AUTOMATIC IDENTIFICATION OF ANIMALS IN THE WILD: A COMPARATIVE STUDY BETWEEN C-CAPSULE NETWORKS AND DEEP CONVOLUTIONAL NEURAL NETWORKS.

Joel Kamdem Teto

Ying Xie

Kennesaw State University

Follow this and additional works at: https://digitalcommons.kennesaw.edu/cs_etd

Part of the Artificial Intelligence and Robotics Commons, Numerical Analysis and Scientific Computing Commons, Other Animal Sciences Commons, and the Other Computer Engineering Commons

Recommended Citation


https://digitalcommons.kennesaw.edu/cs_etd/20

This Thesis is brought to you for free and open access by the Department of Computer Science at DigitalCommons@Kennesaw State University. It has been accepted for inclusion in Master of Science in Computer Science Theses by an authorized administrator of DigitalCommons@Kennesaw State University. For more information, please contact digitalcommons@kennesaw.edu.
AUTOMATIC IDENTIFICATION OF ANIMALS IN THE WILD: A COMPARATIVE STUDY BETWEEN C-CAPSULE NETWORKS AND DEEP CONVOLUTIONAL NEURAL NETWORKS.

A Thesis Presented to
The Faculty of the Computer Science Department

by

Joel Kamdem Teto

In Partial Fulfillment
of Requirements for the Degree

Master of Computer Science

Kennesaw State University 2018.11
In presenting this thesis as a partial fulfilment of the requirements for an advanced degree from Kennesaw State University, I agree that the university library shall make it available for inspection and circulation in accordance with its regulations governing materials of this type. I agree that permission to copy from or to publish, this thesis may be granted by the professor under whose direction it was written, or, in his absence, by the dean of the appropriate school when such copying or publication is solely for scholarly purposes and does not involve potential financial gain. It is understood that any copying from or publication of, this thesis which involves potential financial gain will not be allowed without written permission.

________________________________________
Joel Kamdem Teto
Notice to Borrowers:

Unpublished theses deposited in the Library of Kennesaw State University must be used only in accordance with the stipulations prescribed by the author in the preceding statement.

The author of this thesis is:

Joel Kamdem Teto

1675 Roswell Road, Apt 518

Marietta, GA 30060

The director of this thesis is:

Prof. Ying Xie

Prof. Selena He

Prof. Sumit Chakravarty

363 J Building, 1100 South Marietta Pkwy

Marietta, GA 30060

Users of this thesis not regularly enrolled as students at Kennesaw State University are required to attest acceptance of the preceding stipulations by signing below. Libraries borrowing this thesis for the use of their patrons are required to see that each user records here the information requested.

Name of user Address Date Type of use (examination only or copying)
AUTOMATIC IDENTIFICATION OF ANIMALS IN THE WILD: A COMPARATIVE STUDY BETWEEN C-CAPSULE NETWORKS AND DEEP CONVOLUTIONAL NEURAL NETWORKS.

An Abstract of

A Thesis Presented to

The Faculty of the Computer Science Department

by

Joel Kamdem Teto

In Partial Fulfillment
of Requirements for the Degree

Master of Computer Science

Kennesaw State University 2018.11
Abstract

The evolution of machine learning and computer vision in technology has driven a lot of improvements and innovation into several domains. We see it being applied for credit decisions, insurance quotes, malware detection, fraud detection, email composition, and any other area having enough information to allow the machine to learn patterns. Over the years the number of sensors, cameras, and cognitive pieces of equipment placed in the wilderness has been growing exponentially. However, the resources (human) to leverage these data into something meaningful are not improving at the same rate. For instance, a team of scientist volunteers took 8.4 years, 17000 hours at a rate of 40 hours/week to label 3.2 million images from the Serengeti wild park.

For our research, we are going to focus on wild data and keep proving that deep learning can do better and faster than the human equivalent labor for the same task. Moreover, this is also an opportunity to present some custom Capsule Networks architectures to the deep learning community while solving the above-mentioned critical problem. Incidentally, we are going to take advantage of these data to make a comparative study on multiple deep learning models, specifically, VGG-net, RES-net, and a custom made Convolutional-Capsule Network. We benchmark our work with the Serengeti project where Mohammed Sadegh et al. recently published a 92% top-1 accuracy [23] and Gomez et al. had a 58% top-1 accuracy [12]. We successfully reached 96.4% top-1 accuracy on the same identification task. Concurrently, we reached up to 79.48% top-1 testing accuracy 33 on a big, complex dataset using capsule network, which outperformed the best results of Capsule networks on a complex dataset from Edgar Xi et al. with 71% testing accuracy [8,33,27].
AUTOMATIC IDENTIFICATION OF ANIMALS IN THE WILD: A COMPARATIVE STUDY BETWEEN C-CAPSULE NETWORKS AND DEEP CONVOLUTIONAL NEURAL NETWORKS.

A Thesis Presented to
The Faculty of the Computer Science Department

by

Joel Kamdem Teto

In Partial Fulfillment
of Requirements for the Degree

Master of Computer Science

Kennesaw State University 2018.11
ACKNOWLEDGEMENTS

I am very grateful to my family for all the support and love they gave to me, it was a critical part of my life as a student abroad.

I am also thankful and grateful to Professor Ying Xie who was my professor of Data mining during my undergraduate and then Advanced Database systems. He was the first one to introduce me to Artificial intelligence and intelligent system and he took me under his arm during all my Graduate studies.

Thanks to Dr Mingon Kang for all the key concepts he sharpened in my head through his machine learning class.

Thanks to my committee members for the Support, Dr Selena He and Dr Sumit Chakravarty.

Thanks to Dr Dan Lo, for whom I was an undergraduate Research assistant and who brought me into the research field. I had the opportunity to get a publication with him: "The impact of defensive programming on cybersecurity".

Thanks to Pablo Ordonez and Kirk Inman in helping me with IT support and providing me with the necessary I.T resources as much as they could.

Thanks to the graduate college for helping me support my studies through the Graduate Research Assistant Program.

Special Thanks to Kennesaw State University for providing an adequate environment for this research and all the necessary resources to make a significant progress.
Table of Contents

**CHAPTER I: Introduction** ............................................................ 12
  Problem statement ................................................................. 12

**CHAPTER II: Literature Review** ................................................ 15
  a. Capsule Network ................................................................. 16
  b. Deep Convolutional Neural Networks .................................... 21

**CHAPTER III: Methodology** ..................................................... 29
  a. Models .................................................................................. 30
  b. Dataset ................................................................................ 36
  c. Preprocessing ....................................................................... 37
  d. Environment ......................................................................... 39

**CHAPTER IV: Experiments and results** ....................................... 41
  a. Capsule network ................................................................. 41
  b. Deep Convolutional Neural Network ................................... 47
  c. Summary ............................................................................. 49
  d. Discussion of results: ........................................................... 50

**CHAPTER V: Future work and Conclusion** .................................. 51
  a. Future Work ....................................................................... 51
  b. Conclusion .......................................................................... 51

**REFERENCES** ........................................................................ 53
List of Figures:

Figure 1: Inverse graphic example ........................................................................................................... 17
Figure 2: Primary capsules' prediction of the next layer's output ............................................................... 19
Figure 3: Routing by agreement .................................................................................................................. 20
Figure 4: Multilayer Perceptron (http://pubs.sciepub.com/ajmm/3/3/1/figure/4) .................................. 22
Figure 5: Convolutional Window Example ................................................................................................. 23
Figure 6: Convolution window applied on an image ............................................................................... 24
Figure 7: Convolution filter applied to a 7*7*3 matrix .......................................................................... 25
Figure 8: Max pooling example ................................................................................................................ 26
Figure 9: Convolutional Neural Network- General overview ................................................................. 28
Figure 10: VGG16 detailed architecture ................................................................................................. 31
Figure 11: Inception V3 graph ................................................................................................................... 34
Figure 12: Example of Double gray-scale transformation ....................................................................... 38
Figure 13: Global Capsule network architecture with reconstruction ................................................... 43
Figure 14: detailed capsule network architecture. .................................................................................. 44
Figure 15: Capsule Network Experiment 2 architecture ....................................................................... 46
List of Tables

Table of Contents......................................................................................................................................................8

Table 4: Mohamed Zadeh et al. results summary[23].................................................................................................16

Table 1: Comparison between capsule network and Traditional Neuron .................................................................21

Table 2: Summary of the various VGG architecture [18].........................................................................................32

Table 3: Inception V3 summary architecture. [6]........................................................................................................35

Table 5: Summary of results capsule network experiment I .......................................................................................45

Table 6: Summary table for VGG16 network ..............................................................................................................48

Table 7: Summary Results for Inception V3 ................................................................................................................49

Table 8. Summary of Results .......................................................................................................................................49
List of Equations:

Equation 1: Squash function. 18

Equation 2: Number of parameters of a convolution layer. 27

Equation 3: Height of a convolution layer 27

Equation 4: Width of a convolution layer 27
CHAPTER I: Introduction

Problem statement

It takes on average 350 milliseconds to a healthy human to identify a known object in a clear image. Ignoring the fatigue and the resting time, it will take up 2 days of non-stop, no fatigue, no scrolling time of human labour to classify 500 thousand clear images. Whereas, with a deep learning model, it would take less than 30 minutes on a Tesla k20 GPU to perform the same. Nonetheless, that idealistic situation can never exist. That is why it took 8.4 years, with 40 hours per week and about 40 000 human volunteers to label these 3.2 million images performing a 96.6% accuracy [23]. This appeared as a big opportunity, both to save and reassign the human labour to less strenuous tasks and to share some discoveries with the deep learning community. We found the need to build models that can accurately monitor animals in a wild ecosystem to a certain extent defined by the scope of this research.

Monitoring the Ecosystem includes several variables that can eventually reduce the ability of a human to recognize species from a picture taken in a wild ecosystem. Imagine a picture taken at night, in a bad weather, with the animals to detect further away. These variables are: the weather that will affect the quality of an image. The time of the day, because identifying an animal in the night is not as easy as during the day. The distance from the camera, how far is the animal in the image can also influence the decision onto what species we have. And miscellaneous objects (trees, stones, etc..) that can be on the way of the animals you are trying to identify/track. Thus, interfering between the camera and the closest animal. These difficulties are not solely problems for human but can also impact on the decision of a computer system built around a deep learning
model. Throughout our research, the first axe will be to provide a solution to effectively identify these animals in a wild ecosystem.

On the other hand, we focus our research on understanding the behavior of a recent deep learning technology called Capsule Network. We leverage this complex dataset to first compare the performances, pros and cons of Capsule networks versus well-known state of the art Deep learning models. Then we will try to produce the best possible model achieved on large and complex dataset like ours.

Hence the models to be developed in this research are of two "categories": deep CNNs and Capsule Networks. These categories will incidentally cover the two axes of this research as we are going to compare the aforementioned categories applied to Serengeti dataset while trying to provide the best evaluation matrixes. After spending some time reviewing Convolutional Neural Networks and applying some basic classification tasks with it, we had two observations. First, CNN had quite a few hypothetical shortcomings (rotation invariance). Secondly, one of the less accurate tasks to perform with deep learning was detecting animals in the wilderness with live camera feed. From these two shortcomings identify, we decided to dig more into these two aspects while trying to find a unique solution for both issues. Theoretically, we have discovered a methodology that attempt to solve some shortcomings of CNN, that might help to improve several tasks on animals in the wilderness. Most of the deep architectures using CNN will usually be made up of several pooling layers (using max pooling), using RELU as activation function for several layers as well and do the Softmax to output the probability of each class. Typically, that will be the case for several well-known architectures like Inception V3 and VGGnet. This typical architecture will make these networks less likely to be rotation invariant and to enforce the relationship between
main features within an image. However, the new methodology, developed by Hinton, seems to be more appropriate for such complex tasks. Hinton proposes a concept called Capsules (see table 1.) and Capsule's layer with a dynamic routing between these two. This new model allows us not only to keep information between pixels but to group pixels into entities of an object and allow the network to identify the appropriate relationship between these entities to confirm the presence or absence of the object in the image.

Based on all this information gained from various readings, we decided to apply these techniques to animals’ detection and classification. More explicitly, it drove us to compare all the advanced neural network architectures that achieved state of the art performance with custom built capsule network architectures applied to animals’ classification. The hypothesis is that we expect the capsule network to outperform all other models on all the tasks we set ourselves to achieve. Hypothesis based on the conceptual attributes of a capsule layer.
CHAPTER II: Literature Review

In the deep learning community, a lot of researchers have attempted to perform a research similar or related to this one in some kind of ways. We have some who attempted identification on the same dataset using various type of networks, others that attempted to customize capsule network as we did, and even some that also tried capsule network on simple or complex datasets.

One of the key related work is Hinton et al. in their publication of Dynamic routing Between Capsules (2017). This paper is actually one of the breakthroughs of Capsule network, it successfully reached a 0.25% error on the MNIST dataset, thereby performing better than previous state-of-the-art models. However, that was on a quite “simple” dataset.

We also have Mohammed Sadegh et al. in their attempts to identify and count the animals wild animals with camera traps, explore some state of the art models and obtained some quite interesting results. Indeed, with 94.9% top-1 accuracy on identification and 99.1% on top-5 accuracy for identification as well. (see table below)
Table 4: Mohamed Zadeh et al. results summary[23]

Gomez et Al. 2016[22], they used transfer learning (using previously trained weights from another dataset to model new datasets) from the 1.3 million images with 1,000 classes from ImageNet datasets for feature extraction; before adapting the end of the model specifically to animal classification. However, transfer learning is often used to compensate for the lack of data for training, which is not the case for this research.

Hung Nguyen et al. (DSAA, 2017) attempted to automate the wild-life monitoring, but on a much smaller scale and with fewer classes. Indeed, they used only 55 000 images from the South-Central Victoria dataset from Australia for training. And they perform well on only three classes where they achieve a 90.4% accuracy.

Chen et al. also tried to automatically classify animals but on much smaller and less diverse datasets. The dataset was composed of around 20,000 images and 20 classes. He tried to harness the power of CNNs, but the results presented 38% of accuracy [8,33,27].

On the other end, Gomez et al. also tried to achieve a similar task using deep neural networks, on the same Serengeti dataset, but they obtained only about 58% accuracy on identifying the animals. [22]

We are going to give details explanation about the key concepts involve here: CNN and Capsules network.

a. Capsule Network

A Capsule network is a neural network that does an inverse graphic to learn features from an image. An inverse graphic is a computer vision concept that allows you to identify what objects
are in an image and what is the instantiation parameter of that object given the object. Capsule network effectively uses inverse graphic to produce a multi-dimensional vector that incapsulates various components of an object. The shape, the thickness, the length, the depth, localized skew, localized parts, and many other variants of an object.

![Image showing instantiation parameters](image)

**Figure 1: Inverse graphic example.**

The key component of a capsule network is called the capsule. A capsule is a function that tries to identify any given instantiation parameter of an object. A capsule is composed of an activation vector. It typically has two dimensions:

- the length, representing the probability of the object embodied by the capsule to be detected.
- the orientation, that characterizes the pose parameter of the object embodied by the capsule.
However, Activation vector may have many more dimensions depending on the goal and configuration of the capsules.

Nonetheless, let us come back on one of the dimensions of the activation vector, the length. The length is representative of the probability of an object being present based on its instantiation parameters. This means it should always be less or ideally equal to 1. For this purpose, a new activation function was designed for the capsule. The so-called Squash function. The squash function is designed to ensure that the activation vectors are always less than 1.

\[ v_j = \frac{\|s_j\|^2 s_j}{1 + \|s_j\|^2 \|s_j\|} \]

Where \( s_j \) = Total Input, \( v_j \) = Vector output of capsule \( j \).

**Equation 1:** Squash function.

Moreover, every capsule from the first layers of capsules, predicts the output of the next layer. This means, during training, the network will try to learn the transformation matrices from one vector in the first layer to another one in the next layer. What does this actually mean?

Let suppose we have a capsule network that is supposed to correctly classify different kinds of objects amongst which we have a bicycle and a vehicle. Amongst the capsules of the first layer, we have one that detects and identifies the instantiation parameters of a wheel oriented. That capsule will predict that two of the next layer’s capsules will output a bicycle and a vehicle respectively oriented at the same angle as the first layer circle capsule. Another example will be that of a house and a boat to be predicted. Let’s say amongst the first layer of capsules (primary capsules, the is a capsule in charge of detecting the Triangle component of either the boat or the House. If that capsules effectively detects a triangle, it will predict the output of the capsules in
the next layer. The house capsule will output a house oriented at an angle of similar to its primary capsule, and the boat capsule will do the same.

![Diagram](image-url)

*Figure 2: Primary capsules' prediction of the next layer's output.*

After each capsule in the primary capsules, layer has contributed to the construction of the next layer, the capsules of the last capsules layer have to decide what they actually identify. It is a democratic process between capsules. Meaning that the majority of the capsules have to agree on what output should be. This process is called **routing by agreement.** Indeed, in the last capsule layers, all the capsules depending on the inputs and predictions of the previous layer, vote on what is the most suitable object detected in the image provided. This vote is based on all the instantiating parameters the capsules were set to detect.

Indeed, if we have to predict between a house, a boat and a car, each primary capsule will vote as to what is supposed to be output based on what they identify. The object having the highest vote
will be the one predicted. When all the capsules involved in the process all agree on the same object, we talk about **strong agreement**.

![Agreement!](image)

*Figure 3: Routing by agreement.*

All these features and mechanism behind the Capsule networks allow them to preserve at least the location and pose of objects throughout the network. Hence, Capsule network is **Equivariant**.

We will use these key concepts to build custom networks and perform several experiments with capsules-based networks.

The table below resumes a brief comparison between capsules network and traditional networks:
b. Deep Convolutional Neural Networks

Convolutional Neural Network (CNN) has been one of the greatest advancements in the artificial intelligence field in the past decade. It has achieved State-of-the-art in computer vision, speech detection, Natural language processing, Intelligent agents, and several other areas. It is proven that CNN can be very valuable in extracting a plethoric number of features out of several types of dataset and actually learn properly for it. You can teach a car to drive, teach a drone to fly, teach a computer system to recognize license plates, bad weather and even animals in the wild.

CNN came as an improvement to the limitations and shortcomings of simple Neural Networks (NN) or Multilayer Perceptron (MLP). Indeed, MLP only accepted vectors and was made up of fully connected layers. This required a lot of parameters (computationally expensive) and was not
able to capture special features in a complex image. On the other end, CNN can accept matrices on several dimensions and came with sparsely connected layers using feature maps and pooling to extract features and reduce the number of parameters necessary to learn.

Here we are going to provide an overview of the key concepts behind the CNN.

**Key concepts**

- *Convolutional layer*
  
  Original, MLP was the main candidate in deep learning to perform image processing.

  A simple architecture composed of an input layer, one or multiple hidden layers and an output layer was used to simultaneously extract features from images and learn how to identify objects in images. (see figure 4)

  ![Multilayer Perceptron](http://pubs.sciepub.com/ajmm/3/3/1/figure/4)

  *Figure 4: Multilayer Perceptron (http://pubs.sciepub.com/ajmm/3/3/1/figure/4)*

  As an improvement to MLP, CNN came with a new type of hidden layers called convolutional layers. A convolutional layer is a layer made up of a set of filters that produces features maps in charge of extracting a particular feature in regions of the image.
From this definition, we understand that the method used by the CNN is “divide and conquer”. In each convolutional layer, each filter is in charge of extracting a specific pattern and reducing the dimensionality of the image at the same time.

Each filter has two principal components, the height and the width. These allow filters to be called, convolutional windows. (See figure 5)

![Convolutional Window Example](image)

This convolution window will slide horizontally and vertically over the image input. At every step, the convolution window will reduce the whole section under the image into a single point by multiplying each value in the convolution window by the corresponding entry in the input image and then sum over all the multiplication. In most of the case, an activation function will be applied after computing the multiplications and summations.
A critical aspect of convolutional filter or feature map is that each of them represents a single pattern; hence each of them is only looking for a single pattern in an image and can only extract that pattern. As there is a need to uncover several patterns to properly identify an object, there is usually the need to use many filters for the purpose.

One other key parameter that conditions the way patterns are learned and dimensions are reduced with convolutional filter is the Stride and the Padding.

- **Stride and padding.** The Stride is simply the amount by which the filter slides over the image. The Higher the Stride, the smaller the resulting feature map. However, the bigger your stride the more you lose information but the more you reduce the dimension of the input.

  On the other end, there are some cases where your filter does not totally fit at the edges of your images. The Handling of those cases is defined by the padding. Padding defines whether you leave out un data that do not completely fit into the filters at the edges or
padding the image with zeros at the edges to fit the convolution window. This decision is made usually by deciding whether the padding is ‘VALID’ or ‘SAME’. A ‘VALID’ padding is when you decide to ignore the incomplete edges, whereas a ‘SAME’ padding is when you decide to pad the image with zeros at the edges.

Figure 7: Convolution filter applied to a 7*7*3 matrix

- **Pooling Layer.** As more filters will be required in the convolutional layer for complex images, we are going to have bigger stacks of features maps which imply higher dimensionality and more parameters for the network to train. To prevent the dimensionality to keep getting higher and bigger, a pooling layer is often placed in-between convolutional
layers. There are two common types of pooling layers, Max pooling layer and Global average pooling layer.

**Max pooling layer** takes as input a stack of feature maps. Then, a window size and a stride are defined to regulate the transformation between the convolutional layer to the max pooling layer. For every pixel within the window, the pixel with the highest value is selected to be the value of the corresponding node in the max pooling layer.

![Figure 8: Max pooling example.](image)

**Global average pooling layer**, it also takes as input a stack of feature maps. However, there are no parameters to be specified. It takes the average of each features map and gives you a unique value for each feature map in the stack of feature maps. It is a more extreme dimensionality reduction technique. For example, a Global average pooling layer can take a 3D array and turn it into a vector.
To build more sophisticated architecture and control the complexity of the network, there are some formulas to derive the number of parameters in a convolutional layer and also to derive the shape of a convolutional layer. Respectively equation 2, 3 and 4.

\[ n_p = K * F * F * D_{in} + K \]

**Equation 2:** Number of parameters of a convolution layer.

Where $n_p =$ the number of parameters of the convolution layer; $K =$ the number of filters; $F =$ the height and width of the convolution filter; $D_{in} =$ the depth of the previous layer.

For a padding set to ‘SAME’ the shape of the convolution layer is given by:

\[ \text{height} = \text{ceiling} \left( \frac{H_{in}}{S} \right) \]

**Equation 3:** Height of a convolution layer

\[ \text{Width} = \text{ceiling} \left( \frac{W_{in}}{S} \right) \]

**Equation 4:** Width of a convolution layer

Where $W_{in} =$ the Width of the input filter; $H_{in} =$ Height of the input filter; $S =$ the Stride of the convolution.
Let us now get into particular use cases of successful Deep convolutional Neural networks. The two interesting use cases for our research are VGG and Inception V3 Networks.

*Figure 9: Convolutional Neural Network - General overview*
CHAPTER III: Methodology

The main goal of our research is to be able to automatically identify animal in the wild. Incidentally, we take advantage of this research to create the opportunity of exploring capsule networks in large and complex dataset. That is why our comparative study between capsules and Convolutional neural network will be the second axis of our research.

To covert and exhaust the two axes of our research, we will first explore state-of-the-art architectures and key pre-processing techniques to help us achieve our main goal. This will give off a strong base to start as to what will be the contouring factor that will ameliorate state of the art models on the same task. We should note that we are looking at full automation, so we will be more interested in top-1 accuracy all along the research. Moreover, explore capsule networks; by engineering custom models and comparing the performances.

After gathering all the performances of our multiples experiments, we will first benchmark our work with several results obtain from people that performed a similar task, then compare our performances between each other. More precisely, we will first compare our result with recent research that attempted this kind of classification task using state-of-the-art models. Then we will compare our capsule network results with that of research that have attempted it with large and complex dataset. Finally, we will compare our own results with each other and get the best out of it. This will allow us to give a prospective analysis as to what to improve and what are the shortcomings of using one methodology over the other.
Achieving the best possible results for this work will highly be dependent on the Models built, the Dataset, the Preprocessing techniques, and the Environment used.

a. Models
To achieve our research goal, we are going to explore 3 sets of models: VGG networks, Inceptions Networks, and Capsule network. VGG and Inception fall into the class of Deep convolutional neural networks. Our choice of network is based on past results of each of these networks achieved on specific problems.

- **VGG:**
A very Deep Convolution Neural Network for Large Scale Image Recognition. Karen Simonyan and Andrew Zisserman of the University of Oxford created a multi-layer (11,16,19) layers CNN that strictly used 3x3 filters with stride and pad of 1, along with 2x2 max-pooling layers with stride 2. This architecture is known for achieving state of the art large-scale animals' classification. Karen Simonyan et al. submitted a series of new Deep neural network architecture that was based on the depth of the network[18]. Indeed, they advocated for a series of 3 by 3 convolutional filters to capture as many inter-dependent features as possible [18,6]. These architectures allowed them to win the ImageNet competition in 2014 and to publish in ICLR conference of 2015. On top of that, they were 1st runner up at ILSVRC (ImageNet Large Scale Visual Recognition Competition) 2014 in classification Task. In 2014 at ILSVRC, VGG beats Google net on the Localization task.

The motivation behind VGG network is that I achieved well in both localization and classification task at the same time, which is a key part of this research. Moreover, VGG introduced very small filters which explores more regions on images and allows to reduce the typical number of parameters for this type of network by about 60%.
For this research, we are going to explore both VGG16 and VGG19 for their performances.

![VGG16 detailed architecture](image)

*Figure 10: VGG16 detailed architecture*

The figure above illustrates an example of VGG network with VGG16. Nonetheless, VGG has various architectures that have been successful from one problem to another; see the table below for the summary of the different VGG architectures.
Inception:

The idea behind inception was basically to find a novel way to improve CNNs performances without getting deeper and deeper with the number of convolutional layers.

Table 2: Summary of the various VGG architecture [18]

- **Table 2:**

<table>
<thead>
<tr>
<th>ConvNet Configuration</th>
<th>A-LRN</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 weight layers</td>
<td>11 weight layers</td>
<td>13 weight layers</td>
<td>16 weight layers</td>
<td>16 weight layers</td>
<td>19 weight layers</td>
</tr>
<tr>
<td>Input (224 × 224 RGB image)</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
</tr>
<tr>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
</tr>
<tr>
<td>maxpool</td>
<td>conv3-128</td>
<td>conv3-128</td>
<td>conv3-128</td>
<td>conv3-128</td>
<td>conv3-128</td>
</tr>
<tr>
<td>maxpool</td>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-256</td>
</tr>
<tr>
<td>maxpool</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
</tr>
<tr>
<td>maxpool</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
</tr>
<tr>
<td>maxpool</td>
<td>FC-4096</td>
<td>FC-4096</td>
<td>FC-1000</td>
<td>soft-max</td>
<td></td>
</tr>
</tbody>
</table>
The problem with deeper network sometime is overfitting and they are computationally exhausting.

To get around this, Christian Szegedy et al. designed a network that will be “wider” instead of stacking convolutional layers linearly \([5,6]\). They proposed an architecture where a layer will have different kernel sizes and will learn different sets of features in a single iteration. In our exploration of more efficient networks in terms of resources and classification performances, the key features that drove us to select this network are its memory efficiency and its performances on ImageNet.

Indeed, Inception Networks are based on inception modules. Inception modules were created to allow the model to train multiple convolutional layers simultaneously. Typically, convolutional layers are linear transformation with non-linear activation function. However, with inception modules, training convolutional layers simultaneously and stacking their features allow the modules to produce a nonlinear transformation. The convolutional filters trained simultaneously are composed of 1by1, 3by3, and 5by5 convolutional layers. On top of tap, there is a 3by3 max pooling layer. The whole architecture increases the sparsity of the network and allow the network to learn different kind of patterns.
As we can see, the architecture is fully composed of inception modules. Inception Modules are the computations of multiple kernel size in the same convolution layer. They increase the learning ability of each convolution layer. Indeed, having different types of filters in the same building block allows the network to have more opportunity to learn any pattern in different localities in the image. Thus, it reduces the number of layers needed to achieve the same learning with a deeper network. Moreover, the number of learning parameters of the network is lesser than similar networks like VGG and the computational cost is therefore reduced.

Here is a detailed summary of the Inception V3 architecture:

**Figure 11: Inception V3 graph[5,6]**
Table 3: Inception V3 summary architecture. [6]

<table>
<thead>
<tr>
<th>type</th>
<th>patch size/stride</th>
<th>output size</th>
<th>depth</th>
<th>#1x1</th>
<th>#3x3 reduce</th>
<th>#5x5 reduce</th>
<th>pool proj</th>
<th>params</th>
<th>ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>convolution</td>
<td>7x7/2</td>
<td>112x112x64</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.7K</td>
<td>34M</td>
</tr>
<tr>
<td>max pool</td>
<td>3x3/2</td>
<td>56x56x64</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>convolution</td>
<td>3x3/1</td>
<td>56x56x192</td>
<td>2</td>
<td>64</td>
<td>192</td>
<td></td>
<td></td>
<td>112K</td>
<td>360M</td>
</tr>
<tr>
<td>max pool</td>
<td>3x3/2</td>
<td>28x28x192</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inception (3a)</td>
<td>28x28x256</td>
<td>2</td>
<td>64</td>
<td>96</td>
<td>128</td>
<td>16</td>
<td>32</td>
<td>159K</td>
<td>128M</td>
</tr>
<tr>
<td>inception (3b)</td>
<td>28x28x480</td>
<td>2</td>
<td>128</td>
<td>128</td>
<td>192</td>
<td>32</td>
<td>96</td>
<td>380K</td>
<td>304M</td>
</tr>
<tr>
<td>max pool</td>
<td>3x3/2</td>
<td>14x14x480</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inception (4a)</td>
<td>14x14x512</td>
<td>2</td>
<td>192</td>
<td>96</td>
<td>208</td>
<td>16</td>
<td>48</td>
<td>364K</td>
<td>73M</td>
</tr>
<tr>
<td>inception (4b)</td>
<td>14x14x512</td>
<td>2</td>
<td>160</td>
<td>112</td>
<td>224</td>
<td>24</td>
<td>64</td>
<td>437K</td>
<td>88M</td>
</tr>
<tr>
<td>inception (4c)</td>
<td>14x14x512</td>
<td>2</td>
<td>128</td>
<td>128</td>
<td>256</td>
<td>24</td>
<td>64</td>
<td>463K</td>
<td>100M</td>
</tr>
<tr>
<td>inception (4d)</td>
<td>14x14x528</td>
<td>2</td>
<td>112</td>
<td>144</td>
<td>288</td>
<td>32</td>
<td>64</td>
<td>580K</td>
<td>119M</td>
</tr>
<tr>
<td>inception (4e)</td>
<td>14x14x832</td>
<td>2</td>
<td>256</td>
<td>160</td>
<td>320</td>
<td>32</td>
<td>128</td>
<td>840K</td>
<td>170M</td>
</tr>
<tr>
<td>max pool</td>
<td>3x3/2</td>
<td>7x7x832</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inception (5a)</td>
<td>7x7x832</td>
<td>2</td>
<td>256</td>
<td>160</td>
<td>320</td>
<td>32</td>
<td>128</td>
<td>1072K</td>
<td>54M</td>
</tr>
<tr>
<td>inception (5b)</td>
<td>7x7x1024</td>
<td>2</td>
<td>384</td>
<td>192</td>
<td>384</td>
<td>48</td>
<td>128</td>
<td>1388K</td>
<td>71M</td>
</tr>
<tr>
<td>avg pool</td>
<td>7x7/1</td>
<td>1x1x1024</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dropout (40%)</td>
<td>1x1x1024</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>linear</td>
<td>1x1x1000</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000K</td>
<td>1M</td>
</tr>
<tr>
<td>softmax</td>
<td>1x1x1000</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

a. Capsule Network:

As our choices of network has been based on diversity and performances. We are selecting Capsule network as our 3 type of model to explore. Indeed, we saw VGG as very deep network with very small 3by3 filter. Followed by Inception which was network advocating sparsity and simultaneous convolution on top of parameter efficacy. Capsule network comes with a completely different way of learning from images. Instead of dealing with pixels, it deals with instantiation vectors as we discussed in Chapter II.

Capsule network theory provides some confidence to image classification as the models are built to be rotation invariant and performs inverse graphic to ensure that the proper vectorial features of an object in an image are extracted. However, the best result achieve with Capsule network was a
0.25% error on the MNIST dataset. Motivated by the promising future of this new technique we decided to explore it on more complex dataset compare to MNIST.

We are going to perform two set of experiments on Capsule network.

- The First one will be with a very low-resolution image and a 3-layer network. This will be to explore the efficiency of capsule on low resolution image and the effect of image alteration on capsule network performances.
- The second one will be a more customize Capsule network with 11 layers. Here we are going to use a more complex image and try to explore how our custom build network can perform on a more complex image (300by300).

Moreover, for each of our Capsule network, we will perform image reconstruction to minimize the changes of overfitting and to ensure that our network actually learns the general patterns in the images. The details of these experiments will be given on the next chapter for experimentation.

### b. Dataset

The dataset used for these experiments is the Serengeti Dataset. Originally it is composed of 3.2 million images, 48 classes for a resolution of 1024 by 1024 pixel per image. Due to limited resources and the exhaustive nature of the capsule network, we had to create a significant subset of this dataset. We noticed a huge data imbalance in the distribution of classes where few classes accounted for most of the images. Hence, we decided to go with the 10 most occurring classes for this research: Zebra, Wildebeest, Impala, Human, Hartebeest, guineafowl, Giraffè, Gazelle Thompson, Elephant, Buffalo. These 10 classes account for 500,000 images (about 1/6th of the whole dataset.). For these experiments, we split the data into training, validation and testing. 80% of the data went towards the training and validation set, and 20% went towards the testing set. Out
of the 80% allocated to testing and validation, 20% of that went towards validation and the other 80% was purely training set.

c. Preprocessing

Preprocessing has been a keep part of the work we did here as it allowed us to have better models, and it increased the performances from one operation to another and it eased our workflow. We identified a few things that might be a problem for our experiment.
The first thing was to identify what the class distribution from the metadata file on the Serengeti official website. Once that was done, we selected the 10 most occurring classes which happened to account for a good portion of the images. Then, we need to prepare a script to download these images and classify them by folders based on their label. This operation really eased the processed when I came to input the data into the models.
Secondly, the size of each image was big due to the resolution of each image. Keeping 1K(1,920x1,080) images would be computationally expensive for every single operation on every single image. As a solution, we decided to resize all the image to 300 by 300. 300 by 300 because most of the state-of-the-art model needed less than this to achieve a good result. For example, Inception which requires the biggest image size, use 299 * 299 images as input.
The next issue faced was that some images were in the day, others were in the dark. We emitted a hypothesis that this could bias the model in case there is no balance between day and night or the model will tend to learn more about one versus the other. To solve this, we did a color alteration on all the images, transforming them to grayscale and then applying another grayscale transformation. This would remove the presumably biased on all the images.
The first set of images represent the first transformation applied, where we convert all the images to gray-scale once while keeping the 3 color channels.

The second image represents the second grayscale transformation that eliminates any biased linked to the time of the day or the weather.

This transformation also enables us to get rid of our third problem, the weather. These transformations will allow the model not to be biased by the weather but to focus only on the shapes and hard color variation(intensity) of the animals.

After these basics transformation on the images, we had to reshape the images again before passing them to each model. These were necessary because each of the models needed different input sizes from 28 by 28 to 300 by 300. During this operation, we also performed batch normalization to all
our images before inputting them to our model. There was a key process because of the large amount of data we had, the size of each image and the complexity of our models. Normalization allowed us to run our model faster while keeping the stability and reducing the chances of overfitting.

Uniform aspect Ratio was also a key component of our preprocessing as we had to make sure that every time, we scale down the images, the ratios between the height and the width is a 1:1 ratio.

d. Environment

The type of resources we used for this research was key factor in the achievements or non-achievements we have had so far. In terms of hardware, we used two different devices.

- The first one was a Mac Pro, with a CPU of 3.5ghz 6-core intel Xeon E5, two graphic cards from AMD having 3 Gb of memory each; 32 Gb of RAM and a storage of 512Gb SSD. This was the initially hardware used at the early stage of our research. It helps us investigate the pros and cons of most of our model and perform the literature review. However, when we tried to start the actual experiments, this hardware was quite limited for the following reasons. The Storage on the device was not big enough to hold the dataset; there was also no GPU on the devices which did not allow us to use Cuda and the parallel processing abilities of the graphical processing unit. Finally, it served as a terminal station to ssh into the Main server that we are presenting below.

- The second hardware was a server hosted by Kennesaw State University and managed by the College of Computing and Software Engineering. It had 1 Tb of storage allocated to me and **4 Tesla M40 GPUs**. This is the device that allow us to host our data and perform all our experiments to an extent allow by our computational limitations.
The Software environment used for this research was comprised of the following tools:

- OSX Yosemite: Operating System hosted on the Mac pro.
- Linux server: Operating System of the CCSE server where the GPUs are hosted.
- FileZilla: File Transfer and SSH software used to graphically interface with the server’s file.
- Jupyter Notebook: Integrated development environment used to build and visualize the models.
- Pycharm: Integrated development environment used to preprocess the data.
- Python 3.5: Programming language used for the research
- Tensorflow-GPU: Machine learning framework used to build some of the models. Efficient for matrix operations and GPU usages.
- OpenCV: Computer vision framework used for preprocessing.
CHAPTER IV: Experiments and results

a. Capsule network

For capsule network, we conduct two sets of experiment. The first set is using a very low-resolution image derived from the original 1K resolution image. We reduce the number of channels to 1 and resize the image to 28 by 28 pixels. For the second set of experiment, we build another custom capsule network architecture with a higher image resolution and with more layers in the network.

- Experiment I:

In the initial experiment, we will use a simple architecture with two convolutional layers and one fully connected layer. The first convolution has 256 filters with a kernel size of 9 by 9, a stride of 1 and ReLU(Rectifier Linear Unit) as the activation function. The second convolution layer will have also 256 filters with kernel size 9, a stride of 2 this time, padding set to valid and ReLU as the activation function. The layers receive as input images of 28 by 28 pixels. The Convolution layers will serve as features extractor in some specific localities of the images and then send the results as input to the primary capsules to start the process of inverse Graphics.

The second part of the network will be composed of the primary capsules that will be the receptors of the convolutions output. The primary capsule layer will be 32 channels convolutional 8D capsules.

On the side of our capsule network, we add a small decoder network that will help us preserved all the necessary information required to render the original image throughout the network. It is called Reconstruction. It is a multi-layer perceptron composed of three fully connected layers that will try to reconstruct the input images from the information given by the output of the capsule network. It serves as a regularization technique that will help us avoid overfitting and prone
generalization. To achieve this, we place a **Mask** to receive only the output vector of the capsules for the specified output classes and we masked out all those vectors except from the longest one (representing the predicted output on the capsule network). Indeed, we want to mask the longest one because, as stated in the Capsule network review, the longest vector in a capsule represents the predicted output which we want to keep. That predicted output becomes the reconstruction target and is passed to the decoder network for reconstruction.

The decoder network will try to decode the vector given by the mask and understand what to reconstruct. The reconstruction process conducted by the Decoder network will produce an image that is supposedly from the same class as the output vector. Finally, the reconstructed image will be compared with the input image. This comparison will be used to produce a new loss that will be the reconstruction loss. The Reconstruction loss is obtained by computing the square difference between the input image and the reconstructed image.

Finally, we will merge the loss given by the main capsule network and the reconstruction loss to have our final loss which will be one of the bases to evaluate how good our model is learning or not. (See figure below)
Figure 13: Global Capsule network architecture with reconstruction

A more detailed architecture of the transition between the Convolution layer and the capsule layer, as well as the architecture of the capsule layer is well presented on Fig. 14 in the following page.
Figure 14: detailed capsule network architecture.

Pictures from Serengeti dataset converted to gray and resized into 28 by 28 for experiment.
This first part yielded a 79.6% accuracy after 100 Epochs, with a loss of 9.89. whereas the benchmark for capsule networks on big-data sets is 71% accuracy. As mentioned by Edgar Xi et al., there is a scarcity of resources, therefore limiting the number of combinations we can try and explore based on our hypothesis.

The table below will present the best results obtained with this architecture.

<table>
<thead>
<tr>
<th>Number of Epochs</th>
<th>Learning Rate</th>
<th>Batch Size</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.1</td>
<td>128</td>
<td>57.34</td>
</tr>
<tr>
<td>100</td>
<td>0.01</td>
<td>64</td>
<td>79.6</td>
</tr>
<tr>
<td>100</td>
<td>0.001</td>
<td>32</td>
<td>71.48</td>
</tr>
</tbody>
</table>

Table 5: Summary of results capsule network experiment I

- **Experiment II:**

In the Second set of this capsule experiment, our hypothesis was that we a better-quality image and a deeper and wider network, the capsules would learn better. We decided to use a higher resolution image, 300 by 300 and to build a more complex network.

To stay consistent, we thought about using the convolutions as feature extractor and image scaler. Indeed, we align our convolution layer in such a way that It will trim the image down to 28 by 28 while extracting the necessary components to help capsules perform better inverse graphics.

Here is the architecture we propose.
However, there is another key difference in the global architecture of this experiment. We complexify the decoder network as it will try to reconstruct a more complex image. We use 3 fully connected layers but we more node in each layer including the output layer (Architecture to be detailed.)

For this one, we only run 10 epochs on the same dataset as we do not have enough computing resources to go beyond that for now. This was executed on a GPU server with 4 Tesla M40.

We succeeded in achieving a validation accuracy of 29.96% and a loss of 10.9. Moreover, the testing accuracy was 34%. This is encouraging because some of the parameters used to
train this model had to be limited because of resources problems. For example, the batch size used for this model was 8 and the number of epochs was limited to 10. We decided to call this model the C-capsule-9, partly because of the number of layers before the capsules.

- **Reconstruction Loss**

  For all our capsule experiments, we applied some image reconstruction and added up the reconstruction loss to the training loss. This is a key part of it as we want to avoid overfitting and we want to make sure that the network actually learns from real images that we pass it to. As capsule networks try to understand and decompose objects as vectors through inverse Graphic, we want to minimize the information loss during that process and make sure that there is almost no difference between the image given and the reconstructed image. This is a critical step to improve the performance of a capsule network.

b. **Deep Convolutional Neural Network**

For the deep convolutional neural networks, we also divide our work into two sets of experiments. The first ones are on VGG architectures and the second ones are on Inception V3.

- **VGG:**

  For our VGG model, we first resize all the images to 224 by 224 as it is the input required for the model. But more importantly, we perform some key modifications based on the characteristic of the model. We know that VGG is a very complex and heavy network with
more than 14 million parameters to train. Moreover, due to how deep the network is it is prone to overfitting, so we performed the following:

- Applied Batch normalization to the data.
- Removed the mean of the RGB channels to all the pixels of the images
- Added a dropout layer in-between every fully connected layer of the network, with a dropout probability of 20%.

These changes allow our model to bit models of researchers that attempted to achieve the same task.

<table>
<thead>
<tr>
<th>Model</th>
<th>Batch Size</th>
<th>Learning Rate</th>
<th>Epochs</th>
<th>Top-1 Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG-16</td>
<td>32</td>
<td>0.01</td>
<td>100</td>
<td><strong>96.48%</strong></td>
</tr>
<tr>
<td>VGG16</td>
<td>64</td>
<td>0.1</td>
<td>100</td>
<td>89.1%</td>
</tr>
<tr>
<td>VGG-19</td>
<td>32</td>
<td>0.01</td>
<td>100</td>
<td>86%</td>
</tr>
<tr>
<td>VGG-19</td>
<td>64</td>
<td>0.1</td>
<td>100</td>
<td>92.02%</td>
</tr>
</tbody>
</table>

Table 6: Summary table for VGG16 network

- **Inception V3:**

For the Inception model, we had to reshape all the images again to 299 by 299. On top of all the pre-processing techniques we globally used across this research, we also remove the mean of the RGB channels to all the pixels. However, we did not apply dropout to any of the inception modules or fully connected layers.

Here are the top two results yielded by the research:
<table>
<thead>
<tr>
<th>Batch Size</th>
<th>Learning Rate</th>
<th>Epochs</th>
<th>Top-1 Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>0.05</td>
<td>100</td>
<td>95%</td>
</tr>
<tr>
<td>64</td>
<td>0.01</td>
<td>100</td>
<td>90%</td>
</tr>
</tbody>
</table>

Table 7: Summary Results for Inception V3

c. Summary

<table>
<thead>
<tr>
<th>Model</th>
<th>Input size</th>
<th>Top-1 accuracy</th>
<th>Hyperparameters (batch size, epochs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG-16</td>
<td>224<em>224</em>3</td>
<td>96.48%</td>
<td>32, 100</td>
</tr>
<tr>
<td>Inception-V3</td>
<td>299<em>299</em>3</td>
<td>95%</td>
<td>64, 100</td>
</tr>
<tr>
<td>Basic Capsule</td>
<td>28<em>28</em>1</td>
<td>79.6%</td>
<td>64, 100</td>
</tr>
<tr>
<td>C-capsule-9</td>
<td>300<em>300</em>3</td>
<td>34%</td>
<td>8, 10</td>
</tr>
</tbody>
</table>

Table 8. Summary of Results
d. Discussion of results:

Across the experiment performed, we noticed that VGG outperforms the other models with a top-1 accuracy of 96.6% before Inception V3 that comes with 95% accuracy. This results quite satisfactory as it outperforms the best model that was built so far for the same problem. Indeed, Mohammed Sadegh et al. recently published a 92% top-1 accuracy [23] and Gomez et al. had a 58% top-1 accuracy [12].

On the other end, our capsules respectively yield their best results for 79.6% and 34% top-1 accuracy. This result is not what was theoretically expected as we were trying to outperform VGG and Inception. However, we believe that there is still a lot of refining that can be done on these capsules for a complex dataset like Serengeti. Nonetheless, we also have an ounce of satisfaction because we reach up to 79.48% top-1 testing accuracy on a big complex dataset using capsule network, which out-perform the best results of Capsule networks on a complex dataset from Edgar Xi et al. with 71% testing accuracy [8,33,27].
CHAPTER V: Future work and Conclusion

a. Future Work

With the improvements and availability of more resources, we will explore More areas of this research. How to widen capsule network and the impact it has on complex datasets. Using transfer learning for feature extraction and feed the features to the capsules. Explore new preprocessing techniques that might improve our models. Study the behavior of capsule networks without prior convolutions applied. Finally, Repeat the whole experiment of the Full update Serengeti dataset.

The first perspective we plan to explore next is actually the impact of having a bigger or smaller capsule. Along with the number of capsules per layers. This inspiration is driven by the idea behind Inception networks [2,3,13], however, there is a doubt that is will reduce the computational needs of the network as in Inception.

Our second perspective is to use the well-trained CNNs as feature extractors on high-quality images and then pass the features to the capsules for the whole dataset. This will drastically reduce the computational cost of having a whole capsule network built to receive a 300*300 image.

The Third perspective is about optimizing all the previous and aforementioned processes to run them on the full update Serengeti Dataset.

b. Conclusion

We have explored a very big dataset with the so-called Serengeti project and tried to out-perform existing authors that attempted the same identification task. Moreover, we have explored a quite complex network in implementation and computation by implementing a couple of Capsule network variation a big dataset (coloured images). For the former, we have succeeded to have our
top-1 accuracy up to 96.48%, where the best record in top-1 identification for the same dataset was 92%.

As for the later, we have shown how capsule networks have a big potential in recognizing complex objects in very low-quality images. However, the more depth you give to the capsule, the higher the computational expenses rise; they rise exponentially. We have also experienced manual and convolutional dimensionality reduction applied to a capsule network. A capsule network learns better from a convolutional cropping. Indeed, it gives it the ability to be built for any size and resolution of images.

More importantly, reaching 96.48% on automatically identifying animals in the wilderness is a great advancement as it performs as good as a human expert on a clear picture and better than human experts on “dark pictures”. Recalling that it took more than 8 years to human experts to classify the SS dataset, it is with a scientific optimism that we believe we can even do better by refining a little more our pre-processing techniques and optimizing our networks. Indeed, one of the difficulties of this research was the exploratory analysis of the capsule network as it is computationally greedy. The more width and depth you add to the network, whether at the level of capsules or convolutions, the more the required computational power will increase.

The capacities of the capsules seem only limited because of the huge computational needs to run larger networks on a larger dataset. However, we expect that not to be a problem in the coming years as computational power becomes more available to the common public.
REFERENCES


[29] Saurabh Srivastava & Prerna Khurana & Vartika Tewari. Identifying Aggression and toxicity in Comments using Capsule Network. TCS research


[35] Youngjoo Kim, Peng Wang, Yifei Zhu, and Lyudmila Mihaylova. (September, 2018) A Capsule Network for Traffic Speed Prediction in Complex Road Networks. Department of Automatic Control and Systems engineering, the university of Sheffield, United Kingdom