Automatic Classification of Radio Sources in the Galactic Plane Using B-CNN

By Vivian Adhiambo

Supervised By: Prof. Melvin Hoare

Submitted in accordance with the requirements for the degree of

Master By Research

The University of Leeds

October 2020.

Acknowledgment

I want to thank Prof. Anna Scaife for the supervision support in developing the classifier. Many thanks to Prof.Melvin Hoare for the unwavering academic support. This journey would not have been possible without the massive financial support from DARA big data. Much more thanks to Linzi and the whole Big data community for the catch-up meetings, which helped in motivating during the pandemic lockdown. My appreciation goes to Tonye, who helped me understand the dataset and availed the needed image cutouts. I want to thank my family for their unconditional love and support. Many thanks to my colleagues at the University of Leeds. My gratitude goes to the University of Leeds, particularly the department of physics and the HPC ARC supporting team. Last but not least, my appreciation goes to Goonhilly earth station for giving access to their GPU clusters.

Abstract

The ever-increasing number of sky surveys calls for the need for more efficient data processing alternatives. CNN has been widely used in computer vision and is currently being incorporated in classifying astronomical objects. A Branch Convolution Neural Network (B-CNN) is a hierarchical-based Convolution Neural Network (CNN) that outputs multiple predictions of coarse to fine levels. This thesis involved developing a B-CNN-based classifier for, automatic classification of objects discovered in the Co-Ordinated Radio 'N' Infrared Survey for High-mass star formation (CORNISH) south survey. First, a 2-level model was developed and trained on similar CORNISH north sources, with additional multi-wavelength data from the IR and the sub-millimeter surveys. From the catalogue, sources used were the ultra-compact HII (UCHII) regions, HII regions, planetary nebulae (PN), radio stars, and extra-galactic background sources. The developed classifier was then tested on a sample of already classified CORNISH south sources with an accuracy of 67% for the coarse level and 33% for the fine level.

Contents

List of Tables

List of Figures

List of Symbols and Abbreviations

Adams	Adaptive moment estimation
AGN	Active Galactic Nucleus
AI	Artificial Intelligence
ANN	artificial neural network
ATCA	Australian telescope compact array
B-CNN	Branch-Convolution Neural Network
BT-Strategy	branch training strategy
CLARAN	Classifying Radio sources Automatically with Neural networks
CNN	Convolution neural network
CORNISH	Co-Ordinated Radio 'N' Infrared Survey for High-mass star formation
FC	Fully Connected
FIRST	Faint Images of the Radio Sky at Twenty-Centimeters
FR-I	Fanaroff-Riley Class I
FR-II	Fanaroff-Riley Class II
HD-CNN	Hierarchical Deep CNN
IR	Infra-Red
ML	Machine learning
MSE	Mean Square Error

NVSS	NRAO VLA Sky Survey
PIL	Python Imaging Library
PN	planetary nebulae
ReLU	Rectified linear unit
SGD	stochastic gradient descent
SKA	Square-Kilometer Array
SUMSS	Sydney University Molonglo Sky Survey
TB-CNN	Tree Based CNN
VLA	Very Large Array
WENSS	Westerbork Northern Sky Survey

Chapter 1

Introduction

Our view of the universe is unique to the band, wavelength, and polarization through which the sky is mapped. Given astrophysical features are observed at given wavelengths. As a result, astronomical surveys are mainly targeted at addressing a particular astrophysics hypothesis. For instance, a multi-wavelength map of the central part of our galactic plane in figure (1.1) shows different objects across varying wavelengths. The galactic center is prominent in the submillimetre. Its visibility decreases with increasing wavelength, becoming less visible in the optical due to a supermassive black hole resulting in high energy phenomena. The different observational wavelengths are such that the optical probes warmer stars, planetary nebulae, and galaxies. X-rays and gamma rays target high-energy phenomena such as black holes, neutron stars, hot gas, and supernova remnants. Ultra-violet shows hot stars and quasars, whereas infra-red (IR) probes cool stars, star formation regions, and cold interstellar dust. Radio gives information on cold molecular clouds, active galactic nucleus (AGN), and radiation from the very early universe (Samal et al., 2010). Combining data from these surveys gives a complete picture of the universe.

Advances in technology have contributed to an improved type of survey types of equipment, increasing the number of sky surveys. Earlier radio surveys were of poor sensitivity and sky coverage (Condon, 1999). Distant extra-galactic luminous sources dominated these surveys. It was not easy to study their multi-wavelength counterparts needed to extract their intrinsic features. These limitations have been overcome by the new generation of radio surveys, impacted by the use of arrays of telescopes interconnected to form an interferometric array. The surveys have better sky coverage and improved sensitivity to nearby usable sources. For example, The NRAO VLA Sky Survey (NVSS) (Condon et al., 1998) covered approximately 82% of the hemisphere, with a declination of $\delta > -40^{\circ}$ at a resolution of 1.4 GHz. There were 2.0×10^6 sources cataloged from the survey. The proposed

upcoming Square-Kilometer Array (SKA) (Schilizzi, 2005) will enable improved radio surveys of the Southern Plane (Simpson, 2017). It will comprise two thousand dishes with million antennas, with a data rate of up to one hundred and sixty terabytes per day. Its precursor telescope, MeerKat, will be included as its mid-frequency component of the SKA phase1. MeerKat has sixty-four antenna arrays of diametre $13.5m$ with a data handling rate of up to $\sim 275GB$ per second. From these, it is predictive that, soon, there will be an inflow rate of petabytes to exabytes of radio data in a day (Devine, Goseva-Popstojanova and Pang, 2018). Relying on old methods of data handling techniques such as classifying radio sources using visual inspection will be practically impossible for this kind of big data. Hence, the need to develop and explore new techniques.

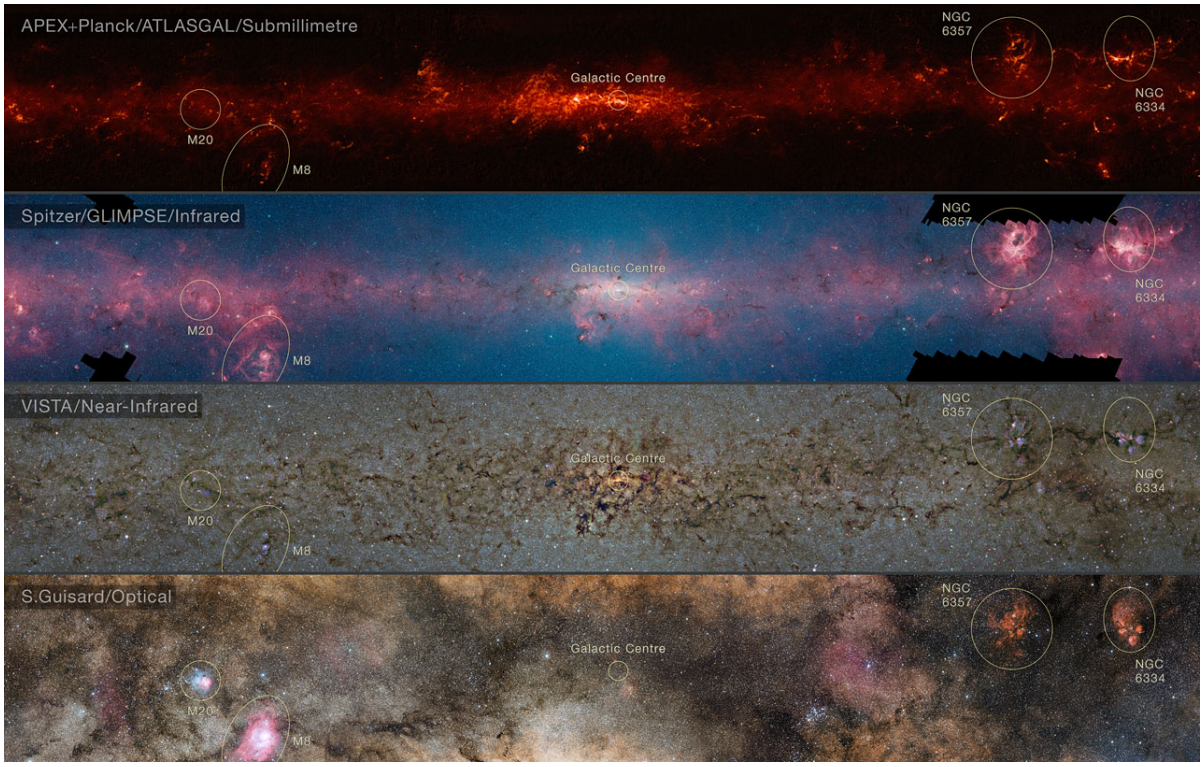


Figure 1.1: Multi-wavelength images of the central part of the milky way. Extracted from information@eso.org (*Comparison of the central part of the Milky Way at different wavelengths (annotated)*)

Machine learning (ML) is a data science tool that has proved to be a great tool in astronomy. Deep learning is a highly automated subset of ML. Over the decades, deep learning neural networks have had a significant impact on computer vision, more so image recognition and classification (Khan et al., 2020). These techniques are aiding in automating data handling pipelines and reducing on-time costs compared to if manual alternatives were to be considered. Deep learning has been used in classifying morphologies of radio sources. For example, software such as Classifying Radio sources Automatically with Neural networks (CLARAN) was developed to aid users in the automatic identification of

radio sources by using radio images with their infrared counterparts (Wu et al., 2019). These methods have also been used for identifying and classifying radio galaxies, transients, and some other astrophysical objects (Lukic, De Gasperin and Brüggen, 2019). A convolution neural network (CNN or ConvNet) is a deep learning algorithm that has successfully classified images, including those of astrophysical objects. In this work, we developed a classifier for classifying radio sources from Co-Ordinated Radio 'N' Infrared Survey for High-mass star formation (CORNISH) survey. The classifier has been developed using Branch-Convolution Neural Network (B-CNN), a hierarchical CNN type.

1.1 CORNISH Survey

CORNISH was a $\sim 5GHz$ radio continuum blind survey of the inner galactic plane, targeted at probing compact ionized gas (Hoare et al., 2012). It was part of a multi-wavelength survey of the inner galactic plane, coordinated with the Spitzer GLIMPSE survey. CORNISH was a two-part survey, CORNISH North and CORNISH South. CORNISH North covered the North galactic plane at longitude $(l) 10^{\circ} < l < 65^{\circ}$ and latitude (b) $|b| < 1°$, with a resolution of 1.5" . CORNISH South covered the South galactic plane at $295° < l < 350°$ and $|b| < 1°$, with a resolution of 2.5". The CORNISH North observations were by the Karl G. Jansky Very Large Array (VLA) in B and BnA configurations, with a sensitivity of < 0.4 mJybeam^{-1}. CORNISH South was by the Australian telescope compact array (ATCA). The primary focus of the CORNISH survey was to study the formation of massive stars. At $\sim 5GHz$, the free-free emission is less obscure, allowing for the probing of ionized gas at all depths. These make the surveys sensitive to detecting bright radio objects such as ultra-compact HII (UCHII) regions, planetary nebulae (PN), ionized winds from evolved massive stars, non-thermal emission from active stars, active galactic nuclei, and radio galaxies.

Objects detected from the CORNISH North survey have been identified and classified. The first source catalog was created and published by Purcell et al. (2013). They identified 3062 sources of above 7σ signal-to-noise. Objects identified in this catalog include HII regions, UCHII regions, PN, evolved stars, active binaries, radio lobes from external galaxies, AGN, and quasars (Purcell et al., 2013). Kalcheva et al. (2018) and Irabor et al. (2018) further worked on the CORNISH database, cataloging UCHII regions and PN, respectively. From Kalcheva et al. (2018) work, they identified 239 UCHII regions validated by observational properties and calculated spectral index. These results showed $3-4$ less UCHII in the galactic plane than previously thought, forming a

new test model for massive star formation models. Irabor et al. (2018) worked on analyzing and verifying PN in the catalog. This catalog is available for public access on the CORNISH website (cornish.leeds.ac.uk/public/index.php). Visual classification of the CORNISH South catalog is an ongoing classification.

1.2 Galactic Surveys

Faint Images of the Radio Sky at Twenty-Centimeters (FIRST) and Sydney University Molonglo Sky Survey ((SUMSS). Over the decades, there has been an increasing number of galactic plane surveys across the electromagnetic spectrum, particularly in the radio continuum. The Westerbork Northern Sky Survey (WENSS) surveyed the Northern sky at a wavelength of 92cm, with a resolution of 54" and positional accuracy of 1.5" (Rengelink et al., 1997). The (NVSS) complimented WENSS, surveying the Northern Sky at 1.4GHz. There are more upcoming surveys, such as SKA. This has been due to the need for multi-wavelength information, which is necessary for population studies of astronomical objects. Moreover, tremendous technological growth has been a critical driving factor in these developments.

Our radio sky is dominated by distant and extra-galactic radio sources giving a viewing window into the much earlier universe. This makes the radio continuum suitable for mapping large-scale structures of the universe and the evolution of galaxies. Stars are some of the key components of galaxies. Thus their formation and evolution highly influence the dynamics of the host galaxies. The CORNISH motivation was to study the formation of massive stars (Hoare et al., 2012). Some of the surveys that covered the CORNISH survey region are illustrated in the table (1.1). In the context of this work, our interest is limited to the new generation of wide-field surveys targeted at mapping the interstellar medium (ISM) of our galaxy. These were surveys probing diffuse and non-thermal sources. Additionally, understanding compact and thermal radio sources from a galactic perspective is needed, hence the CORNISH survey's complementary contribution.

Table 1.1: Galactic plane surveys covering regions covered by the *CORNISH* Survey. Extracted from (Hoare et al., 2012)

Survey	**Wavelength**	l **Coverage**	b **Coverage**	**Probe**
IPHAS	$H\alpha$	$30° < l < 210°$	$\|b\| < 5°$	Nebulae ,stars
UKIDSS	JHK	$-2° < l < 230°$	$\|b\| < 1°$	Stars, Nebulae
VVV	$ZYJHK$	$-65° < l < 10°$	$\|b\| < 2°$	"
GLIMPSE	$4-8\mu m$	$-65° < l < 65°$	$\|b\| < °1$	Warm Dust
MSX	$8-21\mu m$	All	$\|b\| < 5°$	"
MIPSGAL	$24,70\mu m$	$-65° < l < 65°$	$\|b\| < 1°$	Cool Dust
AKARI	$50-200\mu m$	$All \;\; sky$		"
Hi-GAL	$70-500\mu m$	All	$\|b\| < 1°$	"
JPS	$450,850\mu m$	$10° < l < 60°$	$\|b\| < 1°$	"
ATLASGAL	$850\mu m$	$-60° < l < 60°$	$\|b\| < 1.5°$	"
BOLOCAM	$1100\mu m$	$-10° < l < 90°$	$\|b\| < 0.5°$	"
GRS	$^{13}CO1-0$	$18° < l < 56°$	$\|b\| < 1°$	Molecular Gas
MMB	$6.7GHz$	$-180° < l < 60°$	$\|b\| < 2°$	Methanol Masers
HOPS	$22GHz$	$-180° < l < 60°$	$\|b\| < 2°$	Water Masers
OH	$1.6GHz$	$-45° < l < 45°$	$\|b\| < 3°$	Hydroxide Masers
S/V/CGPS	$21cm$	$-107° < l < 147°$	$\|b\| < 1.3°$	Atomic Gas
MAGPIS	$20cm$	$5° < l < 48°$	$\|b\| < 0.8°$	Diffuse Ionized Gas
MGPS-2	$35cm$	$-115° < l < 0°$	$\|b\| < 10°$	"

The Galactic Legacy Infrared Mid-Plane Survey Extraordinaire (GLIMPSE) was the first Spitzer mid-infrared galactic survey, carried over three phases (Churchwell et al., 2009). GLIMPSE *I* covered the galactic plane in the region of $10^{\circ} < l < 65^{\circ}$ and, with a spatial resolution of 2" operating at wavelengths $3.4\mu m$, $4.5\mu m$, $5.8\mu m$ and $8.0\mu m$. It was designed for studying star formation regions and warm interstellar dust. The Spitzer Multiband Imaging Photometer Galactic Plane Survey (MIPSGAL) covered the GLIMPSE region at $24\mu m$ and $70\mu m$ with a resolution of 6" and 18" respectively (Carey et al., 2009). This survey was designed to provide long-wavelength complementary data to the GLIMPSE survey, sensitive to MYSO (Massive Young Stellar Objects) and interstellar dust. In addition, the Herschel Infrared Galactic Plane Survey (Hi- GAL) also probed embedded sources across six bands, with our particular interest in $70\mu m$ (Molinari et al., 2016). Completing the infrared surveys is the UKIRT (United Kingdom Infrared Telescope) Infrared Deep Sky Survey (UKIDSS), working in the JHK bands (Lucas et al., 2008). UKIDSS covered the region $-2° < l < 230°$ and $|b| < 1°$, mapping stars and nebulae. More surveys for multi-wavelength data covering CORNISH region are the Wide Field Infrared Survey Explorer (WISE) (Yan et al., 2013), ATLASGAL (Schuller et al., 2009) and VVV (Minniti, 2015).

Incorporating data from the above-discussed surveys alongside CORNISH reveals the differ-

ent morphological features, color, and environment features required to classify the different radio sources. HII region exists in complex and clustered environments, and PN is isolated. PN and HII regions are strong IR emitters, thus cross-matching CORNISH sources with the mid-infrared (MID) surveys. The GLIMPSE and WISE surveys will help discriminate them from other radio emitters such as radio stars and other extra-galactic sources, which are not prominent at this particular wavelength. ATLASGAL, 850μm band, traces cool dust in molecular clouds. Sub-millimeter surveys help in discerning PN from HII, which have counterparts in the far-infrared (FIR). For example, in image (1.2), HII regions are bright sources in the far-infrared. They are detected as a bright source in the MIPSGAL 24μm images. These make multi-wavelength data a vital tool for characterizing and classifying radio.

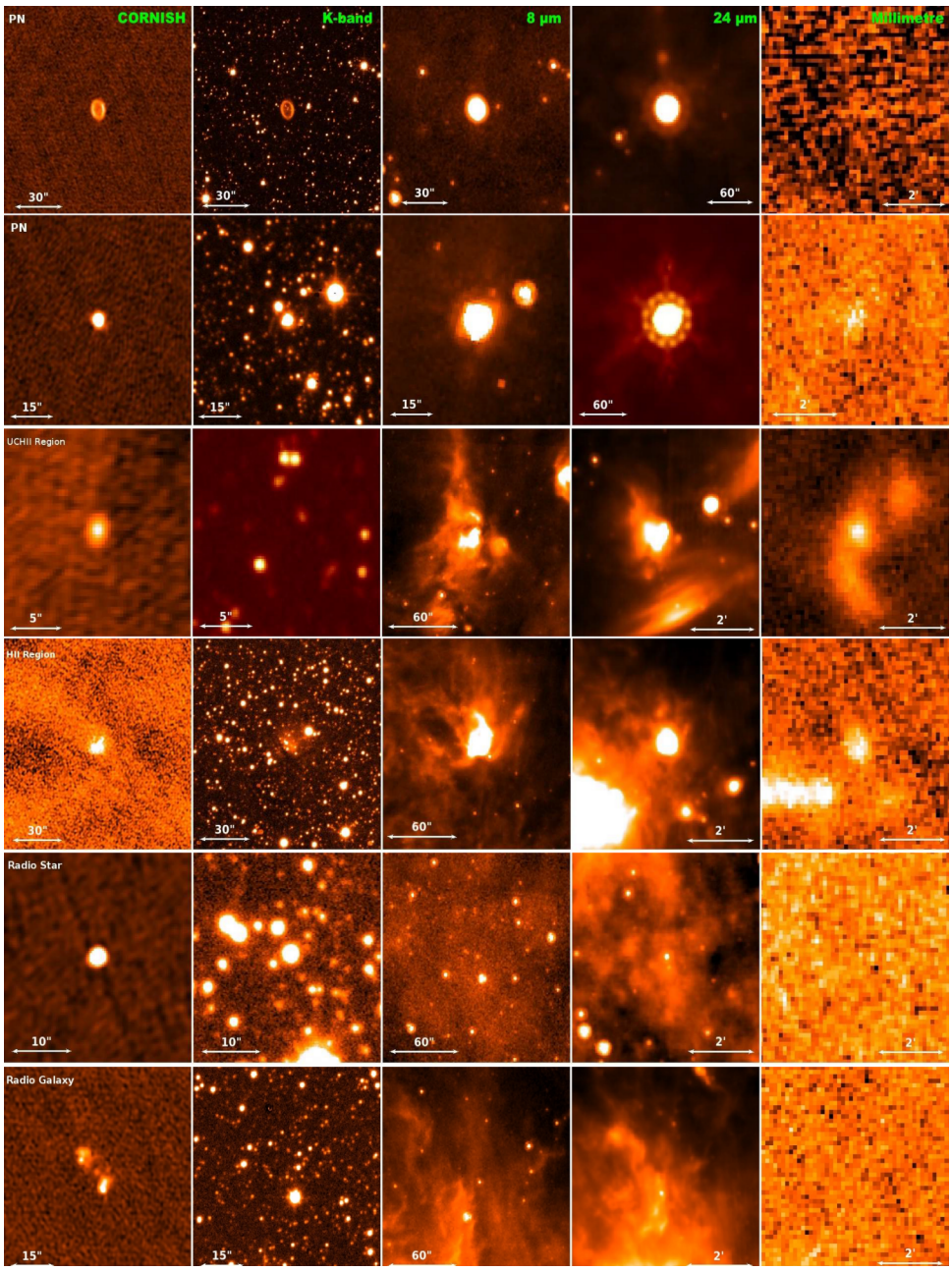


Figure 1.2: Sample *CORNISH* images alongside GLIMPSE and MIPSGAL. Extracted from (Purcell et al., 2013) .

1.3 Deep Learning with Astronomy

The ability of deep learning methods to automatically learn a representation of features in data is being explored in astronomy. Different ML techniques are being implemented to automate the classification of astrophysical objects. Some work was done by Arsioli and Dedin (2020), using ML methods incorporating multi-frequency data to classify blazars. They were successful in their en-devour up to an accuracy of $\sim 93\%$, proving the viability of ML for identifying and classifying astronomy sources. Some other work has been done using ML to identify multi-wavelength counterparts of sub-millimeter

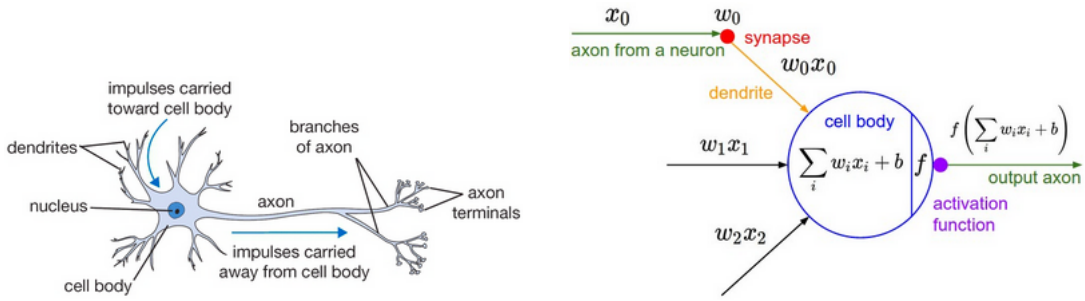
galaxies (An et al., 2018) and classify Transient Radio Frequency Interference (Czech, Mishra and Inggs, 2018).

There have been several research on using CNN to classify radio sources, e.g., Tang, Scaife and Leahy (2019) worked on re-using elements of existing machine learning models to classify radio galaxies. Lukic et al. (2019) tested the performance of CNN to capsule networks in classifying unresolved Fanaroff-Riley Class I (FR-I)and Fanaroff-Riley Class II (FR-II) morphologies. Capsule networks use capsules of grouped neurons containing spatial and spectral information of the image features. CNN outperformed capsule networks to a precision of 94.3% compared to 89.7%. This further verifies why CNN based model is currently the most preferred ML algorithm for classification purposes.

Chapter 2

Machine Learning

Artificial intelligence (AI) is an ongoing computer science-based research that is aimed at imparting and studying the intelligent behavior of artificial objects (Mou and Jin, 2018). It involves enabling machines to acquire some aspect of human-like intelligence and be able to perform given tasks. The Complexity of human behavior makes it challenging to define the specificity of intelligence in AI. Symbolism intelligence, using symbols to operate schemes, was the basis on which earlier AI was built (Garnelo and Shanahan, 2019). It was successful in natural language processing and other symbol-based processes such as logistic inferences. Unfortunately, it had shortcomings in mimicking common sense and performing non-symbolic operations. In this regard, researchers ventured into different natures of intelligence. Connectionism, commonly referred to as artificial neural network (ANN), was developed. This was biologically inspired by neuron connection in the human brain. Neurons are the basic functioning units of the brain. They are interconnected into a circuit by complex microscopic links called synapses. The brain works by each neuron receiving signal link through their dendrites and outputting it at an axon, which connects to dendrites of other neurons through synapses. This is achieved through a series of mathematical transformations illustrated in figures (2.1).



(a) Illustration of the brain's neuron

(b) Mathematical illustration of an active biological neuron.

Figure 2.1: Figures of a cartoon representation of a neuron alongside its mathematical description. Sourced from *CS231n Convolution Neural Networks for Visual Recognition*

Similarly, ANN works by taking in inputs and performing transformations to output their weights. It is the basis on which current AI is built. ML is currently one of the most advanced AI techniques. It is based on using intelligent neurons for statistical learning. Intelligent algorithms have been implemented in everyday applications on most modern websites such as Amazon, Facebook, Netflix for movie recommendation e.t.c. Some of the most common modes of ML are :

- Supervised learning. This involves using available labeled features for training, which is then used to predict labels of similar unlabeled data set. Supervised learning can either be a regression for a continuous dataset or classification for categorical variables. CNN is one of the highly developed methods for supervised classification.

- Unsupervised learning. This particular technique learns the features of unlabeled variables for predictive purposes. It is widely used for natural language processing.

- Reinforcement learning. It works almost similar to supervised learning but is sequential and uses unknown defined labels with reward incentives awarded for a given predictive sequence.

2.1 Building Blocks of CNN

2.1.1 Convolution layer

The convolution layer is the basic building block of CNN, from which the name originates. This layer comprises a kernel applied to smaller regions of an image, sampling pixels to create a smaller representative image called a feature map. The size of the resulting feature map is determined by

kernel size, padding, and stride (Khan et al., 2020). Kernels are matrices of varying sizes used for edge detection, blurring, sharpening images and e.t.c. In CNN, an image of $i(i = 1, 2...\infty)$ channels requires specification of kernel depth of size i. Mathematically, convolution is considered as an integral of a product of two functions $g * f$. In which for discrete cases, convolution

$$G(p,q) = f * g(p,q) = \sum_{k=1}^{m} \sum_{j=1}^{n} g(p-j, q-k) f(j,k) \tag{2.1.1}$$

Where $G(p,q)$ is the convolution function at p,q. g is the kernel (masking) function applied to an image f of size $m \times n$. j,k are row and column positions.

Equation (2.1.1) translates to a matrix transformation of the form (Saha (2018) and "Convolution and Applications")

$$\underbrace{\begin{bmatrix} x_1 & 1 & 1 & 1 & x_3 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & x_2 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ x_4 & 1 & 1 & 1 & x_5 \end{bmatrix}}_{Image(5\times5)} \times \underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}}_{Filter\ \ matrix} = \underbrace{\begin{bmatrix} (x_1+x_2-2) & 0 & (-x_3-x_2+2) \\ 0 & 0 & 0 \\ (-x_4-x_2+2) & 0 & (x_5+x_2-2) \end{bmatrix}}_{Feature\ \ map\ \ (Convolved\ \ features)} \tag{2.1.2}$$

Stride is the number of steps a matrix shifts through an image. In equation (2.1.2), stride= 1, for stride = 2, the convolved matrix becomes

$$\begin{bmatrix} x_1 & 1 & 1 & 1 & x_3 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & x_2 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ x_4 & 1 & 1 & 1 & x_5 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} (x_1+x_2-2) & (-x_3-x_2+2) \\ (-x_4-x_2+2) & (x_5+x_2-2) \end{bmatrix} \tag{2.1.3}$$

Comparing the feature map in equations (2.1.2) and (2.1.3), you can see that some parts of the matrix are truncated in equation (2.1.2). This is because kernel size, with the specified stride size, does not entirely fit the image. This can be corrected by choosing a stride size that allows the kernel

to cover the entire image; like in this case, stride size $= 2$ would be suitable, or padding the image for stride $= 1$. Padding is adding an extra outer layer of zero values on the matrix, such that

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & x_1 & 1 & 1 & 1 & x_3 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & x_2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & x_4 & 1 & 1 & 1 & x_5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & -1 \\ 0 & (x_1+x_2-2) & 0 & (-x_3-x_2+2) & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & (-x_4-x_2+2) & 0 & (x_5+x_2-2) & 0 \\ -1 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1.4}$$

From equation (2.1.4), it is evident that less of the image is lost compared to considering the non-padded approach of using stride$= 2$. This implies that bigger strides require bigger padding sizes to generate better representative image outputs. Multiple convolution layers are used in models. First layers extract low-level features such as color, edge e.t.c, and more added layers extract high-level features. The depth of kernel used should also be equal to the number of channels in a dataset. The size of the resulting feature map is

$$Feature\ \ map\ \ size = \frac{Image\ \ size + 2 \times Pad\ \ size - kernel\ \ size}{Stride\ \ size} + 1 \tag{2.1.5}$$

The depth of a feature map is equal to the number of convolution layers applied. Weighted sums from a convolution layer are then assigned to the activation function.

2.1.2 Activation Functions

The activation function introduces non-linearity in a feature map to produce an activation map. Non-linearity activates different patterns on a feature map facilitating learning of image features. Activation is an element-wise operation and doesn't alter size of a feature map (Mou and Jin, 2018; Khan et al., 2020). They play a crucial role in the performance of a deep learning model and determines its computational efficiency. There are a lot of available activation functions depending on the targeted problem.

Linear activation function takes in weighted inputs and linearly transforms them by using equa-

tion.

$$f_{li}(x) = Ax \tag{2.1.6}$$

Where A is a constant, and x is the weighted parameter of a neuron. $f_{li}(x)$ is linearly proportional to input x and has a constant derivative. This makes the function suitable for feed-forward operation but ineffective for backpropagation. Backpropagation is the essence of neural networks. It involves fine-tuning the weights of a neural network based on the error rate obtained in the previous iteration (further discussed in section 2.2). Also, the linear activation function can't handle complex and highly varying input parameters. Non-linear activation functions have overcome these challenges. Sigmoid is a non-linear function that transforms parameter values to range between 0 and 1, by

$$f_{sig}(x) = \frac{1}{(1+e^{-x})} \tag{2.1.7}$$

The vanishing gradient problem challenges this function. In that, there are no improved accuracies for extreme (very high and low) values. As the computed gradient approaches zero, the model stops learning. Tanh function improves on this by scaling up the sigmoid function such that

$$f_{tan}(x) = 2f_{sig}(2x) - 1 = \frac{2}{(1+e^{-2x})} - 1 \tag{2.1.8}$$

$f_{tan}(x)$ is zero centered, in that its output ranges between 1 and -1. In comparison to sigmoid, it is better performing but computationally expensive for complex models. Rectified linear unit (ReLU) comes into play to ease the computation cost. It imitates biological neurons activation and only activates features with positive values, such that

$$f_{re}(x) = \{^{x,if\ \ x>0}_{0,otherwise} \tag{2.1.9}$$

Inactivated neurons die during backpropagation since they are not updated. This causes poor performance of a model. Updated versions of ReLU have addressed this challenge. Leaky ReLU corrects for inactivated neurons by

$$f_{Le\ re}(x) = \{^{x,if\ \ x\geq 0}_{0.01x,otherwise} \tag{2.1.10}$$

Which makes negative values non-zero, thus activated. Parameterized ReLU tackles dead neurons challenge by introducing a gradient parameter to the negative side, such that

$$f_{pa\,re}(x) = \{^{x,if\ \ x\geq 0}_{ax,otherwise} \tag{2.1.11}$$

a is a learnable parameter. This makes it an all-encompassing function of ReLU and Leaky ReLU features. It can be fixed to 0.01 or 0, to obtain equations 2.1.10 and 2.1.9, respectively. ReLU is currently the most preferred activation function.

Softmax is another widely used activation function. It is a generalization of logistic regression that takes in a vector of scores and outputs a vector of probabilities. It is a combination of multiple sigmoid functions and takes the form.

$$f_{so}(x)_i = \frac{e^{x_i}}{\sum_{i=1}^{k} e^{x_i}} \quad for \quad i = 1, \ldots, k \tag{2.1.12}$$

Where k is the total number of classes, and i is the i^{th} class. Thus, a classification involving three categories will have a vector output of the percentage of their probabilities. This function is often used at the end of a neuron for multi-class classification, whereas sigmoid is used for binary cases.

2.1.3 Pooling layer

The pooling layer downsizes and only extracts vital information from an activation map. It reduces on spatial information of images (Khan et al., 2020). There are different statistical techniques for pooling, such as max pooling, average pooling, and some pooling. Maxpooling involves using filters of given sizes and sample features with maximum values, sum pooling samples by summing up features, whereas average pooling samples average values. Using 2×2 filters, with stride$= 2$, a mathematical illustration of these processes are

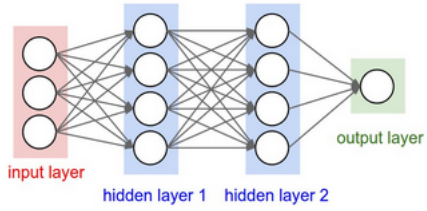
$$\begin{bmatrix} 2 & 4 \\ 1 & 2 \end{bmatrix} \xleftarrow{max\ \ pooling} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 2 \end{bmatrix} \xrightarrow{average\ \ pooling} \begin{bmatrix} 1.25 & 2 \\ 0.25 & 0.75 \end{bmatrix} \tag{2.1.13}$$

$$\downarrow$$

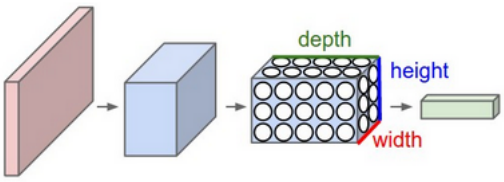
$$\xrightarrow{sum\ \ pooling} \begin{bmatrix} 5 & 8 \\ 1 & 3 \end{bmatrix}$$

2.1.4 Fully Connected layer

After pooling, the feature map is flattened into a vector. The fully connected (FC) layer tunes weighted parameters from the activation map to generate the probability likelihood of given labels. It is fully connected because it takes input from the activation map. It is generated from all the hidden layers of convolution, activation, and pooling layers, forming a neuron(see figure (2.2)). Each neuron is representative of specified class labels. The output of a neuron is passed through an activation function to output labels with the highest probability as classification results. At this stage, softmax or sigmoid function is often used. CNN is encompassed multiple layers of these neurons. Each neuron is representative of the activated feature. Figure (2.2a) shows a neuron structure that takes in three channels of a two-dimensional image. Passes it through two hidden layers and outputs its probability. Figure (2.2b) shows the transformation phases of the image across different layers.



(a) A neuron composed of two hidden layers.



(b) Three-dimensional input image, transformed through a three-layer neural network in figure (2.2a).

Figure 2.2: These are an illustration of processes involved in a three-layer neural network. Sourced from *CS231n Convolution Neural Networks for Visual Recognition*

For effective classification, multiple layers of the above-discussed layers are used, forming a deep neural network. Deep neural networks have a lot of learnable parameters and efficient for computer vision (Khan et al., 2020).

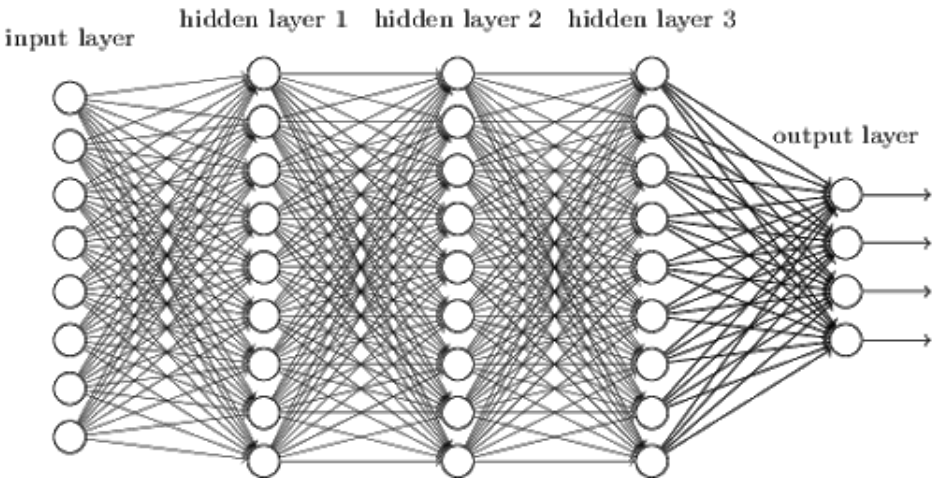


Figure 2.3: Illustration of FC layers of a deep neural network for multi-class classification. sourced from *Fully Connected Layers in Convolutional Neural Networks*

The general structure of CNN is illustrated in figure (2.4):

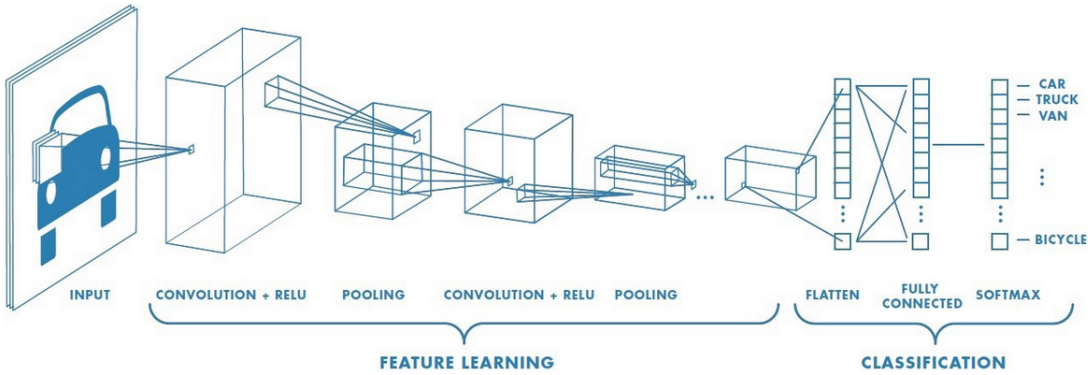


Figure 2.4: CNN Network architecture. Sourced from Saha (2018)

2.2 Model Learning

In this phase, CNN is used to model the dataset. Weights are first randomly initialized at the start of the learning phase. Initialization involves assigning input parameters into a CNN. Input parameters comprise data sets and hyperparameters, which are trainable parameters given outside the model. Network forward-pass these parameters across hidden layers and outputs their weights. Data is split into training and validation sets. The training set is used to learn the weights, whereas the validation set evaluates the trained model's performance. For image classification, CNN uses images. It takes $3-$dimensional images as $2-$dimensional array of pixels characterized by their resolutions from which it extracts image depth. In CNN architecture, depth is the number of channels, width and height are image dimensions. A gray-scale image of pixel sizes 32×32, is interpreted as $1 \times 32 \times 32$.

Whereas an RGB image of similar size is $3 \times 32 \times 32$. Learnable parameters are obtained by summing up all features. For example, an RGB image will result in $3072(3 \times 32 \times 32)$ learnable parameters for the input layer. For the proceeding hidden layers, the number of learnable parameters is outputs of previous layers. For example, they consider an input feature $z \in \mathbb{R}^n$, where n is the number of neurons. The output of a neuron is computed by: Computing the weighted sum of each dimension of z. The resulting weights $w \in \mathbb{R}^n$ are the parameters of the neuron. A bias can be introduced into the weighted sum by adding a term $b \in \mathbb{R}^n$ as in equation (2.2.1). An activation function is then applied to the weighted sum as described by equation (2.2.2)

$$x = w^T z + b \tag{2.2.1}$$

$$y = f(x) \tag{2.2.2}$$

whereby for a linear activation function

$$x = f(x)$$

This is similar to equation (2.1.6), since the scaling factor A can be absorbed during weight computation as in equation (2.2.1); A process illustrated in figure (2.5).

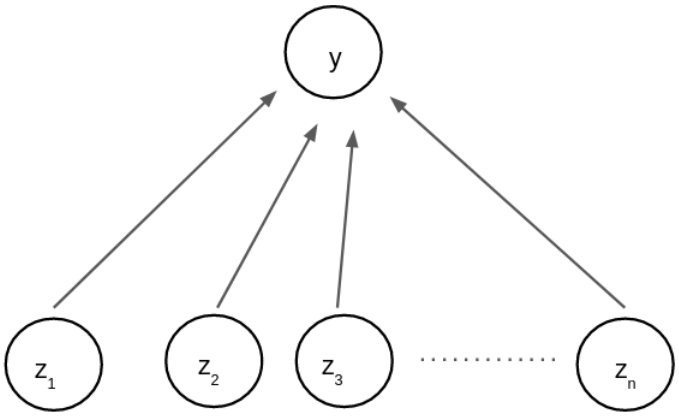


Figure 2.5: Schematic illustration of computation of a neuron output, modified from Mou and Jin (2018).

Outputs are evaluated by the use of a defined loss function to gauge the performance of the

weights. Loss functions quantify how far off is, the correct result from the actual output of a model. There are a lot of available loss function algorithms. Mean square error (MSE) is the mean square difference between the true label and the actual predicted truth. It is suitable for regression-based problems and defined as

$$F_{mse} = \frac{1}{2N}\sum_{i}^{N}\left|y^i - t^i\right| \tag{2.2.3}$$

Where N is the number of data points, y^i is neuron prediction, and t^i is the true label of the i^{th} data-point. Use of subscript indexing i.e. t^i denotes multinomial distribution adopted from Mou and Jin (2018). For classification-based problems, cross-entropy is often used. Mathematically, considering binary classification cross-entropy loss function(F_{ce}) for classifying label t , which is denoted as $t \in [0,1]$ is

$$F_{ce} = \frac{1}{N}\sum_{i=1}^{N}\left[-t^i \log p(y^i) - (1-t^i)\log(1-y^i)\right] \tag{2.2.4}$$

For multi-class classification $t^i_{id} \in \{1,2,\ldots,C\}$, where C is the number of categories. t_{id} is indexing representation i.e. representing the target as a vector, with each element indicating whether the data point belongs to this category $t^i_{one\ hot} \in \{0,1\}^C$. Whereby, *one hot* is one hot representation. In that, categorical data is numerically encoded data such that color targets can be encoded as in table (2.1).

Table 2.1: One-hot encoding of color variables.

Red	Green	Blue
1	0	0
0	1	0
0	0	1

Loss of each label is obtained independently per observations and summed up such that

$$F_{ce} = -\frac{1}{N}\sum_{i=1}^{N}\log y^i_{t^i_{id}} \tag{2.2.5}$$

Model complexities are controlled by introducing regularization. This involves introducing an additional auxiliary loss into the loss function. Lasso (Least Absolute Shrinkage and Selection Operator) and ridge Regression are some of the widely used regularization techniques. Lasso regression, referred to as $L1$, is a linear regression that uses shrinkage. It adds an absolute value of the magnitude of coefficient as penalty term to the loss function, i.e. $L1 = |\theta|$. Where θ is a vector of all

parameters, assuming that matrices are vectorized and concatenated. In this regard, Lasso shrinks the less important feature's coefficient to zero thus, removing some features altogether. Ridge regression technique, often known as *L2*, is similarly based on shrinkage but adds squared magnitude of coefficient as penalty term

$$L2 = |\theta|_2^2 = \sum_i \theta_i^2 \tag{2.2.6}$$

Therefore, the overall task of learning involves solving for loss function F defined by

$$F = F_{ce} + \lambda L_2 \tag{2.2.7}$$

With an additional term λ, which is a hyperparameter that keeps the two losses in balance. Learning is carried out by iteratively computing gradients of F until the function converges.

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial F}{\partial \theta_i} \tag{2.2.8}$$

α is the learning rate and adjust the steps at which gradients are updated . Previously, gradient $\frac{\partial F}{\partial \theta_i}$ was computed numerically by perturbing F with a small value ε such that as $\varepsilon \to 0$ the gradient

$$\frac{\partial F}{\partial \theta_i} = \frac{F(\theta_i + \varepsilon) - F(\theta_i - \varepsilon)}{2\varepsilon} \tag{2.2.9}$$

This method is ineffective for a large number of learnable parameters. Since it is computationally expensive when solved numerically. As a result, an analytical approach by use of backpropagation was adopted.

2.2.1 Backpropagation

It is an algorithm that analytically calculates gradient $\frac{\partial F}{\partial \theta_i}$. Backpropagation recursively iterates through a network. Calculating gradient for all layers, starting from the output layer. For a MSE, taking a single data point, equation (2.2.3) becomes

$$F_{mse} = F = \frac{1}{2}|y - t|^2 \tag{2.2.10}$$

$$\frac{\partial F}{\partial y} = y - t \tag{2.2.11}$$

$\frac{\partial F}{\partial y}$ is recursively solved, such that lower activation values become inputs for upper layers, denoted as z. Thus, $\frac{\partial F}{\partial z}$ needs to be computed. Using equations (2.2.1), (2.2.2) and chain rule

$$\frac{\partial F}{\partial x} = \frac{\partial F}{\partial y} \cdot \frac{\partial y}{\partial x} \tag{2.2.12}$$

$$\frac{\partial F}{\partial x} = w^T \frac{\partial F}{\partial y} \tag{2.2.13}$$

$$\frac{\partial F}{\partial w} = z^T \frac{\partial F}{\partial x} \tag{2.2.14}$$

$$\frac{\partial F}{\partial b} = w^T \frac{\partial F}{\partial y} \tag{2.2.15}$$

Backpropagation terminates once equations (2.2.13), (2.2.14) and (2.2.15) are computed for all layers. This algorithm is linear and following, multiplicative and additive rules. Gradients of shared weights are added or multiplied depending on the nature of connections in layers. As a consequence, this might result in huge gradients causing explosion or vanishing of gradients. An optimizer updates weights of computed gradients to minimize loss.

2.2.2 Optimization

For gradient-based learning, the overall learning process is illustrated in figure (2.6). The terms cost and loss functions almost refer to the same meaning. But loss function mainly applies for a single training set compared to the cost function, which deals with a penalty for several training sets or the complete batch. In short, the cost function is an average of loss functions from multiple groups of training.

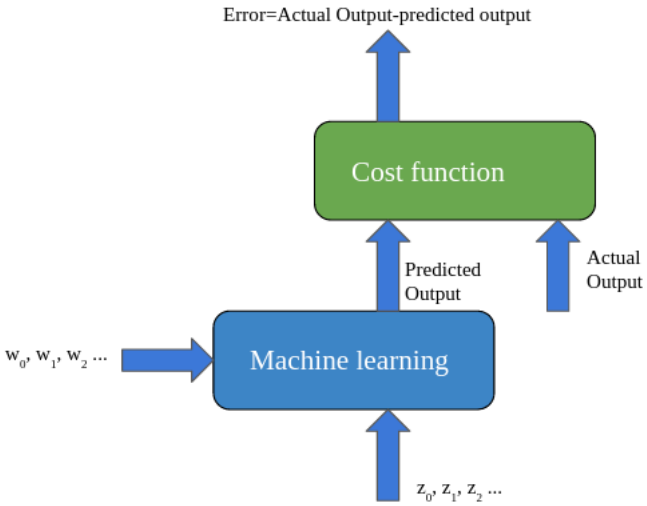
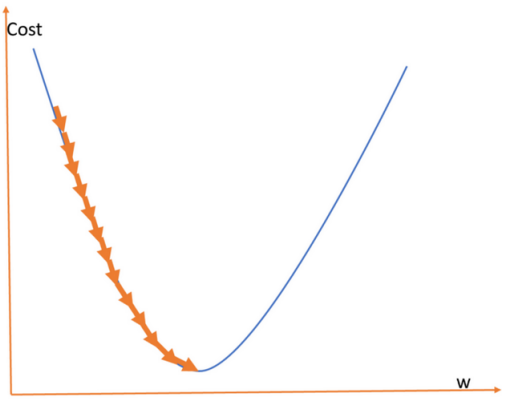
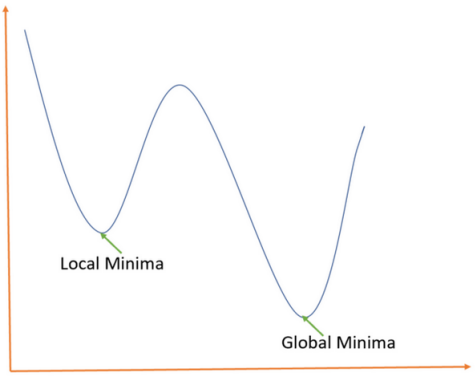


Figure 2.6: Depiction of a gradient descent learning method. Extracted from *Overview of different Optimizers for neural networks | by Renu Khandelwal | Data Driven Investor | Medium*

In this approach, an optimizer iteratively updates weighted parameters with computed gradients as in figure (2.7a). Parameters are updated in regards to negative gradient direction, as defined by equation (2.2.8). But real-life scenarios are complex, often resulting in noisy weights (see figure (2.7b)). Local and global minima characterize a Valley-like gradient path. The optimizer might sometimes get stuck at local minima, confusing it as global minima, producing an under-representative model. This is avoided by choosing an optimizer that suits your network. Learning is successful when global minima are reached.



(a) Depiction of gradient descent update of a gradient.



(b) The figure shows different valley points along with gradient update that confuses an optimizer. The global minimum is the actual convergence point, whereas the local minimum is the minimum of a given sample set.

Figure 2.7: Illustration of optimizer walking along with a loss function, updating the weight parameters gradient. Sourced from Khandelwal (2020)

There are three main types of gradient descent optimizers.

- Full-batch gradient descent - Updates gradient after all weights have been calculated. This is not effective for deep networks with a large number of learnable features.
- Mini-batch gradient descent - It samples gradient across loss functions of specified weight sizes, called batch size. It is computationally cost-effective and is suitable for GPU implementation.
- Stochastic gradient descent (SGD) - The algorithm works by updating gradients by randomly picking on the calculated loss functions of a given weight sample. It iterates through the whole selection, updating for the loss function of each weight. It is currently one of the widely used optimizers. This increases the number of updating iterations required for convergence, increasing learning time. Frequent update of weights also leads to wide fluctuations in the loss function.

Momentum was introduced to dampen the oscillations resulting from fluctuating loss function. It speeds up convergence by curving an optimizer towards the most sloping direction. It can be used with the above-discussed optimizers. Besides, there are other well-performing optimizers, such as the adaptive moment estimation (Adams) optimizer. It is currently the most preferred optimizer and works by the adaptive learning rate.

2.2.3 Batch Normalization

Deep neural networks are sensitive to initialized training parameters (Khan et al., 2020). During training, deep hidden layers may fluctuate for mini-batch updates (Ioffe and Szegedy, 2015). This causes the cost function to take forever to converge. Batch normalization is an algorithm that standardizes layer inputs per mini-batch, accelerating on a model convergence.

2.2.4 Dropout

Deep networks have a tendency of over-fitting when training on few datasets. Overfitting is when a model learns a data set too well because the model structure is too complex for the given data set. Underfitting is the opposite and results when the specified data set is complex for the model. Dropout layers help in regularizing the number of layers. Dropout randomly drops out layers to reduce the network depth.

2.3 B-CNN

To improve the performance of CNN models, some features can be modified to get desired results. Increasing number of filters and use of non-linear activation improve model accuracy. However, these are limited within the boundaries of a network. For more efficient models, hierarchy based CNN models were developed. These models incorporate the use of hierarchical architecture in the already existing CNN model blocks. Tree-Based CNN (TB-CNN) is a hierarchical based CNN which incorporated use of sub-tree kernels for convolution(Mou and Jin, 2018). Hierarchical Deep CNN (HD-CNN) model is designed such that, it first splits data into two parts, coarse and fine details. It then first classifies classes based on coarse features before proceeding to finer features for finer classification (Yan et al., 2015). Ma et al. (2019)designed TB-CNN based encoder to classify morphologies of radio AGN's, comparing performance to use of visual inspection. This process is time-consuming since it requires pre-training and tuning of coarse and fine training parameters (Zhu and Bain, 2017). B- CNN is a hierarchical based CNN, in that it has multiple network layers along with the main convolution layer which contains hierarchy information of different classes. CNN does not take account of hierarchy in classes and treat each data exclusively and only has a single flat output layer. B-CNN is implemented by use of branch training strategy (BT-Strategy), where low-level parameters are first activated thus learning coarse features before proceeding to activation of high-level parameters learning the finer features. It is a step-by-step kind of training, in which data is ordered into general- to-specific classes. This top-down approach type of classifier network architecture of B-CNN is illustrated in figure (2.8), (Zhu and Bain, 2017).

Feature of a CNN model can be modified to improve its performance. Increasing the number of filters and the use of non-linear activation improve model accuracy. However, these are limited within the boundaries of a network. For more efficient models, hierarchy-based CNN models were developed. These models incorporate the use of hierarchical architecture in the already existing CNN model blocks. Tree-Based CNN (TB-CNN) is a hierarchical-based CNN that uses sub-tree kernels for convolution (Mou and Jin, 2018). The hierarchical Deep CNN (HD-CNN) model is designed to splits data into two parts, coarse and fine details. It then first classifies classes based on coarse features before finer features for more refined classification (Yan et al., 2015). Ma et al. (2019) designed TB-CNN based encoder to classify morphologies of radio AGN's, comparing performance to visual inspection performance. This process is time-consuming since it requires pre-training and

tuning of coarse and fine training parameters (Zhu and Bain, 2017). B- CNN is a hierarchical-based CNN in that it has multiple network layers and the main convolution layer, which contains hierarchy information of different classes. CNN does not account for hierarchy in classes, treats each data exclusively, and only has a single flat output layer. B-CNN is implemented using branch training strategy (BT-Strategy), where low-level parameters are first activated, thus learning coarse features before activating high-level parameters learning the finer features. It is a step-by-step kind of training in which data is ordered into general- to specific classes. This top-down approach type of classifier network architecture of B-CNN is illustrated in figure (2.8), (Zhu and Bain, 2017).

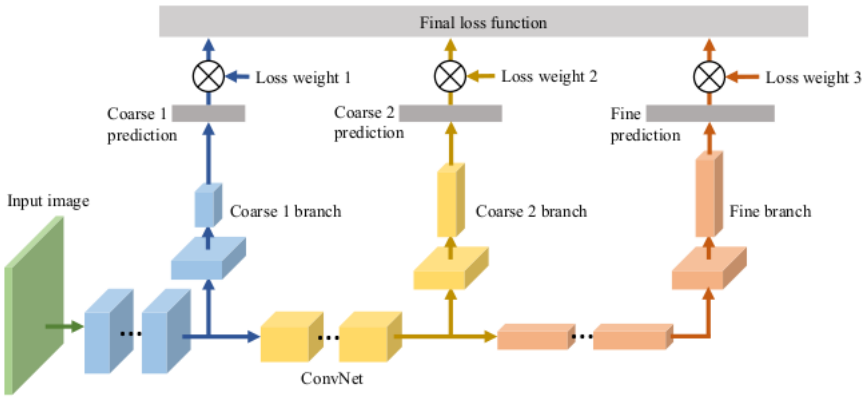


Figure 2.8: B-CNN Network architecture, with three hierarchy levels. Source (Zhu and Bain, 2017)

Chapter 3

Methodology

The primary motivation for this work was to develop a classifier for sources in the CORNISH survey. This has been achieved by exploiting the hierarchical properties of CNN, developing the B-CNN network structure discussed by Zhu and Bain (2017). Neural networks have been trained and tested on CORNISH North sources incorporating IR counterparts. The developed model is based on the PyTorch algorithm (PyTorch, 2021). This is a low-level pythonic machine learning algorithm suitable for small classification models. It gives users a lot of freedom in implementing and developing neural networks, with a broad community for support. The functioning model was then used on the CORNISH South sources. For the whole process, the following tools were used: anaconda libraries (using Python 3.7), PyTorch(GPU version), torch, *sklearn*, pandas, PIL, and other basic python modules. Developing a classifier involves:

- Data pre-processing
- Defining Model network architecture(B-CNN network architecture)
- Training the model
- Model evaluation
- Prediction (Implementing the trained model)

3.1 Data processing

Data is the main backbone of a CNN network and determines the performance of a model. This work involved using a multi-wavelength dataset from radio and IR. From the CORNISH North catalog,

2631 were selected as trainable sources. These were fits files of varying image sizes. Some of the cutouts are from tiles with partial beam coverage at the edge of the survey coverage. They are referred to as near the edge of the survey and have empty pixels of regions outside the survey beam coverage. In this work, this cutouts will be referred to as ´near the edge surveys´ (see table (3.1)). Statistics of IR counterpart are in table (3.1) obtained from Irabor et al. (2018) and Hoare et al. (2012). All images were of 2 arcsecondss angular size.

There were some sources available in the CORNISH North catalogue but missing in some given channels. These are presented in the table (3.1) as missing sources (M_1) obtained by $M_1 = \frac{N_i}{N_{CORNISH}} \times 100\%$, where N_i is the number of missing sources in a given channel i and $N_{CORNISH}$ is the total number of sources in CORNISH North. On the other hand missing M_2 are missing sources relative to ATLASGAL, $M_2 = \frac{N_i}{N_{Atlasgal}} \times 100\%$. This was necessary because there was a high percentage of missing ATLASGAL sources in comparison to CORNISH North. Hence we needed to evaluate models with input data set available in ATLASGAL. Specified image sizes are of single dimensions.

Table 3.1: Statistics of the multi-wavelength dataset.

Survey	Wavelength	Spatial resolution (arcseconds)	Image size (Pixel size in Single dimensions)	Total number of survey sources selected for training	No. of near the edge surveys	Percentage of Missing Sources (M_1)	Percentage Missing sources (M_2)
CORNISH North	$6cm$	1.5	601 to 603	2631	15	0.0	0.0
MIPS24	$24\mu m$	6	70 to 148	2621	0	0.38	0.45
MIPS70	$70\mu m$	18	35 to 92	2593	0	1.44	1.53
GLIMPSE(I_3)	$5.3\mu m$	2	149 to 302	2631	0	0.0	0.0
GLIMPSE(I_4)	$8\mu m$	2	130 to 302	2631	0	0.0	0.0
UKIDDS K	k ($2.20\mu m$)	0.8	601 to 895	2573	9	2.2	2.16
Hi-GAL	$70\mu m$	10	80	2582	32	1.86	0.4
Hi-GAL	$500\mu m$	35	23	2579	32	1.98	0.54
ATLASGAL	$850\mu m$	19	43	2224	10	15.46	0.0

3.1.1 Data pre-processing

The above-downloaded sources were grouped into folders with source names. One of the most vital things in machine learning is data quality. As a start, near the edge surveys were identified and

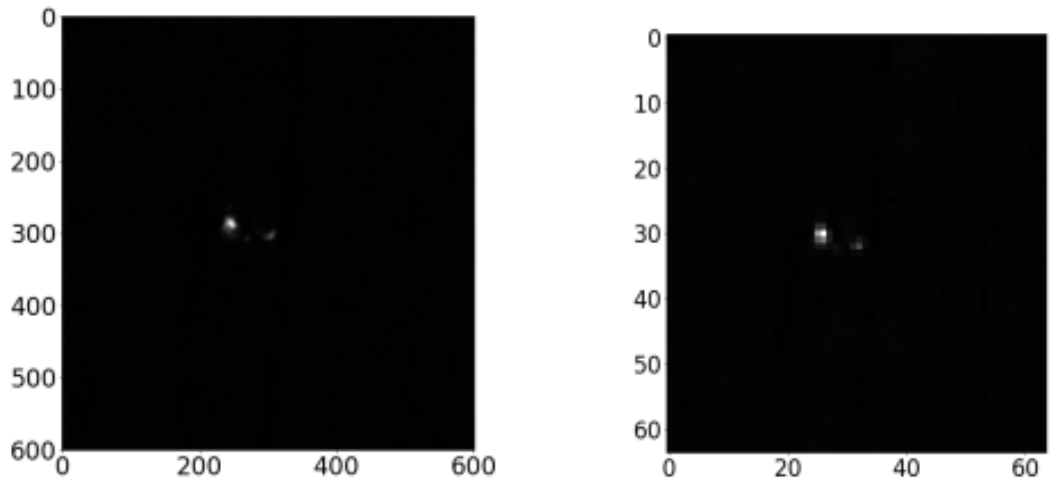
processed as discussed in subsection (3.1.1.1). Images from the MIPS70 channel were found to be corrupted and discarded. Pixel values substantially vary across the different channels. For example, a data point has a minimum and maximum values of -17.467087 and 6348.166, respectively, for the MIPS24. Similar data-point has a minimum and maximum values of -0.0016523073 and 0.0028295908 for the CORNISH North channel. In this context, data-point refers to an image of the same object (source point). Big differences in an image escalate variations in training features, making it hard for a model to learn. This makes it necessary for data points to be scaled for stabilized learning. Images were normalized by (Tang, Scaife and Leahy, 2019):

$$X_{i_{new}} = \frac{X_i - X_{min}}{X_{max} - X_{min}} \tag{3.1.1}$$

X_i is the original pixel, $X_{i_{new}}$ normalized pixel, X_{min} is the minimum pixel and X_{max} is the maximum pixel of the image. Normalization reduces values to range between 0 and 1 . The normalized images were then scaled by

$$X_{new} = AX_{i_{new}} \tag{3.1.2}$$

Where $A = 255$ is constant. Re-scaled images were re-sized to 64×64 (see figure (3.1)). These processes reduce dynamic range in the images suppressing low-level features considered as noise. After which they are converted to Python Imaging Library (PIL) images. Except for Hi-GAL($500\mu m$), which was resized to 32×32 due to its smaller size. PIL is a module for processing images in python (Lundh and Clark, 1995; Clark, 2014). It was deprecated in 2011 and replaced with Pillow, which is compatible with the newer versions of Python (version 3 onward). Pillow is built on top of PIL, thus supports similar operations. This module supports a wider variety of image formats, including JPEG, PNG, HDF5 e.t.c.



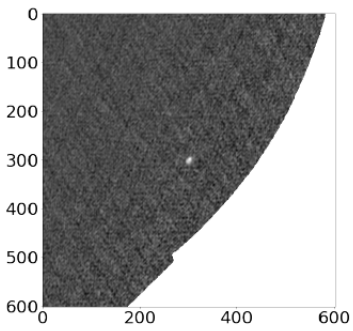
(a) Original image

(b) Normalized and scaled image.

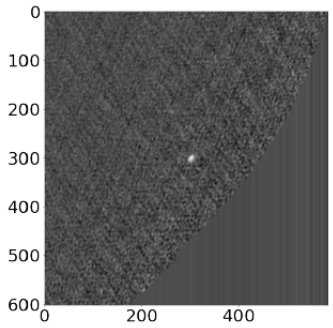
Figure 3.1: An image of UCHII region from CORNISH survey (figure (3.1a)), normalized and scaled in (in figure (3.1b))

3.1.1.1 Near the edge survey

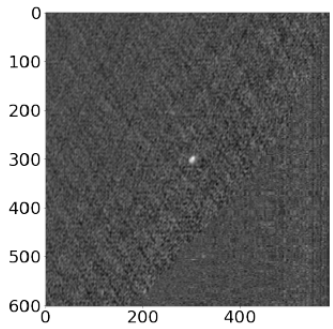
Available catalogs had a limited data set, making it vital to use most available images. Near-the- edge-survey images have bad pixels, which compromise the performance of a classifier. This can be rectified by correcting for bad pixels up-to-given threshold. Python has some already existing algorithms that aid in imputing these values (Badr, 2019). In this work, SimpleImputer and KNNImputer algorithms were tested on imputing images with up to 50% bad pixels. Images in figure (3.2) are results of techniques of imputing missing values. Figure (3.2a) is an original near the edge survey image. Figure (3.2b) is an illustration of figure (3.2a) imputed by the use of SimpleImputer. This algorithm entails filling missing values using constant values or statistics such as mean, median, or mode. In this particular case, the constant value used was mean. Figure (3.2c) is an image imputed using the k-Nearest neighbors approach. KNNImputer algorithm entails using values of nearest neighbors to impute values of empty pixels. The distance limit to the nearest neighbor was arrived at by trial and error. Bigger distances would result in pixels being imputed by far-off values. Comparing outputs of the two algorithms, KNNImputer with a specified distance of n_neighbors=3 was better fitting.



(a) Original near the edge survey image



(b) Image imputed using SimpleImputer



(c) Image imputed using k-Nearest Neighbors approach.

Figure 3.2: Imputed images alongside the original image of a PN from the CORNISH survey.

3.1.1.2 Missing channels

When uploading input data, the model expects to find each image of specified channels. However, as pointed earlier, some image counterparts were missing. It would have been easier to discard these sources, but there were a limited number of available sources. Therefore, empty images of size 64 were created to fill in for the missing channel images. Created images were as in figure (3.3).

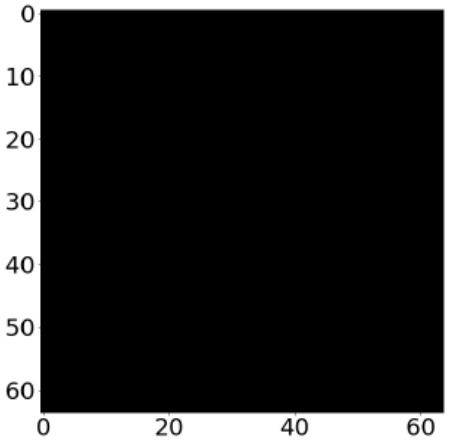


Figure 3.3: Empty image for filling in missing channel images.

3.1.2 Training and testing dataset

CNN uses labeled data. Allowed labels from the CORNISH North catalog are presented in table (3.2). Class 1, Class 2, and Class 3 are levels for the different hierarchy levels of the classifier. For example, class 1 is for the first branch of the B-CNN, Class 2 and Class 3 are for the second branch and fine branch, respectively. Nebulous categorization are sources with cloud-like features in

the radio wavelength, whereas non-nebulous are distant point-like sources. This was adopted from Neilson (2018) work. The labeled hierarchy structure is illustrated by the figure (3.4), indicating level class.

Table 3.2: Classes of the cataloged CORNISH North Sources

Sources	Class Labels		
	Class 1	**Class 2**	**Class 3**
Dark HII Region	Nebulous	All HII Regions	HII Region
Diffuse HII Region			
HII Region			
UCHII			UCHII
PN		PN	PN
Galaxy	Non-Nebulous	IR-Quiet	Extra-Galactic Sources
Radio Galaxy (Central Source)			
Radio Galaxy (Lobe)			
Radio Star		IR-loud	Radio Star

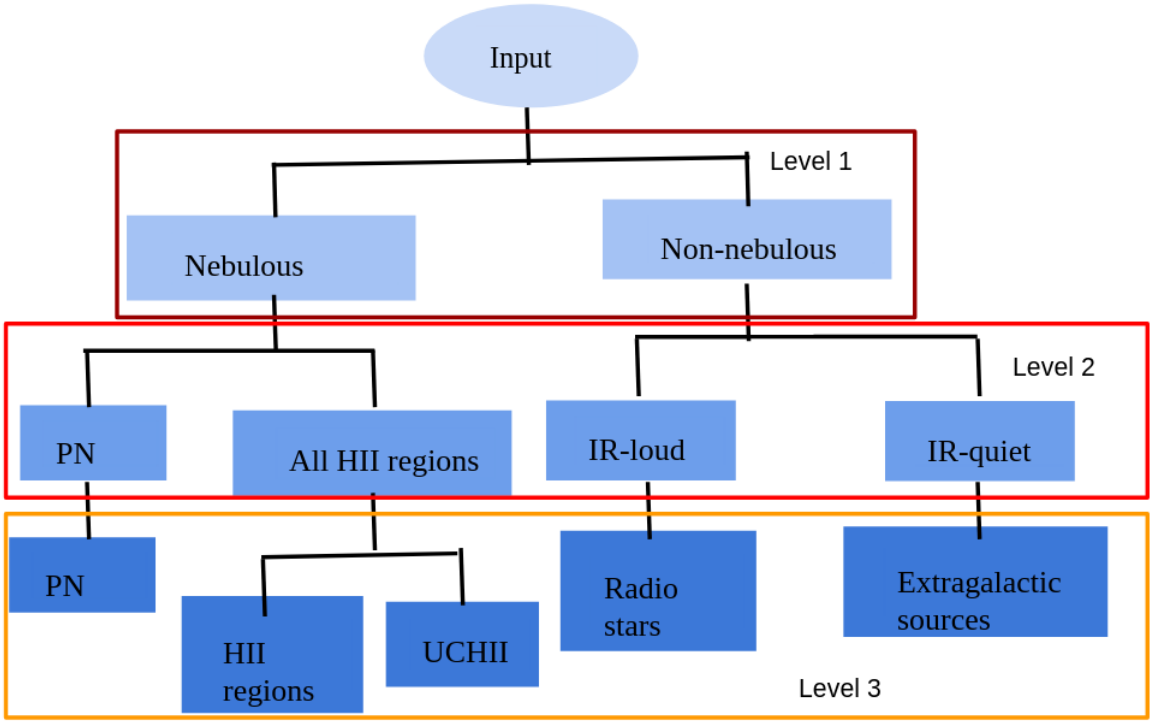


Figure 3.4: Hierarchy structure of the classifier.

The number of trainable label heads of the reference channel is tabulated in the table (3.3), whereby Training 2 are available sources regarding ATLASGAL.

Table 3.3: Statistics of sources in the CORNISH North dataset.

Source Type	Training 1	Training 2
Extra-galactic Sources	2043	1669
UCHII	239	234
PN	170	158
Radio-Star	130	113
All HII Region	49	45

Data sets were split into training and test set, test size of 0.30 for each unique label of the dataset. 30% of each source type in the CORNISH North catalog was used as a test set and the remaining percentage as the training set. Machine learning algorithms only work with numerical values. Thus categorical values of labels have to be encoded into a numerical format. Encoded labels for each class was as presented in the table (3.4).

Table 3.4: Encoded class labels.

Class 1 labels	
Source	Encoded numerical value
Nebulous	0
Non-Nebulous	1
Class 2 labels	
PN	3
All HII Region	0
IR-loud	1
IR-quiet	2
Class 3 labels	
Extra-galactic Sources	0
HII Region	1
PN	2
Radio Star	3
UCHII	4

3.1.3 Dataloader

PyTorch is a tensor-based language. Hence data must be converted into tensors to be uploaded onto a network model . Often, many data sets can not fit in processor memories. These require data to be loaded in batches. You can opt to define your algorithm to accomplish these tasks or build on already existing modules. PyTorch has a class module, Dataloader that allows for automatic batching of data (Chilamkurthy, 2021; Pytorch, 2021). It is constructed by specifying the uploaded dataset,

batch size, and other additional instances. Such as shuffle-For specifying whether you want to shuffle data or not; sampler-Formulated for defining sample retrieval method, geared at balancing loading data of imbalanced classes; the num_workers-Enables defining number of processors. Dataloader works by fetching data from the specified dataset and feeding it into the training network in batches. Dataset is a PyTorch class that is used to load and convert data into PyTorch understandable format. Its customizable features were employed to develop a data loader, described in figure (3.5).

The loader takes in labels, PIL objects, channels, and the location of channel images. The module loops through, loading PIL objects and creating Objects for the missing channels. Channel objects are then loaded, coordinated by loaded CORNISH data. Every loaded object is transformed, with the specified transforms (Torch, 2017b; Torch, 2017a).

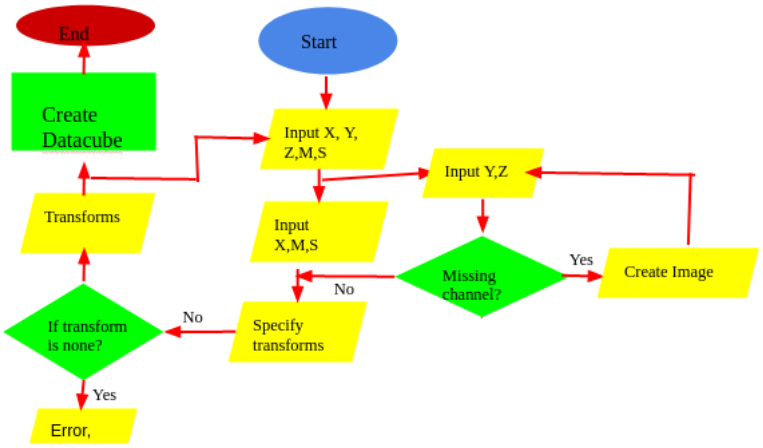


Figure 3.5: Algorithm flow of customized data loader, MyFitsDataset. Input X are Labels and PIL objects of specified reference channel(in this case, CORNISH). Y is a list of channels to load. Z is the path to the location of the Y images. M and S are the mean and standard deviation of the channels.

Rotations, tensor, resize, horizontal and vertical flip were used. Transforms are defined separately for training and test set. This is because training requires more feature variations for the model to learn on as many outlooks of a given scenario as possible. Which in turn improves on its performance capabilities.

Input mean and standard deviation were for standardization. Transformed tensors were standardized to centralize on feature distribution and reduce noise during training by

$$Image = \frac{Image - mean}{std}$$

The standardized tensors were then loaded into memory, from which Dataloader feeds into the

network. Finally, Dataloader outputs a rank four tensor entailed of batch size, channels, length, and image width.

3.2 B-CNN Model

The choice of neural network architecture highly influences the performance of a classification model. Therefore, a B-CNN network architecture in figure (2.8) was employed as the backbone of our classifier, with network flow defined in figure (3.6).

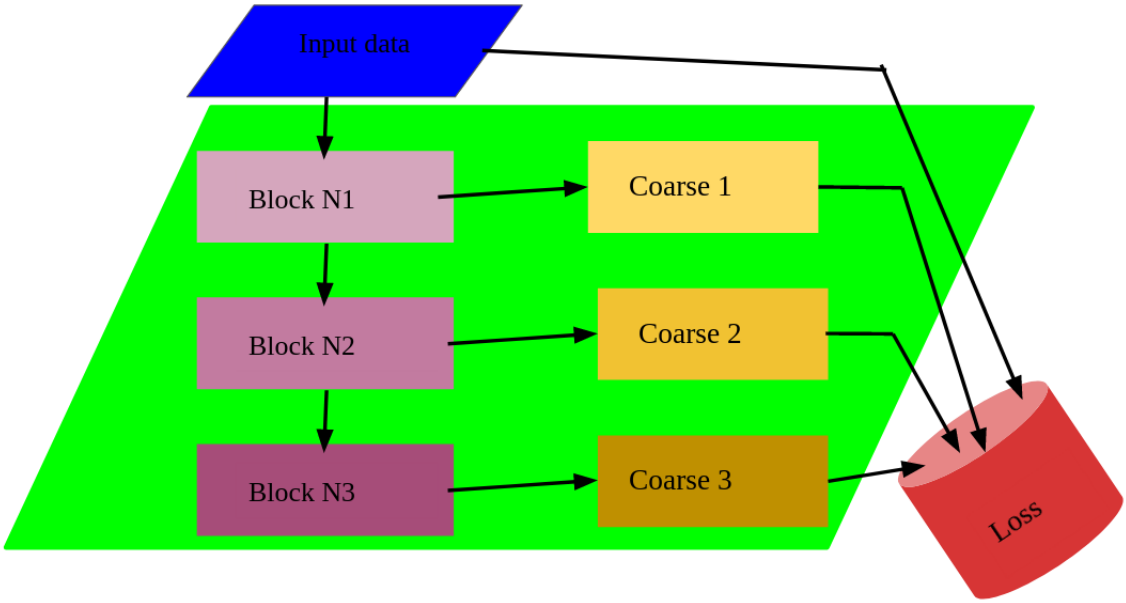


Figure 3.6: B-CNN layers define the region covered by the green parallelogram. Loss cylinder entails loss calculation.

Input data were rank four tensors from the dataloader. The region covered by the green parallelogram is depictive of layers defining B-CNN. Blocks N1, N2, and N3 comprise multiple sequentially connected Conv2 layers. In which, Conv2 layers are unit layers comprising convolution layers, ReLU, and batch normalization layers. Padding and kernel sizes of 1×1 and 3×3 respectively were used throughout the training. Coarse 1, 2, and 3 are blocks of fully connected layers comprising of multiple coarse layers. Coarse layers are entities of linear layer, ReLU, batch normalization, and dropout layer. From there, network structures in the table (3.5), (3.6) and (3.7) were developed. The number of hidden layers was influenced by Zhu and Bain (2017) work. Input1 are parameters obtained after max-pooling of block N1 whereas, Input2 are parameters obtained from max-pooling of block N3. A complete network of Block N1 and coarse 1 is a CNN network for level 1 classes, similar to levels 2

and 3 in figure (3.4).

Table 3.5: Network structure of a 3-level B-CNN, with 4 Conv2 layers in the second level.

Model 1		
Level 1	Level 2	Level3
Input	Input1	Input2
Conv2 (32)	Conv2 (192)	Conv2 (576)
Conv2 (32)	Conv2 (192)	Conv2 (576)
Conv2 (64)	Conv2 (576)	Conv2 (576)
Conv2 (64)	Conv2 (576)	Conv2 (576)
Maxpool2d	Maxpool2d	Maxpool2d
Flatten		
coarse (192*X*X)	coarse (1728*Y*Y)	coarse (4032*Z*Z)
coarse (192)	coarse (1728)	coarse (4032)
FC-2(Linear)	FC-4(Linear)	FC-5(Linear)
Softmax		

Table 3.6: Network structure of a 3-level B-CNN, with 3 Conv2 layers in the second level.

Model 2		
Level 1	Level 2	Level3
Input	Input1	Input2
Conv2 (32)	Conv2 (192)	Conv2 (576)
Conv2 (32)	Conv2 (192)	Conv2 (576)
Conv2 (64)	Conv2 (192)	Conv2 (576)
Conv2 (64)		Conv2 (576)
Conv2 (192)		
Maxpool2d	Maxpool2d	Maxpool2d
Flatten		
coarse (384*X*X)	coarse (960*Y*Y)	coarse (3264*Z*Z)
coarse (384)	coarse (1728)	coarse (3264)
FC-2(Linear)	FC-4(Linear)	FC-5(Linear)
Softmax		

Table 3.7: Network Structure of a 2-level B-CNN.

Model 3	
Level 1	Level 2
Input	Input1
Conv2 (32)	Conv2 (192)
Conv2 (32)	Conv2 (192)
Conv2 (64)	Conv2 (192)
Conv2 (64)	
Conv2 (192)	
Maxpool2d	Maxpool2d
Flatten	
coarse (384*X*X)	coarse (960*Y*Y)
coarse (384)	coarse (1728)
FC-2 (Linear)	FC-4 (Linear)
Softmax	

3.2.1 Loss function

Model learn by solving loss function. The loss cylinder in figure (3.6) computes loss. For the B-CNN structure, the loss is the sum of losses of all hierarchy levels. Each hierarchy level is a complete CNN. Summing up losses results in increased losses that cause the model to explode. This requires regulating summed up losses by defining a loss weight modifier, which helps in scaling updated losses. Loss weight modifier is a regularizing operator, giving the flexibility of tuning on levels of learning rates. It is a matrix of size equal to hierarchy levels (Zhu and Bain, 2017). For a 3-level depth, loss weight modifier=[α,β,γ], with boundary conditions, $\sum(\alpha,\beta,\gamma) = 1$ and $0 \leq \alpha,\beta,\gamma \geq 1$. Loss is obtained by

$$Loss \ = \alpha \times F_{ce}^{(1)} + \beta \times F_{ce}^{(2)} + \gamma \times F_{ce}^{(3)} \tag{3.2.1}$$

Where $F_{ce}^{(1)}$, $F_{ce}^{(2)}$ and $F_{ce}^{(3)}$ are cross-entropy loss functions for levels 1, 2 and 3, respectively, loss weight modifier dictates the contribution of each level to the final loss. For instance, $[0.1, 0.2, 0.7]$ implies that the model will focus on training fine features with minimal training on coarse features from levels 2 and 1. For $[1,0,0]$, the model will only activate the level 1 network and train on coarse features to contribute to final classification levels. From figure (3.6) and table (3.5), it is evident that when the proceeding levels (2 and 3) are not activated, parameters from level 1 are the contributing factors for the whole network. Thus equation (3.2.1) becomes

$$Loss \ = \alpha \times F_{ce}^{(1)}$$

Which is for label prediction for levels 1, 2, and 3. This applies to all the possible combinations of α, β and γ combinations.

3.2.2 Optimizer

In the earlier phases of training the classifier, an SGD optimizer was used. This was to give an insight into what to expect of this kind of network structure. Later on, Adams optimizer was incorporated.

Chapter 4

Model Training and Evaluation

The training was done in two phases, phases one and two. Phase one, referred to as pre-training, involved evaluating the network configuration. Phase two entailed fine-tuning the hyperparameters obtained from the pre-training phase. Loss and accuracy plots were used as diagnostic tools for evaluation purposes. These are mean losses and accuracies plotted against the number of epochs. Accuracy is obtained by

$$Accuracy = \frac{Number\ \ of\ \ Correct\ \ Preductions}{Total\ \ Number\ \ of\ \ Preductions} \tag{4.0.1}$$

Thus

$$Accuracy = \frac{True\ \ Positives\ \ +\ \ True\ \ Negatives}{True\ \ Positives\ \ +\ \ True\ \ Negatives + False\ \ Positives\ \ +\ \ False\ \ Negatives} \tag{4.0.2}$$

whereby:

True Positives are cases in which a model´s predicted object label as true is similar to the ground truth.

True Negatives are cases in which a model´s predicted object label as false is similar to the ground truth.

False Negatives are cases in which a model predicts an object label as false, but the ground truth is true.

False Positives are cases in which a model predicts an object label as true, but the ground truth is false.

An epoch is a complete forward and backpropagation of the whole dataset in the network. The

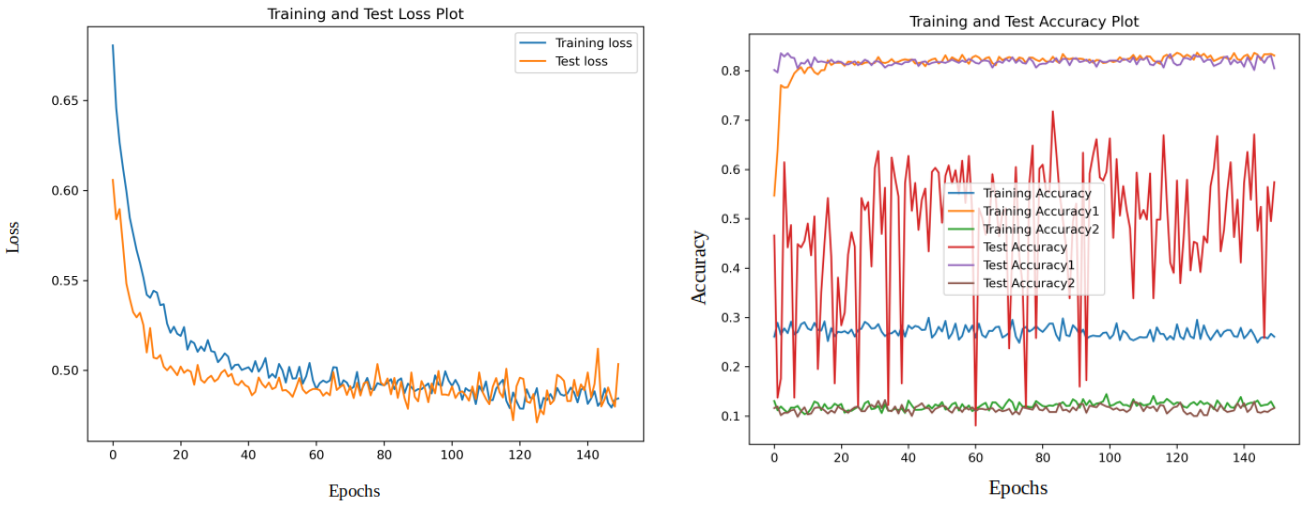
number of epochs and specified batch size influences how long a training session would take to complete. Train accuracy/loss plots show how well the model is learning. Test accuracy/loss are evaluation diagnostic tools of the prediction capability of the trained model.

4.1 Evaluation of model configurations

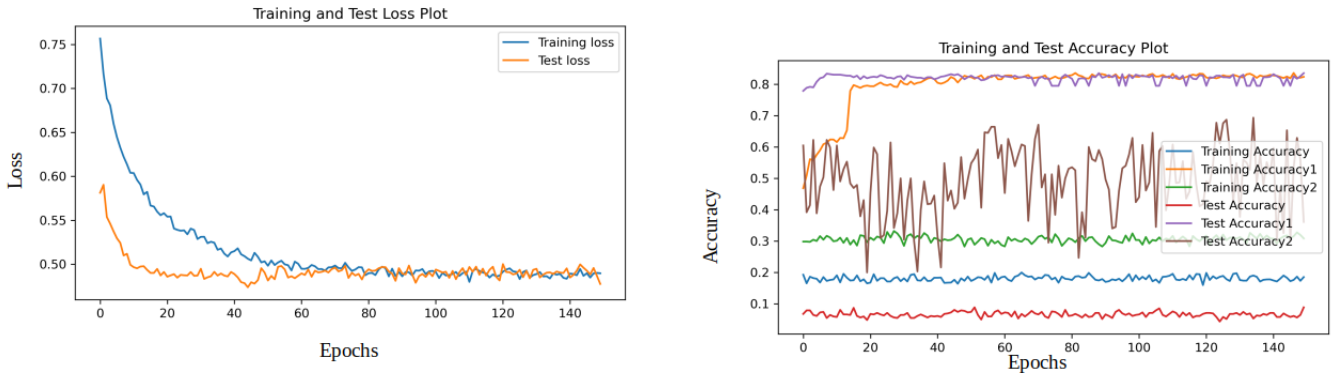
Evaluation of model configuration is a pre-training phase geared at helping ease parameter selection for training the classifier. Only CORNISH North sources resized to size 32 were used for training. Four GPUs from the ARC4 cluster were used, with training sessions ranging between 1 to 2 hours depending on the specified number of epochs. For the model setup, the 3-level model described in table (3.5) (model 1), SGD optimizer, specified loss, learning rate (lr), momentum, weight decay (decay), and loss weight modifier were used.

4.1.1 Batch normalization

Figure (4.1) is loss and accuracy plots showing the effects of batch normalization. The model was trained for 150 epochs, with the following sets of training parameters: lr= 0.001, decay10^{-6}, momentum= 0.8, batch size= 100, loss weight modifier= $[1,0,0]$. Choice of these parameters was random and influenced by Zhu and Bain (2017) work. Training and test loss plots in figures (4.1a) and (4.1b) show that batch normalization is essential for a model´s stability. This is seen by the more in-depth wiggly loss plot in figure (4.1a) compared to figure (4.1b). In figure (4.1a), the level two accuracy plot has more in-depth oscillations showing instability, whereas accuracy plots in figure (4.1b) are better defined. The instabilities result from changing the distribution of inputs to layers deep in the network layers as weights are updated after the mini-batch. Batch normalization standardizes inputs to these layers stabilizing the learning process. From this, it is conclusive that batch normalization is essential for regularizing this model, as discussed in subsection (2.2.3).



(a) Loss and Accuracy plots without Batch Normalization.



(b) Loss and Accuracy plots with Batch Normalization layer.

Figure 4.1: These are accuracy and loss plots showing the effects of batch normalization on the model performance. Graphs labeled Training Accuracy, Accuracy 1, and Accuracy 2 are training accuracy plots for level 3 (considered fine level), level 2, and level 1, respectively. Similarly, Test Accuracy, Accuracy 1, and Accuracy 2 are test accuracies for the given levels.

4.1.2 Sampler

The CORNISH dataset has an imbalance in class distribution, causing an imbalance of source distribution per mini-batch upload. This influences model training, more so causing asymmetric per class test accuracy. Pytorch dataloader has some existing algorithms that aid in customizing batch uploads. For example, a sampler is an algorithm that uses weighted probabilities for class sampling, Which introduces symmetry in the batch upload. Figure (4.2) shows the effects of incorporating a sampler in the model in figure (4.1b), with weighted sampling rate (W)

$$W = \frac{C}{N}$$

Where N is the total number of sources and C is the per class sources. The model is unstable compared to if random shuffling is used, as in figure (4.1b). Table (4.1), extra-galactic class (1532 training set and 511 test) has an accuracy of 9% compared to 71% for shuffle-based methods. It improved on accuracy classification of lower levels and smaller-sized classes but reduced for larger classes. Thus, as much as it improves on the classification of low-level classes, it seems to be introducing unpredictable biases. The table columns are correlated to the hierarchy schematic in figure (3.4), whereby Level refers to hierarchy levels and classes are the sub-branches in the different hierarchy levels.

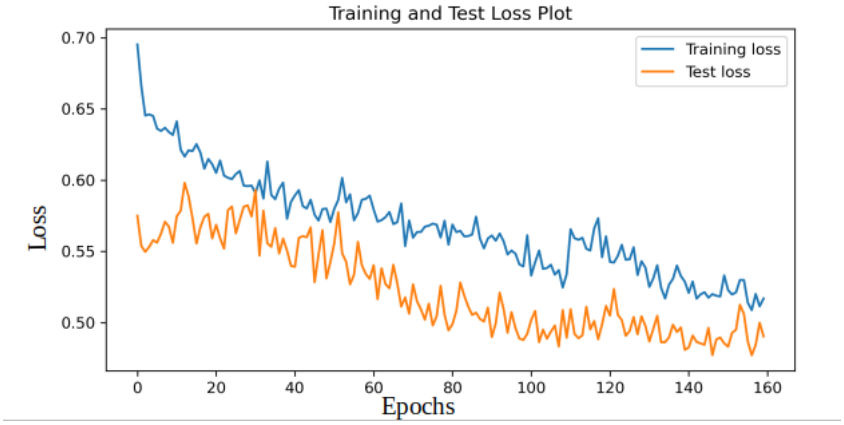


Figure 4.2: Loss and Accuracy plot of a model with symmetric class distribution per batch upload.

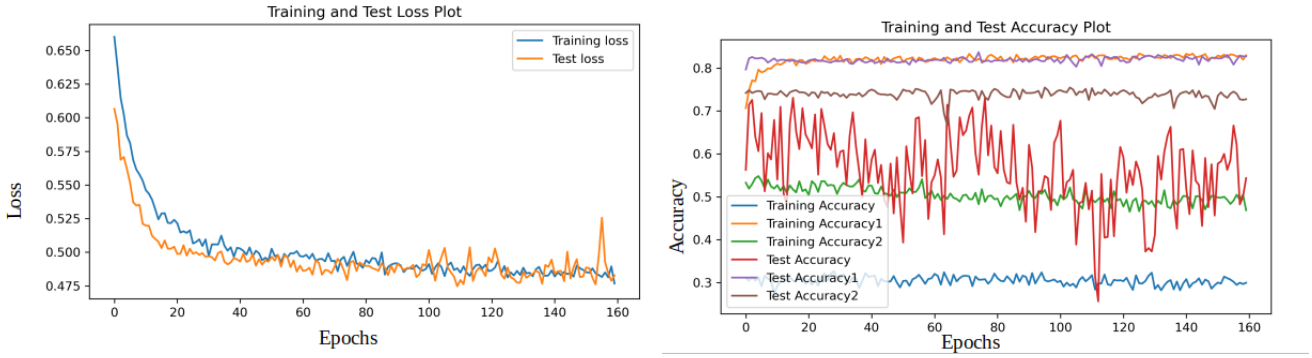
Table 4.1: Classification accuracy of a sampler-based model in figure (4.2) and shuffle-based model in figure (4.1b).

Level	Class	Classification accuracy(%)-Sampler	Classification accuracy(%)-Shuffle
1	Nebulous	24	5
	Non-nebulous	97	99
2	IR-loud	53	0
	IR-quiet	4	0
	All HII regions	31	62
	PN	5	0
3	Extra-galactic sources	9	71
	Radio stars	48	40
	PN	5	0
	HII regions	0.0	14
	UCHII regions	16	25

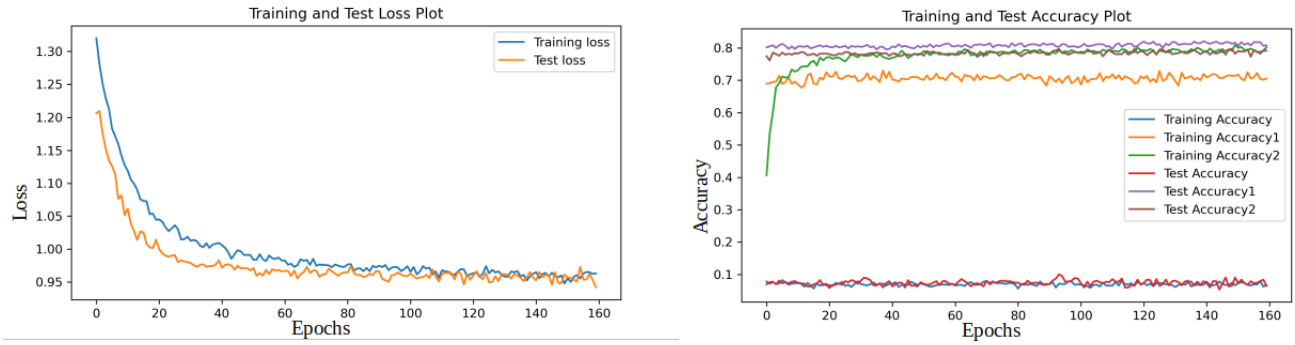
4.1.3 Loss weight modifier

Characterizing the influences of loss modifier on the model is vital. The model was tested for extreme values of α, β and γ, shown in figure (4.3). The loss plot in figure (4.3a) shows that the model

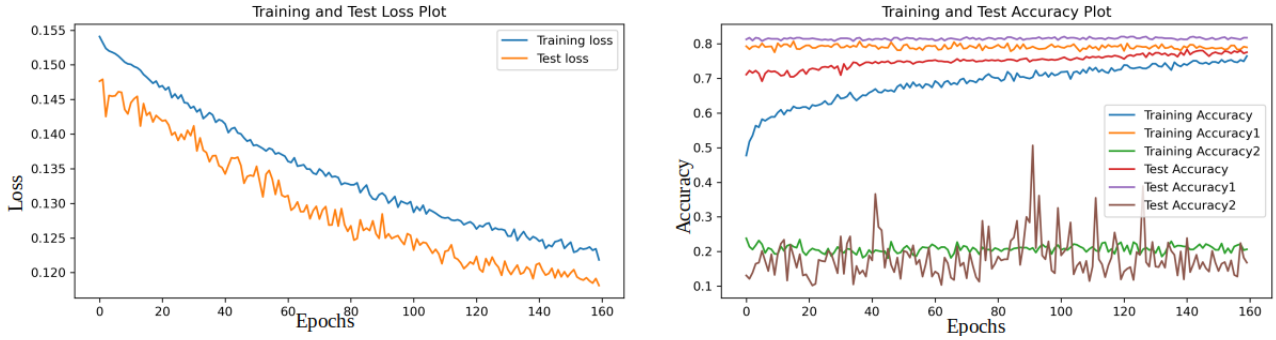
converges at a much earlier epoch of ~ 90. On the other hand, accuracy plots show that the model in figure (4.3c) has better level class performance (see table (4.2)) compared to models in figures (4.3a) and (4.3b). On further training for 250 epochs, the model in figure (4.3c) converges much later after 160 epochs. This implies that loss weight modifier$= [0,0,1]$, requires more training time in comparison to loss weight modifier$= [0,1,0]$ and loss weight modifier$= [1,0,0]$. Nevertheless, it leads to overall improved performance, compared to the other two, when further trained for more epochs. As discussed in (3.2.1), loss weight modifier $[1,0,0]$ implies $\alpha = 1, \beta = 0, \gamma = 0$, and is an input for the whole network and not only level 1. Thus columns 3, 4 and 5 in table (4.2) are outputs of 3 separately trained models, and not an output of a single model. In that, loss weight modifier=$[1,0,0]$ corresponds to model 1, loss weight modifier=$[0,1,0]$ to model 2, and loss weight modifier=$[0,0,1]$ to model 3.



(a) Loss and accuracy plots for weight=[$\alpha = 1, \beta = 0, \gamma = 0$]



(b) Loss and accuracy plots for weight=[$\alpha = 0, \beta = 1, \gamma = 0$]



(c) Loss and accuracy plots for weight=[$\alpha = 0, \beta = 0, \gamma = 1$]

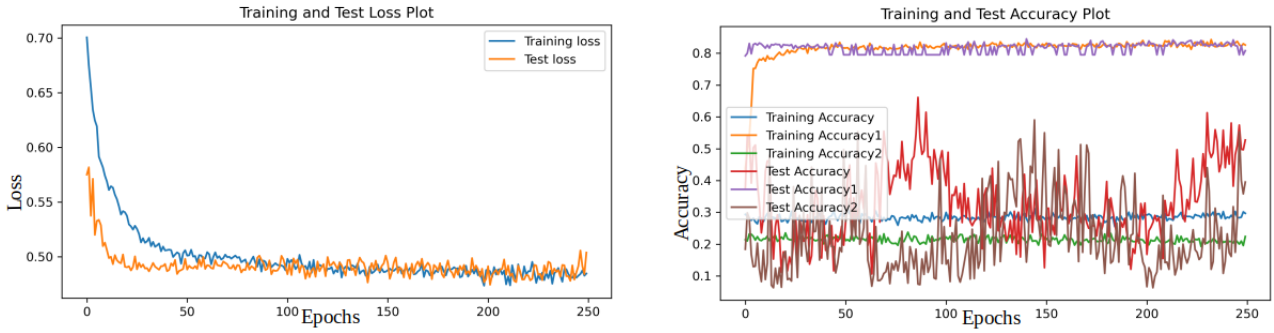
Figure 4.3: Effects of loss modifier. Training parameters:lr= 0.003, decay10^{-6}, momentum= 0.8, batch size= 100

Table 4.2: Per class classification accuracies for models in figure (4.3).

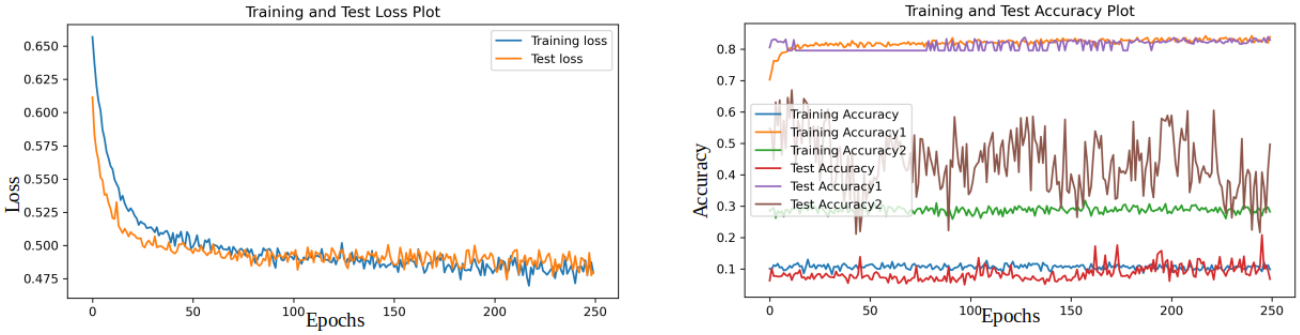
Level	Class	Classification accuracy(%, loss weight modifier=[1,0,0])	Classification accuracy(%, loss weight modifier=[0,1,0])	Classification accuracy(%, loss weight modifier=[0,0,1])
1	Nebulous	18	18	8
	Non-nebulous	99	97	99
2	IR-loud	95	98	90
	IR-quiet	0	0	0
	All HII regions	4	43	62
	PN	0	0	0
3	Extra-galactic sources	68	1	98
	Radio stars	0	0	0
	PN	3	0	5
	HII regions	14	43	0
	UCHII regions	24	32	38

4.2 Weight decay

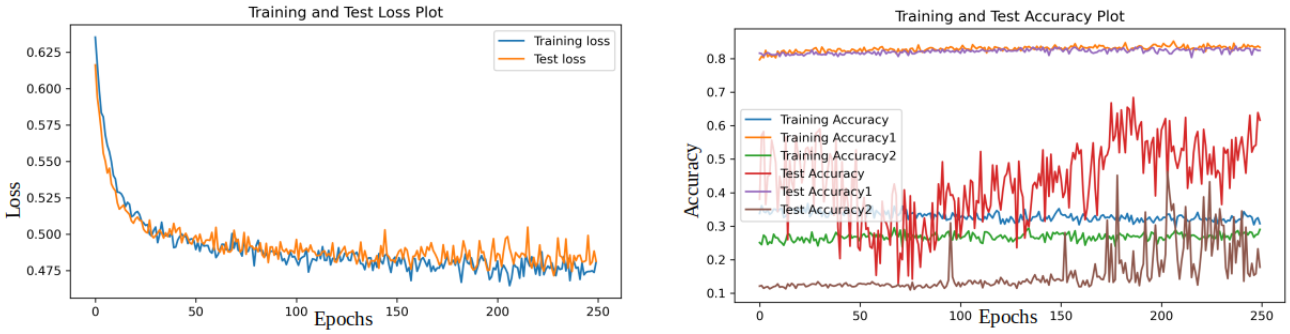
Figure (4.4) are loss and accuracy plots for varying weight decays. Loss plot for decay=10^{-12} converges ~ 60 epochs, and accuracies plots are finely defined. Whereas, for decay=10^{-6}, loss converges much more later at ~ 110, and accuracy plots show more complexity present in the model. Decay $= 10^{-6}$ has higher per class accuracies for the fine level classes. Similarly, decay$= 10^{-8}$ has higher per class accuracies for level 2 classes. This observation implies that much smaller values of decay regularize weight resulting in less complex models. However, this oversimplification reduces the model's classification capability. Thus, its value should be defined within a range of not too small to oversimplify and not too big to generalize, just enough to reduce some complexities. For this case, decay=10^{-8} and 10^{-10} would be a good fit.



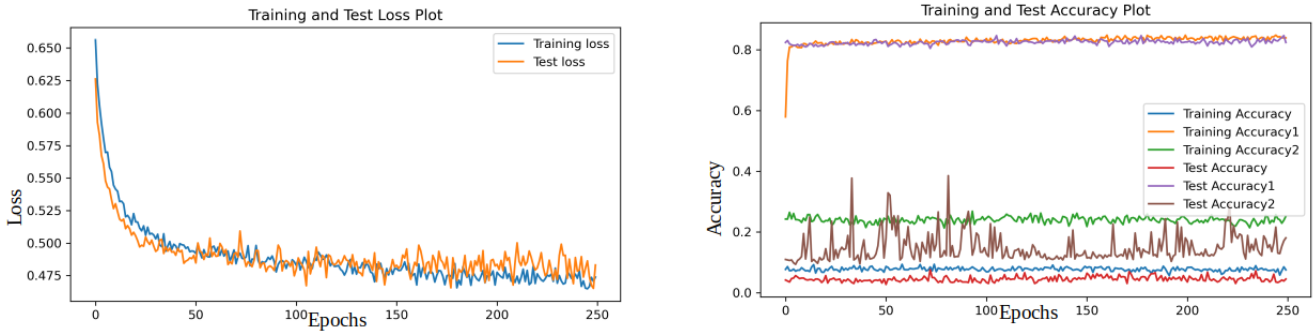
(a) Decay=10^{-6}



(b) Decay=10^{-8}



(c) Decay=10^{-10}



(d) Decay=10^{-12}

Figure 4.4: Effects of weight decay. Training parameters:lr= 0.003, momentum= 0.8, batch size= 100, $\gamma = 1$ (loss weight modifier=$[0,0,1]$).

Table 4.3: Per class accuracy for models in figure (4.4), (decay= $10^{-12}, 10^{-10}, 10^{-8}, 10^{-6}$)

Level	Class	Classification accuracy(%, decay=10^{-12})	Classification accuracy(%, decay=10^{-10})	Classification accuracy(%, decay=10^{-8})	Classification accuracy(%, decay=10^{-6})
1	Nebulous	15	19	25	9
	Non-nebulous	99	98	98	99
2	IR-loud	13	10	63	48
	IR-quiet	4	0	4	0
	All HII regions	57	65	43	62
	PN	0	0	0	0
3	Extra-galactic sources	1	78	2.5	68
	Radio stars	0	0	0	12
	PN	0	6	8	48
	HII regions	71	14	0	2
	UCHII regions	22	7	18	0

4.2.1 Pre-training analysis

This type of network structure has good performance, even on the few datasets with a small image size of 32. Batch normalization plays a significant role in stabilizing the network. The network can generalize classes on its own, and sampling is not necessary to introduce symmetry. Learning the fine features result in improved accuracy for fine level classes but poor performance on the coarse level. Focusing on the coarse features increases coarse class level accuracies but reduces fine levels. In comparison, focusing on coarse features leads to higher accuracy limited to the coarse levels as shown by the model of weight decay of 10^{-12} having higher accuracies for level 1 and 2 classes (see table (4.3)). On the other hand, the model with weight decay 10^{-6} has higher accuracies for the fine level. This is further shown by the differences in performance in varying the loss modifier (see table (4.2)). Further investigation shows that the in-between values of weight decay and loss modifier give a balanced per-level class classification. Thus, an optimal model is obtained by trading off on higher per class accuracy and balancing per-level accuracy.

4.3 CORNISH classifier

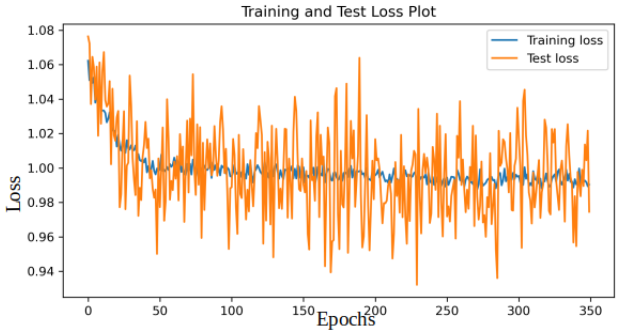
This phase entailed incorporating multi-wavelength and using the above-obtained configurations to fine-tune the classifier. In addition, loss and loss weight modifier were added onto the network as learnable parameters. This increases the number of trainable parameters, which further complicates the model. As a result, the 3-level model with a reduced number of hidden layers described in table

(3.6) with images size 32. And later on, adjusted to size 48 to test on the 2-level model in table (3.7).

4.3.1 Learnable loss

Adding loss as a learnable parameter introduces more complexities and requires more training configuration. The model requires a much lower learning rate to converge. Figure (4.5a) is a loss plot of the 3-level model with loss as a learnable parameter, trained on single-channel data (CORNISH north). The model explodes and does not converge even when trained for over 500 epochs. However, the model converges in figure (4.5b) when the learning rate is reduced to 10^{-7}. With additional scheduling for a further reduction once the loss gets to a minimum. The model is complex for single-channel data and has poor classification performance. On other tests, reducing the learning rate after a given epoch interval significantly improved the model. Thus, the additional parameter, lr step, specify the epoch step size to facilitate the learning rate.

(a) Loss plot of a model with learnable loss trained for 250 epochs. Training parameters: lr=0.0001, decay=10^{-12}, batch size=100, weight=$[1,0,0]$



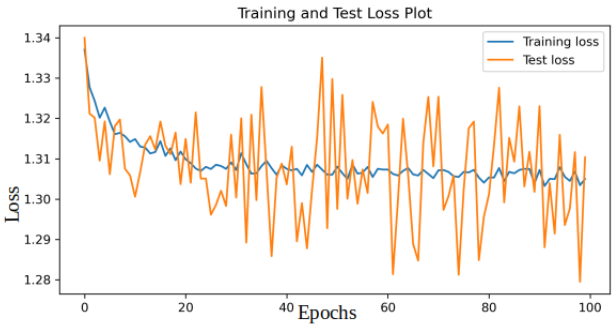
(b) Loss plot of a model with learnable loss trained for 350 epochs. Training parameters: lr=10^{-7}, decay=10^{-12}, batch size=100, weight=$[1,0,0]$

Figure 4.5: Effects of learnable loss on the 3-level model.

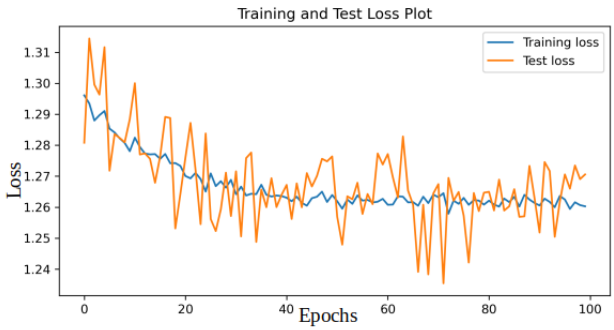
4.3.2 Channel selection

Additional channels onto the network improve on model's classification efficiency. However, it increases the number of trainable parameters, which introduce more complexities. Figure (4.6) has loss plots for various models. Figure (4.6a), (4.6b) and (4.6c) are based on the 3-level model, with image size 32 . Whereas figure (4.6d) is the 2-level model, with image size 48. Using a bigger image size with the 3-level model was hindered by limited GPU memory. The different models were of various configurations because each additional channel results in a model requiring new sets of hyperparameters. Fewer channels require a lower learning rate compared to a higher number of channels. From

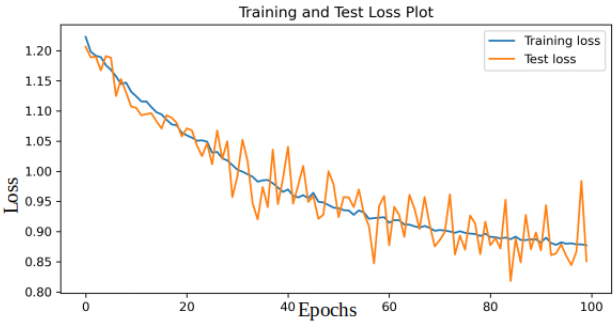
table (4.4), the varying number of channels in the 3-level model increase per class accuracy, but with a skewed class distribution similar to the single-channel model. The classification behavior of the 2-level model is similar to the performance of the coarse levels of the 3-level model. As a result, it was adopted to reduce complexities. Model in figure (4.6a) shows better capability in classifying IR-quite sources, which had been challenging for other tuned models. Unfortunately, the MIPS70 data set was corrupted thus discarded.



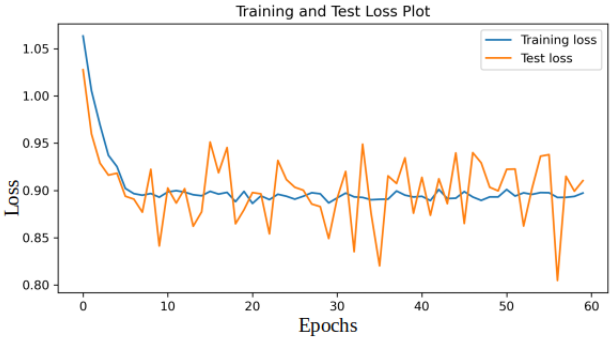
(a) Input channels are CORNISH, MIPS24, MIPS70, GLIMPSE (I3 and I4). Training parameters: weight=$[0.1, 0.2, 0.7]$, lr=10^{-7}, decay= 10^{-8}.



(b) Input channels are CORNISH, MIPS24, Hi-gal ($70\mu m$), UKIDDS, GLIMPSE (I4). Training parameters: weight=$[0.1, 0.2, 0.7]$, lr=10^{-7}, decay= 10^{-12}.



(c) Input channels are CORNISH, MIPS24, Hi-gal ($70\mu m$ and $500\mu m$), UKIDDS K, GLIMPSE (I4), ATLASGAL. Training parameters: weight=$[0.3, 0.3, 0.4]$, lr=0.0003, decay= 10^{-12}.



(d) 2-level model. Input channels are CORNISH, MIPS24, Hi-gal ($70\mu m$), UKIDDS, GLIMPSE (I4), ATLASGAL. Training parameters: weight=$[0.5, 0.5]$, lr=0.0003, decay= 3×10^{-8}.

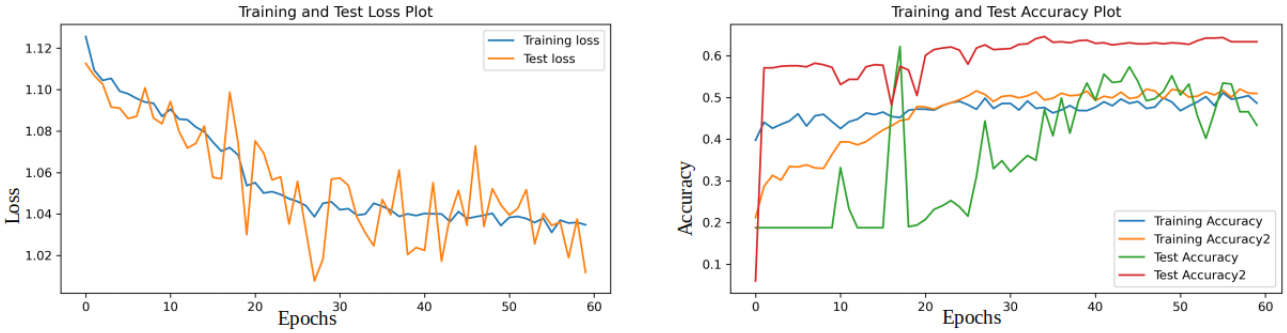
Figure 4.6: Various models of different channel inputs. Figures (4.6a)(4.6b) and (4.6c) are based on 3-level model. Figure (4.6d) is implemented in a 2-level model. For all the models: Batch size=100, lr step=5.

Table 4.4: Classification accuracy of various channels and configurations, models presented in figure (4.6).

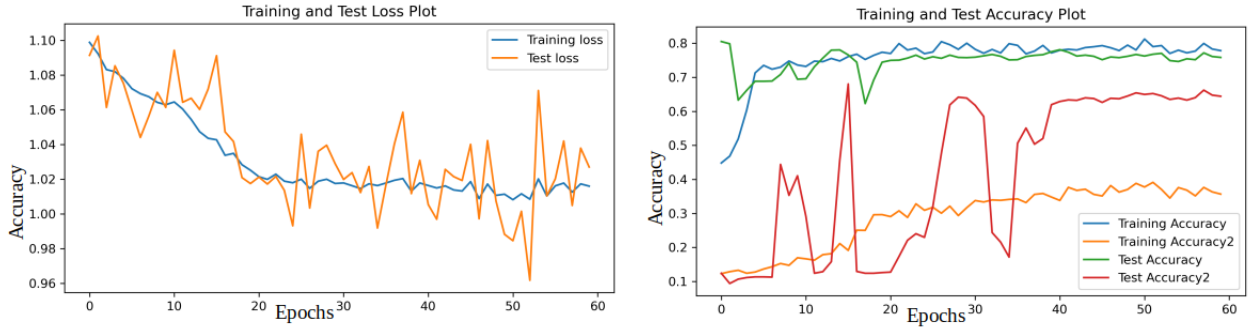
Level	Class	Classification accuracy(%) of model in figure (4.6a)	Classification accuracy(%) of model in figure (4.6b)	Classification accuracy(%) of model in figure (4.6c))	Test accuracy (%) of model in figure (4.6d)
1	Nebulous	64	0	47	49
	Non-nebulous	62	100	97	77
2	IR-quiet	39	51	98	82
	IR-loud	54	0	0	0
	All HII regions	56	76	70	50
	PN	0	21	0	3
3	Extra-galactic sources	1	51	98	
	Radio stars	0	0	0	
	PN	0	11	0	
	HII regions	36	29	0	
	UCHII regions	70	74	65	

4.3.3 Batch size and learning rate

The batch size determines the sample of loaded data at a time. From figure (4.7), accuracy plots show that increasing batch size improves test accuracy to a given extent. On the other hand, comparing test accuracy in figures (4.7b) and (4.7a), see accuracy stabilizing after 30 epochs. Similarly, the loss plots plateau ~ 30 epochs, which is the time scheduled for the learning rate to be reduced further, increasing the learning rate for the model in figure (4.7b) from 0.0003 to 0.0007. The model stabilizes as shown by the accuracy plots in figure (4.8). This implies that there needs to be a balance between learning rate and batch size to optimize the model performance.



(a) Batch size= 200.



(b) Batch size= 250

Figure 4.7: The effects of batch size. Training parameters: lr= 0.0003, weight=$[0.4, 0.6]$,lr step=20, decay= 10^{-16}.

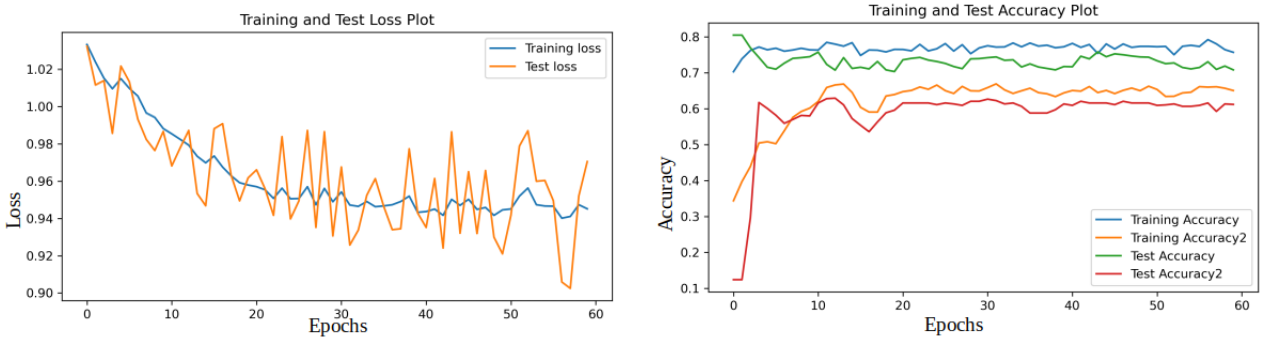
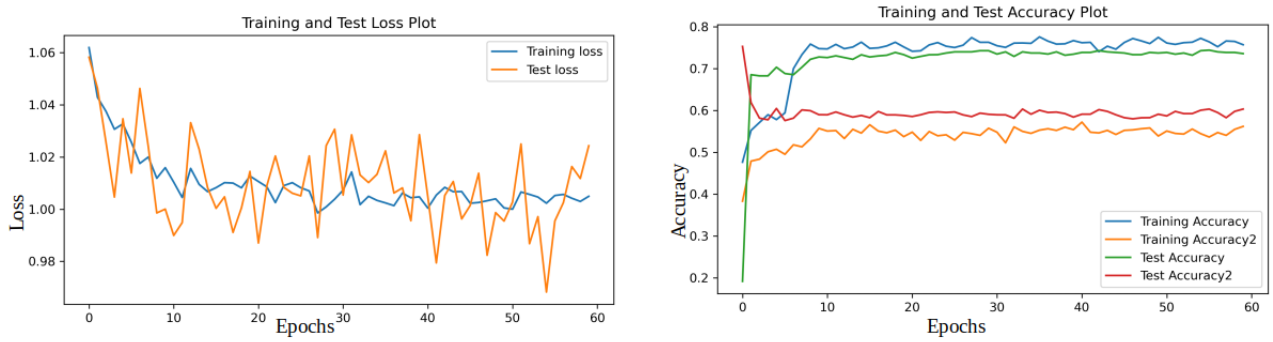


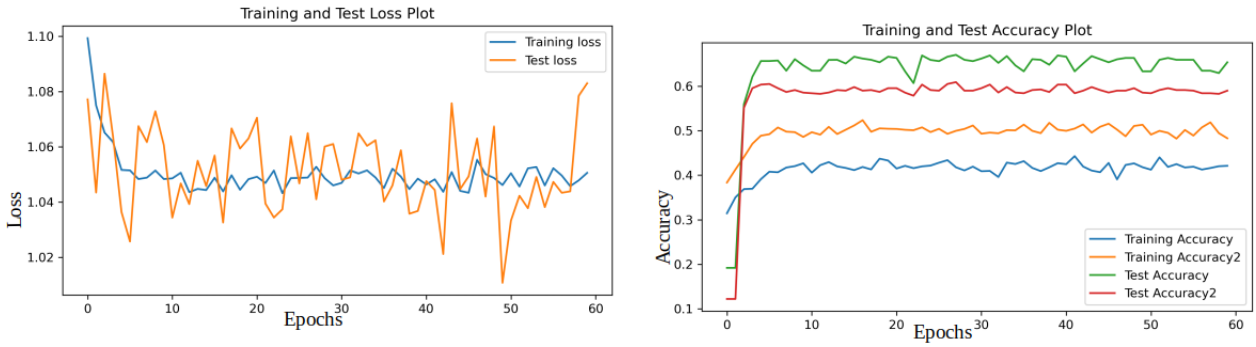
Figure 4.8: Batch size= 250. Training parameters: $lr = 0.0007$, decay=10^{-16}, weight= $[0.4, 0.6]$

4.3.4 Model selection

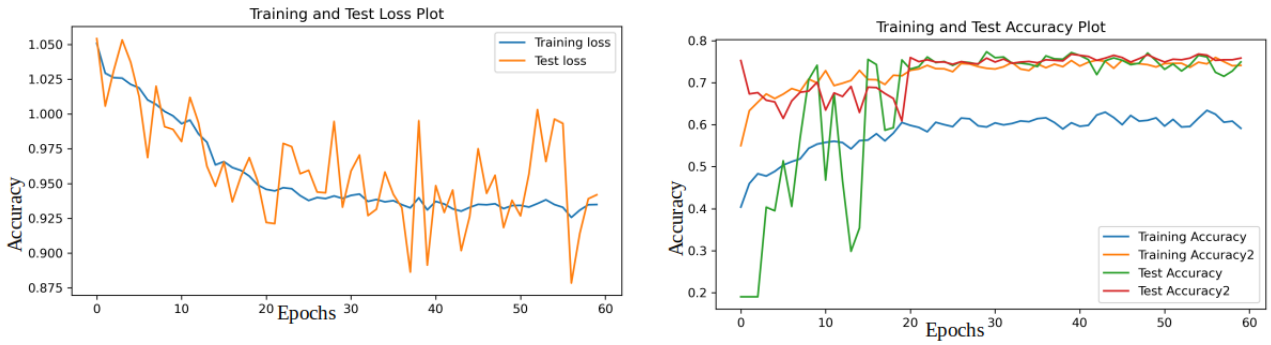
Models in figure (4.9) were the best performing after a series of trial and error in balancing the hyperparameters. Their classification accuracy in the table (4.5) shows a trade-off between a high accuracy rate and distributed classification accuracy.



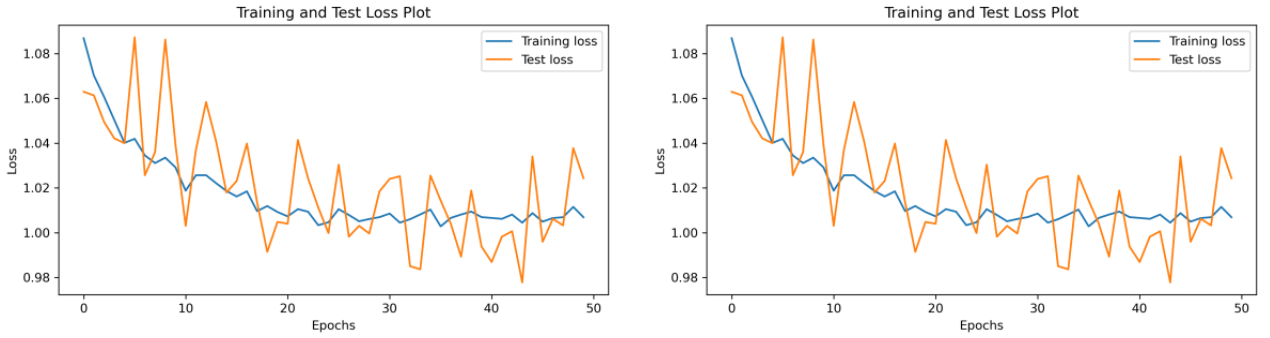
(a) Loss and accuracy plots. Training parameters: Batch size=120, lr=0.0003, lr step=10, weight=$[0.4, 0.6]$, decay=10^{-16}.



(b) Loss and accuracy plots. Training parameters: Batch size=120, lr=0.0003, lr step=5, weight=$[0.4, 0.6]$, decay=10^{-17}.



(c) Loss and accuracy plots. Training parameters: Batch size=150, lr=0.0003, lr step=30, weight=$[0.4, 0.6]$, decay=10^{-16}.



(d) Loss and accuracy plots of training parameters: Batch size=150, lr=0.0003, lr step=10, weight=$[0.4, 0.6]$, decay=10^{-17}

Figure 4.9: Candidate models for CORNISH sources.

Table 4.5: Classification accuracies of models in figure (4.9)

Level	Class	Classification accuracy(%) of model in figure (4.9a)	Classification accuracy(%) of model in figure (4.9b)	Classification accuracy (%) of model in figure (4.9c)	Classification accuracy(%) of model in figure (4.9d)
1	Nebulous	51	74	82	64
	Non-nebulous	81	66	72	78
2	IR-loud	76	69	88	73
	IR-quiet	16	0	0	6
	All HII regions	11	70	81	3
	PN	11	0	0	23

4.4 Classification

The model described in figure (4.9d) was used to classify some already classified CORNISH south sources. This was to test the classification capability of the developed classifier. MIPS24, VVV, ATLASGAL, Hi-gal($70\mu m$) and GLIMPSE I4 channels were incorporated. Mean classification accuracy for the different sources are in table (4.5a), whereby for each class label,

$$Classification\ \ accuracy = \frac{Correct\ \ \ predictions}{Total\ \ \ predictions} \times 100\% \tag{4.4.1}$$

The model has a mean test accuracy of 67% for coarse level 1 and 12% for level 2. Averaging test errors of all epochs obtain test accuracy during model evaluation. For each epoch, test accuracy is obtained as described in equation (4.0.2). After which

$$Mean\ \ test\ \ \ accuracy = \frac{\sum test\ \ accuracy}{\sum epochs} \times 100\%$$

Mean averaging classification accuracies can not obtain test accuracy in the table (4.5a).

Table 4.6: Model test on CORNISH south sources.

(a) Classification accuracy of the model in figure (4.9d) on CORNISH south sources, on the table (4.5b).

Level	Class	Classification accuracy(%) of model in figure (4.9d)
1	Nebulous	99
	Non-nebulous	0
2	IR-loud	0
	IR-quiet	36
	All HII regions	0
	PN	52

(b) Statistics of CORNISH south sources

Name	Number of sources
HII regions	179
UCHII	202
Radio star	60
Extragalactic sources	221
PN	178

4.5 Discussions

The performance of a CNN model is evaluated based on classification precision. From the sampled cases, the best performing model had an accuracy of 67% for coarse level 1 and 12% for level 2. However, this was tested on 840 sources of the CORNISH South dataset and may not necessarily reflect classification performance on the whole dataset comprising $\sim$ 5000 sources. For instance, test loss plots in figure (4.9) are unstable, implying the dataset is under-representative of the training data set. This might be due to the use of empty images for replacing missing data across the channels. Also, the use of imputed data on images impacted by the near-the-edge surveys might be a contributing factor. But these additional images are necessary to minimize further loss of the already minimal available labeled data.

The obtained classification accuracy is almost similar to manual classification but faster, thus a better alternative for a vast dataset. This classifier performs comparably for both CORNISH south and CORNISH north sources. From table (4.5a) and (4.5), the model performance is almost similar for both north and south sources. This shows that the model can generalize and classify sources from the different surveys unhindered by their different resolutions. But the low precisions are not conclusive enough. Thus there needs to be further investigations and improvement in fine-tuning the training parameters.

B-CNN models are complex, and fine-tuning of hyperparameters is vital in improving model performance. Parameter selection is complicated due to loss addition which contributes to more complexities. Learning rate plays a significant role in model convergence. A high learning rate

causes the model to explode. On the other hand, a very low learning rate causes overfitting and more instabilities. Reducing the learning rate at given intervals, initialized at $\leq 10^{-7}$ and $\leq 10^{-3}$ for single-channel and multiple channels, respectively, offered the best performance.

Additionally, the choice of batch size too, influences the selection of learning rate. A large batch size results in more parameters with a lower learning rate, giving a more stable model. But this is computationally expensive and requires a significant amount of computer memory; Which is often not available. Hence there need to choose an optimal batch size with an accompanying learning rate.

Dataset significantly determines the outcome of a model. A complex dataset is very dynamic and would often result in unpredictable instabilities. CNN networks are like very long tunnels with zero visibilities. You can only tell what enters and predict what to expect at the exit but wouldn´t be sure of what happens inside the tunnel. A real-life dataset is complex and comprises noise-causing factors that cause instabilities. As a result, it is not necessarily confident about how data is handled inside the CNN network.

Moreover, B-CNN is a compilation of multiple CNN networks, thus more complicated. Model training and test show that any additional channel would result in a completely different model. Therefore fine-tuned parameters for a single channel dataset can not be directly transferred as fined tuned parameters for other channels. But can be used as guidance in tuning parameters when adjusting the number of channels. From this, it is conclusive that many channels do not necessarily mean a better-performing model. However, it offers the advantage of the more additional dataset. Hierarchy levels of B-CNN didn´t seem to play a significant role in the model performance. A $3-$ level model has almost similar performance to $2-$ level models for levels 1 and 2, and only advantageous by classifying level 3 classes.

4.6 Conclusion and recommendations

This work was focused on developing a classifier for CORNISH South sources. A $3-$level and $2-$level B-CNN models were trained and tested on CORNISH north sources with additional multi-wavelength data. A $2-$level model, trained on CORNISH North, MIPS24, Hi-Gal ($70\mu m$), UKIDDS, GLIMPSE (I4), and ATLASGAL datasets was the optimal performing model. It was trained with the following set of hyperparameters: Batch size$=$ 150, learning rate (lr)$=$ 0.0003, epoch size for reducing learning rate (lr step)=10, loss weight modifier ($[\alpha,\beta]$=$[0.4,0.6]$), decay$=$ 10^{-17}. The result-

ing model was tested on multi-channel data comprising CORNISH South, MIPS24, Hi-Gal ($70\mu m$), VVV, GLIMPSE (I4), and ATLASGAL. UKIDDS does not cover the CORNISH South region. Hence the channel was replaced by VVV, which surveyed in the same wavelength, covering the Southern region. The resulting classifier had a test accuracy of 67% for the coarse level and 12% for the fine level.

This was the most optimal model amongst the trained models. However, more work is still needed to fine-tune the model parameters. For example, increasing images from 48 to $\sim 100-200$ will improve the efficiency of the classifier. However, this was not possible in this work due storage of GPU/CPU memory. Using fewer channels might also lead to fewer complexities which might lead to a more stable model. This will also reduce training time and bad images such as imputed and empty images, critical hindrances to classification efficiency. Evaluating the models on the labeled CORNISH South dataset might also contribute to selecting the optimal model. Instead of training and evaluating the models based on CORNISH North and only testing South sources for classification purposes, Which was the case in this work.

Additionally, the coarse level prediction is independent of the fine level output. This is because each level of the B-CNN is a complete CNN by itself. This property can be taken advantage of by fine-tuning different B-CNN models to focus on specific classes. Results from each model can then be used in improving classification accuracy. And also, a $3-$ level model will be a better alternative since it incorporates accuracies for the fine level.

Transfer learning by using weights of the already trained CORNISH North sources to train on CORNISH South sources. This method has been tested by Tang, Scaife and Leahy (2019) for classifying radio galaxies. But this requires CORNISH South sources to be already classified and labeled. Another alternative will be to use the already pre-trained parameters to develop a TB-CNN-based model (Mou and Jin, 2018). TB-CNN incorporates the use of conditional logistics, which will give a better prediction precision. Its primary disadvantage is the need for already pre-trained weights. However, this is already done in this particular work.

Bibliography

An, FangXia et al. (July 2018). "A machine-learning method for identifying multi-wavelength counterparts of submillimeter galaxies: training and testing using AS2UDS and ALESS". In: *ApJ* 862.2, p. 101.

Arsioli, Bruno and Pedro Dedin (Sept. 2020). "Machine Learning Applied to Multifrequency Data in Astrophysics: Blazar Classification". In: *Monthly Notices of the Royal Astronomical Society* 498.2, pp. 1750–1764. DOI: 10.1093/mnras/staa2449.

Badr, Will (Jan. 2019). *6 Different Ways to Compensate for Missing Data (Data Imputation with Examples)*. URL: https://towardsdatascience.com/6-different-ways-to-compensate-for-missing-values-data-imputation-with-examples-6022d9ca0779.

Carey, S. J. et al. (Jan. 2009). "MIPSGAL: A Survey of the Inner Galactic Plane at 24 and 70 μm". In: *PUBL ASTRON SOC PAC* 121.875, pp. 76–97. DOI: 10.1086/596581.

Chilamkurthy, Sasank (2021). *Writing Custom Datasets, DataLoaders and Transforms — PyTorch Tutorials 1.9.0+cu102 Documentation*. URL: https://pytorch.org/tutorials/beginner/data_loading_tutorial.html.

Churchwell, Ed et al. (2009). "The *Spitzer* /GLIMPSE Surveys: A New View of the Milky Way". In: *PASP* 121.877, pp. 213–230.

Clark, Alex (2014). *Pillow · PyPI*. URL: https://pypi.org/project/Pillow/2.5.0/.

Condon, J. J. (1999). "Very large radio surveys of the sky". In: *PASP* 96.9, pp. 4756–4758.

Condon, J. J. et al. (1998). "The Nrao VLA Sky Survey: Lessons Applied". In: *Observational Cosmology*. Ed. by M. N. Bremer, N. Jackson and I. Pérez-Fournon. Astrophysics and Space Science Library. Springer Netherlands, pp. 37–44. DOI: 10.1007/978-94-011-5238-9_6.

"Convolution and Applications". In: (), p. 40.

CS231n Convolution Neural Networks for Visual Recognition. URL: https://cs231n.github.io/neural-networks-1 (visited on 24/10/2020).

Czech, Daniel, Amit Mishra and Michael Inggs (Mar. 2018). “A CNN and LSTM-Based Approach to Classifying Transient Radio Frequency Interference”. In: *arXiv:1803.02684 [eess]*.

Devine, Thomas, Katerina Goseva-Popstojanova and Di Pang (Aug. 2018). “Scalable Solutions for Automated Single Pulse Identification and Classification in Radio Astronomy”. In: *Proceedings of the 47th International Conference on Parallel Processing*, pp. 1–11.

Fully Connected Layers in Convolutional Neural Networks. *Fully Connected Layers in Convolutional Neural Networks: The Complete Guide*. URL: https://missinglink.ai/guides/convolutional-neural-networks/fully-connected-layers-convolutional-neural-networks-complete-guide/ (visited on 31/10/2020).

Garnelo, Marta and Murray Shanahan (2019). “Reconciling deep learning with symbolic artificial intelligence: representing objects and relations”. In: *Current Opinion in Behavioral Sciences* 29, pp. 17 –23.

Hoare, M. G. et al. (2012). “The Co-ordinated Radio and Infrared Survey for High Mass Star Formation (The CORNISH Survey) - I. Survey Design”. In: *PASP* 124.919, pp. 939–955.

information@eso.org, ESO. *Comparison of the central part of the Milky Way at different wavelengths (annotated)*. URL: https://www.eso.org/public/images/eso1606e/ (visited on 31/10/2020).

Ioffe, Sergey and Christian Szegedy (2015). “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *arXiv:1502.03167 [cs]*.

Irabor, T. et al. (2018). “The Coordinated Radio and Infrared Survey for High-mass Star Formation. IV: A new radio selected sample of compact Galactic Planetary Nebulae”. In: *MNRAS* 480.2, pp. 2423–2448.

Kalcheva, I. E. et al. (2018). “The Coordinated Radio and Infrared Survey for High-Mass Star Formation III. A catalogue of northern ultra-compact H II regions”. In: *AA* 615, A103.

Khan, Asifullah et al. (2020). “A survey of the recent architectures of deep convolutional neural networks”. In: *AIR*.

Khandelwal, Renu (Jan. 2020). *Gradient Descent*. URL: https://arshren.medium.com/gradient-descent-5a13f385d403 (visited on 31/10/2020).

Lucas, P. W. et al. (Nov. 2008). “The UKIDSS Galactic Plane Survey”. In: *Monthly Notices of the Royal Astronomical Society* 391.1, pp. 136–163. DOI: 10.1111/j.1365-2966.2008.13924.x.

Lukic, V., F. De Gasperin and M. Brüggen (Dec. 2019). "ConvoSource: Radio-Astronomical Source-Finding with Convolutional Neural Networks". In: *arXiv:1910.03631 [astro-ph]*.

Lukic, V. et al. (Aug. 2019). "Morphological Classification of Radio Galaxies: Capsule Networks versus Convolutional Neural Networks". In: *Monthly Notices of the Royal Astronomical Society* 487.2, pp. 1729–1744. DOI: 10.1093/mnras/stz1289.

Lundh, Fredrik and Alex Clark (1995). *Image File Formats — Pillow (PIL Fork) 8.2.0 Documentation*. URL: https://pillow.readthedocs.io/en/stable/handbook/image-file-formats.html.

Ma, Zhixian et al. (Feb. 2019). "A Machine Learning Based Morphological Classification of 14,245 Radio AGNs Selected From The Best-Heckman Sample". In: *ApJS* 240.2, p. 34.

Minniti, Dante (Aug. 2015). "The VVV Survey". In: 29, p. 2257043.

Molinari, S. et al. (July 2016). "Hi-GAL, the Herschel Infrared Galactic Plane Survey: Photometric Maps and Compact Source Catalogues. First Data Release for Inner Milky Way". In: *A&A* 591, A149. DOI: 10.1051/0004-6361/201526380.

Mou, Lili and Zhi Jin (2018). *Tree-Based Convolutional Neural Networks: Principles and Applications*. SpringerBriefs in Computer Science. Singapore: Springer.

Neilson, Stephen (2018). "CORNISH object classification using a CNN- Final report". In: *Final Thesis report*, p. 29.

Overview of different Optimizers for neural networks | by Renu Khandelwal | Data Driven Investor | Medium. URL: https://medium.com/datadriveninvestor/overview-of-different-optimizers-for-neural-networks-e0ed119440c3 (visited on 31/10/2020).

Purcell, C. R. et al. (2013). "The Co-ordinated Radio and Infrared Survey for High-Mass Star Formation - II. Source Catalogue". In: *ApJS* 205.1, p. 1.

Pytorch (2021). *Datasets & Dataloaders — PyTorch Tutorials 1.9.0+cu102 Documentation*. URL: https://pytorch.org/tutorials/beginner/basics/data_tutorial.html.

PyTorch (2021). *Welcome to PyTorch Tutorials — PyTorch Tutorials 1.9.0+cu102 Documentation*. URL: https://pytorch.org/tutorials/.

Rengelink, R. B. et al. (Aug. 1997). "The Westerbork Northern Sky Survey (WENSS): I. A 570 Square Degree Mini-Survey around the North EclipticPole". In: *Astron. Astrophys. Suppl. Ser.* 124.2, pp. 259–280. DOI: 10.1051/aas:1997358.

Saha, Sumit (2018). *A Comprehensive Guide to Convolutional Neural Networks*. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (visited on 24/10/2020).

Samal, M R et al. (2010). "Multi-wavelength studies of Galactic HII regions". In: p. 6.

Schilizzi, R. T. (2005). "The Square Kilometre Array". In: *EAS Publications Series* 15, pp. 445–463. DOI: 10.1051/eas:2005170.

Schuller, F. et al. (Sept. 2009). "ATLASGAL – The APEX Telescope Large Area Survey of the Galaxy at 870 μm". In: *Astronomy & Astrophysics* 504.2, pp. 415–427. DOI: 10.1051/0004-6361/200811568.

Simpson, Chris (2017). "Extragalactic radio surveys in the pre-Square Kilometre Array era". In: *RSOS* 4.7, p. 170522.

Tang, Hongming, Anna M. M. Scaife and J. P. Leahy (2019). "Transfer learning for radio galaxy classification". In: *MNRAS*.

Torch, Contributors (2017a). *Torchvision — Torchvision 0.10.0 Documentation*. URL: https://pytorch.org/vision/stable/index.html.

— (2017b). *Torchvision.Transforms — Torchvision 0.8.1 Documentation*. URL: https://pytorch.org/vision/0.8/transforms.html.

Wu, Chen et al. (Jan. 2019). "Radio Galaxy Zoo: ClaRAN - A Deep Learning Classifier for Radio Morphologies". In: *MNRAS* 482.1, pp. 1211–1230.

Yan, Lin et al. (Jan. 2013). "Characterizing the MID-Infrared Extragalactic Sky With *WISE* AND SDSS". In: *AJ* 145.3, p. 55. DOI: 10.1088/0004-6256/145/3/55.

Yan, Zhicheng et al. (2015). "HD-CNN: Hierarchical Deep Convolutional Neural Network for Large Scale Visual Recognition". In: *arXiv:1410.0736 [cs, stat]*.

Zhu, Xinqi and Michael Bain (2017). "B-CNN: Branch Convolutional Neural Network for Hierarchical Classification". In: *arXiv:1709.09890 [cs]*. arXiv: 1709.09890.