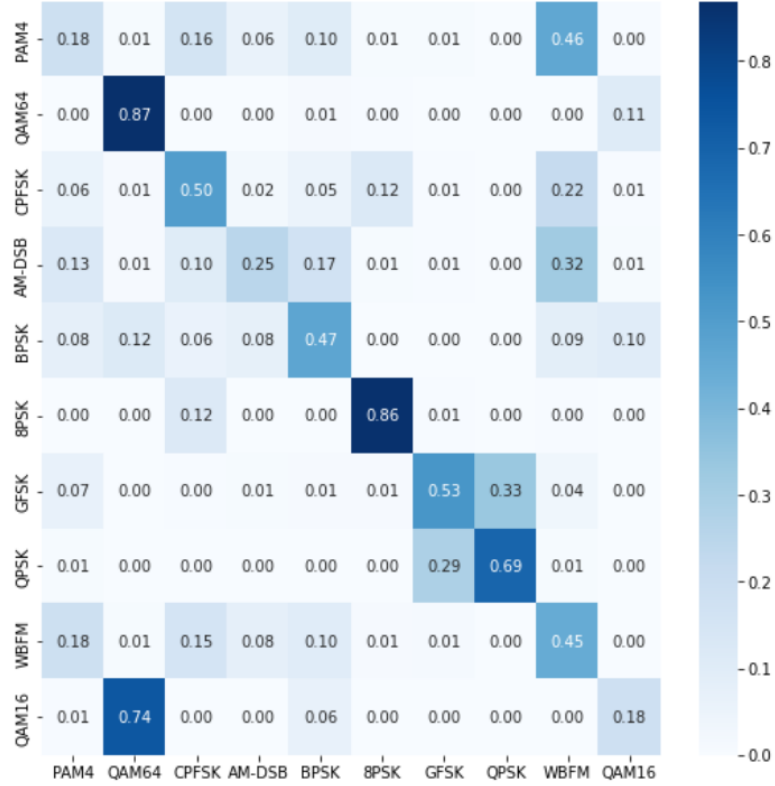


Figure 6.7: Confusion matrix for SNR=-6db(Hybrid1)

Accuracy at -6db = 51.32%

The results from the the model prediction at low SNR value -6db is shown in the confusion matrix. PAM4 modulation signal is classified most accurately than other signals. QAM16 has maximum false prediction percentage. At low SNR value Model sometimes behave randomly at prediction because of the noise.

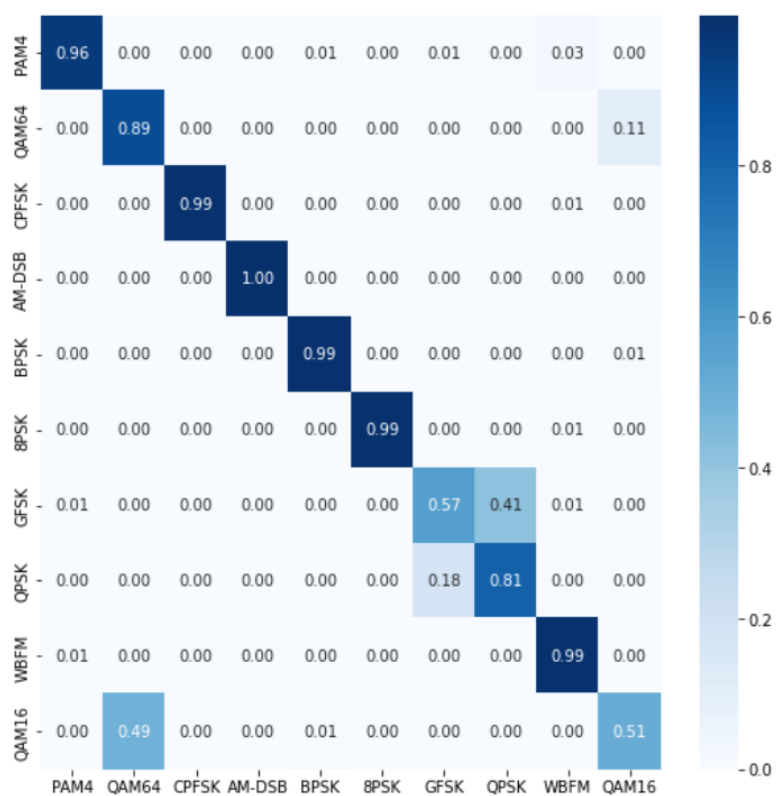


Figure 6.8: Confusion matrix for SNR=6db(Hybrid1)

Accuracy at 6db = 87.96%

At 6db SNR value model is able to predict classes same as true classes. QAM16 modulation signal is predicted wrong by approximately 50% to QAM64. QPSK and GFSK also has some

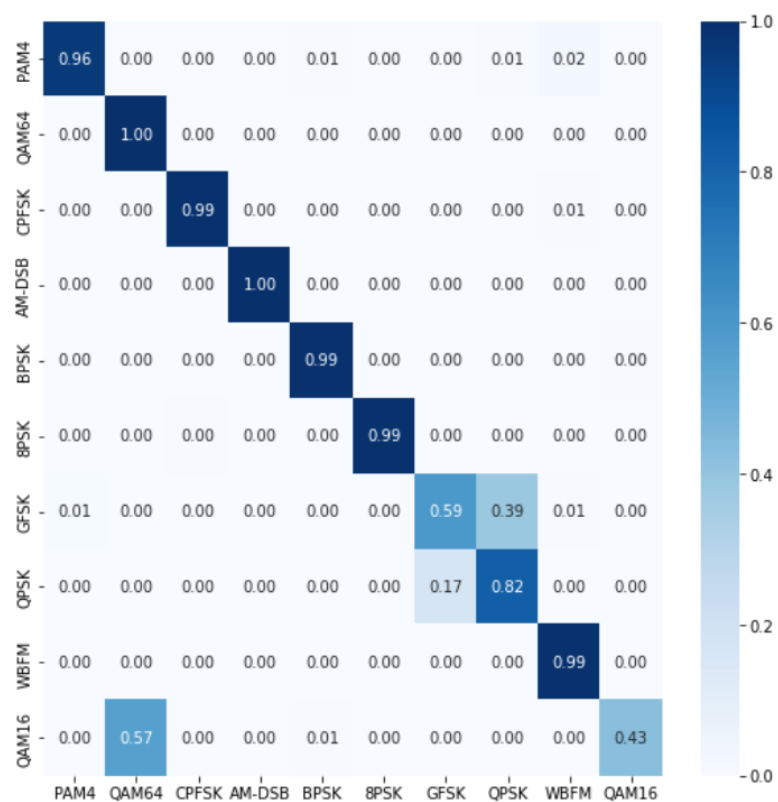


Figure 6.9: Confusion matrix for SNR=18db(Hybrid1)

Accuracy at 18db = 87.89%

Confusion matrix shown in the above figure is for 18db. At high SNr value model is performing



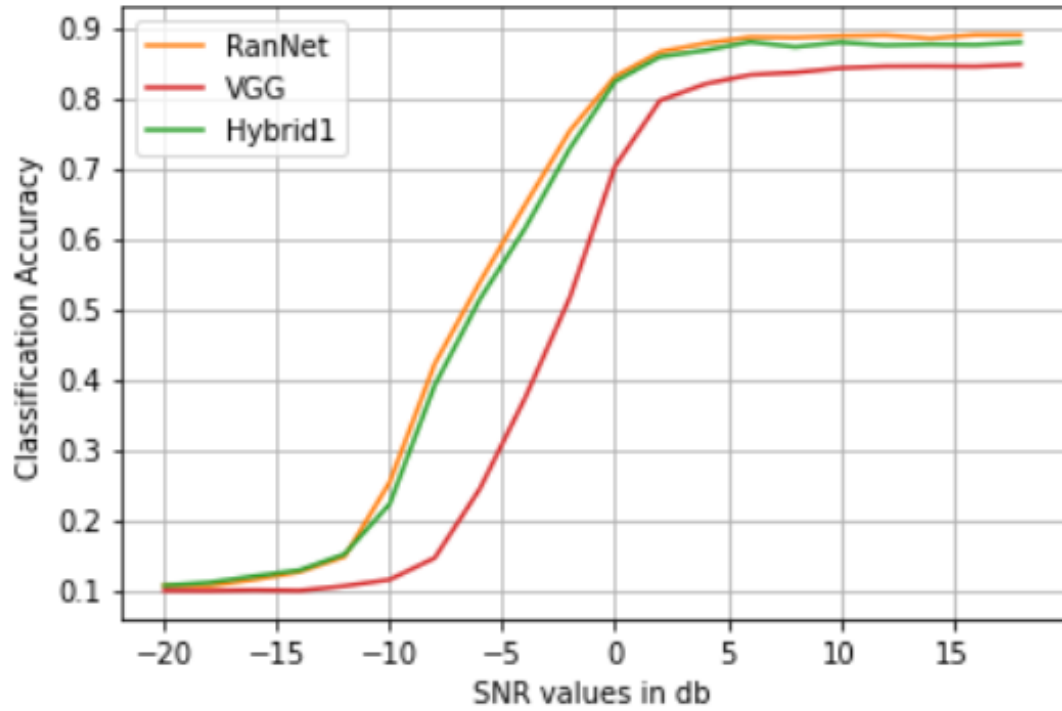


Figure 6.10: Architecture accuracy v/s SNR(Hybrid1)

This is a comparison of performance of the proposed model by us with the benchmark models that has outperform one the model to a certain extent. This model is behaving approximately same as the RanNet model with a bit difference.

## 6.3 Hybrid2 architecture

The Next Hybrid model includes a new block called X block. The block involves convolution layers and their element-wise addition. This block is inspired by one of the blocks from the RefineNet architecture called as **Chain residual pooling**. We have build chained residual convolution block (includes only convolution layers) that has been named as X block, one ran block and 1 Feablock. In this hybrid network we have tried adding new block to the RanNet architecture and modified it to capture background context from a large image.

The X block is built as a chain of multiple convolution blocks. Each convolution block in X block is having kernel size as 3x3. Output from one convolution block goes to the next convolution block as its input. Therefore, the current convolution block is able to re-use the result from the previous convolution operation and thus access the features from a large region without using a large pooling window. The output feature maps of all convolution blocks are fused together with the input feature map through summation of residual connections[7].

Model	Accuracy at 18db(%)	Learnable parameter
RanNet	88.96	114746
VGG	84.7	186570
Hybrid 2	87.69	209578

Table 6.5: Comparison Matrix of RanNet,VGGNet,Hybrid2 Architecture

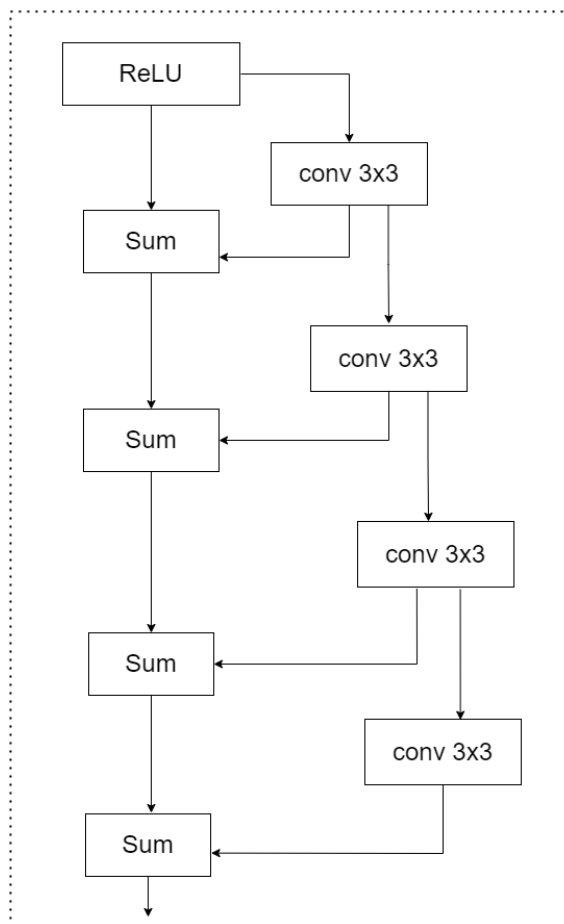


Figure 6.11: X Block[[7]]

### 6.3.1 Architecture

Layers		Description	Output Shape	Parameters used
Preblock		conv(64,(4,7),stride=2) conv(64,(1,7),stride=2)	(None, 1, 128, 64) (None, 1, 64, 64)	31104
Ranblock1	Feablock	m)pool((1,2),stride=2) a)conv(64,(1,3),stride=1) conv(32,(1,3),stride=1) b)conv(16,(1,1),stride=1) c)conv(16,(1,5),stride=1) X)concatenate(a,b,c)	(None, 1, 32, 64) (None, 1, 32, 64) (None, 1, 32, 32) (None, 1, 32, 16) (None, 1, 32, 16) (None, 1, 32, 64)	25216
	Attention	GAP(keepdims=True) d)conv(16,(1,1),stride=1) e)Sigmoid activation function Multiply(d,e) f)conv(64,(1,1),stride=1) g)Sigmoid activation function h)Multiply(g,X) i)Add(h,m)	(None,1,1,64) (None,1,1,16) (None,1,1,16) (None,1,1,16) (None,1,1,64) (None,1,1,64) (None,1,32,64) (None,1,32,64)	2448
X block		1) maxpool((1,2),stride=2) 2)Relu 3)conv(16,(1,2),stride=1) 4)Add(2,3) 5)conv(16,(1,2),stride=1) 6)Add(4,5) 7)conv(16,(1,2),stride=1) 8)Add(6,7) 9)conv(16,(1,2),stride=1) 10)Add(8,9)	(None,1,16,64) (None,1,16,64) (None,1,16,64) (None,1,16,64) (None,1,16,64) (None,1,16,64) (None,1,16,64) (None,1,16,64)	147712
	Attention	„	(None,1,16,64)	2448
FC		GlobalAveragepooling Dense(10) Softmax	(None,64) (None,10) (None,10)	650

Table 6.6: Hybrid 2 Model Architecture

Total number of learn able parameters = 209578

### 6.3.2 Results

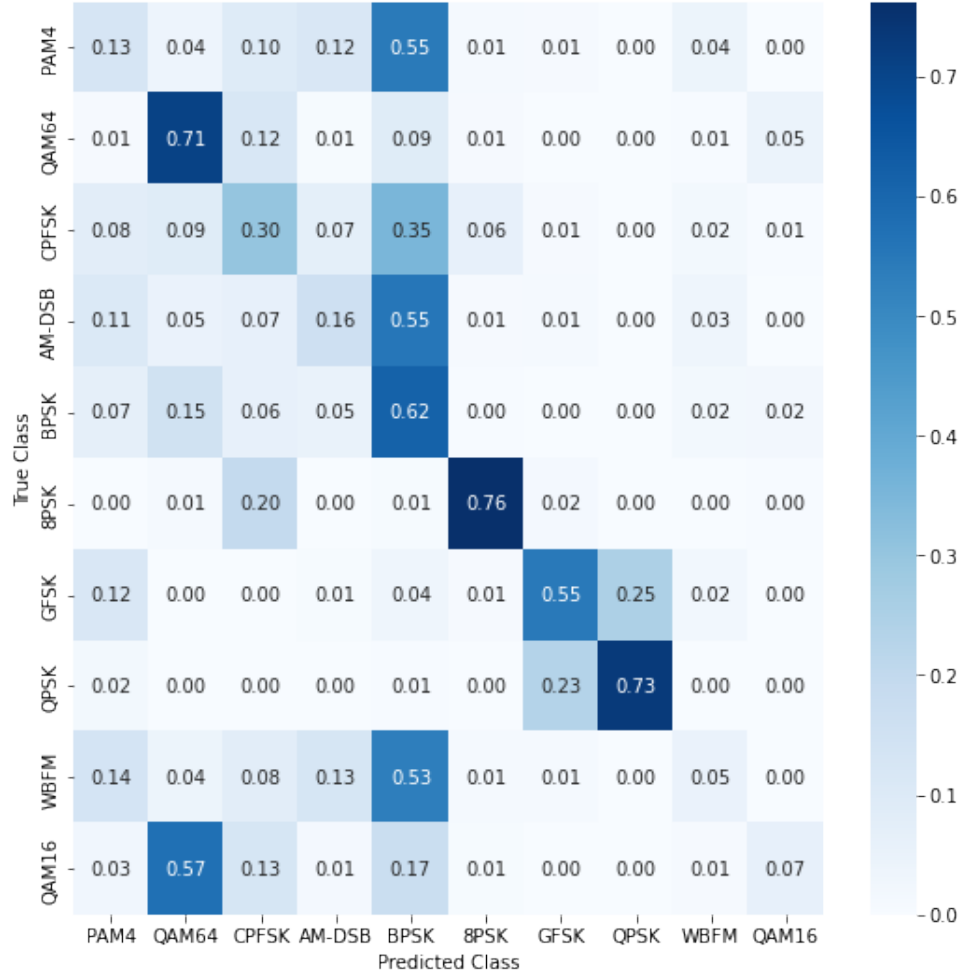


Figure 6.12: Confusion matrix for SNR=-6db(Hybrid2)

Accuracy at -6db = 40.78%

The confusion matrix between true class and predicted class depicts at low SNR value for this hybrid model including a new block called as X block performs 40% accurate. PAM4, QAM16, WBFM also not predicted correctly. Because signal strength is very low at low SNR values.

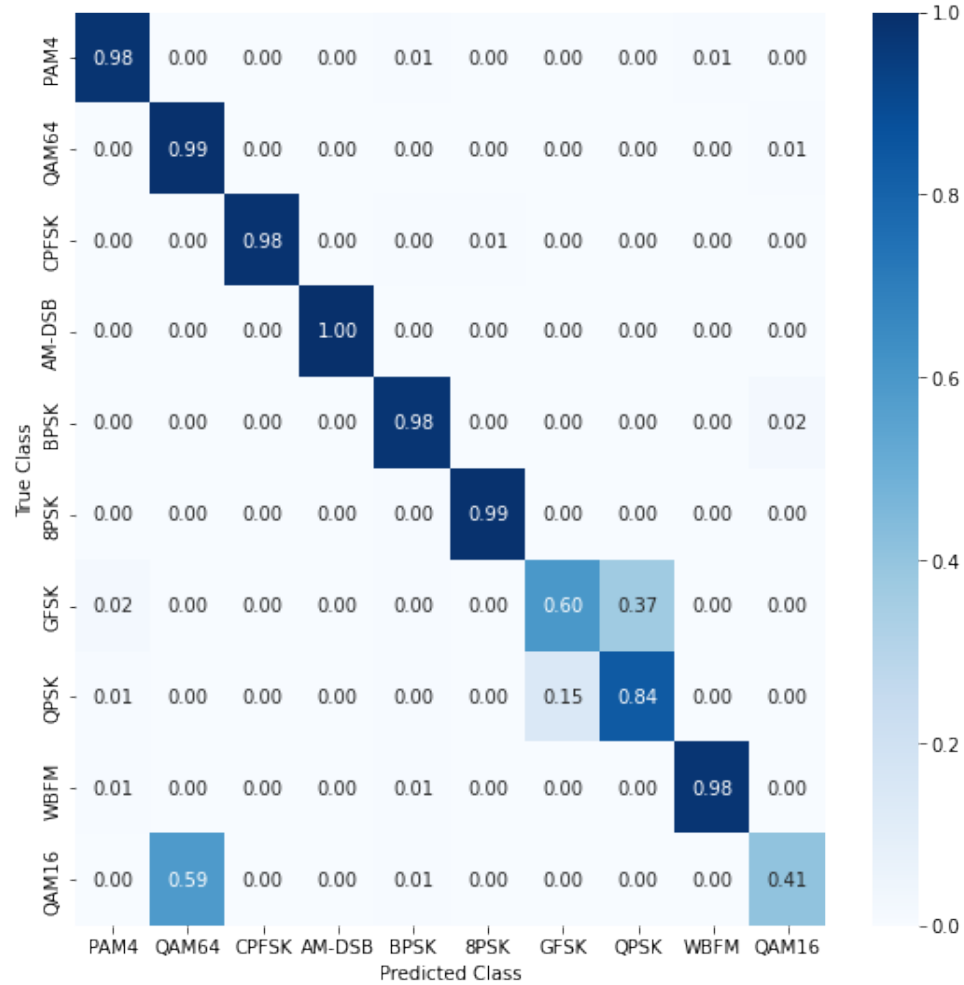


Figure 6.13: Confusion matrix for SNR=6db(Hybrid2)

Accuracy at +6db = 87.5%

At SNR value = 6db model is able to predict most of the classes accurately, overall accuracy of the model prediction is 87%. Although QAM16 incorrectly predicted to QAM64 with a percentage 59%.

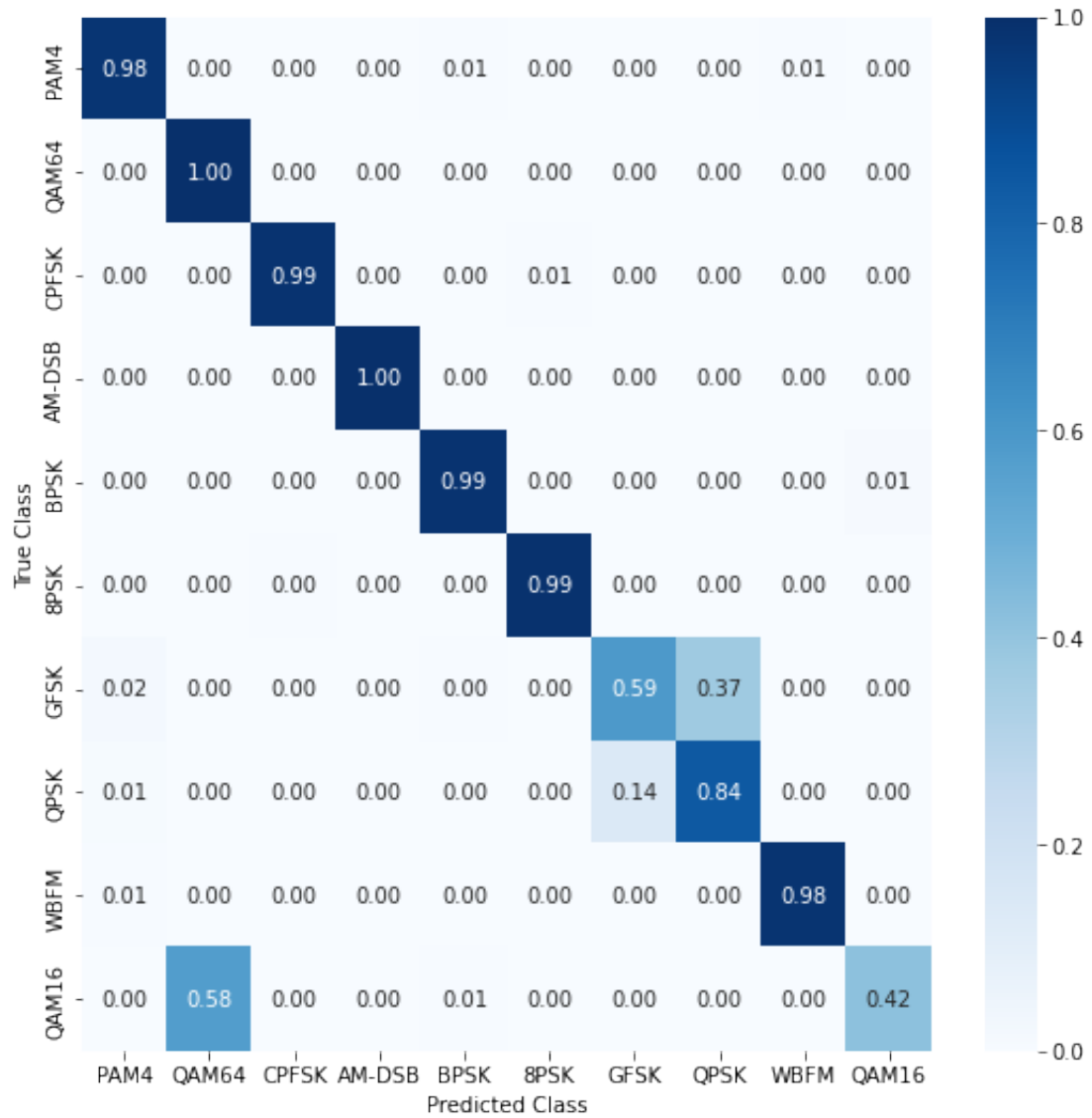


Figure 6.14: Confusion matrix for SNR=18db(Hybrid2)

Accuracy at 18db = 87.69%

At high SNR value 18db classification accuracy of model is high. Some of the modulation signals is predicted to same as their true classes 100%. Some false prediction is still there for GFSK,QPSK and QAM16 modulation signals.

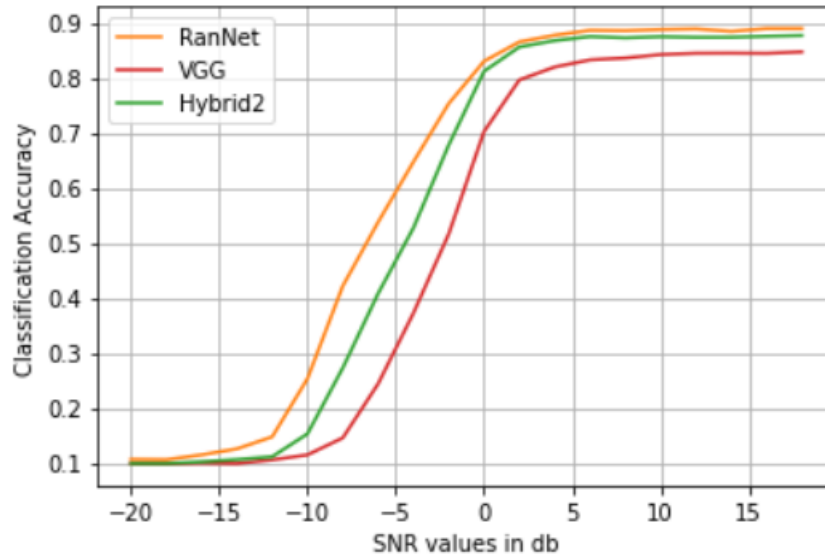


Figure 6.15: Accuracy v/s SNR(Hybrid2)

The above figure 6.14 shows the characteristic of the Hybrid model along with the benchmark models. It is clear from the curves that the hybrid architecture follows the RanNet model after 0db. And before 0db behaves as a average of both the model. Clearly this model is able to outperform VGGNet. The hybrid model curve is visible in green colour in above figure.

These are the results obtained while combining one Feablock from RanNet and one X block.



## 6.4 Hybrid3 architecture

The Hybrid architecture is made by using RanNet architecture incorporating with X block. The RanNet architecture's block Feablock is replaced with X block and all rest layers will remain the same. We have use two Ranblocks each having a X block instead of Feablock. With increase in number of blocks, number of parameters also increases that inturns increase the complexity of the model. So we can not use more blocks so as to increase accuracy, because after certain point classification accuracy saturates.

Model	Accuracy at 18db(%)	Learn able parameter
RanNet	88.96	114746
VGG	84.7	186570
Hybrid 3	81.32	332074

Table 6.7: Comparison Matrix of RanNet,VGGNet,Hybrid3 Architecture

### 6.4.1 Architecture

Layers		Description	Output Shape	Parameters used
Preblock		conv(64,(4,7),stride=2) conv(64,(1,7),stride=2)	(None, 1, 128, 64) (None, 1, 64, 64)	31104
X block		1) maxpool((1,2),stride=2) 2)Relu 3)conv(16,(1,2),stride=1) 4)Add(2,3) 5)conv(16,(1,2),stride=1) 6)Add(4,5) 7)conv(16,(1,2),stride=1) 8)Add(6,7) 9)conv(16,(1,2),stride=1) 10)Add(8,9)	(None,1,16,64) (None,1,16,64) (None,1,16,64) (None,1,16,64) (None,1,16,64) (None,1,16,64) (None,1,16,64) (None,1,16,64)	147712
	Attention	GAP(keepdims=True) d)conv(16,(1,1),stride=1) e)Sigmoid activation function Multiply(d,e) f)conv(64,(1,1),stride=1) g)Sigmoid activation function h)Multiply(g,X) i)Add(h,m)	(None,1,1,64) (None,1,1,16) (None,1,1,16) (None,1,1,16) (None,1,1,64) (None,1,1,64) (None,1,32,64) (None,1,32,64)	2448
X block		1) maxpool((1,2),stride=2) 2)Relu 3)conv(16,(1,2),stride=1) 4)Add(2,3) 5)conv(16,(1,2),stride=1) 6)Add(4,5) 7)conv(16,(1,2),stride=1) 8)Add(6,7) 9)conv(16,(1,2),stride=1) 10)Add(8,9)	(None,1,16,64) (None,1,16,64) (None,1,16,64) (None,1,16,64) (None,1,16,64) (None,1,16,64) (None,1,16,64)	147712
	Attention	„	(None,1,16,64)	2448
FC		GlobalAveragepooling Dense(10) Softmax	(None,64) (None,10) (None,10)	650

Table 6.8: Hybrid 3 Model Architecture

Total number of learnable parameters = 332074

## 6.4.2 Results

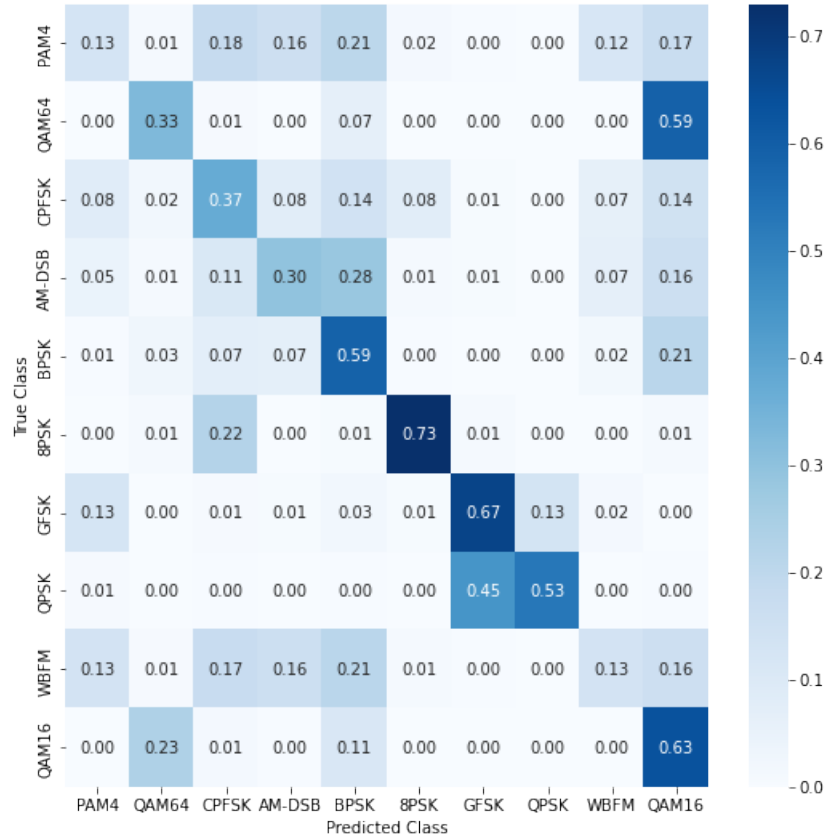


Figure 6.16: Confusion matrix for SNR=-6db(Hybrid3)

Accuracy at -6db = 44.19%

The result from the above confusion matrix shows that some of the modulation signals have not been classified accurately, their classification accuracy is very less. QAM64 has been wrongly predicted as QAM16 and the accurate result percentage for this is less than the false prediction. 8BPSK is classified accurately to some extent at low SNR value -6db.

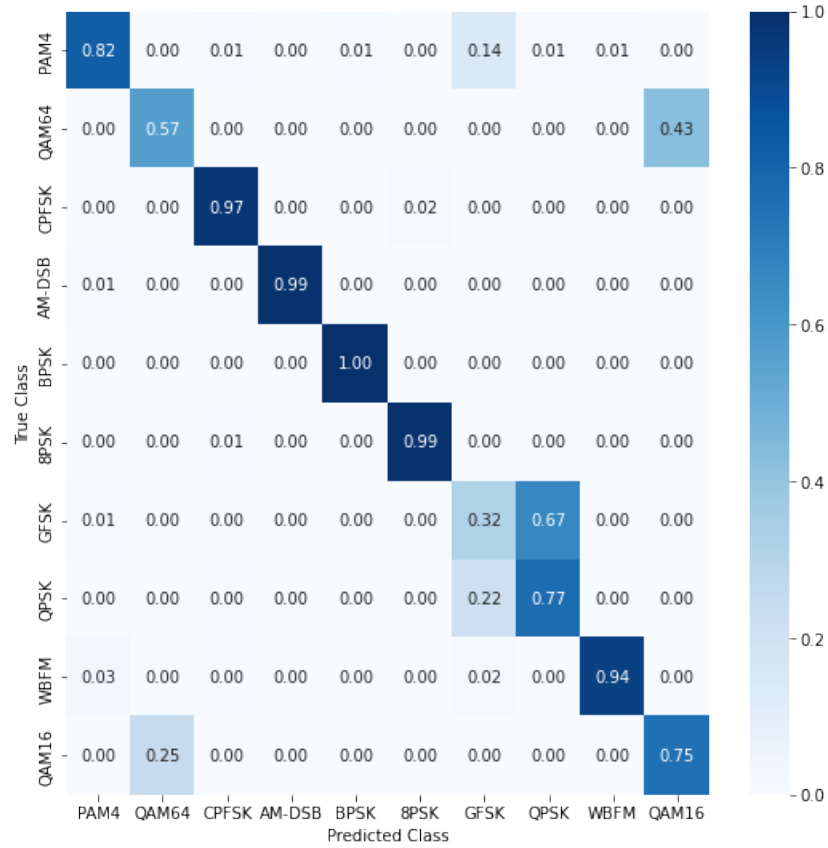


Figure 6.17: Confusion matrix for SNR=6db(Hybrid3)

Accuracy at +6db = 81.23%

At 6db the model is able to predict most of the classes more accurately comparatively at low SNR values. The percentage by which QAM64 is predicted false as QAM16 at 6db is 43%. QAM16 also predicted false by 25% as QAM64. There has been some discrepancy on prediction of GFSK and QPSK also.

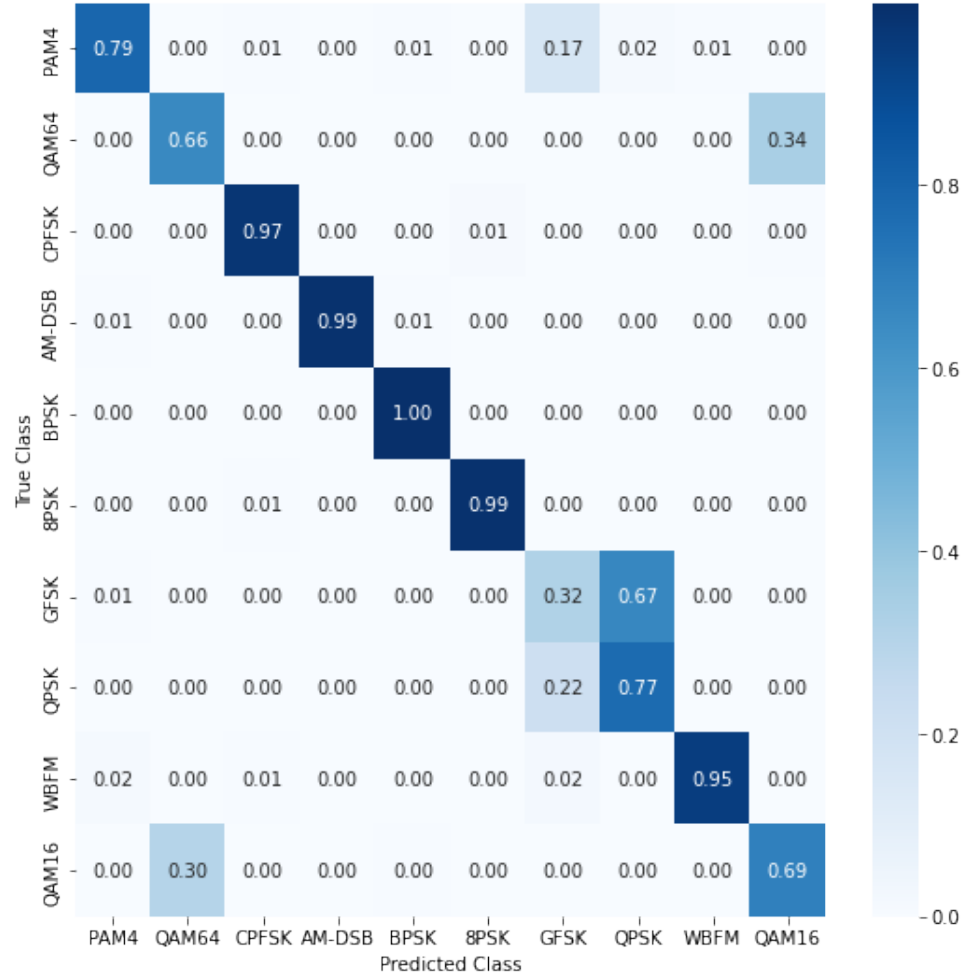


Figure 6.18: Confusion matrix for SNR=18db(Hybrid3)

Accuracy at 18db = 81.32% At high SNR value classification accuracy has increased. In comparison with low SNR values model is able to predict true classes at 18db. Declassification occurs in QAM16, QAM64, PAM4, QPSK and GFSK.

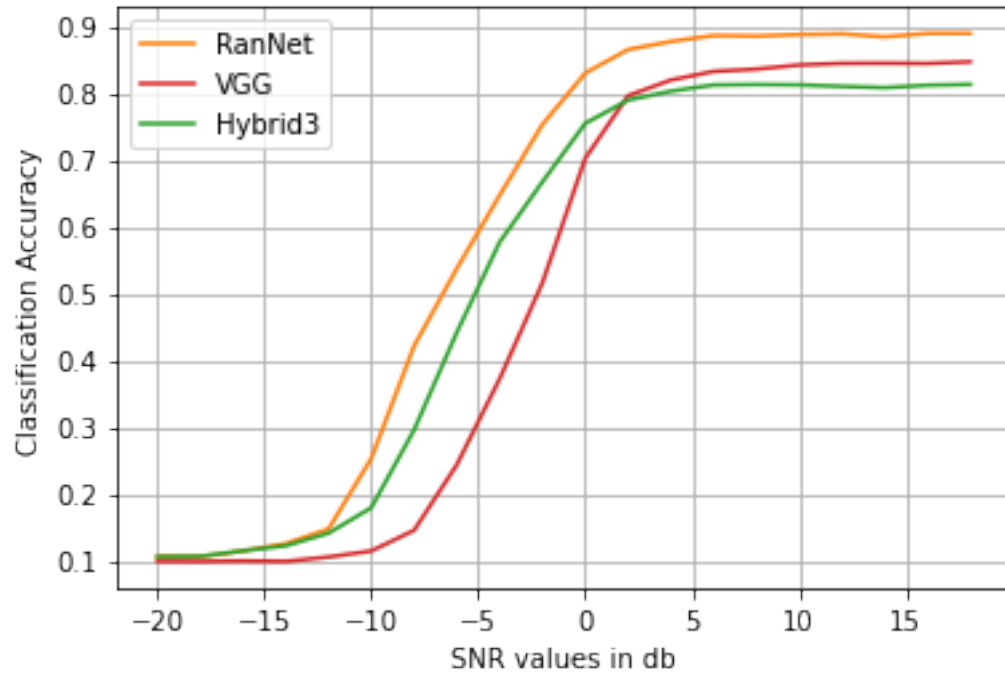


Figure 6.19: Accuracy v/s SNR(Hybrid3)

In this hybrid architecture model two X blocks have been used in Comparison with the hybrid 2 model that used one Feablock and one X block. The classification accuracy for hybrid 3 model is less than the hybrid 2 model. Which means that the incorporating the new X block is not performing well as required.

Tabulation of all the results got so far:

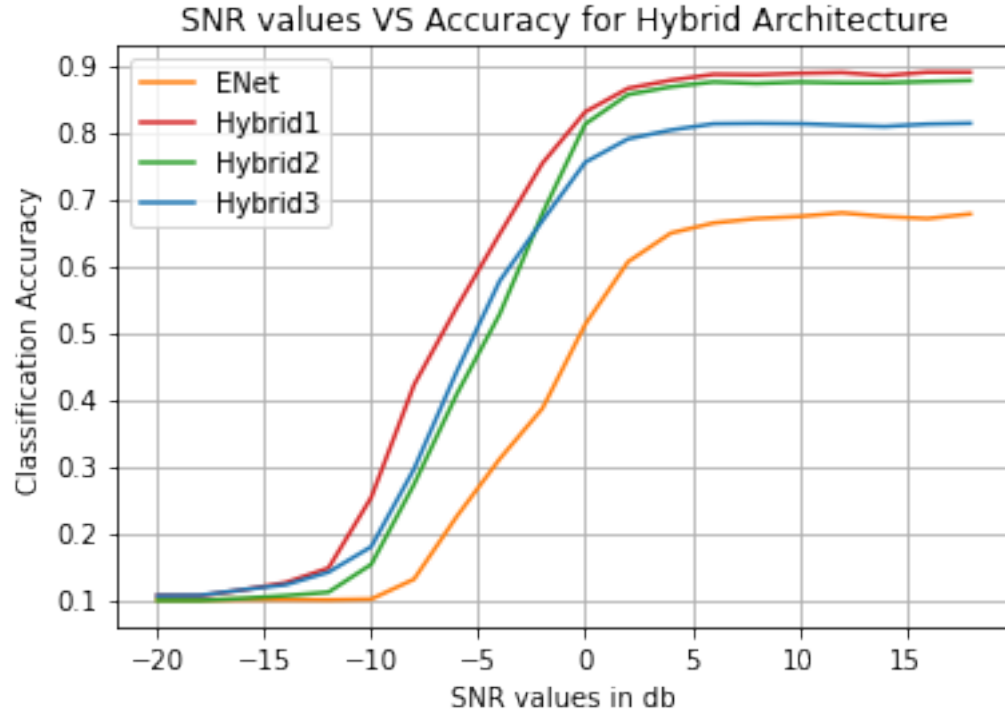


Figure 6.20: Accuracy v/s SNR for all proposed models

Model	Accuracy at 18db(%)	Learnable Parameters
RanNet	88.96	114746
VGG	84.7	186570
ENet	67.73	136148
Hybrid 1	87.89	145322
Hybrid 2	87.69	209578
Hybrid 3	81.32	332074

# Conclusion

Deep learning algorithms are being widely used now to identify and classify images and emulate them with the real life objects. Various architectures have been proposed regarding this classification and two such models are v and r architectures.

In this work we have developed a new hybrid deep learning architecture from the benchmark model architectures which are VGGNet and RanNet. The classification accuracy for this hybrid architecture is 87.6% which is higher than the benchmark architecture. Several other hybrid models have been proposed which combines two blocks, residual blocks from RanNet and and chained residual pooling block that contains only convolution layers. Different combinations of these block have been proposed in which one hybrid model architecture with one residual block and one X are used. In block (chained residual pooling block) the classification accuracy that is obtained is 87.8%. Another hybrid model with two X blocks show an accuracy 81.32%. Further this work is concluded with future scope that can used to further this work and develop new models.

## **Future scope of this work**

- Other different benchmark models can be used to form new set of hybrid models
- Using data augmentation on data set we can increase the accuracy of this hybrid model.



# Bibliography

- [1] M. Stewart, “Simple introduction to convolutional neural networks,” Jul 2020. [Online]. Available: <https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac>
- [2] F. Yu and V. Koltun, “Multi-scale context aggregation by dilated convolutions,” 2015. [Online]. Available: <https://arxiv.org/abs/1511.07122>
- [3] C.-F. Wang, “A basic introduction to separable convolutions,” Aug 2018. [Online]. Available: <https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728>
- [4] “2d global average pooling: Peltarion platform.” [Online]. Available: <https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/global-average-pooling-2d>
- [5] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1026–1034.
- [6] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, “Enet: A deep neural network architecture for real-time semantic segmentation,” 2016. [Online]. Available: <https://arxiv.org/abs/1606.02147>
- [7] G. Lin, A. Milan, C. Shen, and I. Reid, “Refinenet: Multi-path refinement networks for high-resolution semantic segmentation,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 5168–5177.
- [8] T. J. O’Shea, T. Roy, and T. C. Clancy, “Over-the-air deep learning based radio signal classification,” *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 168–179, 2018.

- [9] T. Huynh-The, Q.-V. Pham, T.-V. Nguyen, T. T. Nguyen, D. B. d. Costa, and D.-S. Kim, “Rannet: Learning residual-attention structure in cnns for automatic modulation classification,” *IEEE Wireless Communications Letters*, vol. 11, no. 6, pp. 1243–1247, 2022.
- [10] J. Kwon, D. Jung, and H. Park, “Traffic data classification using machine learning algorithms in sdn networks,” in *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, 2020, pp. 1031–1033.
- [11] J. Kwon, J. Lee, M. Yu, and H. Park, “Automatic classification of network traffic data based on deep learning in onos platform,” in *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, 2020, pp. 1028–1030.
- [12] A. Bali and V. Mansotra, “Deep learning-based techniques for the automatic classification of fundus images: A comparative study,” in *2021 3rd International Conference on Advances in Computing, Communication Control and Networking (ICAC3N)*, 2021, pp. 351–359.
- [13] J. H. Lee, K.-Y. Kim, and Y. Shin, “Feature image-based automatic modulation classification method using cnn algorithm,” in *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIIC)*, 2019, pp. 1–4.
- [14] J. Li, Q. Meng, G. Zhang, Y. Sun, L. Qiu, and W. Ma, “Automatic modulation classification using support vector machines and error correcting output codes,” in *2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, 2017, pp. 60–63.
- [15] T. Huynh-The, C.-H. Hua, Q.-V. Pham, and D.-S. Kim, “Mcnet: An efficient cnn architecture for robust automatic modulation classification,” *IEEE Communications Letters*, vol. 24, no. 4, pp. 811–815, 2020.
- [16] D. Zhang, W. Ding, B. Zhang, C. Xie, H. Li, C. Liu, and J. Han, “Automatic modulation classification based on deep learning for unmanned aerial vehicles,” *Sensors*, vol. 18, no. 3, p. 924, 2018.
- [17] K. Team, “Keras documentation: Conv2d layer.” [Online]. Available: [https://keras.io/api/layers/convolution\\_layers/convolution2d/](https://keras.io/api/layers/convolution_layers/convolution2d/)

- [18] D. Mishra, “Transposed convolution demystified,” Jul 2021. [Online]. Available: <https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba>
- [19] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International conference on machine learning*. PMLR, 2015, pp. 448–456.