# AUTOMATED SCENARIO DESCRIPTION FOR IMAGES

**A PROJECT REPORT**

*Submitted By*


**SATISH P 312212104095**

**SIDDHARTH G 312212104102**

**VIJAYALAKSHIMI MLS 312212104121**


*in partial fulfillment for the award of the degree*

*of*

**BACHELOR OF ENGINEERING**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**


**SSN COLLEGE OF ENGINEERING**

**KALAVAKKAM 603110**


**ANNA UNIVERSITY :: CHENNAI - 600025**

**April 2016**

# ANNA UNIVERSITY : CHENNAI 600025

## BONAFIDE CERTIFICATE

Certified that this project report titled **"AUTOMATED SCENARIO DESCRIPTION FOR IMAGES"** is the *bonafide* work of "**SATISH P (312212104095)**, **SIDDHARTH G (312212104102)**, and **VIJAYALAKSHIMI MLS (312212104121)**" who carried out the project work under my supervision.


**Dr. Chitra Babu**
**Head of the Department**
Professor,
Department of CSE,
SSN College of Engineering,
Kalavakkam - 603 110

**Dr. R S Milton**
**Supervisor**
Professor,
Department of CSE,
SSN College of Engineering,
Kalavakkam - 603 110


Place:
Date:

Submitted for the examination held on.....

**Internal Examiner** **External Examiner**

# ACKNOWLEDGEMENTS

We thank GOD, the almighty for giving us strength and knowledge to do this project.

We would like to thank and express deep sense of gratitude to our guide **Dr. R S MILTON**, Professor, Department of Computer Science and Engineering, for his valuable advice and suggestions as well as his continued guidance, patience and support that helped us shape and refine our work.

Our sincere thanks to **Dr. CHITRA BABU**, Professor and Head of the Department of Computer Science and Engineering, for her words of advice and encouragement and We would like to thank our project Coordinator **Dr. SHEERAZUDDIN**, Professor, Department of Computer Science and Engineering for his valuable suggestions throughout this first phase of project.

We express our deep respect to the founder **Dr. SHIV NADAR**, Chairman, SSN Institutions. We also express our appreciation to our Principal, **Dr. S. SALIVAHANAN**, for all the help he has rendered during this course of study.

We would like to extend my sincere thanks to all the teaching and non-teaching staffs of our department who have contributed directly and indirectly during the course of our project work. Finally, we would like to thank our parents and friends for their patience, cooperation and moral support throughout our life.

# ABSTRACT

The project aims at automatically generating a description for an image using machine learning and machine translation techniques. Restricting to cartoon images (especially that of Pokemon), bounds the number of objects to be detected and the number of actions that can be performed by a particular object. Given a labeled training set of images of Pokemon and their actions, we train a classifier to segment and detect them in an image and finally relate the actions with the doer objects to form a meaningful sentence in human readable form. The ultimate vision of the work is to design computing systems with image articulation skills similar to humans.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

CHAPTER 1

# INTRODUCTION AND BACKGROUND

The intersection of Machine Learning and Computer Vision is witnessing its first dawn after the advent of Deep Learning. Although Deep Learning and Neural Networks were proposed years ago, computational intractability and necessity to have intelligent visual agents were almost absent and so tangible results could not be reached, which is possible today, albeit slowly. Computer Vision aims at understanding a 2-D array of numbers which are images and Machine Learning tries to perform a task by learning from experience. We realize a confluence of these two fields along with Deep Learning in our project, all the while trying out various techniques in the quest for further visual intelligence, ranging from the age old $k$-NN to even a little bit of Graph Theory! To start off, we first understand what Machine Learning is.

## 1.1 Machine Learning

Machine Learning is formally defined as

> Given a set of couples $(x_i, y_i)$ where each $x_i \in X$, $y_i \in Y$ and a class of functions $f_\theta : X \to Y$ as well as a loss function $L : Y \times Y \to R^+$, find the function $f_\theta$ which minimizes $\Sigma_i L(f_\theta(x_i), y_i)$.

A more intuitive understanding can be seen in this age old definition:

> A computer program is said to learn from experience $E$ with respect to some class of tasks $T$ and performance measure $P$ if its performance at tasks in $T$, as measured by $P$, improves with experience $E$.

In this project, we ultimately minimize a loss function, either softmax loss or hinge loss, evaluated over a complex, continuous, differentiable polynomial developed by a Convolutional Neural Network, using variations of the solver, Gradient Descent, given a set of images as the input couples.

**Convolutional Neural Network:** Convolutional Neural Networks (CNN) are biologically-inspired variants of Multi-Layer Perceptrons. From Hubel and Wiesel's early work on the cat's visual cortex [16], we know the visual cortex contains a complex arrangement of cells. These cells are sensitive to small sub-regions of the visual field, called a receptive field. The sub-regions are tiled to cover the entire visual field. These cells act as local filters over the input space and are well-suited to exploit the strong spatially local correlation present in natural images. Additionally, two basic cell types have been identified. Simple cells respond maximally to specific edge-like patterns within their receptive field. Complex cells have larger receptive fields and are locally invariant to the exact position of the pattern. The animal visual cortex being the most powerful visual processing system in existence, it seems natural to emulate its behavior. Hence, many neurally-inspired models can be found in the literature, among which are NeoCognitron [17], HMAX [18] and LeNet-5 [19].

## 1.2 Organisation

In Chapter 2, we discuss the literature studied and surveyed in order to conceptualise the project and define its objectives. We discuss the various implementation strategies used and design our model correspondingly.

In Chapter 3, we discuss the basic architectural design of the proposed model and explain the functions of each module. The architectural design focuses on the main modules including Pokemon Localisation, Pokemon Classification, Attacks Classification and Sentence Generation. Each of these modules are explained in brief in this chapter and in detail in subsequent chapters.

In Chapter 4, we explain in detail about the various datasets prepared and used. Broadly, three kinds of datasets were prepared: Pokemon image dataset, Battle image dataset and Attacks image dataset.

In Chapter 5, we discuss the Selective Search Algorithm which is used to segment images and localise Pokemon. Further, fine tuning techniques for the algorithm are discussed in order to improve its results. A Pokemon/ Parts/ Other classifier is discussed as it helped circumvent the undesirable effects of Selective Search Algorithm.

In Chapter 6, we discuss various Feature Extraction methods such as the use of 1D RGBA Vectors, 3D Colour Histograms and Histogram of Oriented Gradients. We also discuss a number of classification methods including $k$-Nearest Neighbour and Convolutional Neural Networks.

Further, we explain a concept called Connected Colour Components used to visualise cartoon images as graphs and compare the same for similarity.

In Chapter 7, we discuss Classification of Attacks using GoogleNet, which is initialized using MIT Places 205 Weights. Classification of Attacks is explained as a scene recognition process rather than an object recognition one. Further, we discuss Triple Generation and Caption Generation using Sentence Templates in chapter 8. Chapter 9 concludes and suggests possible future direction for further work.

CHAPTER 2

# LITERATURE SURVEY

We discuss the literature studied and surveyed in order to conceptualise the project and define its objectives. We also consider the various implementation strategies used and design our model correspondingly.

## 2.1 Automatic Caption Generation for News Images

This system is based on the fundamental idea that the ability to link images with textual descriptions could facilitate the retrieval of multimedia data as well as increase the accessibility of the web for visually impaired users who are unable to access the web in the same ways as sighted users can. Image description generation is achieved through a 2-stage process. First stage, content selection, involves identification of the event or object described by the image. Second stage, surface realization, renders the description of the image. Human written templates are used to achieve the same in this unsupervised technique. Use of dictionaries that contain text descriptions for images has been written off to be exhaustive and minimally efficient. An abstractive and extractive caption generation model is alternatively proposed using the probabilistic image annotation model which suggests keywords for every image. Previously, similar systems involved segmentation of objects in an

image and consequent generation of description, usage of triples that rendered visual information, and usage of image annotation as a classification task for images (supervised technique).

**Image annotation model**

A hypothetical system is provided with an image and a set of keywords. The hypothetical system generates a description for the image using the accompanying document along with the specified keywords.

**Extraction caption generation**

The basic idea is to extract the most important sentences from a document and concatenate them while paying minimal attention to linguistic analysis. Various methods are used to achieve the same, including Word overlap based sentence selection, Vector space based sentence selection and Topic based sentence selection.

**Abstraction caption generation**

This form of caption generation can be implemented using methods such as Word based and Phrase based caption generation.

## 2.2 Automatic Image Captioning

The aim of the work in [20] was, given a dataset of images with human annotated captions, to learn associations of the words in the captions with blobs in the image. The images were separated into blobs based on feature vectors regarding the colour, texture, shape, size, position and the blobs were assigned blob tokens.

### Dataset

The dataset consists of set of images $I = \{I_1, I_2, \ldots, I_N\}$ and each image $I_i$ has a set of words (captions) $W_i = \{W_{i1}, W_{i2}, \ldots, W_{iL_i}\}$ and a set of blobs $B_i = \{B_{i1}, B_{i2}, \ldots, B_{iM_i}\}$. The blobs are formed using G-Means algorithm, an adaptive clustering method.

### Input representation

They form two matrices, an unweighted data matrix and a uniqueness weighted data matrix. The unweighted data matrix is of the form $D_0 = [D_{W0}|D_{B0}]$ is a $N \times (W + B)$ matrix, where $(i, j)$, an element of the $N \times W$ matrix $D_{W0}, D_{B0}$ is the count of term $w_j$ in image $i$, blob $b_j$ in image $i$. Uniqueness weighted data matrix $D = [D_W|D_B]$ is formed from this matrix by multiplying each term by the log of reciprocal of the fraction of the dataset in which the word or blob appears.

**Caption generation**

The next step was to find a correlation between the words and blob vectors and hence formed 4 matrices, corr, cos, svdcorr, svdcos. An image with $l$ blob-token $B_0 = (b_{01},\ldots b_{0l})$, can be captioned as follows. First, form a query vector $q = [q_1,\ldots q_B]$, where $q_i$ is the count of the blob-token $b_i$ in the set $B^{'}$. Then, compute the term-likelihood vector $p = Tq$ , where $p = [p_1,\ldots p_W]^T$, and $p_i$ is the predicted likelihood of the term $w_i$. Finally, select the $m$ captioning terms corresponding to the top-$m$ $p_i$'s in the $p$-vector. They do not form complete English sentences, but only tags for the image.

## 2.3 Deep visual-semantic Alignments for Generating Image Descriptions

The model was proposed for the purpose of generating description for images. CNN was used for image classification and object detection while RNN was used for language modelling. In order to facilitate inter-nodal relationships, RNN was trained to combine a word and its previous context to produce the next word. The words in the sentences are also stored in the $n$-dimensional space, as vectors, where images were stored. For this purpose, BRNN (Bidirectional Recurrent Neural Network) was used. Further, an image-sentence score was calculated such that a sentence-image pair has a high score if the word has a confident support in the image. Later, Multimodal RNN was used to achieve better performance. MRNN proved to outperform other techniques.

**Dataset**

Data was obtained from Flickr8K, Flickr30K and MSCOCO. 5000 images from MSCOCO were used for validation and testing. Each image used was annotated with 5 sentences using Amazon Mechanical Turk.

**Data processing**

In order to perform data processing, the various processes were performed including converting to lowercase and discarding non-alphanumeric characters.

## 2.4 Baby Talk: Understanding and Generating Image Descriptions

Baby Talk [21] is about understanding and generating image description by building tight connections between image content and the caption generation process. Given any image, it mainly extracts the objects, modifiers (attributes or adjectives) and spacial relationships (prepositions) from it. Then these detections from the images are smoothed and removed of noise with respect to the statistical prior obtained from a corpus of descriptive text. These smoothed results are later used as constraints for sentence generation. This sentence generation is done using simple $n$-gram language models (like Markovian based ones) or simple boilerplate templates of sentences. There are 6 modules namely

Detector, Attribute Classifier, Prepositional relationship formulater, CRF and Label generation, and Sentence generation.

- Detector detects things (bus, car, man, boy etc) and stuff (grass, trees, water, etc.).
- Attribute classifier: Each candidate object regions generated above are processed by an attribute classifier trained on ImageNet to get attributes of images such as color, pattern, shape, texture.
- Propositional relationship formulator: Pairwise propositional relationships are computed like relationship (object1, object2) (Trained on Flickr image descriptions).
- CRF: They are built over 100 hand-labeled images, and are used to handle nodes generated by above three layers that ultimately try to minimize an energy function to generate labels.
- Sentence generation: The triples <objects, prepositions, stuff> generated by CRF are used to build sentences by just inserting gluing words and pre-formated boilerplates.

CHAPTER 3

# SYSTEM ARCHITECTURE

We describe in this chapter the basic architectural design of the proposed model and explain the functions of each module. The architectural design focuses on the main modules including Pokemon Localisation, Pokemon Classification, Attacks Classification and Sentence Generation. Each of these modules are explained briefly in this chapter and in detail in the forthcoming chapters.

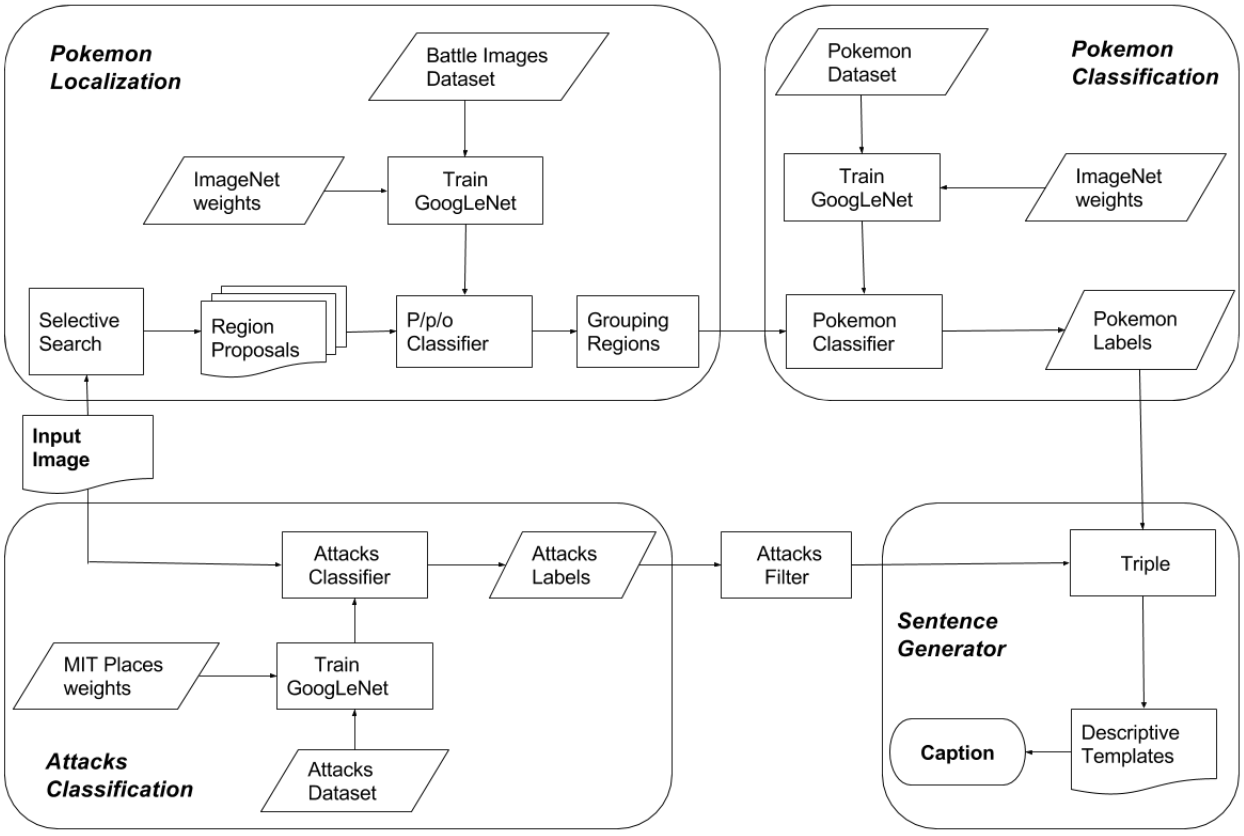


FIGURE 3.1: Architecture of image description pipeline

## 3.1 Architecture Overview

In this section, we present an overview of the components of the architecture of our image captioning system (Figure 3.1). A detailed explanation of the working of each component is presented in the subsequent sections.

- **Input Image**: A Pokemon battle image such as the one shown in Figure 3.2 is the input image for which we would like to generate a caption. This is a cartoon image and usually consists of about two Pokemon, in battle with each other. The Pokemon may also be executing attacks, as seen in the image, based on their type and ability.

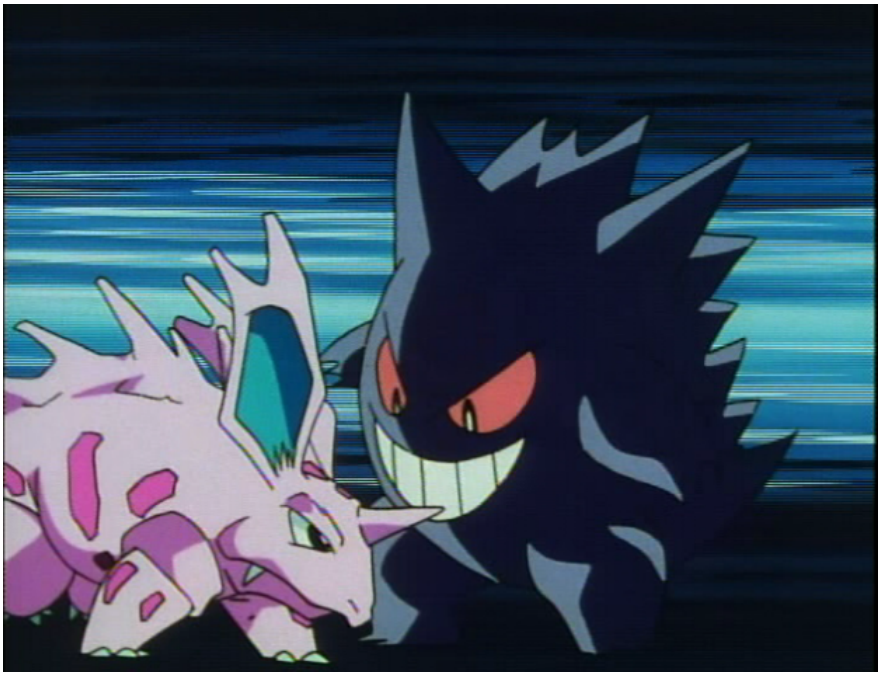

FIGURE 3.2: Input battle image

- **Localization Module**: This module is responsible for segmenting the specific Pokemon images from the input image, to be fed to the classifier, for identification in the next step. We use the *Selective Search algorithm* [6] to identify bounding boxes of Pokemon as shown in Figure 3.3. This produces a set of region proposals, which are all input to the next stage, the classifier.


FIGURE 3.3: Input image with bounding boxes

- **Pokemon Identification**: The input to this component are the cropped images of Pokemon from the input image, as shown in Figure 3.4. We train a *Convolutional Neural Network* (GoogLeNet) [5] using *ImageNet* [3] weights with Caffe [4] and fine-tune it to realize the Pokemon classifier.

FIGURE 3.4: Input image and cropped output labeled as Persian

- **Attacks Identification**: The initial input image as a whole is fed as input to the attacks identification module. Since attacks are not local in an image and they represent a scene, we need to consider the image as a whole, as in Figure 3.5. Again, we train a *Convolutional Neural Network* (GoogLeNet), but this time using *MIT's Places 205 weights* [7] to classify scenes, and not objects.

FIGURE 3.5: Wrap attack

- **Triple Generation**: Pokemon have the limitation that each can execute only a certain type and number of attacks. Using this to our advantage, we filter the attacks received from the previous module *Attacks Identification* so that the resultant attacks are only those which the identified Pokemon can execute. This results in a triple, which has the following structure.

  <Pokemon_A, Pokemon_B, Attack_C>

  Sample: <Charizard, Articuno, Mirror Move>

- **Captioning**: Finally, we use descriptions from a body of text containing Pokemon battle descriptions and insert the triple values at relevant places to form a caption for the image as shown in Figure 3.6

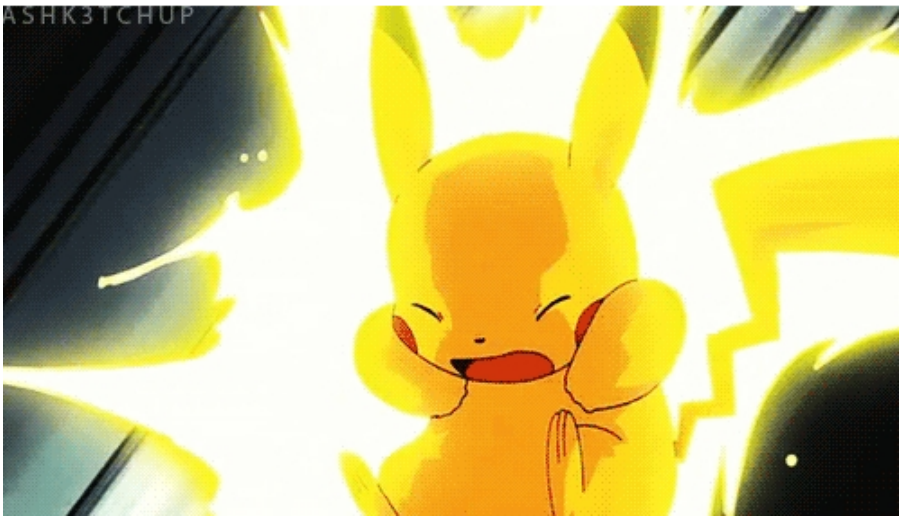


FIGURE 3.6: Pikachu is executing Thunder Wave

CHAPTER 4

# DATASET PREPARATION

This chapter explains in detail about the various datasets prepared and used. Broadly, three kinds of datasets were prepared, Pokemon image dataset, Battle image dataset and Attacks image dataset. Preparation of the dataset was twofold: one for training a *Pokemon* classifier, and another for training an *Attacks* classifier.

## 4.1 Pokemon Images Dataset

We generated two different types of Pokemon Images Dataset, each tailored for a specific purpose. They are: the raw vectorized dataset, where the the images have transparent backgrounds and only the Pokemon is visible holding a clear pose; the backgrounded dataset, where the raw vectorized images are superimposed on Pokemon battle background images; the backgrounded and augmented dataset, where a range of transformations are applied on each raw Pokemon image and then each transformed image is superimposed on top of a background commonly found in battles.

### 4.1.1 Raw Vectorized Dataset

We parsed through the website, `Bulbapedia` [9] and obtained raw Pokemon images with transparent backgrounds. This was necessary because *Connected Color Components* (CCC) is an algorithm which learns how a Pokemon looks based on its color and its color adjacencies. Hence the presence of only the Pokemon in the image with a transparent background was needed as a background might mislead the CCC algorithm to think that was also part of the Pokemon (Figure 4.1).

FIGURE 4.1: Raw vectorized dataset

### 4.1.2 Backgrounded Dataset

The images obtained were superimposed with different backgrounds normally found in battle images resulting in a total of 1,50,000 images of Pokemon to be used in training. The superimposed images were scaled

down to the size of 32 × 32 pixels, using bilinear, bicubic, anti-alias and nearest resizing algorithms (Figure 4.2).

FIGURE 4.2: Snapshot of backgrounded dataset

### 4.1.3 Augmented Dataset

In order to build a model invariant to scale, rotation and translation operations, aspects of Pokemon images such as angle and posture were considered. An image data augmentation tool was developed to achieve the same and resulting in the Augmented Dataset as shown in Figure 4.3.

The basic code was inspired from the online image data augmenter of *Keras* [10], which include

- Horizontal and vertical flip
- Rotation, scaling, translation
- Shear
- Swirl

FIGURE 4.3: A snapshot of the augmented dataset

### 4.1.4 Backgrounded and Augmented Dataset

In order to generate a dataset that is more diverse in terms of Pokemon angles, orientations, scale, position, background, etc., we generated backgrounded and augmented the dataset. Initially 19 common backgrounds found in Pokemon battles, as shown in Figure 4.4, were gathered and the image augmentation tool *Keras* [10] was used to generate more than 40 different occluded and convolved images for each of the 150 Pokemon using the Raw Vectorized Dataset as the base, as shown in Figure 4.5. Later these Pokemon images were superimposed on the handpicked backgrounds to generate the Backgrounded and Augmented Dataset as shown in Figure 4.6. This dataset was especially designed for the *Convolutional Neural Networks (CNN)*, as the performance and the generalization of a CNN mainly depends upon the diversity of the images for each class and the size of the dataset.

FIGURE 4.4: Pokemon battle background images

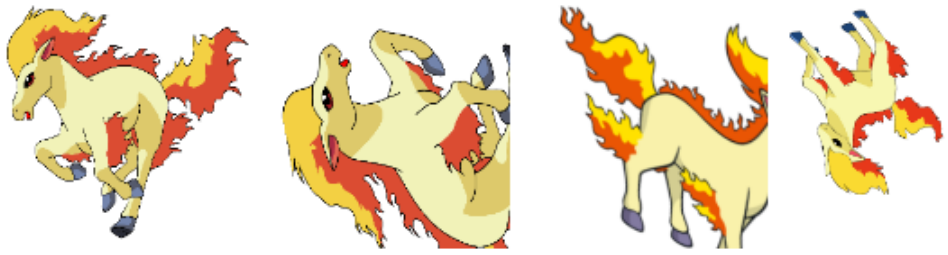

FIGURE 4.5: Raw vectorized augmented images

FIGURE 4.6: Backgrounded and augmented dataset

## 4.2 Additional Augmentation

In addition to the augmentations mentioned above, namely rotate, shear, swirl, displace, flip, and mirror, we also decided on including certain other modifications, after analysing the nature of battle images. We saw many images had a white or black fade, and were either blurred or some pixels were absent in them. To model these irregularities, we added to the previous augmentation model, and included random intensities of black as shown in Figure 4.7 and white tints as shown in Figure 4.8, blurring as shown in Figure 4.9 and pixel removal as shown in Figure 4.10 in the existing backgrounded and augmented dataset and expanded it to 170% its original size. On training with a CNN, this dataset proved weaker in terms of accuracy, and also while deployment it produced undesirable results, hence we decided to drop it.

FIGURE 4.7: Black tinted

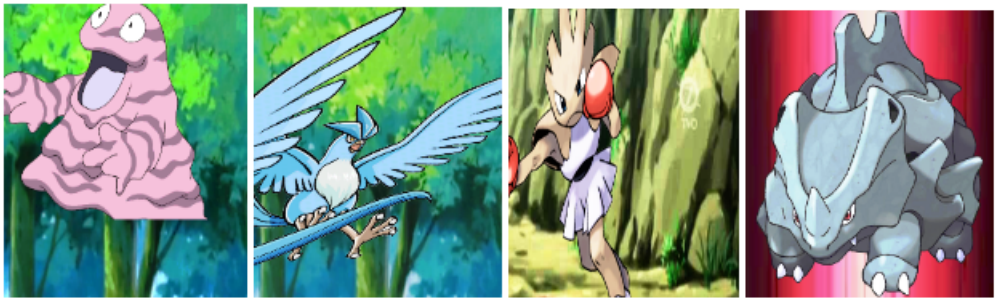

FIGURE 4.8: White tinted

FIGURE 4.9: Blurred image

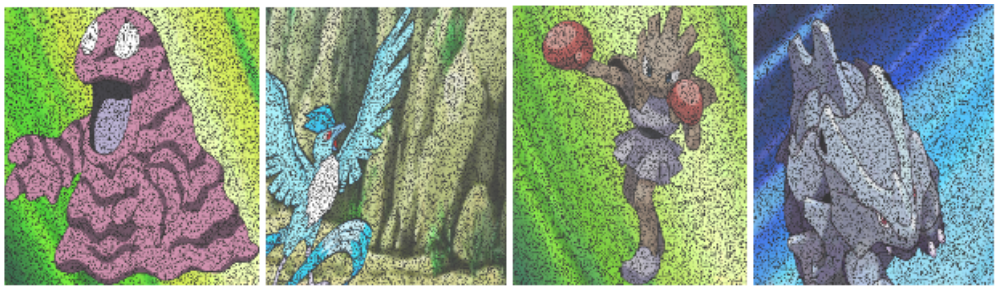

FIGURE 4.10: Pixels removed

## 4.3 Pokemon Presence Dataset

### 4.3.1 Battle Images Dataset

We put together a collection of Pokemon Battle Images (Figure 4.11) scraped from `pokemon.wikia.com` [11] which had a season-wise collection of snapshots from the anime. We initially intended to use this for the attacks classifier but later we found better utility in employing this dataset for the Pokemon/Parts/Other classifier (more on this later). By running an undoctored version of *Selective Search Algorithm* on this dataset, we were left with various segments of each image, which we manually segregated to one of three categories: Pokemon, Parts, Other. The *Pokemon* label is assigned if the segment cropped contains a whole Pokemon, as shown in Figure 4.12; the *Parts* label is assigned if a part of the Pokemon is visible in the segment, as shown in Figure 4.13 and the *Other* label is assigned if there are more than one or no Pokemon present in the segment, as shown in Figure 4.14 .

FIGURE 4.11: Battle images scraped from pokemon.wikia.com

FIGURE 4.12: Segmented *Pokemon* from battle images

FIGURE 4.13: Segmented *Parts* images from battle images

FIGURE 4.14: Segmented *Other* images from battle images

## 4.4 Attack Images Dataset

After studying how attacks are represented in a battle image, we found that most attacks are not situated in a local region in an image. The essence of the above study is that, a bounding box cannot be drawn on attacks executed by Pokemon during battle. We then considered attacks to be scenes; for example, thunderbolt in an image would be a different scene than flamethrower; the former would be mostly yellow and the latter red. In order to formulate the attacks dataset, we did the following.

FIGURE 4.15: Attack dataset

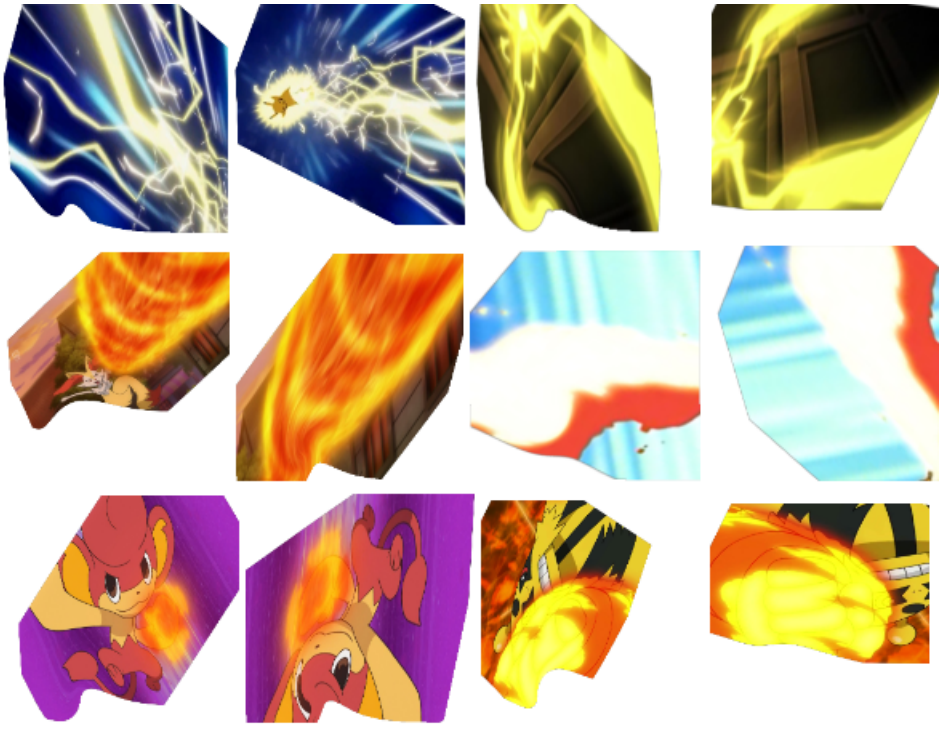

FIGURE 4.16: Augmented attack dataset

### 4.4.1 Augmented Attacks Dataset

We scraped Pokemon attack images from `Bulbapedia` again, which had a maximum of four images per attack. As deep learning architectures require larger datasets, we used the *Keras* data augmentation tool [10] to expand the size of the dataset by a factor of 25. Each attack image now had 25 augmentations to it, resulting in a dataset of size $144 \times 4 \times 25$, which is a total of 14400 images.

## 4.5 Captioned Dataset

The ultimate aim of our system is to caption or describe the scene, a Pokemon battle sequence in a given image. To build this module of our system, we need battle images that have captions assigned to them and this forms the captioned dataset. In order to collect this dataset, initially the subtitled episodes of the Pokemon series were collected. However, as the Pokemon series was telecasted during the 90's, they were directly aired on television, all the episodes were dubbed in English, and no CDs or raw copies of the series were released. As a result, we were not able to find the subtitled episodes. So we opted to do speech processing of the audio track from the episodes and convert it to text and caption each scene of the video. For this purpose, the CMU Sphinx [12] project was studied and pre-trained language models for English were used to convert speech to text. However, this approach did not achieve the

intented result, as the language model did not have most of the Pokemon world words and it was not able to recognize the Pokemon sounds too.

## 4.6 Dataset Summary

TABLE 4.1: Dataset Summary

| Dataset | Classes | Train | | Test | | Total no. of images |
|---|---|---|---|---|---|---|
| | | Images per class | No. of images | Images per class | No. of images | |
| Raw Vectorized Dataset | 150 | 9 | 1349 | 4 | 600 | 1949 |
| Backgrounded Dataset | 150 | 135 | 20209 | 54 | 8100 | 28309 |
| Augmented Dataset | 150 | 33 | 4950 | 14 | 2100 | 7050 |
| Backgrounded and Augmented Dataset | 150 | 628 | 94137 | 269 | 40371 | 134508 |
| Battle Images Dataset | 3 | Full - 210<br>Part - 1666<br>Other - 925 | 2801 | Full - 62<br>Part - 499<br>Other - 277 | 838 | 3639 |
| Augmented Attacks Dataset | 144 | 103 | 14832 | 44 | 6356 | 21188 |

## Dataset used

After experimenting, we settled on using the Backgrounded and Augmented Dataset for Pokemon classification and the Augmented Attacks Dataset for attacks classification.

CHAPTER 5

# CHARACTER LOCALIZATION AND SEGMENTATION

In this chapter, we discuss the Selective Search Algorithm which is used to segment images and localise Pokemon. Further, fine-tuning techniques for the algorithm are discussed in order to improve its results. A Pokemon/Parts/Other classifier is discussed as it helped circumvent the undesirable effects of Selective Search Algorithm.

## 5.1 Selective Search Algorithm

Object localization and segmentation, given an image, has always been an active problem of research and *Selective Search* [2] is one such algorithm for this purpose. Given an image, it proposes various regions of interests that are likely to have an object within it; this was also adopted in RCNN (Region based CNN) for proposing regions of interest which was the initial stage of the RCNN pipeline. It was mainly chosen as it is the fastest to compute and as it combined the advantages of exhaustive search and segmentation. Like segmentation, the image structure was used to guide sampling and as in exhaustive search, it captures all possible object locations invariant to size and scale. Later, hierarchical grouping of region proposals based on colour, texture, size, fill, etc. is performed to propose the best regions of interest, as shown in Figure 5.1. Moreover, the *Selective*

*Search* algorithm uses a data-driven approach to propose objects and pre-learnt models have also been released which can be fine-tuned to suit our needs.

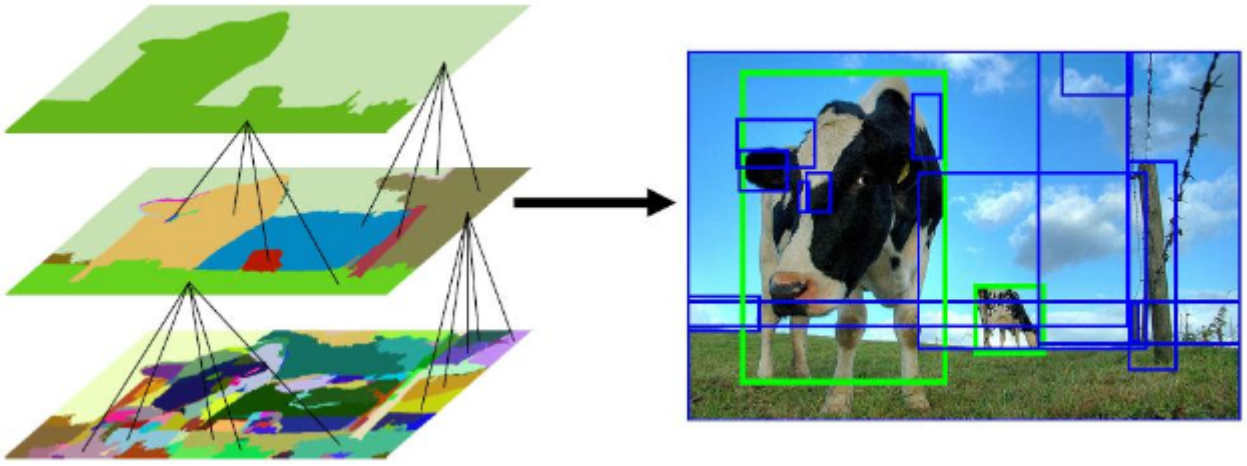

FIGURE 5.1: The overall working of Selective Search algorithm

### 5.1.1 Improvements

#### Fine tuning Selective Search

In order to improve the region proposals to suit the purpose of identifying Pokemon in battle images, the model has 4 parameters to be fine-tuned. To find the optimal combination of these, a greedy grid search approach over these 4 parameters was performed and the best combination is as listed below for a $512 \times 512$ image [2].

- **Scale**: higher the value, larger the clusters in felzenszwalb segmentation (350, 450, 500)
- **Sigma**: width of Gaussian kernel for felzenszwalb segmentation (0.8)

- **Min size**: minimum component size for felzenszwalb segmentation (30, 60, 120)
- **Min area**: minimum area of a region proposed (2000)

Once these parameters were decided, the regions proposed were relevant enough to suit our purpose of Pokemon localization, but the number of regions proposed were really high and mostly approximations and generalizations of each other, as shown in Figure 5.2. In order to reduce the number of regions proposed and to increase the quality of the region proposals, the following methods were devised:

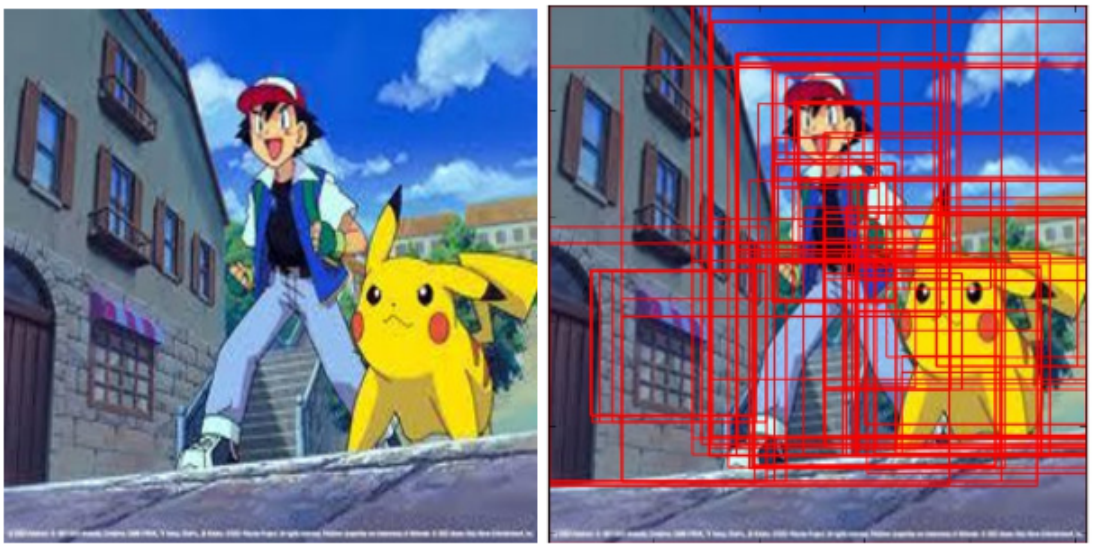

FIGURE 5.2: Selective search with best parameter combination for Pokemon

### Merge Concentric Proposals

Most of the proposals were focusing on the same object with just small variations in the position and areas being covered. Such proposals were

merged together and replaced by the mean rectangle of all the concentric proposals.

### Contained Boxes Removal

Some other proposals were subsets of overall Pokemon regions, each part of the Pokemon was also proposed along with the full Pokemon. So, all of these subsets were removed and only the overall proposals were retained, as shown in Figure 5.3

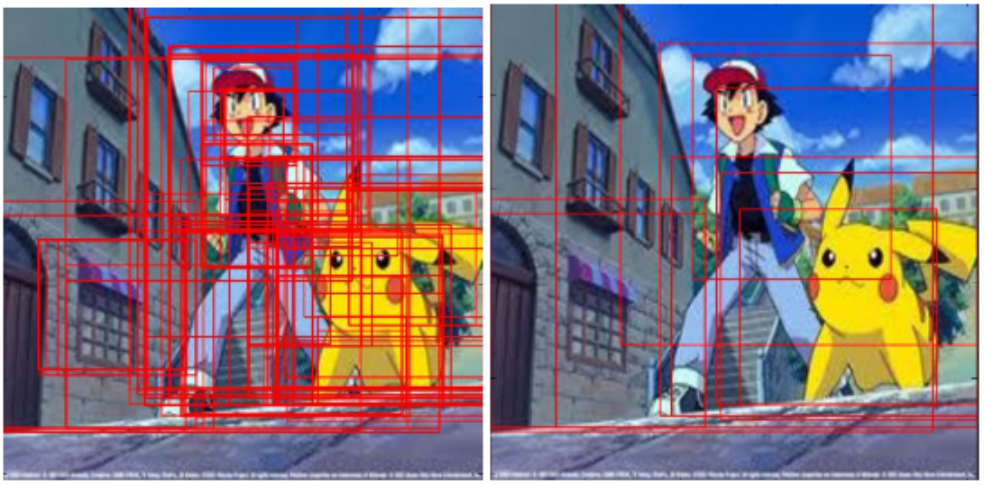

FIGURE 5.3: After merging concentric boxes and removing contained boxes

### Draw Super Box

Other proposals were overlapping each other such that a single Pokemon was proposed as two different overlapping regions. The percentage overlap of such proposals was calculated and thresholded at 20, all those pairs of regions having more than 20 percent overlap were replaced by a single minimal super box that bounded both the proposals (Figure 5.4).

FIGURE 5.4: After drawing super boxes

### 5.1.2 Results Obtained

Given a battle image, the above proposed localization algorithm proposes regions of interest, ranging from 2 to 10 proposals per image. These proposals also include non-Pokemon objects which, when a trained CNN is applied on them, gets low scores. Given the regions, by enforcing a thresholded prediction and using the trained CNN model, we are able to successfully localize and identify the Pokemon present in a given battle image, as shown in Figure 5.5.

FIGURE 5.5: The segmented images on which CNN will be applied

## 5.2 Pokemon Presence Detection

### 5.2.1 Pokemon/Part/Other (P/P/O) Classifier

Selective Search was giving regions which would not make sense to the Pokemon Classifier, but still classified as one of the 150 Pokemon. To circumvent this undesirable effect, we trained a classifier which detects whether the region cropped by Selective Search is a *Pokemon*, or a *part of a Pokemon* or *Other* which constitutes multiple Pokemon in the same segment or no Pokemon at all. We used *GoogLeNet's CNN architecture* and froze the learning of all the initial layers. The final fully connected layer's learning rate was determined empirically and its weights, allowed to learn by training on the dataset mentioned above. The classifier was able to achieve a 75% accuracy given we had to only classify images into three classes.

### 5.2.2 Grouping Regions

From the output of the classifier, the region proposals with *Part* and *Pokemon* (Figure 5.6) labels are merged (Figure 5.7) to form a complete bounding box around a Pokemon. We do this to the effect that the result of Selective Search is tailored to discover Pokemon in the image as a whole. Previously ungrouped regions were sprawled all over the place in an image, and the consequence of the P/p/o classifier is a problem specific robustness in cropping Pokemon.

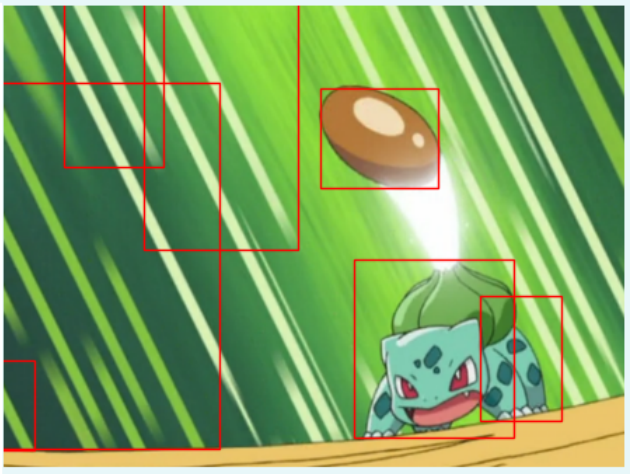

FIGURE 5.6: Pokemon, Part and Other bounding boxes

FIGURE 5.7: Pokemon and Part boxes combined

CHAPTER 6

# CHARACTER CLASSIFICATION

In Chapter 6, we discuss various Feature Extraction methods such as use of 1D RGBA Vectors, 3D Colour Histograms & Histogram of Oriented Gradients, and Pokemon Classification methods including *k*-Nearest Neighbour and Convolutional Neural Networks. Further, we explain a concept called Connected Colour Components used to visualise cartoon images as graphs and compare the same for similarity.

## 6.1 1D RGBA Vectors

### Feature Extraction

The Backgrounded Dataset has 1,50,000 images — each image has $1,024 (32 \times 32)$ pixels, with a RGBA value associated with each pixel, resulting in a total size of $1024 \times 4 = 4,096$ pixels per image. This two dimensional array of RGBA values was reshaped into a single dimensional integer array of size 4,096 for each image, resulting in a dataset of size $1,50,000$ with 150 classes. The resulting 1D RGBA vectors were used to represent the images in the dataset.

### Classification Algorithms

Then the Backgrounded Dataset was split into training and test data in 70:30 ratio. `NumPy` was used to load the data as `NumPy` arrays and `SciPy` was used to train the classifiers. The following training models were used.

$k$-NN: Using $k = 1$, a 1-NN classifier was trained which evaluates the distance between two image vectors based on the Euclidean distance and it achieved a 96% accuracy on the test data.

## 6.2 3D Colour Histograms

### Feature Extraction

A feature extraction technique known as *3D colour histograms* [4] was applied on the *Backgrounded Dataset* to convert any given $32 \times 32$ image of 1024 pixels into a 3D histogramic vector. For each of the three RGB channels, $N$ equally divided bins over the $[0,255]$ range were created and each of the 1024 pixel values in the range of $[0,255]$ were binned to one of these bins to form histograms. Thus a 3-channel image with $N$ bins yielded a $N \times N \times N = N^3$-sized feature vector. These features were extracted for every image in the dataset and were used to index the images, as shown in Figure 6.1.

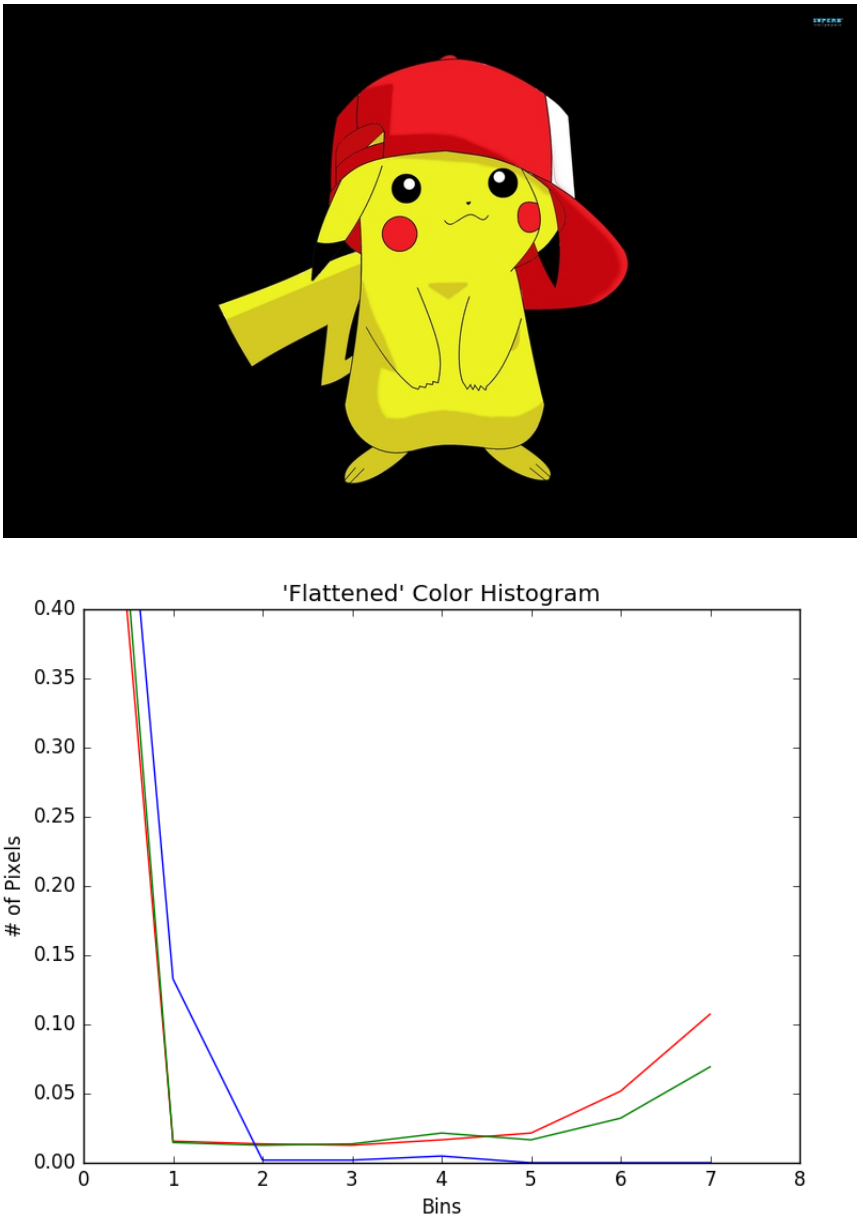


FIGURE 6.1: A sample Pikachu image and its flattened 3D histogram

## Classification Algorithm

Nearest Neighbour classifier was run over these vectors. The best result was obtained for $N = 8$-bin 3D histograms, using the Bhattacharyya distance [12], The final accuracy recorded was 41.2%.

**Reasons for low accuracy**

The 3D histogram was merely a frequency coded colour representation of an image. Therefore Pokemon with similar colour frequency distributions got misclassified into each other's class. Moreover, Pokemon have a unique characteristic where those belonging to the same type (grass, water, fire, etc.) have similar colour distributions. For example, Pikachu and Electabuzz have similar color frequency distribution as shown in Figure 6.2.

FIGURE 6.2: Pokemon having similar colour frequency distributions (Pikachu and Electabuzz)

## 6.3 Histogram of Oriented Gradients

### Feature Extraction

Histogram of Oriented Gradients (HOG) is one of the most famous feature extraction techniques for images by Navneet Dalal and Bill Triggs [13]. It works on grayscale images and is mainly used for object detection

purposes. This technique counts occurrences of gradient orientations in localized portions of an image. The *Scikit-Image* implementation was used for computing the HOG features for an image and this was applied on $32 \times 32$ images. The default parameter settings were used to compute the HOG features as per [13]. A visual representation of the various steps involved in computing HOG features has been illustrated using Figures 6.3 to 6.6.

FIGURE 6.3: Initial gray scale Image

FIGURE 6.4: Magnitude of Gradient

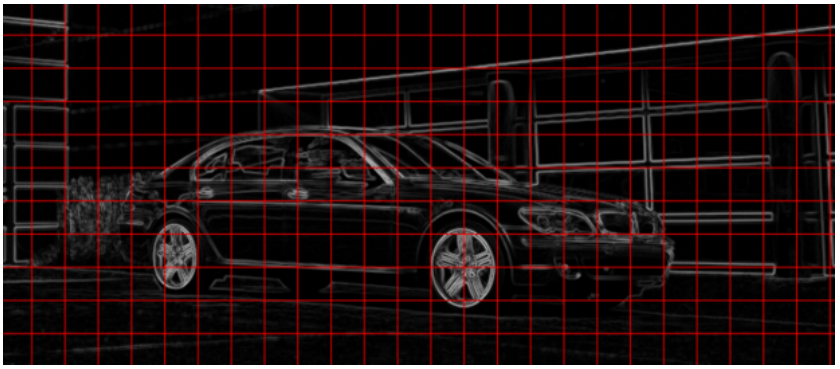

FIGURE 6.5: Cell division Example

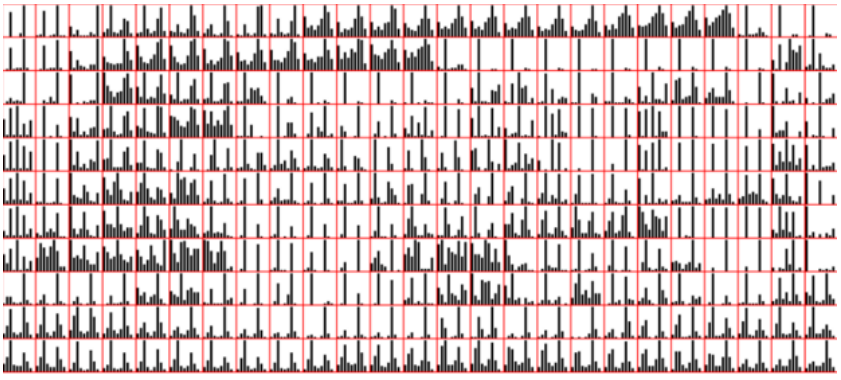

FIGURE 6.6: HOG Descriptor (Normalized inside each cell)

## Classification Algorithms

Once the features were extracted, *Nearest Neighbour Classifier* was used with standard Euclidean distance, resulting in an accuracy of 85.5%. Further, using a $k$-NN classifier, cross validation and a hyper-parameter, $k = 12$, an accuracy of 97.2% was achieved. Various other classification algorithms were used such as Random Forest, Perceptron, Naive Bayes, etc. Except for *Random Forest Classifier* which achieved an accuracy of 98.4%, the other results were not promising. The results are shown in Table 6.1.

| Classifier Description | Result (Accuracy in %) |
|---|---|
| $k$ Nearest Neighbour ($k = 12$) | 97.2 |
| Random Forest (100 estimators) | 98.4 |
| Perceptron (50 iterations) | 75.6 |
| Passive Aggressive Classifier (50 iterations) | 85.3 |
| SGD Classifier | 77.5 |

TABLE 6.1: Benchmark results with HOG

**Drawbacks**

The high accuracy yielded was limited to the Backgrounded Dataset. However, when applied to other battle images, accuracy dropped. Hence, the models built seemed to overfit the dataset. For example, if a Pikachu was positioned like a Rhyhorn, it misclassified it as Rhyhorn, without consideration for its colour information, the corresponding case is illustrated in Figure 6.7.

FIGURE 6.7: Example image with same orientation but different colour

## 6.4 Connected Colour Components

Cartoon images have the properties that they have sizable blocks of color and sharp transition between the blocks of colors. In contrast, real life photos or images are continuous in tone and quality and there is a relatively flatter gradient in color transition. Using these properties of cartoon images to our advantage, we developed an algorithm to reduce the representation of an image from raw RGB pixel data to a graph which is described below.

Graph $G$ extracted from a cartoon image has the colors (RGB value) of the input image as its vertices (The words *color* and *vertex* are used interchangeably throughout to mean the same thing). This entails a limit of $256^3$ possible vertices a graph can have (given there are only $256^3$ digitally representable colors) although we do not expect a graph to be even nearly that large. Each vertex has a *node weight* attribute attached to it. The value of node weight of a particular vertex is the fraction of the total image covered by that vertex's color. Expanding on this, the value of node weight is the total number of pixels of that color divided by the total number of pixels in the image. The *edges* between two vertices in the graph (in this case, edges between any two colors) represent adjacency between the two colors in the image. More clearly, there is an edge between color $A$ and color $B$, if color $B$ is present in one of the eight adjacent pixels surrounding color $A$. Adding to the detail, the graph is undirected. Each edge also has a *edge weight* which is computed as follows. We add a value of 1 to the weight of the edge between two colors $A$ and $B$, if $B$ is present in one of the eight cells adjacent to $A$, and vice

versa. This results in adding the edge weight twice for a single adjacency, so we divide the final edge weights by 2. Finally, each edge weight is normalized by dividing its value with the sum of the weights of all the edges in the graph.

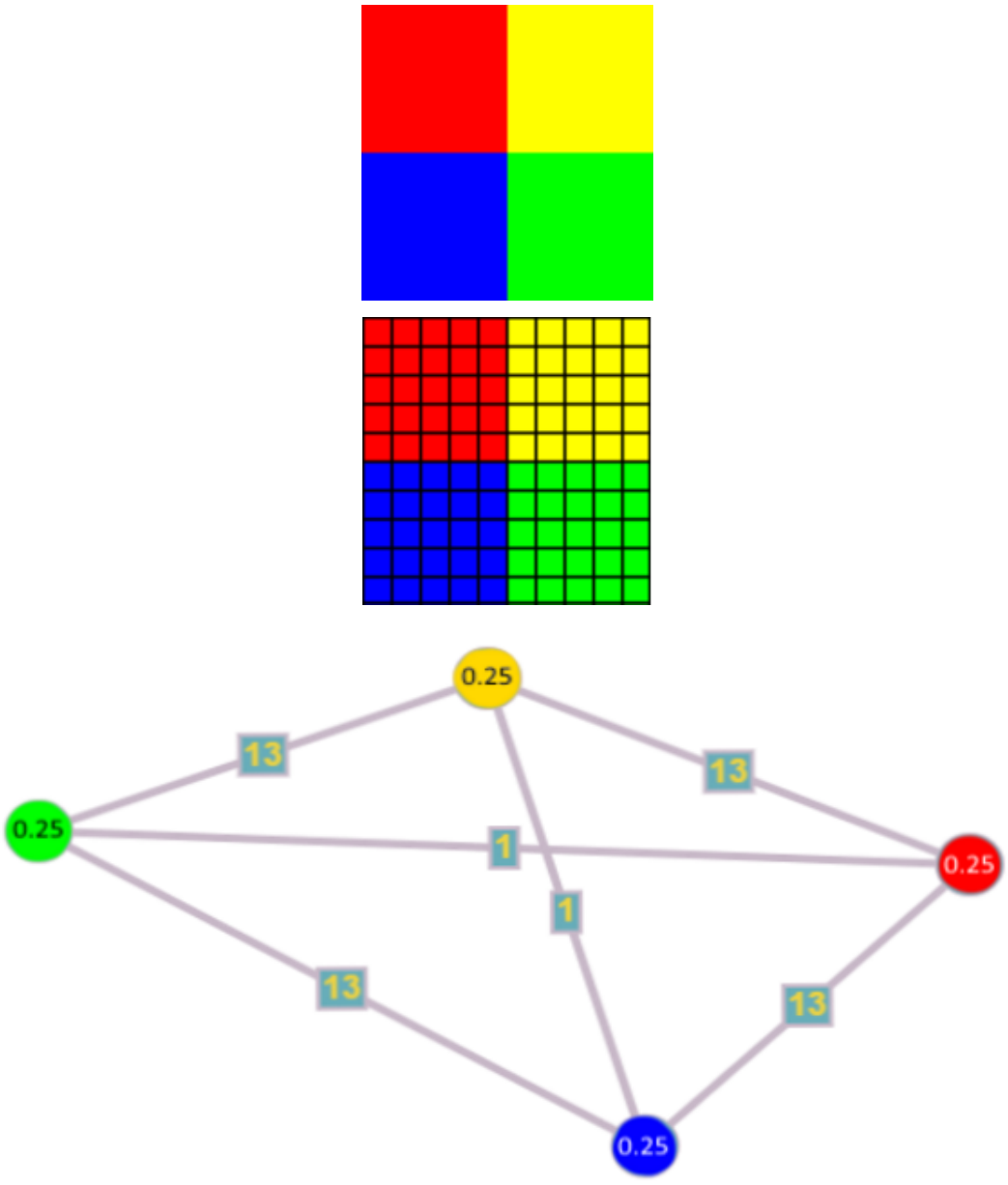


FIGURE 6.8: The CCC graph produced for the above input image

**Input:** $I$: Image represented as 2D array of (RGBA) tuples
**Output:** $G$: CCCGraph (undirected) with node attribute (count) and edge weights

```
begin
    G ← null
    pixelCount ← 0
    for i,j ∈ I.height,I.width do currentPixel ← I[i,j] ;
    if currentPixel is not transparent then
        pixelCount ← pixelCount + 1
        currentPixel ← bandLimit(currentPixel, 8)
    end
    if currentPixel ∉ G.nodes() then
        G.add_node(currentPixel, count ← 1)
    else
        G.node[currentPixel]['count'] +← 1
    end
end
```

**Algorithm 1:** CreateGraph: Compute G

**Distance Metric:** In order to compute how similar (or how dissimilar) two graphs are (which in turn generalizes to how similar two Pokemon images are), we also developed a *distance metric* to measure the distance between two graphs. A cursory explanation is given in Algorithm 3,

```
for all pixel indices a,b adjacent to i,j do
    neighborPixel ← I[a,b] ;
    if neighborPixel is not transparent then
        neighborPixel ← bandLimit(neighborPixel, 8) ;
        if neighborPixel ∉ G.nodes() then
            G.add_node(neighborPixel, count←0);
        end
        if Edge(currentPixel, neighborPixel) ∉ G.edges() then
            if currentPixel != neighborPixel then
                G.add_edge(currentPixel, neighborPixel, weight←1) ;
            end
        end
    end
end
for node ∈ G.nodes() do  G.node[node][count] /= pixelCount ;
totalEdgeWeight ← 0 ;
for edge ∈ G.edges(data=True) do  totalEdgeWeight +← edge[weight] ;
for edge ∈ G.edges(data=True) do  edge[weight] /= totalEdgeWeight ;
return G;
```

**Algorithm 2:** CreateGraph: Compute G (continued)

followed by its formal definition.

```
Input: G1: CCCGraph
Input: G2: CCCGraph
Output: distance: float
begin
    edge_distance ← Compute_Edge_Distance(G1.Edges, G2.Edges);
    node_distance ← Compute_Node_Distance(G1.Nodes, G2.Nodes);
    distance ← edge_distance + node_distance;
    return distance
end
```

**Algorithm 3:** Compute Distance: Find distance between 2 CCCGraphs

We start by defining how and when two graphs are exactly similar. This occurs when both have exactly the same *nodes*, the same *node weights* for the corresponding nodes, the same *edges*, the same *edge weights* for the corresponding edges. The distance between two graphs is the sum of the distances between their corresponding edges, and the distances between their corresponding nodes described by the algorithm below.

Algorithm 4 (`Compute_Edge_Distance`) was used to compute the edge distance between two given CCC graphs. The Euclidean distance between each of the `CommonEdgePairs` of the two graphs was estimated and added to current `edge_distance`. Further, Euclidean distance between the `UncommonEdges` and the origin was estimated and added to the edge_distance.

Algorithm 5 (`Edge_Intersect`) was used to identify `Common_Edge_Pairs` from two sets of edges by comparing the corresponding node pairs between which the edge exists.

Algorithm 6 (`Compute_Node_Distance`) was used to compute the node distance between two given CCC graphs. The difference in weight between nodes of each node pair belonging to the `CommonNodePairs` of the two graphs was estimated and added to current `node_distance`. Further, the node weight of each of the `UncommonNodes` was added to the `node_distance`.

Algorithm 7 (`Node_Intersect`) was used to identify `Common_Node_Pairs` from two sets of nodes by directly comparing them. The identified node pairs were stored in a $2D$ array.

**Input:** E1: Set of edges where each edge e is of the form (n1, w, n2)
**Input:** E2: Set of edges where each edge e is of the form (n1, w, n2)
**Input:** 1: Node with node weight n1.weight
**Input:** n2: Node with node weight n2.weight
**Input:** w: Weight of edge
**Output:** edge_distance: float
**begin**
    CommonEdgePairs ← Edge_Intersect(E1, E2)
    edge_distance ← 0
    **for** edgePair ∈ CommonEdgePairs **do**
        p1 ← (edgePair[0].n1.weight, edgePair[0].w,
          edgePair[0].n2.weight)
        p2 ← (edgePair[1].n1.weight, edgePair[1].w,
          edgePair[1].n2.weight)
        edge_distance ← edge_distance + Euclidean_Distance(p1, p2)
    **end**
    UncommonEdges ← (E1 - E2) (E2 - E1)
    **for** edge ∈ UncommonEdges: **do**
        p ← (edge.n1.weight, edge.w, edge.n2.weight)
        edge_distance ← edge_distance + Euclidean_Distance(p, 0)
    **end**
    return edge_distance
**end**

**Algorithm 4:** Compute_Edge_Distance : Find edge distance between 2 CCCGraphs

**Implementation**

The non-superimposed image dataset was resized to 100 × 100 pixels and fed to Algorithm 1 and 2 (`CreateGraph`) to generate the graphs of the 1155 images in the dataset. These graphs along with their labels were stored

**Input:** E1: set of Edges
**Input:** E2: set of Edges
**Output:** L: 2n array of Edge Pairs where where $\forall$ a $\in$ L; a[0] = a[1]
**begin**
    L $\leftarrow$ [];
    **for** e1 $\in$ E1 **do**
        **for** e2 $\in$ E2 **do**
            **if** e1.n1 = e2.n1 and e1.n2 = e2.n2 **then**
                L.append((e1,e2))
            **end**
        **end**
    **end**
    return L
**end**

**Algorithm 5:** Edge_Intersect : Return common edge pairs between two edge sets

in a pickle file. With a 70-30 split for training and test data, we ran a 1-NN (Nearest Neighbor) classifier on the dataset using the `ComputeDistance` method as distance metric.

**Results Obtained**

The resulting accuracy of 5% was not impressive and not close to any of the other state-of-the-art methods. We aim to modify this and hope to raise the bar to present comparable results in the future.

**Input:** N1: set of nodes of form n, its weight accessed by n.weight
**Input:** N2: set of nodes of form n, its weight accessed by n.weight
**Output:** node_distance: float
**begin**
    node_distance ← 0;
    CommonNodePairs ← Node_Intersect(N1, N2);
    **for** nodePair ∈ CommonNodePairs **do**
        node_distance += (nodePair[0].weight - nodePair[1].weight)
    **end**
    UncommonNodes ← (N1 - N2) (N2 - N1);
    **for** node ∈ UncommonNodes **do**
        node_distance ← node.weight;
    **end**
    return node_distance
**end**

**Algorithm 6:** Compute_Node_Distance : Returns node distance between two CCCGraphs

## 6.5 Convolutional Neural Networks

The Convolutional Neural Network [3] is a Deep Learning method that has been gaining popularity recently due to its exceptional performance on image based machine learning tasks. The very concept of CNN is inspired from the actual working of the human eye and how our visual neurons process images that the eyes see. Another main strength of using a CNN is that it works as a feature extractor with deep learnt features as well as a classifier. The CNNs are able to model extremely complex polynomials in the process of modeling the data for classifying them. The deeper and wider the network, the more complex the polynomial being modeled will be.

**Input:** N1: set of Edges
**Input:** N2: set of Edges
**Output:** L: 2D array of Node Pairs where $\forall$ a $\in$ L; a[0] = a[1]
**begin**
    L $\leftarrow$ []
    **for** n1 $\in$ N1 **do**
        **for** n2 $\in$ N2 **do**
            **if** n1 = n1 **then**
                L.append((n1,n2))
            **end**
        **end**
    **end**
    return L
**end**

**Algorithm 7:** Node_Intersect : Return common node pairs between two node sets

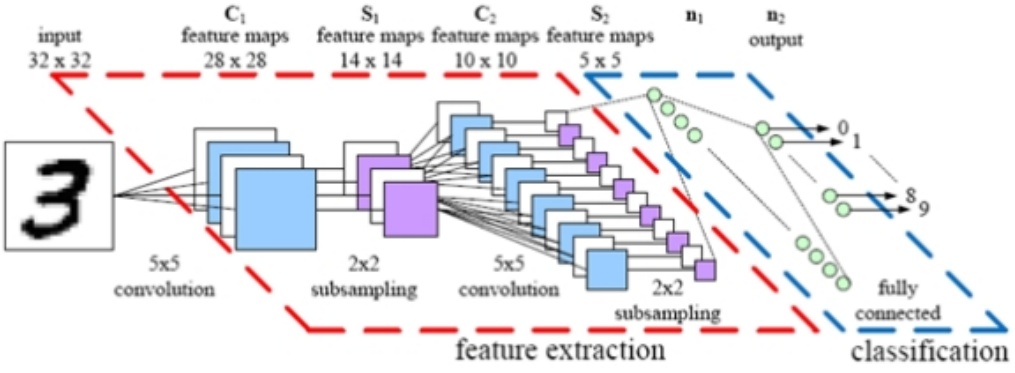


FIGURE 6.9: The overall layout of a simple CNN architecture [4]

### 6.5.1 Why Deep Features?

Generally, the way images are vectorized into features has always been hand-crafted, but now with increasing problem complexity, these

deep-learnt features are capable of adapting themselves on focusing about, what to look in the images, given the requirements, instead of hand-crafting it. With just minimal or no pre-processing, a hierarchy of features as shown in Figure 6.10 can be learnt and with less effort. They have proved to outperform hand-crafted features in many cases and also behave invariant to various types of distortions and occlusions in images.

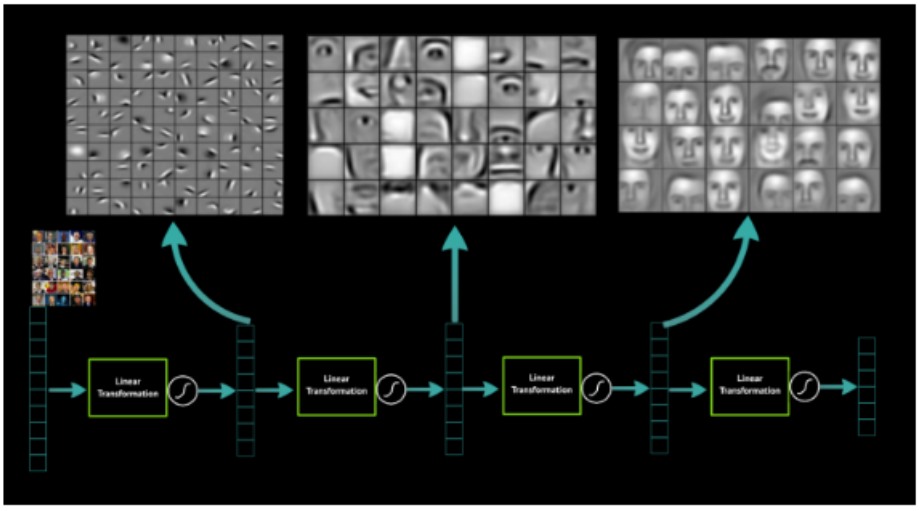


FIGURE 6.10: The hierarchy of Deep Learnt features [5]

### 6.5.2 The GoogLeNet Architecture

The CNNs consist of a variety of layers like Convolution, Pooling, Fully-Connected, LRN, ReLU, Dropout, Image Input, SoftMax Classifier, etc. Each of these layers has a number of parameters that can be configured. The overall blueprint that describes the hierarchy of the layers and the parameter configurations used in each layer is referred to as the CNN Architecture. Designing an architecture takes a lot of experimenting, time, effort and is moreover an art and hence predefined architectures built and open-sourced by organizations like Google, VGG, BVLC, etc. are

generally used as the base for building and training the network for our own data.

We use the `GoogLeNet` [3],[7] a 22 layers deep network (refer Figure 6.12), which is actually an incarnation of the convolutional neural network architecture codenamed *Inception* (refer Figure 6.11) developed by Google engineers that set the state of the art performance in the ImageNet Large-Scale Visual Recognition Challenge 2014. This network's architecture has a carefully crafted design that has a very deep and wide network but at the same time keeps under control the computational resources needed, thus making it one of the most efficient and powerful implementations of CNN and it was mainly designed for object classification and detection problems. So, we planned to adopt it for Pokemon Identification and Classification task.

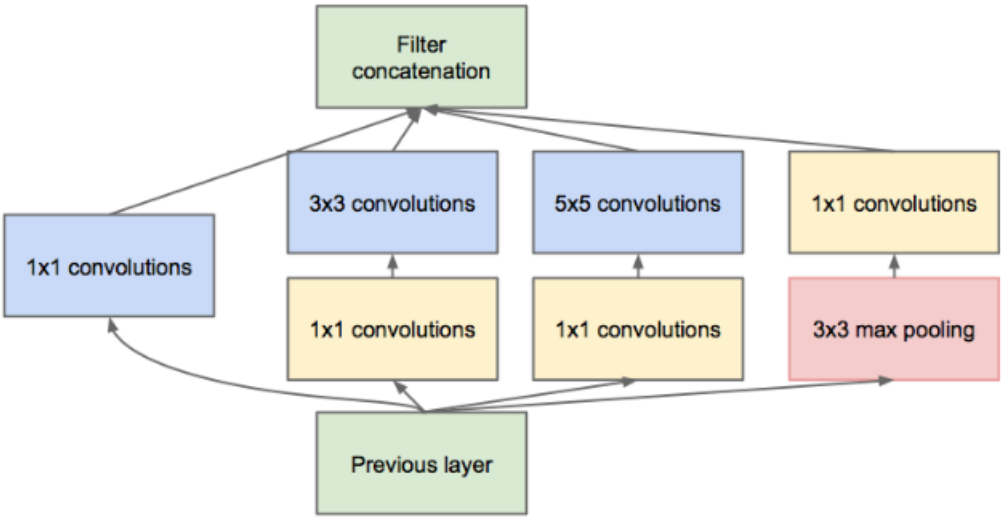


FIGURE 6.11: The Inception module of `GoogLeNet`

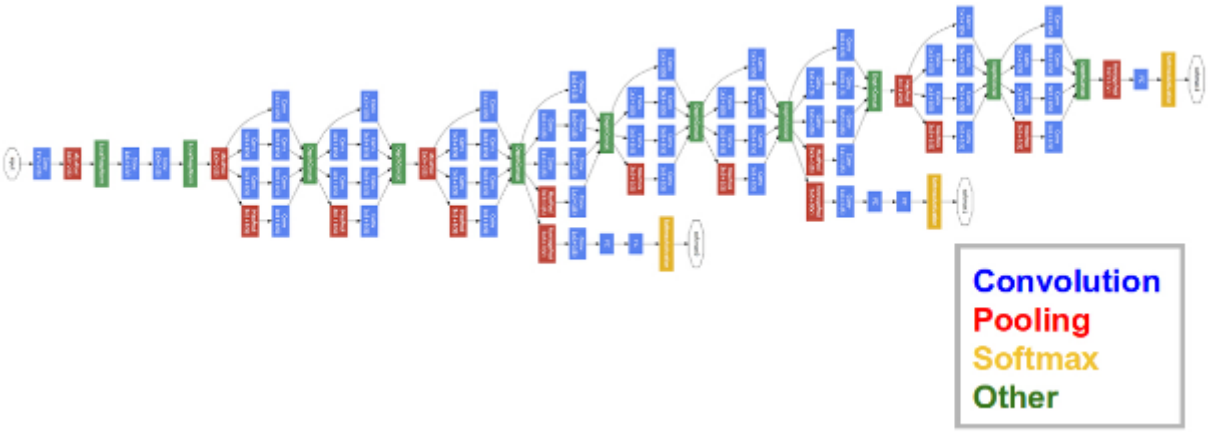


FIGURE 6.12: Overall GoogLeNet architecture (9 Inception Modules) [14]

### 6.5.3 Training from Scratch

In order to build, configure, train, test and deploy a convolutional neural network, the *Caffe Deep Learning framework* developed by BVLC [4] was used. Setting up this environment involved configuring Nvidia Driver 352 + Nvidia GPU + CUDA 7.5 + cuDNN 4.0 + Caffe framework and many other dependencies to be installed. Finally a GPU version of Caffe with `PyCaffe` Python interface was setup. The convolution, pooling and other operations are performed very efficiently and quickly on GPUs using CUDA rather than normal CPUs. The computation becomes 50 times faster than the normal CPU performed ones. Once this high performance, parallelizable Deep Learning environment was setup, the `GoogLeNet` training network was designed and configured based on [5], using Caffe's `protobuf` specification file. Then an SGD (Stochastic Gradient Descent) solver was configured and the deploy network was designed from the training network, using the same `protobuf`

specification file. The deploy network will be used when the model is used in realtime applications for classification purposes.

Once the network was built, the 1,50,000 images from the backgrounded and augmented dataset was scaled to 224 dimensions for fitting into the network specifications and the weights of the links in the network was randomly initialized. Once all of this was setup, backpropagation techniques were used to train the network with a *SGD solver* with a learning rate of 0.01 and step size of 320000. The batch size used was 128 with the network being tested for every 500 iterations and being trained for 66 epochs approximately. However, the loss function was fluctuating and overshooting when the network was being trained. Finally, it ended up in an accuracy of 6 percent.

### 6.5.4 Fine-tuning the Network using Pre-Learnt GoogLeNet Weights

Usually very deep architectures like `GoogLeNet` [5] were designed with extremely huge datasets in mind, that are hundreds of times bigger than our dataset. So, for us to train such deep architectures from scratch with just the 1,50,000 images we have, would only lead to poor results as above. This is where a concept called *Transfer Learning* comes into picture, where the weights and filters learnt by the convolutional neural networks on some other dataset could be fine-tuned and transfer-learnt to suit our needs. This concept works well because of the hierarchical feature learning of a CNN, as discussed above. The initial layers of a CNN

architecture try to learn image structures like small lines, image color blobs, etc. that make up a more complex image, which is something similar to a Gabor Filter. As the network progresses through the depth of it, it starts learning higher level features that become specific to the dataset being trained on. So, the initial layers of a CNN that has been trained on huge datasets like the `ImageNet` would have learnt really rich filters that becomes impossible to be learnt with smaller datasets like ours.

In order to achieve transfer learning in our case, the `GoogLeNet` was initialized with the pre-trained weights and filters that were trained on the `ImageNet` images [3] for setting the state of the art in the *ImageNet Large-Scale Visual Recognition Challenge 2014*. This approach of avoiding random weight initialization in a network has proved to be effective in almost all cases. With this in place, the networks parameters had to be modified for transfer learning to happen, the initial layers of the network where just the features were being learnt had the rich low level filters of the `ImageNet` images initialized. Therefore, the learning rate for such layers was set to 0 and as the network progressed into its depth, the already available base learning rates were doubled as they had to quickly cope with, in learning the Pokemon dataset specific features compared to the previous layers. Once this was done all the fully connected layers preceding the SoftMax classifier were initialized to random weights and 150 outputs in our case, as this has to be learnt from scratch. The previously designed SGD solver's parameters, shown in Figure 6.13, were modified such that the rate at which the network learnt was reduced using a learning rate of 0.001 and step size of 3200 with the same 66 epochs.

```
net: "googlenet/train_val.prototxt"
test_iter: 1407
test_interval: 4000
test_initialization: false
display: 40
base_lr: 0.001
lr_policy: "step"
stepsize: 32000
gamma: 0.1
max_iter: 10000000
momentum: 0.9
weight_decay: 0.0002
snapshot: 4000
snapshot_prefix: "bvlc_googlenet_pokenet"
solver_mode: GPU
```

FIGURE 6.13: The GoogLeNet SGD solver configuration (for fine tuning)

### 6.5.5 Going Deeper and Results Obtained

The fact about convolutional neural networks is that, as the architecture becomes deeper, the precision and recall of the model being trained increases, as deeper architectures form complex polynomials to model the data. In order to realize this statement, we split the `GoogLeNet` into 3 equal parts and tested its accuracy at three levels namely at $\frac{1}{3}$, $\frac{2}{3}$ and the full network (Figure 6.14). Along with this, the best results we have obtained for Pokemon classification has been tabulated below. The CNNs were way more superior than all of our other models that gave great accuracies around 95 percent, as they were able to generalize to battle images and not just overfit the training data. The classification results of CNN on sample Pokemon images is given in Table 6.2.

TABLE 6.2: Results obtained with CNN

| Levels of testing in the GoogleNet | Augmented Dataset | | Backgrounded and Augmented Dataset | |
|---|---|---|---|---|
| | Top 1 | Top 5 | Top 1 | Top 5 |
| Level 1 ($\frac{1}{3}$ of Network) | 74.39% | 91.8% | 79.22% | 93.99% |
| Level 2 ($\frac{2}{3}$ of Network) | 76.28% | 92.3% | 82.06% | 95.32% |
| Level 3 (Full Network) | 85.39% | 96.43% | 91.16% | 98.03% |

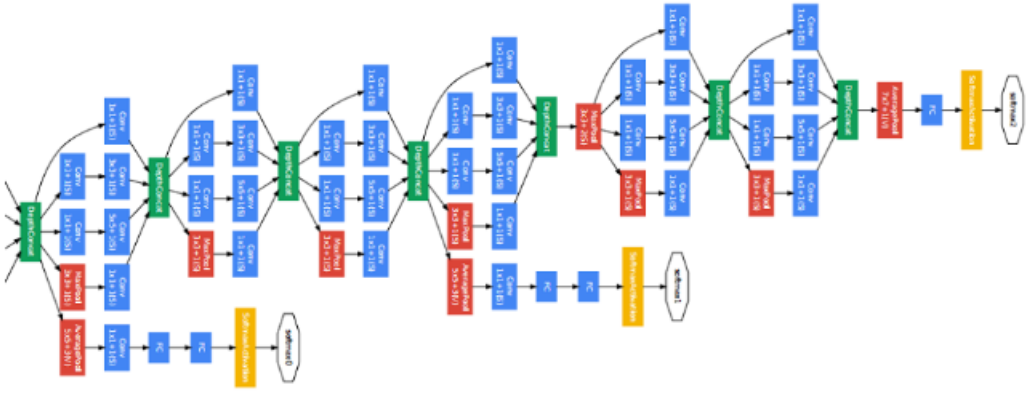

FIGURE 6.14: Going deeper with CNNs, network within a network (tested at three levels) [7]

| | | |
|---|---|---|
|  |  |  |
| 'Electrode'<br>'Voltorb'<br>'Chansey'<br>'Horsea'<br>'Gengar' | 'Blastoise'<br>'Rhydon'<br>'Rhyhorn'<br>'Golduck'<br>'Wartortle' | 'Haunter'<br>'Mankey'<br>'Nidorino'<br>'Arcanine'<br>'Seadra' |

Table 2. Classification results of Pokemon Images

### 6.5.6 Conclusion

Among all the approaches tried, deep learning works the best and gives the highest accuracy along with a good generalization to battle images. Hence, we chose to use `GoogLeNet`'s architecture with `ImageNet` weights to classify Pokemon.

CHAPTER 7

# CLASSIFYING AND FILTERING ATTACKS

This chapter describes Classification of Attacks using GoogleNet, which is initialized using MIT Places 205 Weights. Classification of Attacks is explained as a scene recognition process rather than an object recognition one. Further, we discuss Triple Generation and Caption Generation using Sentence Templates .

## 7.1 Classifying Attacks

While testing out the approaches for classifying Pokemon, we settled upon the *Convolutional Neural Network* as it had produced the best results. As a result, we digressed from testing and employing all other methods and directly used deep learning to achieve good results.

### 7.1.1 Convolutional Neural Network

All the arguments for using a CNN for classifying Pokemon also hold here. The only difference we perceived was that attacks were not present as localized segments in the image; they spanned the entire image, and were present as the orientation or pose of Pokemon in the image. We turned to considering attacks as whole scenes rather than objects in an image. The problem was transformed into a scene classification problem.

We had scraped attack images from `Bulbapedia`, numbering at about 4 per attack, as shown in Figure 7.1. Again, we used the data augmentation tool by *Keras* to augment and expand the dataset 25-fold. The resultant dataset was used to train the attacks classifier explained below.

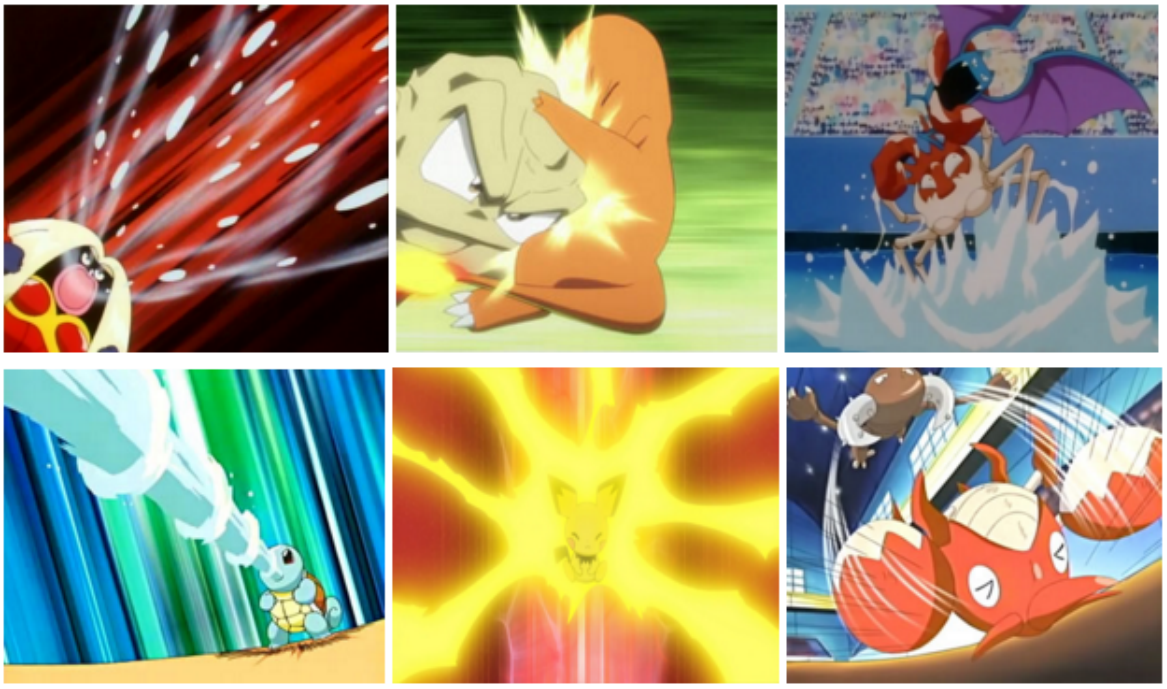

FIGURE 7.1: Pokemon attacks

#### 7.1.1.1 MIT Places 205 Weights

Fortunately, MIT had trained a dataset of scenes which had 205 scenes [7] using `GoogLeNet` and released its weights online for free. This was called the *weights of the MIT Places 205* [7] dataset and we used these weights as initialization for fine-tuning `GoogLeNet`. We froze all the layers except the final fully connected layer, by setting their learning rates to 0.

#### 7.1.1.2 Results Obtained

At the deepest layer of the network, we obtained and 88% top-5 accuracy and a 55% top-1 accuracy. The accuracy actually gets sharper, once we filter attacks according to the Pokemon-Attacks mapping (Figure 7.2).

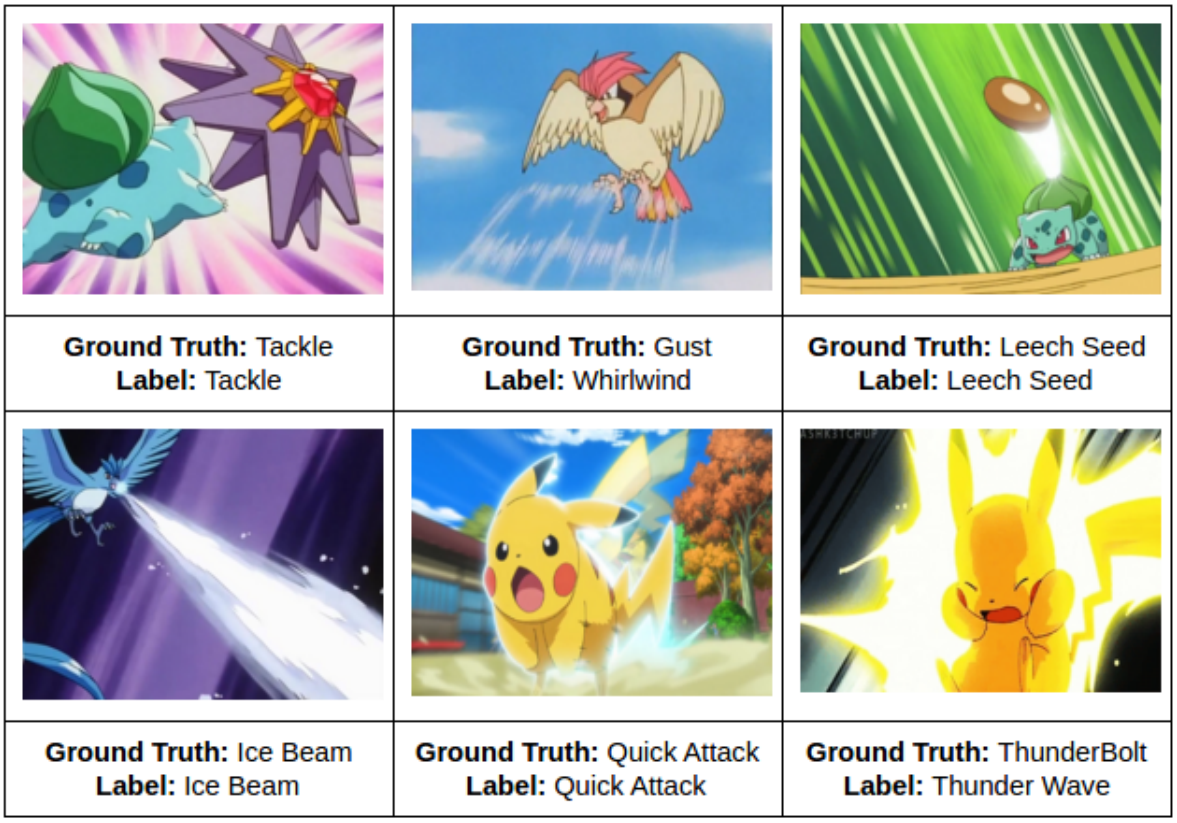


FIGURE 7.2: Attacks classifier results

## 7.2 Filtering attacks

We chose the domain of Pokemon as it is restrictive in the number of labels one can assign to attacks made by a single Pokemon. In simpler terms, each Pokemon can execute only a small number of attacks. We formed a database of Pokemon-Attacks mapping, and used this in removing attacks

which will not be executed by the Pokemon identified in the image, to make inference simpler.

## Examples

**Pikachu**: [ThunderBolt, ThunderWave, VoltTackle, IronTail, Tackle, Agility, Thunder, TailWhip, Growl, ElectroBall]

**Charmander**: [FlameThrower, Tackle, Ember, TailWhip, Growl, Scratch, SmokeScreen, FlameBurst, FireSpin, Inferno]

**Squirtle**: [WaterGun, Withdraw, Tackle, TailWhip, WaterPulse, HydroPump, IronDefense, SkullBash, Bubble, AquaTail]

CHAPTER 8

# TRIPLE GENERATION AND CAPTIONING

## 8.1 Triple Generation

To give an overview of the system at this stage (Figure 8.1), we have a localizer and segmenter; Selective Search, a Pokemon Classifier; `GoogLeNet` CNN with `ImageNet` weights, an Attack Classifier; `GoogLeNet` CNN with MIT Places 205 weights. The outputs at this stage are a set of Pokemon names and a list of attacks in the image.

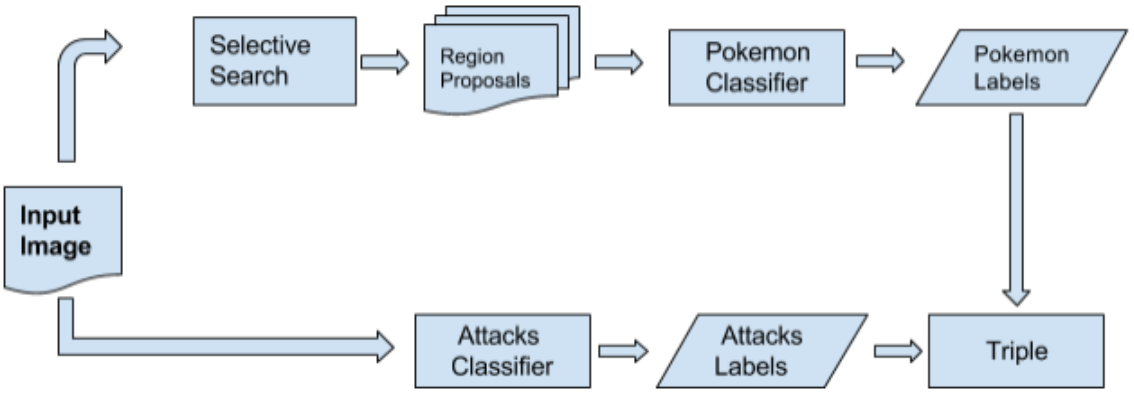


FIGURE 8.1: Concise view of system

These are got after thresholding the scores so that confidence level of output is high enough. We restrict our output to a maximum of two Pokemon per image and only one attack per image. After the thresholding is done, we finally produce the triple which is of the following structure.

<PokemonA, PokemonB, AttackC>

## 8.2 Captioning

### 8.2.1 Recurrent Neural Networks

An RNN requires that a vocabulary of words represented as one-hot vectors be fed in as input and it computes the probability of the $(n+1)^{\text{th}}$ word occurring in the sentence, given the probability that the previous $n$ words occurred in that particular sequence.

$$P(w_1,...,w_n) = \prod_{i=1}^{n} P(w_i|w_{i-1},...,w_1)$$

As the Pokemon world's vocabulary is restrictive and manageably finite, we tried to fabricate the vocabulary ourselves, with the names of the Pokemon, the attacks and some standard English language verbs, articles, prepositions, conjunctions, adjuncts and adjectives. Finally, this vocabulary is further restricted to removing the Pokemon and attack names which are not present in the image, after image classification. This is required for accurate language generation.

However, since RNN is a deep learning framework, it required a large corpus of data. We were unable to generate a corpus large enough so that RNN does not generate random babble, and so we dropped this idea.

### 8.2.2 Template Based Generation

While trying to generate a large body of text for an RNN, we were left with a very small text file of sentence templates, which we had planned to train

the RNN with. The sentence templates as shown in Figure 8.2, served as the framework to hold the results received from the Pokemon and Attacks Classifier, forming the natural language description of the image.

```
$pokemonA attacks $pokemonB using $attackC
$pokemonA attacked $pokemonB using $attackC
$pokemonA is attacking $pokemonB using $attackC
$pokemonA uses $attackC on $pokemonB
$pokemonA used $attackC on $pokemonB
$pokemonA is using $attackC on $pokemonB
$pokemonB is attacked by $pokemonA using $attackC
$pokemonB was attacked by $pokemonA using $attackC
$pokemonB is being attacked by $pokemonA using $attackC
$attackC is used by $pokemonA on $pokemonB
$attackC was used by $pokemonA on $pokemonB
$attackC is being used by $pokemonA on $pokemonB
$pokemonA has attacked $pokemonB using $attackC
$pokemonA has used $attackC on $pokemonB
$pokemonB has been attacked by $pokemonA using $attackC
$attackC has been used by $pokemonA on $pokemonB
$pokemonA attacks $pokemonB utilizing $attackC
$pokemonA attacked $pokemonB utilizing $attackC
$pokemonA is attacking $pokemonB utilizing $attackC
$pokemonA utilizes $attackC to attack $pokemonB
$pokemonA utilized $attackC to attack $pokemonB
$pokemonA is utilizing $attackC to attack $pokemonB
$pokemonB is attacked by $pokemonA utilizing $attackC
$pokemonB was attacked by $pokemonA utilizing $attackC
$pokemonB is being attacked by $pokemonA utilizing $attackC
$attackC is utilized by $pokemonA to attack $pokemonB
```

FIGURE 8.2: Snapshot of templates used for sentence generation

CHAPTER 9

# CONCLUSION

## 9.1 Summary

We have presented the detailed design and related algorithms for a system to automatically generate scenario descriptions for images. We have used the Backgrounded and Augmented Dataset for Pokemon Classification and the Augmented Attacks Dataset for attacks classification. We tried various approaches to classify Pokemon and observed that deep learning works the best and gives the highest accuracy along with a good generalization to battle images. Hence, we chose to use `GoogLeNet`'s architecture with `ImageNet` weights to classify Pokemon. Similarly, we completed attack classification using `GoogLeNet` along with weights of the MIT Places 205. Finally, we generated captions of the form <Pokemon_A,Pokemon_B,Attack_C> using sentence templates.

## 9.2 Future Prospects

The sentences formed in this project are of rigid structure owing to the hand written templates. This becomes unwieldy and loses generality when venturing into other datasets and applications. With a dataset of Pokemon battle images annotated with a set of captions for each image, we will be able to train a multi-modal RNN with inputs from a CNN to

directly generate sentences without having to use templates as containers for information triplets.

To further extend to context based sentence generation in our work, classifiers to detect the surrounding and environment can be employed, adding more detail to the caption.

With fast hardware and algorithms optimized for scaling up, we can enrich the web experience for the visually challenged by painting a picture of the images in their minds using words. When rewired to caption various frames in a video, we can achieve automated commentary for video such as commentating sports matches.

# REFERENCES


1. Aherne, Frank J. and Thacker, Neil A. and Rockett, Peter I. (1998) *The Bhattacharyya metric as an absolute similarity measure for frequency coded data*, Kybernetika, Vol. 34, No. 4, [363]–368

2. Dalal, Navneet and Triggs, Bill. (2005) *HOG Descriptor: Histograms of Oriented Gradients for Human Detection*, CVPR, Volume 01, Pages 886-893

3. Deng, J. and Dong, W. and Socher, R. and Li, L.-J. and Li, K. and FeiFei, L. (2009) *ImageNet: A large-scale hierarchical image database*, CVPR

4. Jia, Yangqing and Shelhamer, Evan and Donahue, Jeff and Karayev, Sergey and Long, Jonathan and Girshick, Ross and Guadarrama, Sergio and Darrell, Trevor (2014), *Caffe: Convolutional Architecture for Fast Feature Embedding*, 22nd ACM international conference on Multimedia, arXiv:1408.5093, 675 - 678

5. Szegedy, Christian and Liu, Wei and Jia, Yangqing and Sermanet, Pierre and Reed, Scott Anguelov, Dragomir and Erhan, Dumitru and Vanhoucke, Vincent and Rabinovich, Andrew (2015) *Going Deeper With Convolutions*, CVPR, arXiv:1409.4842, 1-9

6. Uijlings, Jasper R. R. and van de Sande, Koen E. A. and Gevers, Theo and W. M. Smeulders, Arnold W. M (2013) *Selective Search for Object Recognition*, IJCV, Volume 104 (2), page 154-171

7. Zhou, B. and Lapedriza, A. and Xiao, J. and Torralba, A. and Oliva, A. (2014) *Learning Deep Features for Scene Recognition using Places*

*Database*. Advances in Neural Information Processing Systems 27 (NIPS) spotlight

8. Pokemon, The Pokemon Company International, http://www.pokemon.com/us/
9. Bulbapedia, a community driven Pokémon encyclopedia, bulbapedia.bulbagarden.net
10. Keras, Francois Collet, https://github.com/fchollet/keras
11. Pokemon Wiki, pokemon.wikia.com
12. CMU Sphinx (Open Source Speech Recognition Toolkit), http://cmusphinx.sourceforge.net/
13. 3D Colour Histograms, http://www.pyimagesearch.com/2014/01/22/clever-girl-a-guide-to-utilizing-color-histograms-for-computer-vision-and-image-search-engines/
14. Basic CNN Architecture diagram, http://parse.ele.tue.nl/education/cluster2
15. Hierarchy of Deep Learnt Features, https://www.datarobot.com/blog/a-primer-on-deep-learning/
16. Hubel DH, Wiesel TN. (1962), *Receptive fields, binocular interaction and functional architecture in the cat's visual cortex*
17. Fukushima, K, (1980), *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position Biological Cybernetic*
18. Serre, T., Wolf, L., Bileschi, S., and Riesenhuber, M. (2007). *Robust object recognition with cortex-like mechanisms*

19. LeCun, Bottou, Bengio, and Haffner, (1998d), *Gradient-based learning applied to document recognition*

20. Jia-Yu Pan, Hyung-Jeong Yang, Pinar Duygulu and Christos Faloutsos, (2004), Automatic Image Captioning, IEEE International Conference on Multimedia and Expo (ICME), June 2004.

21. Girish Kulkarni, Visruth Premraj, Sagnik Dhar, Siming Li, Yejin Choi, Alexander C Berg, Tamara L Berg, Baby Talk: Understanding and Generating Image Descriptions Computer Vision and Pattern Recognition (CVPR), 2011.