CAPSULE NETWORK BASED SUPER RESOLUTION METHOD FOR MEDICAL IMAGE ENHANCEMENT

Shashika Chamod Munasingha

189388N

Thesis submitted in partial fulfilment of the requirements for the

Degree of Masters of Science in Artificial Intelligence

Department of Computational Mathematics

Faculty of Information Technology

University of Moratuwa

Sri Lanka

October 2020

Declaration

I declare that this dissertation does not incorporate, without acknowledgment, any material previously submitted for a Degree or a Diploma in any University and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, to be made available for photocopying and for interlibrary loans, and for the title and summary to be made available to outside organization.

Name of the student: Shashika Chamod Munasingha

Signature: Date:

The above candidate has carried out research for the Masters/MPhil/PhD thesis/ Dissertation under my supervision.

Name of the supervisor: Subha Fernando (PhD)

Signature of the supervisor: Date:

Abstract

Medical imaging has been one of the most attentive research and development areas since the 1950s, particularly due to the contribution to disease diagnosis. Despite the fact that imaging technologies have been advanced in multiple ways, yet resolution limitations can be observed. To overcome the resolution limitations, various image enhancement techniques have been used. Image Super-Resolution (SR) is the latest technique in the list to achieve higher resolution with much lower resolution images. Earlier, frequency based and interpolation based SR techniques were used for SR. The afterward achievements in SR techniques are obtained via Convolution Neural Network (SRCNN) based methods and have several flaws.

Capsule net (Caps Net) is the state of the art alternative methodology for the problems which were previously solved by CNN. One recent attempt was made to assess the Caps Net for SR task. This new area has a lot to be explored. Especially the time inefficiencies of this approach should be addressed along with accuracy improvements.

In this research several capsule network routing mechanisms have been investigated for Super Resolution pipeline with a medical image dataset. Standard Dynamic Routing and Expectation Maximization Routing methods are re-configured to improve the accuracy. Above all, a novel integration of state of the art routing mechanism, Inverted Dot Product based Attention Routing mechanism is introduced for Super Resolution task.

With 300,000 medical image training pairs and 2,500 evaluation pairs, every model was evaluated. Along with different image quality indexes, it was shown that the Dynamic Routing based method outperformed all methods and the newest Attention Routing based approach has shown similar image quality performance to that of the state of the art method FSRCNN and less time complexity to that of the existing Caps Net based approaches. This implies that clinicians can use this system effectively in a clinical setting.

Dedication

I dedicate this thesis to my parents, my grandmother and my wife who are always withstand in my successes and failures.

Acknowledgment

Throughout the completion of this dissertation I have received great deal of helping hand from many people around me.

I would first like to thank my supervisor, Dr. Subha Fernando for her effort, patient, commitment and guidance for the success of this project. Her expertise was invaluable in formulating the research question and the methodology. Your exceptional support and feedback always helped me to bring my work to a higher level.

I also would like to thank Prof. Asoka Karunananda for the guidance he has given to prepare the thesis materials and showing the correct path of conducting the research.

In addition to that, I would like to acknowledge Dr.Sagara Sumathipala for his immense support in the background to conduct the project in timely manner. My sincere gratitude goes to all the other lecturers and non- academic staff members who helped me to make this project a success.

My fellow colleagues, I would like to thank you for your support in completion of this project.

Finally, I would like to thank my parents, wife and my family members for their wise counsel and for keeping up with me. You are always there for me and without your encouragement this project would not end up in great success.

Table of Content

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Description
SR	Super Resolution
SFSR	Single Frame Super Resolution
DR	Dynamic Routing
EM	Expectation Maximization
Conv	Convolution
Net	Network
Caps	Capsule
GPU	Graphical Processing Unit
FSRCNN	Fast Super Resolution using Convolutional Neural Network
SRCNN	Super Resolution using Convolutional Neural Network
GAN	Generative Adversarial Network
(A)NN	(Artificial)Neural Network
GUI	Graphical User Interface
PSNR	Peak Signal to Noise Ratio
LR	Low Resolution
HR	High Resolution
SSIM	Structural Similarity Index
MSSSIM	Multi Scale Structural Similarity Index
MRI	Magnetic Resonance Imaging
CT	Computed Topography

LIST OF APPENDICES

Chapter 1 Introduction

1.1 Prolegomena

‘Digital imaging along with Deep Learning allow to create images limited by hardware but just in our imagination’.

From the earliest stage of computers to the most recent technological advancements in the computing domain, digital images have been playing a significant role in every domain we can think of. Representing visual objects with a numerical matrix format made a new computing area, today identified as computer vision. The greatest of all, the evolutions in the digital imaging domain were found mainly in the medical image domain amongst other domains due to the vast number of applications in that field. With the expansion of ideas in the digital image domain, there are always advancements as well as room for improvements.

The traditional algorithmic methods of digital image processing have been recently replaced and improved with artificial intelligence related technologies, particularly deep learning based methods.

1.2 Background and Motivation

The vitality of medical images in medical diagnostic, disease prevention, treatment and illness management [1] is a trivial fact. Superficial vessel obstructions to hidden early stage tumors can be identified with medical images. The medical images on screen or paper are non-other than a 2-D representation of 3-D internal structures in the human body. To achieve this task, different imaging modalities have been developed [2] starting from the earliest X-Ray to Functional MRI systems [3]. These modalities stand for different medical purposes varying from anatomical structure analysis to functional information analysis. For most of the years in the early development stages of imaging modalities, the researchers had paid particular attention to hardware optimization as the main method of image quality enhancement. With the

improvement of the software field, image quality improvement has become a software post-processing challenge [3].

Among the image quality parameters; the spatial resolution is a very important parameter. The contrast resolution, noise, temporal resolution and radiation luminance (where applicable) are the other quality causing factors. Spatial resolution is about the smallest distinguishable objects that can be seen in the image. According to research, around 40% of the medical malpractices that are reported at the law-suite are due to misdiagnosis [4] and they are mostly a result of insufficient image resolution. Hence, even a small resolution upgrade could drastically positively change the diagnostic results by early detection, low signal to noise ratio (SNR) and increasing accuracy of measurements.

1.3 Aim and Objectives

Aim - To develop a Super Resolution module for medical imaging system, in order to generate low noise and high resolution images from low resolution images.

Objectives –

- To critically review of the existing SR techniques and identification of areas to be improved and further researched
- To select of different Caps-Net Routing techniques for implementing SR framework.
- To implement and train different Capsule Network architectures (layers, routing mechanism) on the data set.
- To evaluate those Capsule Network architectures against state of the art SR methods.
- To perform clinical opinion seeking and publishing the results.

1.4 Problem in Brief

The spatial resolution improvement too has been developed as a software related tasks [6] in the past few decades and achieved breakthrough results.

The state of the art technique using CNN based SR (SRCNN & FSRCNN) has been named as the current winners of SR techniques. Even though there are some competitive methods introduced afterward, using GANs and one attempt using Capsule Network in 2020, still SRCNN and FSRCNN is considered to be the leading methods in terms of training efficiency and output accuracy. Yet, there are some drawbacks in these methods like; requirement of a very large dataset for training, the chessboard effect and the noise induced due to the pre-up sampling process. The lattermost development of Capsule Network [38] based method can be recommended as a solution to overcome those problems, but yet improvable in architectural (layers, routing and reconstruction) design.

1.5 Proposed Solution

Capsule network architecture by Geoffrey Hinton and his team [1] has been proposed as a solution to the inherited drawbacks in CNN.

In this research, a Capsule Network based SR Technique is proposed. It uses different routing mechanisms (Dynamic Routing, EM Routing and Attention based Routing) and layer architectures than the recent research [38] and particularly trained on medical image dataset to embed the domain specific knowledge to the Caps Net.

1.6 Resource Requirements

For the successful completion of the project, as this is a Deep Learning based method, it was required a virtual environment with GPU capabilities. To make a more general model for most of the imaging modalities it was also required several databases of different medical images.

1.7 Outline

The rest of the thesis is outlined as follows. Chapter 2 is dedicated to reviewing the other research work in the domain of Super Resolution highlighting the pros and cons of each method with some introduction to the problem that is being addressed in this research. The next section is describing the core technique that is used in the project. Following that, in the Approach chapter, you can see the overview of the solution that I propose here. In the Design chapter, you could see the high-level workflow of the method proposed. The Implementation chapter describes the end-to-end details of the design showing how it is realized. The results are layout in the Evaluation chapter giving the opening to the final chapter, Conclusion where the discussion of our work and the further development is emphasized.

1.8 Summary

This chapter opens up the research by introducing the area of the research and the motivation for selecting this particular area for the research. It also highlights the project objectives and some introductory details related to the problem to be addressed in this research. The proposed solution and the resources to conduct the research are also mentioned above. The chapter finally explains the outline of the upcoming chapters.

Chapter 2 Super Resolution – Past, Present & Future

2.1 Introduction

In the previous chapter, an introduction to the overall project was given emphasizing the importance of SR in medical imaging. This chapter presents our critical review of research on developments in SR techniques. This chapter is structured under several headings, namely, early development in SR techniques, breakthrough in SR techniques, modern development in SR techniques, challenges in SR techniques and problem definition.

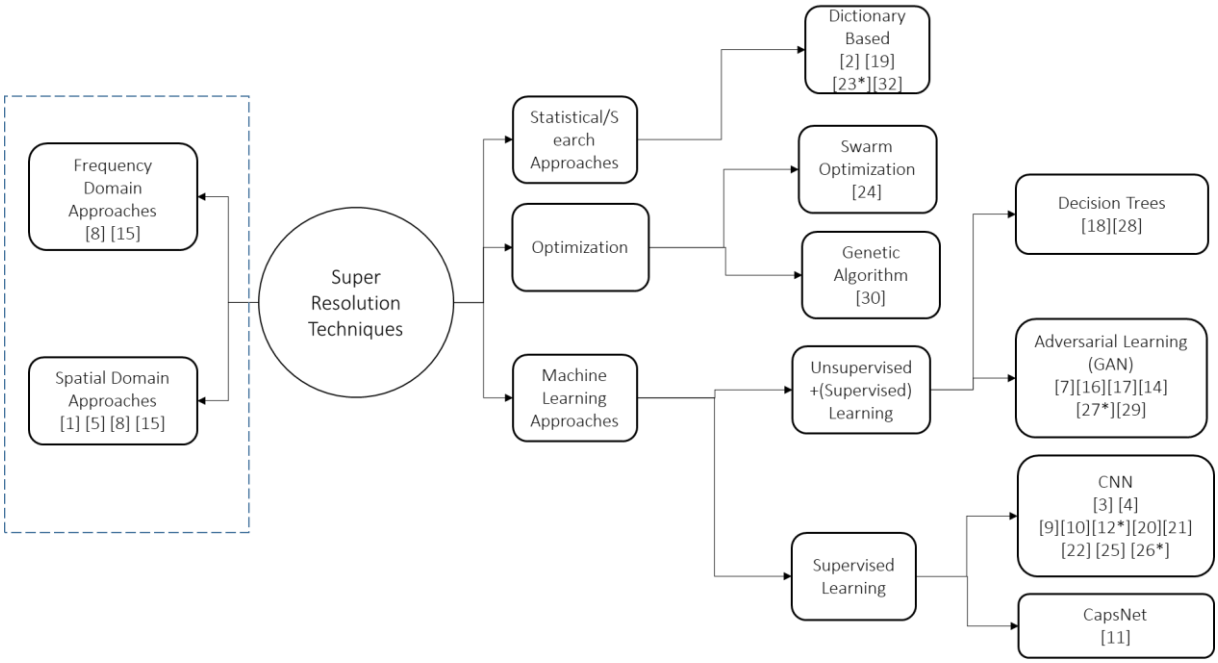


Figure 2.1: Overview of the SR techniques

2.2 Early Approaches to Super Resolution

Super Resolution is defined as the process of generating high resolution (HR) images from one or more low resolution images. Not only limiting to medical imaging [7] [3] [6], SR has been widely used in several other applications such as Satellite and Aerial imaging [8], Face recognition [9], Text Image Improvement [10] and Fingerprint

enhancement [11]. These related fields, themselves nourished the SR techniques and motivated them to experiment on different SR approaches.

In this section, the early approaches of SR are emphasized. Even though they have been addressed as early approaches, to date these methods have been used in applications to large and small extent accordingly. Before explaining the early developments, the terms interpolation and restoration must be distinguished from the term SR. In interpolation, only the size of the image is increased. During image restoration, the image is treated for noise removal and contrast adjustment, etc., but the image size is unchanged. SR does both; size increments and image quality improvement.

2.2.1 Frequency Domain Approaches

The earliest SR methods are based on the frequency domain [12] [13] [14] [15] algorithms. The theory behind frequency domain approaches is trivial. The LR images are first converted to the frequency domain by Fourier transform and estimation of HR image frequency spectrum is obtained with mathematical models. Then, the HR image is reconstructed in the spatial domain by inverse Fourier transformation. The very first SR algorithms [12] [16] by Gercberg and Santis respectively, have used iterative truncation on the frequency domain for SR. This early approach was unpopular until the work of Tsai and Huang's [17] system of satellite image SR. This was one of the first multiple image SR algorithms, where multiple LR images were used to reconstruct the HR images.

The wavelet transform is another frequency domain approach where the LR image is decomposed into sub-images. Nguyen [18] has first proposed interpolation and restoration based wavelet decomposition for SR. With the decomposition in the wavelet domain, it is convenient to explore the similarities in the neighboring pixels and obtain the HR image decomposition. Then, the images are generated via inverse wavelet transform [19]. The wavelet based methods are not computationally efficient as Fourier transform based methods, but give appealing results.

In summary, frequency domain methods are relatively efficient in computation, but they are prone to model based errors and unable to handle complicated motion models.

2.2.2 Spatial Domain Approaches

To overcome the drawbacks in frequency domain methods, spatial domain methods have become the trend. Early spatial domain based approaches used several techniques; non-uniform interpolation [20], iterative back-projection (IPB) [21], projection onto convex sets (POCS) [22], Direct methods [23] and Regularization methods [20]. One similarity to all these methods that they use multiple LR images to reconstruct the higher resolution images.

Iterative Back Projection is one of the earliest spatial methods in SR. Here, an HR image is first guessed by averaging multiple LR images. Then, this initial guess is fine-tuned iteratively. Next, the LR images are simulated with the guessed HR image. Afterward, the observed LR and simulated LR images are subtracted to obtain the error term. This error is back projected to HR coordinate for tuning. This process is repeated over iterations until no change is observed.. The main problem in IBP method is the convergence to a better solution is not guaranteed as and could oscillate between weak solutions [24].

Direct methods using Optical Flow [23], Adaboost [25] and many more techniques have also gained popularization as SR techniques. The following steps are followed in Direct methods in common; an LR image is selected as the reference and the rest of the LR images are registered against the reference image. The reference image is scaled up to the expected scale and the other registered images are injected into the HR grid using registration information. The fusion of all these images happens next and finally, denoising kernels will be applied. These methods outperform the IBP computational wise.

Projection onto Convex Sets (POCS) is another iterative approach [22] [26]. They are using a non-direct cost function for obtaining SR image. In POCS, it is assumed that the LR images could generate knowledge on HR images. The generated knowledge is assumed to be a convex set. To reduce the erroneous results, prior knowledge related to images; luminance variations, boundedness parameters have been used accordingly [22].

The regularized methods [21] [27] are the most popular due to the effectiveness and flexibility. The regularization methods work on a framework where an imaging model is assumed with parameters related to blurring, down-sampling and noise terms. The imaging model to solve SR is an ill-posed problem; where no limited number of solutions available but infinitely many solutions [28]. This is where the regularization term comes to play. It can stabilize the inversion process as well as reconstruction artifacts. Maximum A-Posteriori (MAP) algorithm based method [27] by Irani found an estimate of the HR image with Baye's rule.

During most of the practical scenarios, it is impossible to acquire sufficient LR images of different viewpoints, different camera sensors [28] etc. Hence, Single Frame SR (SFSR) methods have been more applicable over more specific tasks. This is because the images of the same class have close statistics. These algorithms have two basics; reconstruction or learning. During most of the practical scenarios, it is impossible to acquire sufficient LR images of different viewpoints, different camera sensors [28] etc. Hence, Single Frame SR (SFSR) methods have been more applicable over more specific tasks. This is because the images of same class has close statistics. These algorithms have two basics; reconstruction or learning.

The very first learning algorithm called Hallucination algorithm [29] was a neural network based algorithm. The network learns LR to HR relationship of images. The learned knowledge is represented as vectors and embodied in the reconstruction. From there onward, many other improvements have been done with learning algorithms like Feature Pyramids [30] & Belief Networks [31]. The main limitation of these algorithms is that they could mainly improve primitive image features like; edges, ridges, corners, junctions, etc.

The internal similarity comparison & correspondence between LR and HR mapped from external LR-HR dictionaries are the two main pathways of SFSR [28]. Neighbor embedding [32] & Sparse Coding [33] are such improved methodologies of SFSR respectively.

2.3 State of the art Techniques for SR

Deep learning is used as the technique in a state of the art SFSR. Generative Adversarial Networks (GANs) [34] and Convolution Neural Networks (CNNs) [35] [36] [37] are the two network configurations used for SFSR. Both methods have shown promising results, but researches in CNN based SR are moving forward faster due to the optimized HW and maturity of CNN architectures.

With more focus on the recovery of textural details in HR images, a new approach with GAN was introduced as SRGAN [34]. The capability of GANs to generate new images laid the foundation for this approach. This method was capable of generating x4 upscaled images for the first time. Apart from the GAN, special loss functions were used to achieve the SR images. The popular ResNet architecture was used as the backbone of the NN design. With the very deep network architecture, it showed weak time performance in practical cases.

The SRCNN [36] is one of the seminal research in SR with Deep Learning techniques. It was because this method outperforms the popular Sparse Coding [33] method and implements a similar SR pipeline, which is in Sparse Coding. The major drawback of this method is the increment of the computational complexity with the size of the image.

To overcome the problem with SRCNN, another approach was introduced as FSRCNN [37]. This is a very shallow network compared to SRCNN and implemented a novel SR pipeline in the following order; feature extraction, shrinking, non-linear mapping, expanding and deconvolution. To date, this was identified as the most widely used SR method due to its powerful time and accuracy performance.

2.4 Challenges in CNN based SR Techniques

Even though CNN based SR techniques like SRCNN [36] and FSRCNN [37] are the most discussed literature in SR techniques, they have the inherited drawbacks of CNN.

With an example, it would be easier to understand the drawbacks of CNN. Imagine a face. What are the components? We have an oval face, two eyes, a nose and a mouth. For a CNN, the presence of these objects can be a very solid indicator to consider that there is a face in the image. The orientation and relative spatial relationships between these components are not very significant factor to CNN. The CNN approach to solving this issue is to use max pooling or successive convolutional layers that reduce the spatial size of the data flowing through the network and therefore increase the "field of view" of higher layer's neurons, thus allowing them to detect higher order features in a larger region of the input image. These operations lose valuable image information that is useful for more precise outputs. Due to the same reason, a large amount of data is needed to train a CNN to a satisfactory level.

Especially, losing valuable information is a negative impact on image construction tasks like SR, where it is a must to keep all the image information at the input to generate additional information or in other words to generate HR images.

2.5 Literature in Brief

In the literature review, major achievements and issues have been identified by considering most cited researches. These are summarized in Table 2.1

Table 2.1: Summary of literature review

Method	**Basis**	**Pros**	**Cons**
Frequency Domain Approaches	Fourier/wavelet based transform of LR images are mapped to HR images in frequency domain. Mathematical models are built in frq domain. Inverese transform to build HR images.	High computational efficiency	Sensitivity to model errors Difficult to handle more complicated motion models
Spatial Domain Approaches	Interpolation, Regularization	Interpolation – most intuitive and simplest approach	Poor performance when magnification factor increases.

		Regularization -	
Sparse Encoding	Dictionary based method. Extracted LR patches are encoded into LR dictionary value and corresponding HR encoding is obtained from HR dictionary. Then reconstruction of patches.	Accuracy wise effective than regularization techniques. Better performance for single image SR technique.	SR pipeline is only optimized for dictionary building and mapping functions, not on reconstruction. Building up dictionaries is time consuming. Considerable pre/post processing
CNN	Pipeline of Sparse Encoding can be fully represented as convolution operations.	Little pre/post processing No explicit learning of dictionaries	Pre-upsampling process induce noise. Chessboard effect when no upsampling
Caps Net	-Similar to CNN-	No upsampling is required at the beginning Learns with less number of samples than CNN Better accuracy	Computationally slow than SRCNN
GAN	Model learns the mapping guided by the GAN loss (High Res and Super Res)	More appealing to human eye	Low accuracy wise performance than CNN

2.6 Problem Definition

Medical images are a major part of medical diagnosis. Images with higher resolution provide diagnostically better judgments for clinicians. Hardware limited low resolution images can be converted to higher resolution images with Super Resolution

(SR) techniques. State of the art SR techniques use CNN as the core. CNN has its inherited faults; additional noise due to up-sampling & chessboard effect while reconstructing. Anew approach, Capsule Net has overcome these issues and accuracy wise performed better than CNN. This approach claimed to have time inefficiencies due to the complexity of the network mechanism. Because of this, the practical implementation of this technique is also a concern. This novel approach gives a lot of room to explore the accuracy improvements as well. Moreover, previous attempts were tested on a common small dataset and specific application in the medical image domain was not tested.

2.7 Summary

In this chapter, a critical review of SR techniques has been given highlighting the application of SR techniques, evolution and limitations in SR techniques. In the next chapter, a highlight on the technology used; Capsule-Net will be described.

Chapter 3 Capsule Nets – Next Giant

3.1 Introduction

In the previous chapter, it was presented how the state of the art SR techniques were evolved around the Deep Learning techniques. Amongst them, CNN based methods SRCNN [36] and FSRCNN[37] are widely discussed. In this research, the same SR pipeline that has been used for CNN methods is adopted as the center to the implementation. This pipeline can be described in three major steps namely; basic feature extraction, non- linear mapping and reconstruction in a sequence. In one of the very recent researches, the researchers has attempted to use Capsule Networks [38] in the same pipeline and shown accuracy wise impressive results. A distinguishable change to this pipeline was made by introducing some of the newest Capsule network architectures at the non-linear mapping stage.

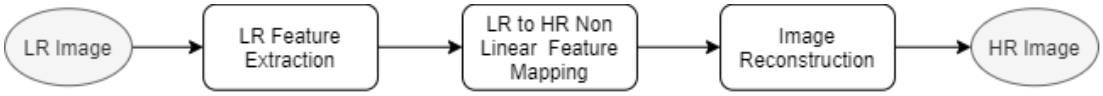


Figure 3.1: Super Resolution Pipeline

The technological content in this chapter can also be described under three SR pipeline components described above. The feature extraction part of the pipeline is supported by 2D convolution operation. The non-linear mapping between low-resolution to high resolution features is done with Caps-Net and finally image reconstruction with feature maps is carried out with De-convolution technique.

3.2 Convolution Neural Networks

A Convolutional Neural Network (CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other [39]. The pre-processing required in a CNN is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, CNNs have the ability to learn these filters/characteristics.

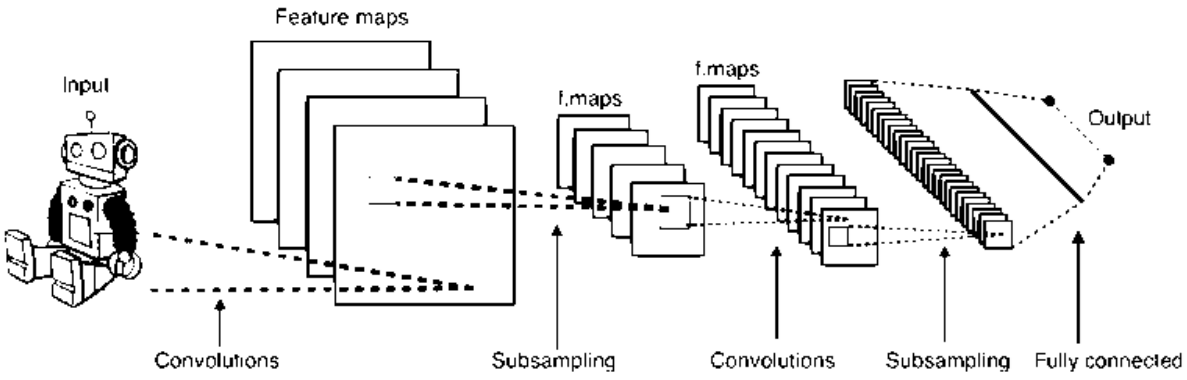


Figure 3.2: CNN Architecture for Image Classification

The simplified architecture of CNN for classification task consists of several 'Convolution layers' followed by sampling layers (pooling) and stacking of few such layers. Lastly, there are several 'Fully connected' layers assembling the final classification output. The objective of the Convolution Operation is to extract the high-level features such as edges, corners and basic shapes from the input image. When getting into deeper layers they capture more and more high-level features of the image giving simple feeds to the fully connected layers. The pooling operations are making the features more robust and invariant to image orientation and scale changes. The head layers of the network take the input features and adjust the flow to do the classification then after.

This smart architecture has driven the Machine Vision tasks to rapid development and most of the improvements that are being experienced today as AI has some links to CNN. Due to this advance layout behind the CNN, it has been used for SR tasks by highlighting the analogous features to Sparse Encoding technique used for SR as mentioned in Chapter 2.

Within the research, the Convolution layers have been used as the early stage feature extraction technology without doing an architectural change due to its strengths in extracting primary features such as edges, corners, lines contrasting regions etc. In order to keep all the information, the previously mentioned pooling operation have

been skipped. Along the SR pipeline, these features set out the basement for extracting high level features in the later part of the NN layer implementation with Capsule layers.

3.3 Capsule Network

The core technology behind this research; Capsule Networks is being described in this sub section emphasizing on three routing mechanisms that have been used for this research namely; Dynamic Routing, EM Routing and Inverted Dot Product Attention Routing.

3.3.1 Inverse Graphics – Backstage of Caps-Net

Computer graphics deals with constructing a visual image from some internal hierarchical representation of geometric data. That internal representation is stored in computer's memory as arrays of geometrical objects and matrices that represent relative positions and orientation of these objects.

Inspired by this idea, Hinton argues that brains, in fact, do the opposite of rendering. He calls it inverse graphics: from visual information received by eyes, they deconstruct a hierarchical representation of the world around us and try to match it with already learned patterns and relationships stored in the brain. And the key idea is that representation of objects in the brain does not depend on view angle.

In 3D graphics, relationships between 3D objects can be represented by a so-called pose, which is in essence translation plus rotation. In order to correctly do classification and object recognition, it is important to preserve hierarchical pose relationships between object parts.

When these relationships are built into internal representation of data, it becomes very easy for a model to understand that the thing that it sees is just another view of something that it has seen before. Capsule network embed these pose relationships explicitly hence be trained with few number of samples.

3.3.2 Capsules

Capsule is a set of neurons which are activated for different image features like, position, size and hue. They encode probability of detection of a feature as the length

of the output vector. If the same feature appears in two different orientations the length will remain the same while changing the vector orientation. Similar to the neurons receive inputs from other neurons and multiply them by weights and summing them and input to nonlinear activation function, capsules also perform analogous operations to output a vector instead of scalar in ANN.

3.3.3 Training in Caps-Net

With all the understanding about capsule process, next thing to be explored is how the training is happened in capsule network. This will adjust the weight matrix values (w_{ij}) and scalars (c_j). This particular mechanism is called routing and in the recent years many researchers have introduced different routing mechanisms. The three routing mechanisms; Dynamic Routing (DR), Expectation Maximisation Routing (EMR) and Attention based Routing are described next.

3.3.3.1. Dynamic Routing

As mentioned, capsule is a group of neurons whose activity vector represents object's visual parameters and the length of the vector represents the probability of presence of that particular object. The capsule operation is analogous to that of traditional NNs where a set of activations from lower level capsule agrees upon higher level feature and activates its neuron.

Dynamic routing[40] is the earliest approach for routing between capsules and introduced in 2017. It is a 'routing by agreement' method where the lower level capsules' output to higher level capsules are determined by the magnitude of the scalar multiplication between lower and higher level capsule vectors.

The main supportive factor that capsule-layers are capable of dynamic routing is that the output of neuron is of vector form. At the first iteration, all the lower level capsules are routed to all the parent capsules. The strength between lower – higher level capsules are determined by a factor called coupling coefficient. The sum of all coupling coefficients from one lower level capsule to all higher level capsules is scaled down to one.

Similar to CNNs the higher layers of capsules cover large regions of the image. In the introductory paper[40] they mention that lower level capsules are 'place-coded'. This

indicates that the object's location information is determined by which capsule is activated. At higher level capsules, the positional information are 'rate-coded'. This implies that the existence of objects are scrutinized at higher level capsules.

Dynamic Routing – Theory

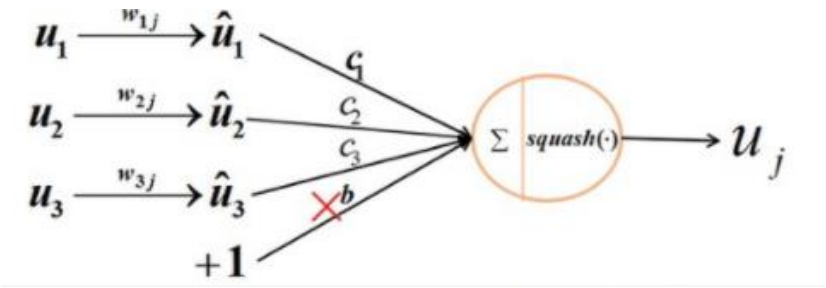


Figure 3.3: Capsule Input-Output

The u_i are the output vectors of previous capsule layer, w_{ij} encodes the relationship between jth capsule and ith feature. For an example, if jth capsule represents a human face inside a picture, and ith capsule represents lower level feature like 'nose' w_{ij} represents the nose-face relationship. After multiplication with w_{ij}s output is the predicted position of the higher level feature w.r.t. lower level feature. If all multiplications give similar output it can be concluded that there is a face in the image. The scalar c depends upon to which higher level capsule the lower level capsule should send the input, it can be understood as this; nose should have a higher c value for face than that of the finger which is not a part of the face. The squash operation in the diagram scale the output to 0-1 range in the same time imposing non-linearity.

The squash function is defined as follows.

$$u_j = \frac{\left||s_j|\right|^2}{1+\left||s_j|\right|^2}\frac{s_j}{\left||s_j|\right|} \quad (1)$$

Here, u_j is the output vector of the capsule and s_j is the summed input to the capsule.

$$s_j = \sum_i c_{ij}\hat{u}_{j|i} \quad (2)$$

$$\hat{u}_{j|i} = w_{ij}u_i \quad (3)$$

The previously mentioned coupling coefficients are indicated as c_{ij} in the above equation. These coupling coefficients are derived in following manner initially. The b_{ij} are the log probabilities that capsule i should be coupled to capsule j.

$$c_{ij} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ik})} (4)$$

The agreement between one lower level capsule and one higher level capsule is determined by the following product.

$$a_{ij} = v_j . \hat{u}_{j|i} (5)$$

The overall routing algorithm pseudo code can be displayed as follows.

---

procedure ROUTING(***u*** ˆ*j*|*i*, *r*, *l*)

for all capsule *i* in layer *l* and capsule *j* in layer (*l* + 1): *bij* ← 0.

for *r* iterations **do**

for all capsule *i* in layer *l*: **c***i* ← softmax(**b***i*) (softmax computes Eq. 4)

for all capsule *j* in layer (*l* + 1): **s***j* ← $\sum_i cij$ˆ**u***j*|*i*

for all capsule *j* in layer (*l* + 1): **v***j* ← squash(**s***j*) (squash computes Eq. 1)

for all capsule *i* in layer *l* and capsule *j* in layer (*l* + 1): *bij* ← *bij* + ˆ**u***j*|*i*.**v***j*

return v*j*

3.3.3.2 EM Routing

The EM Routing [41] was also introduced by Geoffry Hinton and his team in 2017 after few days of introduction of DR algorithm. This routing mechanism seeks more on different set of routing properties than that of DR algorithm.

In EM routing the capsules are grouped to build part-whole relationship using the clustering technique; EM. The basis of the EM routing is to cluster the data points into Gaussian distributions. The lower level capsules which represent basic features of image vote for higher level capsules with transformation matrix multiplication similar

to the Dynamic routing mechanism. This transformation matrix learns over the training iterations with the help of EM algorithm.

The basis of the EM routing is as follows. Assume, there is a need to cluster data points into two clusters; G1 and G2. These clusters are Gaussian distributions defined by mean μ and standard deviation σ. The EM algorithm converge until all the data points in the dataset belongs to two clusters maximizing the probabilities.

$$Max\ (\sum_{j=1}^{m}\sum_{i=1}^{n} P(x_i|G_j))\ (6)$$

$$\text{Here, } P\left(x_i\middle|G_j\right) = \frac{1}{\sigma_j\sqrt{2\pi}} e^{-\left(x_i-\mu_j\right)^2/2{\sigma_j}^2}\ (7)$$

The capsule o/p computation is different in EM routing. Here, the pose matrix of the capsule is also presented as Gaussian distribution. In EM routing, the pose matrix of parent capsule is presented as a Gaussian distribution. One such pose matrix is represented as 16 Gaussians having 16μs and 16σs where μ s are extracted from pose matrix components.

Let v_{ij}^h be the h-th component of vote from child capsule i to parent capsule j. The probability of v_{ij} belongs to capsule j is calculated with following Gaussian distribution.

$$p_{i|j}^h = \frac{1}{\sqrt{2\pi(\sigma_j^h)^2}} exp^{\left(-\frac{(v_{ij}^h-\mu_j^h)^2}{2\left(\sigma_j^h\right)^2}\right)}\ (8)$$

Let's take the log of $p_{i|j}^h$.

$$\ln(p_{i|j}^h) = \ln\left(\frac{1}{\sqrt{2\pi(\sigma_j^h)^2}} exp^{\left(-\frac{(v_{ij}^h-\mu_j^h)^2}{2\left(\sigma_j^h\right)^2}\right)}\right)$$

$$= -\ln\left(\sigma_j^h\right) - \frac{\ln(2\pi)}{2} - \frac{(v_{ij}^h-\mu_j^h)^2}{2\pi(\sigma_j^h)^2}\ (9)$$

The minus of $\ln\left(p_{i|j}^{h}\right)$, or in other words negative of the log likelihood is considered as the cost.

$$cost_{ij}^{h} = -\ln(P_{i|j}^{h}) \text{ (10)}$$

The lower level capsules are not equally linked with the higher level capsules, hence the cost for one capsule from all lower level capsules is calculated as,

$$cost_{j}^{h} = \sum_{i} r_{ij} cost_{ij}^{h} \text{ (11)}$$

With substitution from Eq.(9) cost is derived as,

$$cost_{j}^{h} = \left(\ln\left(\sigma_{j}^{h}\right) + k\right) \sum_{i} r_{ij} \text{ (12), here k is a constant.}$$

Following equation determine whether the capsule j will be activated.

$a_j = sigmoid(\lambda(b_j - \sum_h cost_j^h))$, here $b_j$is referred as the cost of describing the mean and variance of capsule j, and λ as the inverse temperature parameter. As r_{ij}s are trained, λ is increased to steepen the sigmoid curve.

The EM routing is used for two operations inside the capsule network; pose matrix calculation and the capsule output calculation. The overall EM routing algorithm consists of two steps called, E-step and M-step. In E-step, r_{ij} s are calculated, whereas in M-step Gaussian model parameters μs and σs are re-calculated.

3.3.3.3 Inverted Dot Product Attention Routing

In previous section, two most widely used routing mechanisms for Capsule networks; Dynamic routing and EM routing were described. Next, in this section one of the most recent approaches of routing; Inverted Dot Product Attention Routing [42] is presented. This method was introduced to achieve the same classification accuracy with fewer number of training parameters than that of the Dynamic Routing and EM routing with the ultimate goal of getting Capsule Networks into real-world tasks.

In this project, this can be identified as the newest technology integration and holds majority of the novelty which is introduced in this project.

The overall architecture of the Capsule Net is different from the previous methods, obviously in routing mechanisms and also with newly introduced layer normalization. Here, inverted attention mechanism is used to measure agreement between capsules. In normal attention routing [43] the child capsule units compete to get the attention of parent capsule units, whereas in this method parent capsules compete to get the attention of child capsule units. The routing probability of child capsule units to parent capsule units depends upon two factors; parent's pose (from previous iteration) and the child's vote for parent's pose (current iteration). Not limiting to the above differences, clear differences from Dynamic routing and EM routing are identified as follows. In Dynamic routing[40] the pose is expressed as a vector and the activation as it's norm. With EM routing, the pose has expressed as a matrix and the activation is achieved with EM algorithm. With inverted dot product approach, the pose is expressed in a similar matrix form as EM but the activation is not directly obtained.

The Inverted Dot Product Attention Routing mechanism has two steps; the first one is to compute the agreement between child and parent capsules and the next is to update the pose matrix of parent capsule. These two steps are pretty easy to understand when compared to EM routing algorithm.

Computing Agreement:

---

procedure INVERTED DOT-PRODUCT ATTENTION ROUTING($\mathbf{P}^L$, $\mathbf{P}^{L+1}$, $\mathbf{W}^L$)

for all capsule *i* in layer *L* and capsule *j* in layer (*L* + 1):$v_{ij}^L \leftarrow W_{ij}^L . p_i^L$ **vote**

for all capsule *i* in layer *L* and capsule *j* in layer (*L* + 1): $a_{ij}^L \leftarrow p_j^{L+1} . v_{ij}^L$ **agreement**

Vote v_{ij}^L is calculated as the product between transformation matrix W_{ij}^L and pose matrix p_i^L. Next, the agreement is calculated as the product between vote v_{ij}^L and the parent capsule pose p_j^{L+1}.

Pose Update:

Extending the same routing function, three more steps were added to update the pose and for layer normalization.

for all capsule *i* in layer L: $r_{ij}^{L} \leftarrow \exp\left(a_{ij}^{L}\right) / \sum_{j'} \exp(a_{ij'}^{L})$. **routing coefficient**

for all capsule *j* in layer $(L + 1)$: $p_j^{L+1} \leftarrow \sum_i r_{ij}^{L} v_{ij}^{L}$ **pose update**

for all capsule *j* in layer $(L + 1)$: $p_j^{L+1} \leftarrow \ LayerNorm(p_j^{L+1})$. **Normalization**

The routing probabilities r_{ij}^{L} is calculated by applying softmax function over a_{ij}^{L} s which are calculated in the previous step. Then, the parent capsule layer pose is updated with summing all the product between routing probability and votes. Finally, a layer normalization is done for this new pose matrix.

One of the other distinguishable characteristics introduce in this method is the concurrent routing. After the first iteration of forward pass, the rest of the iterations are happened concurrently here. All the capsules get input from previous iteration preceding capsules and do one iteration simultaneously. This is indicated in Fig 3.4.

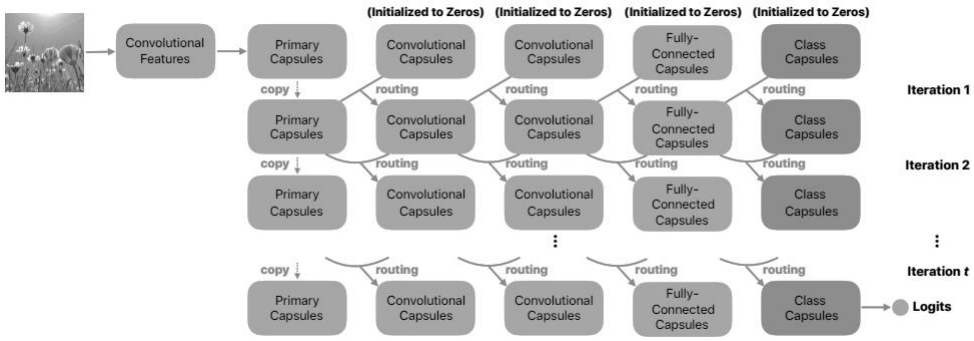


Figure 3.4: Concurrent Routing (Source [42])

However with these modifications in the forward pass, the backward pass or the learning is still based on the Stochastic Gradient Descent algorithm.

In the SR implementation, the capsule layers from Conv-Capsule layer to Fully-connected Capsule (Appendix I) layer have been used. We also implement the concurrent routing mechanism that is introduced above expecting better time performance at training phase.

3.4 Deconvolution (2D)

The deconvolution is a mathematical process to restore the original signal which has undergone the process convolution [44]. In mathematical formula it can be described as follows.

$$f * g = h \text{ (13)}$$

Here f, is the signal to be recovered and * denotes the convolution operation. The convolution filter or the transfer matrix g should be known to recover the original signal f from the convolved output h. When g is unknown, but the form of g is known there are statistical methods to approximate g.

Once g is approximated or known, f can be recovered by the following equation.

$$F = \mathrm{H}/\mathrm{G} \text{ (14)}$$

$$f = IFT(F) \text{ (15)}$$

Here F, G and H are the Fourier transforms of the functions f,g and h respectively. IFT is the Inverse Fourier Transform operation.

In this application, deconvolution operation is carried out in 2D domain after the feature mapping layers to generate the image. With the back propagation, the parameters analogous to G are learnt and F (final image) could be recovered.

3.5 Summary

In this chapter, an insight into CNN, the previous generation giant and why anew approach Caps-Net is used as a better alternation, describing its strengths and limitations are presented. The overall theory behind Caps-Net is also be described in detail. Furthermore, three of the Caps-Net architectures different from the routing mechanism which are used in this research are intensely explained.

Eventually, the reconstruction technique; 2D De-convolution is described. In the next chapter, the overall research approach will be put forward.

Chapter 4 Caps-Net based Approach for SR

4.1 Introduction

Using the technology described in the Chapter 3; Capsule Networks an approach for Super Resolution is proposed as follows.

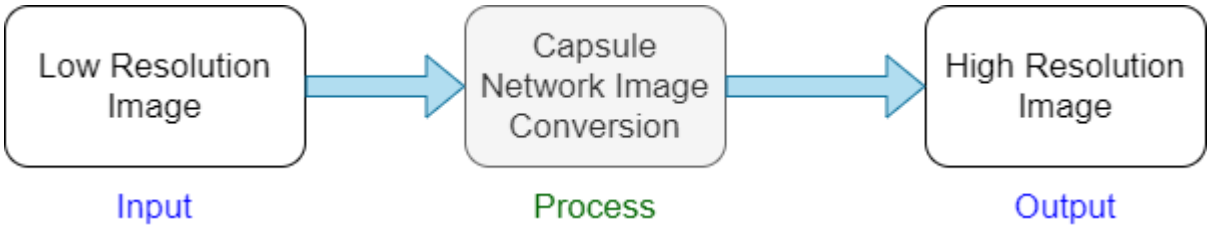


Figure 4.1: Approach

The rest of the chapter will describe the IPO (Input, Output & Process) along with system users and finally the system features.

With this approach, following hypothesis is set for the project.

Hypothesis – The image quality performance of Capsule Net based Super-Resolution model for medical image resolution enhancement is further improved by changing the routing mechanism and the layered architecture.

4.2 Input

The inputs to the system are low resolution medical images of grey scale (single channel) digital images. These inputs are generated from different imaging modalities like; MRI Scanners, PET Scanners or CT Scanners, etc. These images are non-other than cross slices of the human body.

4.3 Output

The output of the system will be a high resolution version of the input image. The output will also be a grey scale image and the scaling factor will be an integer. The new image will include the information interpolated by the Caps-Net model.

4.4 Process

The Caps-Net Module will get the input as low resolution image and after several Capsule conversions and internal reconstruction, it will generate the higher resolution images. In the training phase the difference between the expected high resolution image and the conversion module based output is compared and fed back to the model for learning. After the training, the trained model is used as the process core. With the saved model, the high resolution images can be produced in seconds.

4.5 Users

The proposed solution is particularly focused on a specific type of images; medical images. Hence the system users will be Radiologist, Other medical consultants and Imaging Technicians. The Radiologists and Imaging Technicians will be using the system at the image lab whereas the other consultants will be using the system on PCs at clinics.

4.6 Features

With the proposed approach following features can be given out by the system.

1. Scale up grey scale images to 2x or 4x
2. Used with all imaging modalities

In this chapter, an overview of the Approach using the system Inputs, Outputs, Process, Users and Features has been given. The process which has been described in this chapter will be expanded as the design in the next chapter.

4.7 Summary

The current chapter can be considered as the essence of the overall project as it briefs about the components in the project. Here, the input, the process, and the output generated by the process are described along with the other relevancies of the project; users of the project and special feature stretched out in the implementation. In the next chapter, the overall design of the process described in this chapter is expanded.

Chapter 5 Design of Caps-Net SR

5.1 Introduction

This chapter expands the system process explained in the previous chapter as the system design by dividing the system into sub-modules. The design diagrams will be used for the core Caps-Net module explanation and minor Data Generator module explanation. Even though the Data Generator module provides the input for the Caps-Net module, the latter will be first explained for the reader's clarity.

5.2 Data Generator Module

This module acts as the data supplier for the capsule network. With this module, the dataset used for training and validation van be expanded quickly. Even when there is no standard dataset, data can be generated with the help of this model.

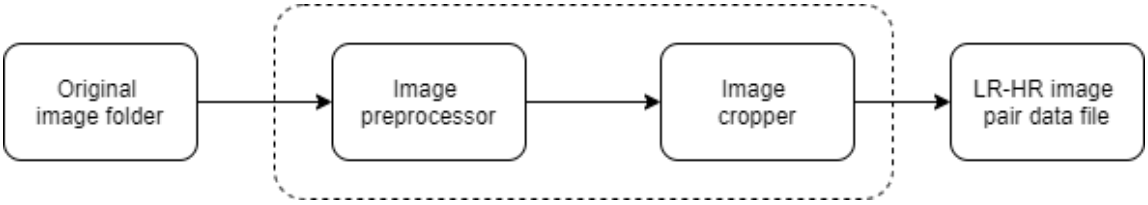


Figure 5.1: Data Generator Module Components

5.2.1 Image Preprocessing Module

This is the first component of the Data Generator pipeline. Here, the input images, which are in RGB format first converted to greyscale images. Then, to obtain a larger amount of training and testing samples, the images were augmented by rotating the images into several different angles. This will increase the number of image samples.

5.2.2 Image cropper

The augmented images are then passed through the image cropper module to obtain the final outputs. It first crops the HR image from the original image that is of the size

of the desired HR image. Then, it scales down the cropped image to generate its LR image. These two LR-HR pair is saved for training and validation.

5.3 Caps-Net SR Module

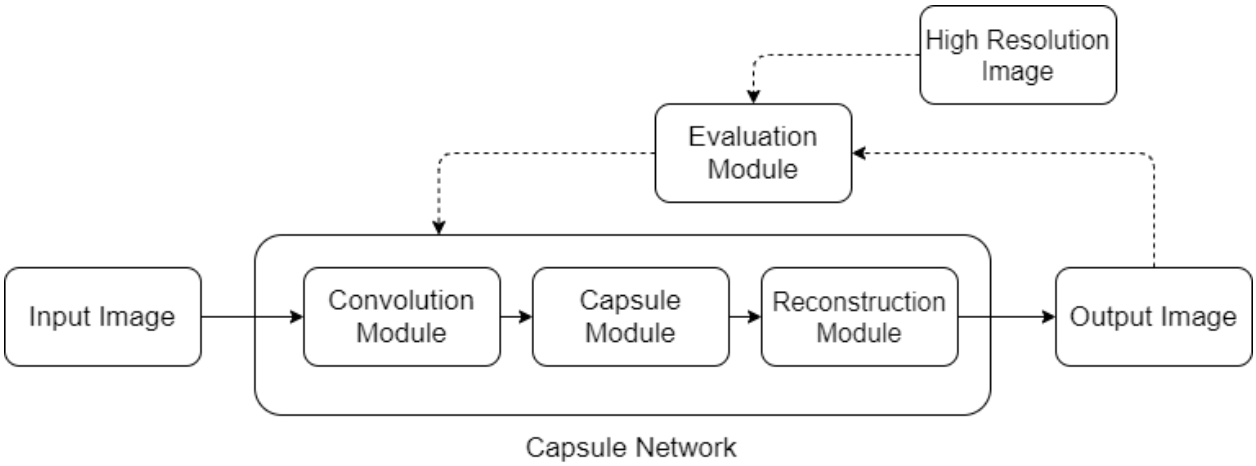


Figure 5.2: Capsule Net Components

5.3.1 Input Image

This is the input to the Capsule Network. It is a low resolution image patch. During the training phase of the network, these images are the lower resolution part of the generated low-high resolution image pairs. Afterward, they act as the input to the system in the validation and in practical. These low resolution input images are of grey scale.

5.3.2 Convolution Module

These are identical to the convolution layers in CNN. The primary feature extraction is achieved in these layers before converting them as capsules. They play a vital role by filtering the most important features of any input image given as input (See 5.2.1).

5.3.3 Capsule Module

The capsule module acts as the core to the new approach. Its function is to embed the richer and more sophisticated features that could interpolate the higher resolution structures. They store pose and probability information about the features input via

Convolution module. These internal representations of the features inside the capsule module will be input to the Reconstruction Module.

5.3.4 Reconstruction Module

The reconstruction module does the final rendering of the higher resolution output. As mentioned in 5.2.3 the feature information generated by the Capsule Module will be input and arranged in such a way that the higher resolution image is output. As an overview of this module, one can imagine this as feature to image mapper.

5.3.5. Output Image

This is the output of the reconstruction module. It is a grey scale image of high resolution. During the training session, this is fed into the Evaluation module, eventually, this will act as the overall system output. As the output of the system, it acts as the reflection module of the system performance.

5.3.6 Evaluation Module

This module will only be used in the training process to evaluate the Caps-Net performance and giving the feedback to the Caps-Net to adjust the weights accordingly. In the case of Caps-Net it compares the two high resolution pairs; reference output and Caps-Net output by generating a Signal to Noise Ratio (PSNR) value. When this value is high, the system performs better.

5.3.7. High Resolution Image

This is the reference/original high resolution image solely used in the training process of the Caps-Net. It acts as an input the Evaluation module and used to calculate the PSNR.

5.4 Evaluation Module

Two evaluation instances have been conducted in this project. One evaluation happens at the time of training itself (See 5.3.6).

For a more comprehensive evaluation, after the training, a new medical image dataset has been used. With his dataset, HR images from different trained Capsule Net models were generated. Next, overall evaluation is performed for different image quality

matrices namely Peak Signal to Noise Ratio (PSNR),Structure Similarity Index (SSIM)[45], Multi Scale SSIM (MSSSIM)[46] and Universal Image Quality Index (UQI)[47].

This part is implemented as two separate sub-modules. The first module is to generate new evaluation data set. It first runs over a medical image dataset and generates up-scaled images for each of the Caps Net module and save them in folders.

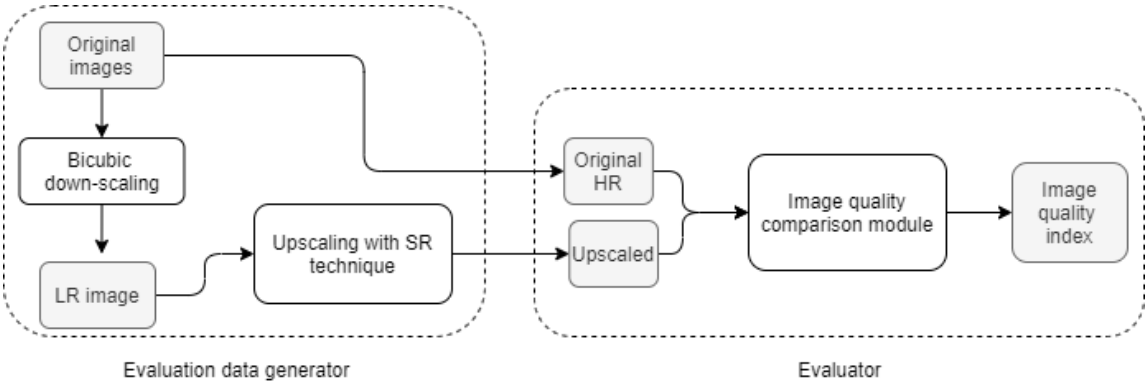


Figure 5.3: Image Quality Evaluation Model

Secondly, the evaluation model gets two inputs, an up-scaled image from the SR technique and the original image to generate the image quality matrix. Inside the image quality comparison module, above mentioned different quality measures have been implemented. The output of the module will be the image quality index by each strategy.

5.5 Summary

This chapter gives an insight into the overall design components of the project. It is prolonged under several major components of the system, namely, Data Generator Module, Caps Net based SR Module, and the Evaluation Module. What tasks are completed by each module and sub-modules are described in detail. In the upcoming chapter, it will be spread out how each of these design components is implemented and realized.

Chapter 6 Implementation

6.1 Introduction

In this chapter, the designs described in the previous chapter are realized. Each module described in the Design chapter is expanded in the following aspects not only limiting to Software, Hardware, Algorithms, Flow Charts and Code segments.

Any special hardware component for this project except the GPU in the PC and cloud environment was not used.

6.2 Data Generator Implementation

6.2.1 Dataset

Similar to any other Deep Learning based project data is a deciding factor for the success of the project. The focus of this project is to implement SR module for particularly in the medical image domain. Hence, it was required a large dataset for the training purpose. The freely available online datasets of medical images were downloaded for the purposes.

Following datasets are used in this project.

- Kaggle Brain MRI dataset for Tumor Detection - https://www.kaggle.com/navoneel/brain-mri-images-for-brain-tumor-detection - 98 Images
- SIIM dataset (CT) - https://www.kaggle.com/kmader/siim-medical-images - 100 Images

These two datasets are used to generate the low-high resolution pairs with our Data Generator Module.

6.2.2 Data Generator

The complete implementation of this project was done with Python programming language, so was the image generator. For this, two separate image sets were used for testing and evaluation purposes, respectively.

The following flowchart describes the overall implementation of the data generator module.

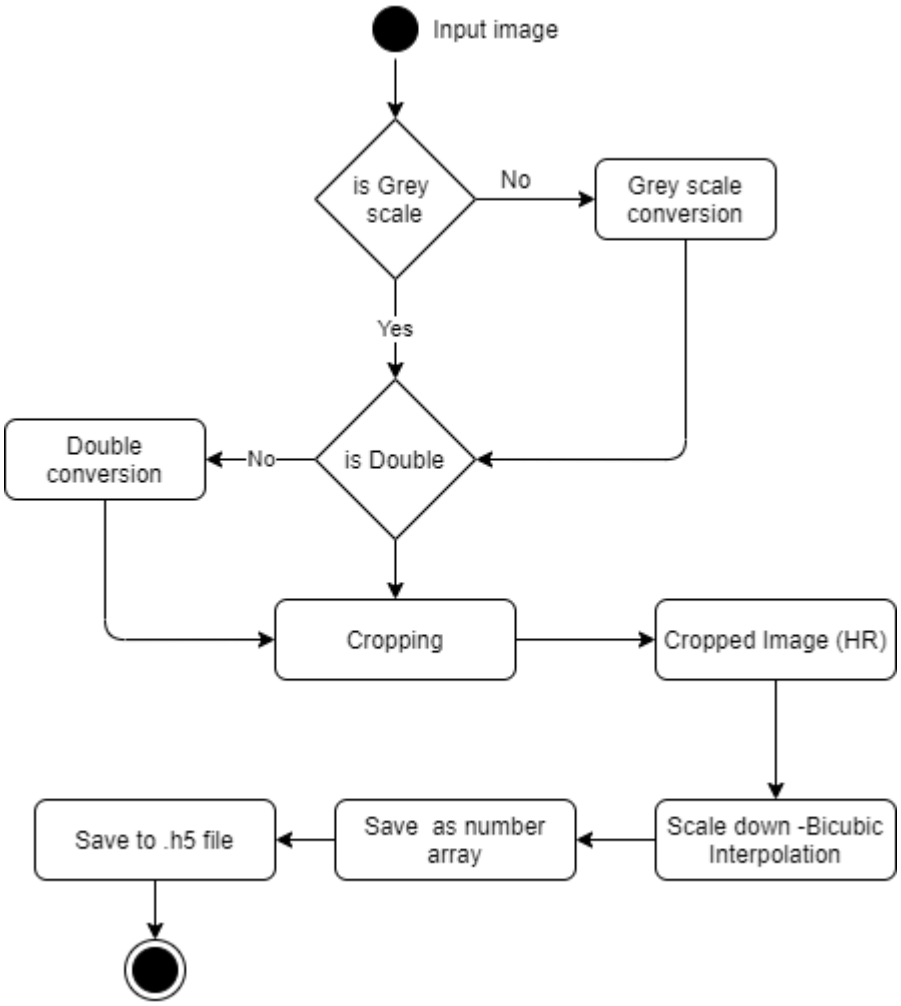


Figure 6.1: Data Generator – Flow Chart

The original input datasets mentioned in 6.1 were sent through the data generator process as above. For this implementation Python packages; *PIL, NumPy* and *h5py* were used. PIL and *NumPy* were used for pixel operations and *h5py* was used for training and evaluation data file generation.

The module checks whether the input images have several layers or one if it is an RGB image, then it will do the grey scale conversion with pil_image conversion module.

After that, if the image was represented as an integer (0,255) it was converted to 0-1 range by dividing it by 255. After the conversion, images were cropped either to 20x20 sub images. These were considered the high resolution reference images. When cropping the images, a stride of 10 pixels was considered.

The reference images were then scaled down using the popular Bicubic interpolation technique. Here a separate function was used for down-sampling of the images. A downsampling factor of 2 was used to obtain 10 x 10 images.

Then, they both reference and down-sampled patches were converted to NumPy array. After iterating through the images in either training or evaluation test, the NumPy array was written to the h5 file.

By running on two image datasets of training and evaluation, training.h5 and eval.h5 files were obtained. (Appendix II)

6.3 Overall implementation

The core of the system is the Caps-Net module. For the new implementation, the official implementation of the FSRCNN [37] with Pytorch [48] was used as the base architecture.

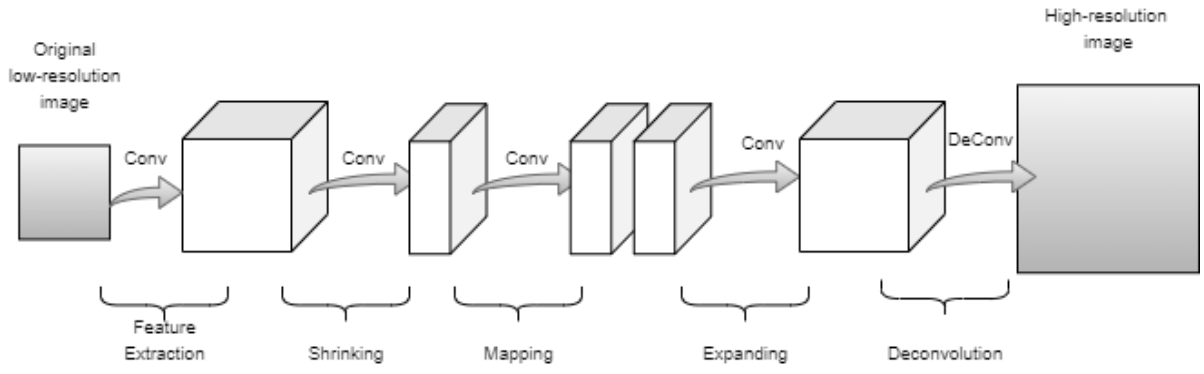


Figure 6.2: FSRCNN Architecture

It consists of two components re-used directly with the new implementation; Convolution Layers and Reconstruction (De-convolution) Layers. The core Capsule model was introduced between these two components bringing the novelty in this research.

6.4 Re-usable Layers

6.4.1 Initial Convolution Layers

The convolution layers were implemented using the standard Pytorch Conv2D function. The number of filters, kernel size, stride and padding parameters were changed to extract different features at different scales. Several parameter settings were used and the highest accuracy figure was given by the following configuration.

Number of filters – 56
Kernel size – 3x3
Stride – 1
With padding – 1 each side

6.4.2 Reconstruction Layers

The Reconstruction Module in FSRCNN is also a reusable component in the new implementation. It is using Deconvolution (ConvTranspose2d) algorithm implemented by Pytorch. The deconvolution layer consists of several configurable parameters; kernel size, output stride and padding.

The DeConv configuration that gives the highest accuracy is as follows.

Kernel Size = 9x9
Output Stride = Scale factor -1
Padding 4

The core of the implementation, Caps Net was implemented along with three routing mechanisms, namely, Dynamic routing, EM routing and Inverted Dot Product based Attention routing. The next section is dedicated to the description of the implementation of these Caps Net models.

6.5 Dynamic Routing

As described in section 3.4.3.1 Dynamic Routing was the first implementation of Caps-Net inside the system. The architecture of the DR layers can be identified by Fig. 6.3. The implementation of the layers was inspired by the official DR implementation (MNIST classification) for PyTorch [49].

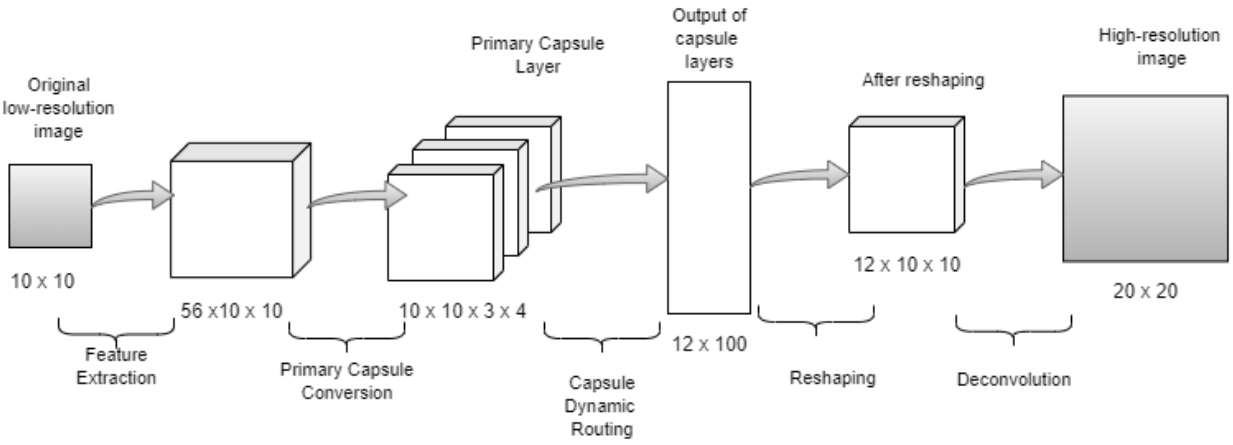


Figure 6.3: DR based Caps-Net Architecture

The following modifications were imposed on the existing DR implementation.

1. Modification of Primary Capsule layer such that kernel size, striding and padding can be configured dynamically.
2. Modification of Routing Capsule layer model such that different input sizes are accepted for the first layer.

After the changes, the highest performance architecture was given as follows. (Appendix I – DR Training)

For the Primary Capsule Layer

Output channels - 12
Capsule dimensionality - 4
Total number of capsules – 12 (12/4 = 3 Capsules blocks)
Kernel size -1

Stride -1
No Padding

Routing Capsules (3 layers). For each layer following same configuration was used.

Total number of capsules - 12
Dimensionality of the output capsule layer – Original image size
(Appendix III)

This capsule implementation was placed in the FSRCNN architecture replacing the middle part of the CNN architecture.

6.6 Expectation Maximization

The EM routing mechanism was implemented as the second method. The architecture of the Caps Conv layers in the EM routing mechanism can be identified by Fig. 6.4. The implementation of the layers was inspired by the official Matrix-Capsule-with EM routing implementation (MNIST classification) for PyTorch [50].

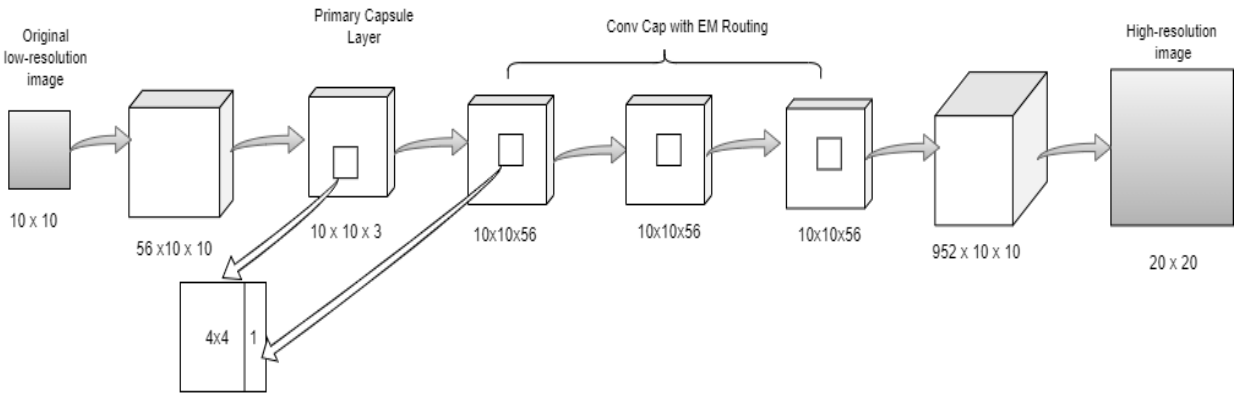


Figure 6.4: EM Routing based Caps Net Architecture

In contrast to Dynamic routing architecture, the Primary capsule layer consists of 3 capsule layers each consisting of a 4x4 pose matrix and an activation. Hence, the capsule dimensionality is 51(= (4 x 4+1) x3).

For the Primary Capsule Layer

Output channels - 3
Pose matrix – 4 x 4

Kernel size -1
Stride -1
No Padding

With comparable to DR Routing technique, 3 Convolution Capsule layers were implemented and routed with EM routing mechanism.

For the Conv-Capsule layers following parameters were used. The number of channels of 56 was obtained with several configuration changes and evaluating their accuracies.

Output channels – 56 (This is set for all 3 Capsule layers)
Pose matrix – 4 x 4
Kernel size -1
Stride -1
No Padding

With this setting, the density of the output channel becomes 952 (= (4x4+1)*56).

After the iterations, the output was reshaped such that it was compatible with the deconvolution layer. Here, the density (952 in this case) was switched as the first dimension of the output. This output was set as the input to the deconvolution layer. (Appendix IV)

6.7. Attention based Routing

The latest implementation of 'Inverted Dot Product based Attention Routing' is introduced in this section. Similar to the previously mentioned mechanisms, the same SR pipeline was used. Only the core of the pipeline was replaced with the Attention Routing Capsules.

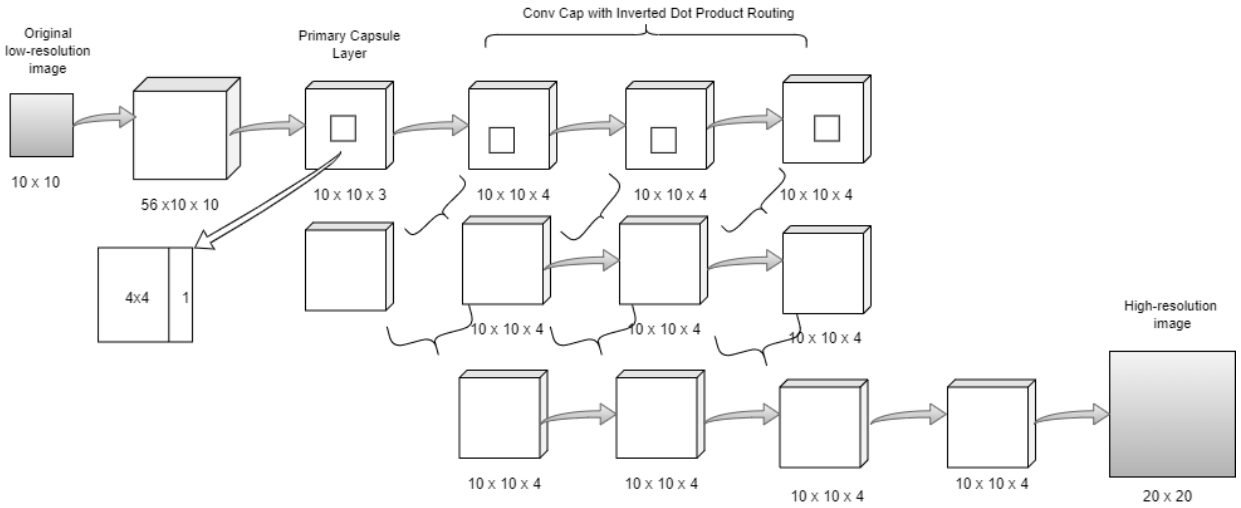


Figure 6.5: Inverted Dot Product based Routing Caps Net Architecture

For this, the code was modified with the help of the official implementation of Inverted Dot-Product Attention Routing [51].

Up to the Primary Capsule layer, the structure was the same as that of the DR architecture. After that, Attention Routing was adopted as follows.

Output channels - 4
Pose matrix – 4 x 4
Kernel size -1
Stride -1
No Padding
(Appendix V)

This same structure was repeated for 3 layers. During the training process, the concurrent routing mechanism was endorsed to reduce the overall training time. Finally, the Conv-Caps output is routed through the De-Convolution layer to generate a higher resolution image.

6.8 Training

The training was performed on a PC with the following specifications.

RAM -16 GB
GPU - 4GB, NVidia (1650Q)
CPU – Intel Core i7 – 9750H CPU of 2.6GHz

All the training was done under the utilization of GPU by compelling models and dataset into GPU arrays with the PyTorch framework based Cuda – GPU commands.

The dataset and training iterations for the training purpose were configured as follows.

Batch size – 16
Epochs – 20
Dataset – 307,520 image pairs (10x10 LR and 20x20HR)
Learning rate - 0.001

After 20 epochs, the model with the best PSNR ratio was saved as the trained model in '.pth' format.

6.9 Summary

The practical realization of the Capsule Network based SR system is explicated here. With the detailed diagrams, the implementation of each design component explained in the previous chapter is described in detail with code segments, network configurations and training configurations, etc. The upcoming chapter describes how the implemented system was evaluated against the desired expectations.

Chapter 7 Evaluation

7.1 Introduction

The main goal in this section is to compare the CapsNet based SR techniques which are introduced in this project against state of the art CNN based methods as well as the traditional highly used SR technique. For the comparison, not only limiting to PSNR, several other image quality assessment indexes have been used. A brief overview of these indices is described in the following section.

In addition to the method comparison over image accuracies, time to train, training accuracy improvement rate parameters are also evaluated for further information.

This chapter describes the evaluation strategies, experimental design and evaluation results in an orderly manner.

7.2 Evaluation Strategy

7.2.1 Evaluation at Training

For this, the evaluation dataset obtained at the data generation phase was used. There, 50,000 LR and HR image pairs have been used for evaluation with a batch of 16. Peak Signal to Noise Ratio (PSNR) has been used for this evaluation and acted as an indicator of the performance of the training phase.

The PSNR calculation function is simply implemented by considering the PSNR formula and called at the evaluation instance. (Appendix VI)

7.2.2 Overall Evaluation

7.2.2.1 Evaluation Data Generator

The evaluation data generator sub-module was implemented as follows. First, a new data set of medical images was selected (https://www.kaggle.com/navoneel/brain-mri-images-for-brain-tumor-detection). An image cropper program was built for this purpose. It was given the HR image window size and the scaling factor as inputs for this program. This program reads all the image files in a given directory and crops the sub images from each image. Then, those sub images are saved in a folder in .png

format. Simultaneously, a downscaled version of those images too is saved in another folder. Image size of 100x100 and scaling factor of 2 were used for this purpose. It indicates that two folders are containing 100x100 and 50x50 image samples in gray scale. (Appendix VII).

Then, with a 'Zooming' application the downscaled images were converted to up-scaled image samples. This 'Zooming' application accepts the trained model file and the LR image as input. As the current set of models were trained for zooming 10 x10 patches into 20 x 20 patches, 10x10 patches were extracted from the LR images. For 50 x 50 images, there are 25(= 5 x 5) such patches. Each patch is up-scaled using different SR models; Bicubic, FSRCNN, DR, EM and Attention routing. Finally, they were merged to generate one 100x100 up-scaled image. These images were saved in different folders corresponding to the SR technique. (Appendix VIII)

7.2.2.2 Data Evaluator

This was developed as a separate Python module. It reads the images from the above mentioned folders, and the original folder to calculate the image comparison matrix. For this, several image quality assessment methods have been used. The implementation was not done from the ground level instead of the python package 'sewar' [52]. The inbuilt modules; 'psnr', 'ssim', 'msssim' and 'uqi' have been used directly inside the evaluator module. (Appendix IX)

After running through all the images, it calculates the average values for all the quality indexes by dividing by the image count. In the next sub section, an overview of image quality indexes is given.

7.2.3 PSNR

The main evaluation method used in the project is the standard image quality comparison method that is used in the industry, called Peak Signal-to-Noise Ratio (PSNR). It measures the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation.

For a gray scale image, Mean Squared Error (MSE) is defined as follows.

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2 \text{(Eq.1)}$$

Here m & n are the width and the height of the image. In our case image is the high resolution SR output image. *'I'* represents the reference image whereas *'K'* represents the image output from our Caps-Net. As indicated by the equation, it is noticeable when the SR technique generated image is closer to the reference image MSE value will be smaller.

$$PSNR = 20.\log_{10} MAX_i - 10.\log_{10} MSE \text{ (Eq.2)}$$

Here, MAX_i is the maximum possible pixel value of the image. It is evident that, for the lower MSE images the PSNR value will be higher.

7.2.4 SSIM

The second evaluation index that is considered for evaluation is the Structural Similarity Index (SSIM) [45]. This index quantifies the image quality degradation from the original image. It's more associated with the human perception of differences between two images, such as luminance, contrast and structure. In contrast to PSNR, SSIM lays its foundation on the visible structures inside the image. This is because it has taken into consideration that the pixels have interdependencies in a small sub area. In some cases, SSIM is recognized to be a more convenient method than PSNR as it reflects human perception.

The formula for SSIM is given as follows.

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

Here, x,y represents the original image and the derived image respectively. The μ and σ have their usual meanings of mean and standard deviation of pixels inside a considered window.

SSIM is dependent upon the distribution of the pixel values. The range of possible values of SSIM is between (-1, 1) whereas 1 indicates perfect structural mapping.

7.2.5 MSSSIM

The Multi Scale SSIM [46] is one of the extensions to the SSIM. The SSIM is derived by combining 3 formulas indicating 3 elements; contrast, structure and luminance. These are highly dependent upon the sub window size of the view scale SSIM (Typically set at 8pixel, 10 pixel windows). This method tries to generalize SSIM by evaluating SSIM at different scales and deriving an index.

7.2.6 UIQ

The Universal Image Quality Index (UIQ) [47] was introduced before SSIM and laid the foundation for SSIM as well. The UIQ is defined as follows.

$$Q(x,y) = \frac{4\sigma_{xy}\mu_x\mu_y}{(\mu_x^2 + \mu_y^2)(\sigma_x^2 + \sigma_y^2)}$$

As indicated by the equation, its resemblance with SSIM is clearly evident. It helps to evaluate two images without considering the luminance conversion of image into HSV plane. This is because, the luminance distortion, contract distortion and loss of correlation is implicitly embedded into the equation.

The upcoming section describes how experimental setup was done to evaluate SR techniques over the above indices.

7.3 Experimental Setup

The core experimental setup can be described with the following steps.

1. Obtain randomly generated 2500 image pairs of low-high resolution of different medical imaging modalities
2. Input low resolution images into particular SR algorithm
3. Image quality index calculation via Evaluation module
4. Comparison among different SR algorithms

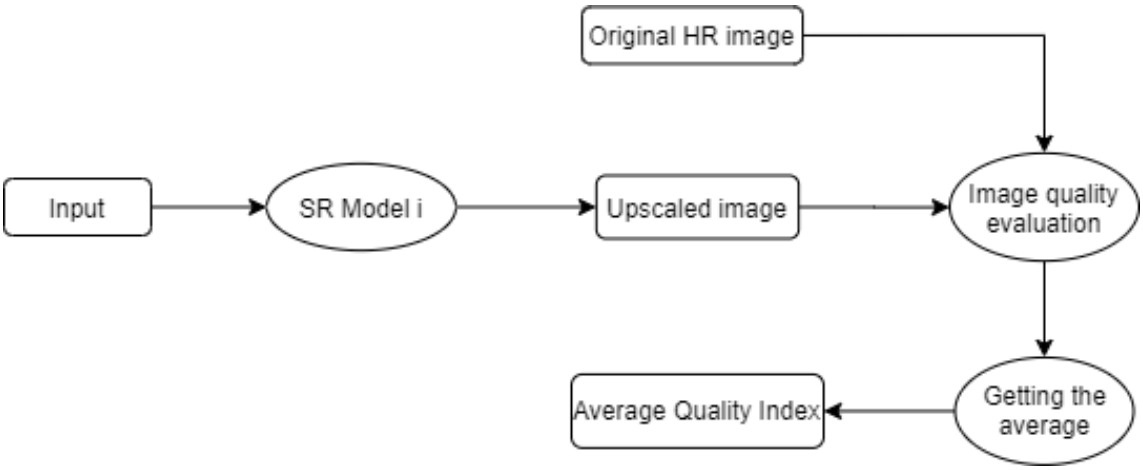


Figure 7.1: Experimental Setup for Evaluation

In extension to the, core evaluation training process and model parameter evaluation was also done with the data obtained at the training time.

7.4 SR Techniques Comparison

Table 7.1 provides a quantitative comparison of the generated dataset explained in Section 6.2. A comparison has been done for the proposed techniques against the state-of-the-art technique FSRCNN and the widely used the old method Bicubic interpolation method. The models were trained for scaling images up to 2x factor only.

Table 7.1: Quantitative Comparison of Results

SR technique	PSNR	SSIM	UQI	MSSSIM
Dynamic Routing	42.10754659	0.974436038	0.985257827	0.995784307
EM Routing	36.27424783	0.916861082	0.850859974	0.98513547
Attention Routing	40.32626865	0.950900289	0.89400693	0.994521225
FSRCNN	40.73684391	0.953492091	0.900293958	0.995011494
Bicubic	38.09835088	0.920734985	0.862577213	0.989270289

As indicated by the results, the Dynamic Routing mechanism shows the most outstanding results amongst all. By all indexes, it is reflected that the DR method leads not only in one index but in all of the indexes that have been used for evaluation. There is a significant difference between PSNR and UQI indices in the DR method. The latest method introduced in this research, the Attention Routing SR technique was performed

slightly below the current state of the art method FSRCNN. The FSRCNN method has the second best performance for the given dataset. However, this slight quality difference between FSRCNN and Attention Routing based method is not qualitatively different as indicated in the following section by the image results. Unexpectedly, the EM routing performance was even lower than the traditional bicubic method. This could be because the increased number of Caps-Conv layers may have resulted in over-fitting the training round evaluation dataset as well. For the versatile new evaluation dataset, it is unable to generate target capsule candidates to input for the reconstruction module.

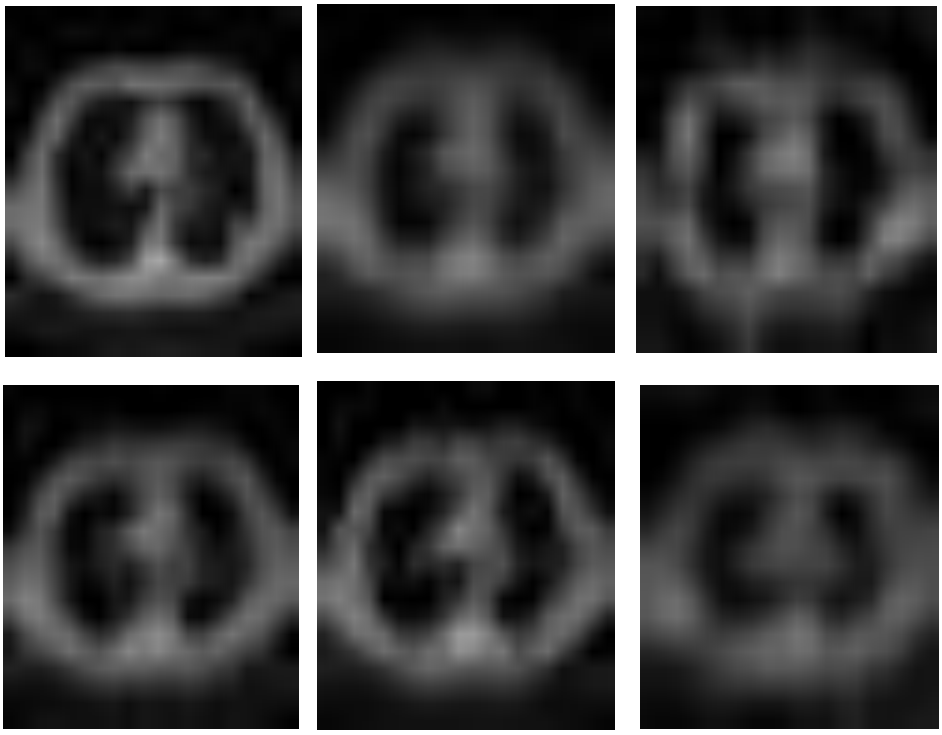


Figure 7.2: Original Image (Top-Left), Caps-Net Attention (Top-Middle), Bicubic (Top-Right), FSRCNN (Bottom-Left), DR (Bottom-Middle), EM (Bottom-Right)

In addition to the quantitative analysis, a qualitative results (Appendix-X) is also presented for further clarification of the system. There, a set of medical sub-images (10 images) were given for a medical doctor for evaluation. The medical doctor has given scores starting from 5 to 1 for each SR method. The highest score is assigned to the highest quality image and vice versa. The cumulative results show the similar implication as the above mentioned quantitative analysis. The DR method scores the

highest among all. The Attention method scores at the second place and in conflicting FSRCNN scores the third while dragging Bicubic and EM Routing methods fourth and fifth places respectively.

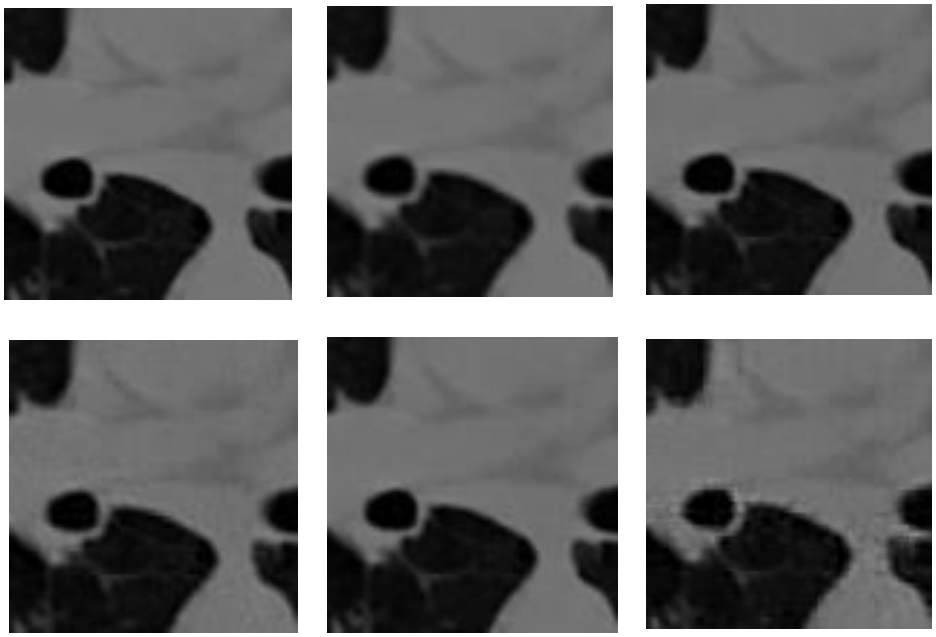


Figure 7.3: Original Image (Top-Left), Caps-Net Attention (Top-Middle), Bicubic (Top-Right), FSRCNN (Bottom-Left), DR (Bottom-Middle), EM (Bottom-Right)

Further to the image quality assessment, the training performance and model size analysis was also carried out. This is particularly important as the computing performance is also an impactful factor in medical imaging applications.

This comparison was done with fully dedicated PC for SR training process (No other program was run during the training process.)

Table 7.2: Training Performance of SR Techniques

Model	**Number of Trainable Parameters**	**Time per Epoch**
FSRCNN	12,809	7-11 minutes
Dynamic Routing	1,442,273	20-25 minutes
EM Routing	10,128	30-35 minutes
Attention Routing	2,274	12-15 minutes

With the analysis, it exhibits that FSRCNN has the best performance during the training and EM routing based SR technique takes the longest time for training amongst the tested methods.

Another important fact to be noticed is that, the number of trainable parameters show no correlation with the training time.

The powerful learning capability of Attention Routing in SR task is emphasized through this simple analysis as well. With 1/6 th of learnable parameters, Attention Routing based SR technique achieves the same level of accuracy of FSRCNN.

In addition to the time performance analysis, for the selected architectures from each of the techniques a training evaluation was also done. The average PSNR was considered to be the evaluation parameter for this.

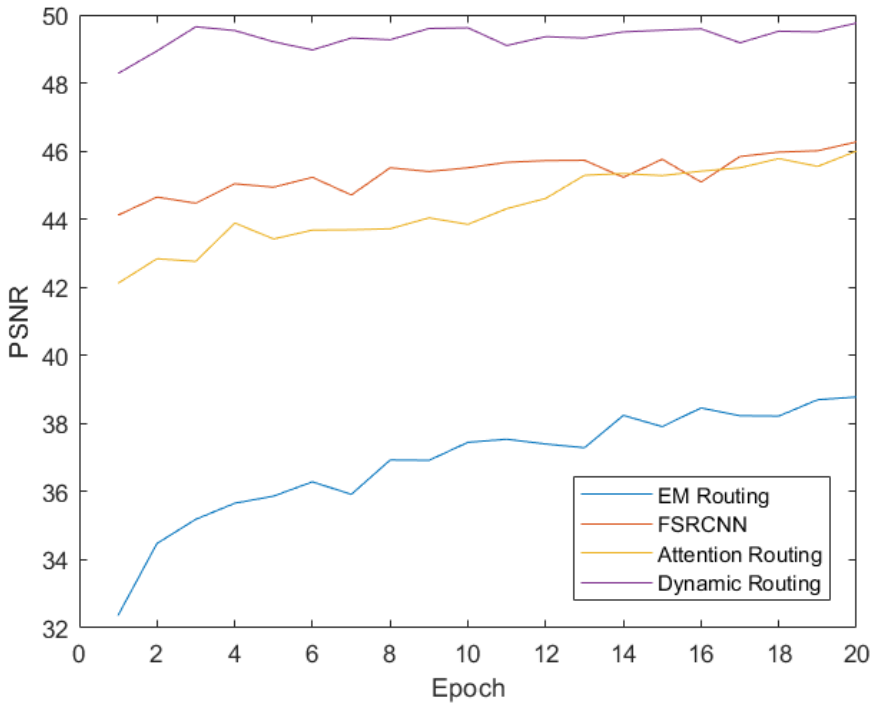


Figure 7.4: PSNR Variation over Epochs

It is worth pointing out that, Dynamic Routing surpasses the other techniques from the very beginning as well. The Attention Routing mechanism started with low PSNR at the beginning and the end of 20 epochs, it has reached the same PSNR ratio obtained by the FSRCNN. Probably, with more iterations, the Attention routing method could transcend the performance of FSRCNN as well.

7.5 Summary

The current method in generating HR images is evaluated against the state of the art method and one of the traditional methods in the domain. This detailed evaluation is explained starting from the strategy to the results through the experimental design. The evaluated results will back-up the overall conclusion that is pointed out in the next chapter.

Chapter 8 Conclusion & Further Work

8.1 Introduction

In this research, a novel integration of state of the art Capsule Net routing mechanism; (Inverted Dot Product based) Attention Routing to one of the existing single image SR pipeline was introduced. Moreover, several architectural modifications to previously introduced Caps Net based approaches were also brought up. Unlike the previous literature where an existing dataset was used, a custom medical image dataset was generated and used for modeling and evaluation. This was done to highlight the relevance of the SR techniques in medical image analysis. The previous chapter, a thorough evaluation was presented on the new SR approach and it was positioned among state-of-the-art techniques. This chapter will conclude the dissertation by emphasizing the achievement of the objectives, drawn conclusions and further work to be conducted in this area of research with the support of the information provided in the evaluation chapter.

8.2 Conclusion

8.2.1 Achievement of Project Objectives

The overall project process was relied on achieving the objectives mentioned in the introduction chapter. By following the proper project process, all of the objectives were achieved as expected.

With a thorough and substantial literature review, several drawbacks in the current SR techniques and attempts to address these issues were identified. By spotting the current research trend in SR methods, one of the recently developed and one time-attempted method was considered to lay down a new solution in the SR domain. The Capsule Network was identified as the learning core of the new solution. The existing SR pipeline established in FSRCNN was altered by introducing the newest Caps Net routing algorithm, namely Inverted Dot Product based Attention Routing. For the completion of the solution, two of the previously used routing

techniques; Dynamic Routing and EM Routing were also used with different configurations.

All of the Caps Net techniques were implemented and integrated into the SR pipeline enabling richer information flow inside the SR pipeline. As indicated in Chapter 7 – Implementation, several configurations were tried for each of the Caps Net methods and the best-trained models were saved for further use. Another important point to be noticed is that a new dataset was generated for the training purpose with one of the freely available medical image datasets.

After the implementation of these methods, each of them was evaluated using a new dataset, which was also generated with another open-source medical database. Several standard image quality indexes were used for the evaluation. Moreover, training time and parameter analysis of each model was also carried out explaining more about this novel approach.

According to the last objective, with the image results, clinician opinion was attained and the responses were positive and pleasing. Finally, at the time of the dissertation submission, one conference paper was drafted fully as 'Super-Resolution Techniques' paper with the study expecting to submit to an upcoming conference.

8.2.2 Overall Conclusion

To overcome the drawbacks like chessboard effect, initial resizing of the image and also to exploit the ability of different Caps Net based SR techniques in the field of medical image domain, several Capsule Network based SR pipelines were introduced. Amongst the 3 Caps-Net architectures namely; Dynamic Routing, EM Routing, and Inverted Dot Product based Attention Routing, the last mentioned technique was introduced. The modified DR based Caps Net outperforms all of the SR techniques that were evaluated in this research. The newly proposed Attention Routing based Caps Net has shown significant results comparable with FSRCNN method as indicated in the Evaluation chapter. This provides evidence that the richness of Caps Net inner structure to map between low resolutions features to high-resolution features. Particularly, achieving the same level of image quality with a much less number of

NN parameters in Attention Routing mechanism indicates the profound information compression and embedding in this SR method.

Showing no significant correlation between the training time and the number of training parameters of the tested models manifests the impact of routing or the complexity of forward passing in a model.

By using a specific medical image dataset for training and evaluation, the networks were specially trained for capturing features related to medical images. According to the feedbacks from the clinicians about the system, the system is capable to deploy in a medical imaging system setting.

With the evaluation results and the step by step concluded outcomes, the project hypothesis that image quality performance of Capsule Net based Super Resolution model for medical image resolution enhancement is further improved by changing the routing mechanism and the layered architecture is proved.

8.3 Limitations and Further Works

Even though the Attention-Routing mechanism inside the SR pipeline significantly reduces the training time compared to DR and EM routing mechanisms, FSRCNN training time is still comparably low even with a higher number of parameters. Further research should be carried out for time optimization of Caps Net based architectures.

The tested models could also be further evaluated not only by changing the core feature learning part, but also the reconstruction methodology of the algorithm. Lastly, to practical deployment of the system in a clinical setting, the models should be integrated with a GUI based application.

8.4 Summary

By bringing the end to the thesis, this chapter describes, to which extent the objectives were achieved, the overall conclusion and the limitations and the further works to be done. Few sideline tasks were identified further to explore the SR field more and to bring the current method into practical ground.

References

[1] S. Fu, M. Zhang, C. Mu, and X. Shen, "Advancements of Medical Image Enhancement in Healthcare Applications," *Journal of Healthcare Engineering*, Mar. 29, 2018. https://www.hindawi.com/journals/jhe/2018/7035264/ (accessed Jun. 02, 2020).

[2] Y. Y. Abdallah, "History of Medical Imaging," *Arch Med Health Sci*, vol. 5, no. 2, p. 275, 2017, doi: 10.4103/amhs.amhs_97_17.

[3] H. Greenspan, "Super-Resolution in Medical Imaging," *The Computer Journal*, vol. 52, no. 1, pp. 43–63, Feb. 2008, doi: 10.1093/comjnl/bxm075.

[4] A. Pinto, "Spectrum of diagnostic errors in radiology," *WJR*, vol. 2, no. 10, p. 377, 2010, doi: 10.4329/wjr.v2.i10.377.

[5] R. L. Morin and M. Mahesh, "The Importance of Spatial Resolution to Medical Imaging," *Journal of the American College of Radiology*, vol. 15, no. 8, p. 1127, Aug. 2018, doi: 10.1016/j.jacr.2018.03.042.

[6] D. Kouame and M. Ploquin, "Super-resolution in medical imaging : An illustrative approach through ultrasound," in *2009 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, Boston, MA, USA, Jun. 2009, pp. 249–252, doi: 10.1109/ISBI.2009.5193030.

[7] J. L. Prince, A. Carass, C. Zhao, B. E. Dewey, S. Roy, and D. L. Pham, "Image synthesis and superresolution in medical imaging," in *Handbook of Medical Image Computing and Computer Assisted Intervention*, Elsevier, 2020, pp. 1–24.

[8] R. Hardie, "A Fast Image Super-Resolution Algorithm Using an Adaptive Wiener Filter," *IEEE Trans. on Image Process.*, vol. 16, no. 12, pp. 2953–2964, Dec. 2007, doi: 10.1109/TIP.2007.909416.

[9] C. Fookes, F. Lin, V. Chandran, and S. Sridharan, "Evaluation of image resolution and super-resolution on face recognition performance," *Journal of Visual Communication and Image Representation*, vol. 23, no. 1, pp. 75–93, Jan. 2012, doi: 10.1016/j.jvcir.2011.06.004.

[10] D. Capel and A. Zisserman, "Super-resolution enhancement of text image sequences," in *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, Barcelona, Spain, 2000, vol. 1, pp. 600–605, doi: 10.1109/ICPR.2000.905409.

[11] H. Lian, "Variational local structure estimation for image super-resolution," Accessed: Jun. 04, 2020. [Online]. Available: https://www.researchgate.net/publication/1764515_Variational_local_structure_estimation_for_image_super-resolution.

[12] R. W. Gerchberg, "Super-resolution through Error Energy Reduction," *Optica Acta: International Journal of Optics*, vol. 21, no. 9, pp. 709–720, Sep. 1974, doi: 10.1080/713818946.

[13] S. P. Kim, N. K. Bose, and H. M. Valenzuela, "Recursive reconstruction of high resolution image from noisy undersampled multiframes," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 38, no. 6, pp. 1013–1027, Jun. 1990, doi: 10.1109/29.56062.

[14] H. Greenspan, G. Oz, N. Kiryati, and S. Peled, "MRI inter-slice reconstruction using super-resolution," *Magnetic Resonance Imaging*, p. 10, 2002.

[15] R. Willett, R. Nowak, I. Jermyn, and J. Zerubia, “Wavelet-Based Superresolution in Astronomy,” p. 10.
[16] P. D. Santis and F. Gori, “On an Iterative Method for Super-resolution,” *Optica Acta: International Journal of Optics*, vol. 22, no. 8, pp. 691–695, Aug. 1975, doi: 10.1080/713819094.
[17] y Tsai and T. Huang, “Multipleframe Image Restoration and Registration,” Greenwich, pp. 317–339.
[18] N. Nguyen and P. Milanfar, “An efficient wavelet-based algorithm for image superresolution,” in *Proceedings 2000 International Conference on Image Processing (Cat. No.00CH37101)*, Sep. 2000, vol. 2, pp. 351–354 vol.2, doi: 10.1109/ICIP.2000.899387.
[19] H. Demirel and G. Anbarjafari, “IMAGE Resolution Enhancement by Using Discrete and Stationary Wavelet Decomposition,” *IEEE Trans. on Image Process.*, vol. 20, no. 5, pp. 1458–1460, May 2011, doi: 10.1109/TIP.2010.2087767.
[20] M. K. Ng, “A Fast MAP Algorithm for High-Resolution Image Reconstruction with Multisensors,” p. 22.
[21] M. Irani and S. Peleg, “Improving resolution by image registration,” *CVGIP: Graphical Models and Image Processing*, vol. 53, no. 3, pp. 231–239, May 1991, doi: 10.1016/1049-9652(91)90045-L.
[22] H. Stark and P. Oskoui, “High-resolution image recovery from image-plane arrays, using convex projections,” *J. Opt. Soc. Am. A*, vol. 6, no. 11, p. 1715, Nov. 1989, doi: 10.1364/JOSAA.6.001715.
[23] S. Baker and T. Kanade, “Super-Resolution Optical Flow.” https://www.researchgate.net/profile/Takeo_Kanade/publication/2449746_Super-Resolution_Optical_Flow/links/57e38d3c08aecd0198de8aea.pdf (accessed Jun. 05, 2020).
[24] M. Irani and S. Peleg, “Super resolution from image sequences,” in *[1990] Proceedings. 10th International Conference on Pattern Recognition*, Atlantic City, NJ, USA, 1990, vol. ii, pp. 115–120, doi: 10.1109/ICPR.1990.119340.
[25] K. Simonyan, S. Grishin, D. Vatolin, and D. Popov, “Fast video super-resolution via classification,” in *2008 15th IEEE International Conference on Image Processing*, San Diego, CA, USA, 2008, pp. 349–352, doi: 10.1109/ICIP.2008.4711763.
[26] M. Elad and A. Feuer, “Restoration of a single superresolution image from several blurred, noisy, and undersampled measured images,” *IEEE Trans. on Image Process.*, vol. 6, no. 12, pp. 1646–1658, Dec. 1997, doi: 10.1109/83.650118.
[27] P. Cheeseman, B. Kanefsky, R. Kraft, J. Stutz, and R. Hanson, “Super-Resolved Surface Reconstruction from Multiple Images,” in *Maximum Entropy and Bayesian Methods*, G. R. Heidbreder, Ed. Dordrecht: Springer Netherlands, 1996, pp. 293–308.
[28] L. Yue, H. Shen, J. Li, Q. Yuan, H. Zhang, and L. Zhang, “Image super-resolution: The techniques, applications, and future,” *Signal Processing*, vol. 128, pp. 389–408, Nov. 2016, doi: 10.1016/j.sigpro.2016.05.002.
[29] E. Mjolsness, “Neural networks, pattern recognition, and fingerprint hallucination,” 1986, doi: 10.7907/M0VQ-DJ43.

[30] S. Baker and T. Kanade, “Hallucinating faces,” in *Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition (Cat. No. PR00580)*, Grenoble, France, 2000, pp. 83–88, doi: 10.1109/AFGR.2000.840616.
[31] W. T. Freeman, “Learning Low Level Vision,” *International Journal of Computer Vision*, vol. 40, no. 1, pp. 25–47, 2000, doi: 10.1023/A:1026501619075.
[32] Hong Chang, Dit-Yan Yeung, and Yimin Xiong, “Super-resolution through neighbor embedding,” in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, Washington, DC, USA, 2004, vol. 1, pp. 275–282, doi: 10.1109/CVPR.2004.1315043.
[33] Jianchao Yang, J. Wright, T. S. Huang, and Yi Ma, “Image Super-Resolution Via Sparse Representation,” *IEEE Trans. on Image Process.*, vol. 19, no. 11, pp. 2861–2873, Nov. 2010, doi: 10.1109/TIP.2010.2050625.
[34] C. Ledig *et al.*, “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, Jul. 2017, pp. 105–114, doi: 10.1109/CVPR.2017.19.
[35] C.-H. Pham *et al.*, “Multiscale brain MRI super-resolution using deep 3D convolutional networks,” *Computerized Medical Imaging and Graphics*, vol. 77, p. 101647, Oct. 2019, doi: 10.1016/j.compmedimag.2019.101647.
[36] C. Dong, C. C. Loy, K. He, and X. Tang, “Image Super-Resolution Using Deep Convolutional Networks,” *arXiv:1501.00092 [cs]*, Jul. 2015, Accessed: Feb. 20, 2020. [Online]. Available: http://arxiv.org/abs/1501.00092.
[37] C. Dong, C. C. Loy, and X. Tang, “Accelerating the Super-Resolution Convolutional Neural Network,” *arXiv:1608.00367 [cs]*, Aug. 2016, Accessed: May 24, 2020. [Online]. Available: http://arxiv.org/abs/1608.00367.
[38] J.-T. Hsu, C.-H. Kuo, and D.-W. Chen, “Image Super-Resolution Using Capsule Neural Networks,” *IEEE Access*, vol. 8, pp. 9751–9759, 2020, doi: 10.1109/ACCESS.2020.2964292.
[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.
[40] S. Sabour, N. Frosst, and G. E. Hinton, “Dynamic Routing Between Capsules,” *arXiv:1710.09829 [cs]*, Nov. 2017, Accessed: Feb. 24, 2020. [Online]. Available: http://arxiv.org/abs/1710.09829.
[41] G. E. Hinton, S. Sabour, and N. Frosst, “Matrix capsules with EM routing,” presented at the International Conference on Learning Representations, Feb. 2018, Accessed: Oct. 16, 2020. [Online]. Available: https://openreview.net/forum?id=HJWLfGWRb.
[42] Y.-H. H. Tsai, N. Srivastava, H. Goh, and R. Salakhutdinov, “CAPSULES WITH INVERTED DOT-PRODUCT ATTENTION ROUTING,” p. 15, 2020.
[43] J. Choi, H. Seo, S. Im, and M. Kang, “Attention Routing Between Capsules,” in *2019 IEEE/CVF International Conference on Computer Vision Workshop*

(ICCVW), Seoul, Korea (South), Oct. 2019, pp. 1981–1989, doi: 10.1109/ICCVW.2019.00247.
[44] "Deconvolution," *Wikipedia*. Sep. 21, 2020, Accessed: Oct. 30, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Deconvolution&oldid=979490499.
[45] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity," *IEEE Trans. on Image Process.*, vol. 13, no. 4, pp. 600–612, Apr. 2004, doi: 10.1109/TIP.2003.819861.
[46] Z. Wang, E. P. Simoncelli, and A. C. Bovik, "Multiscale structural similarity for image quality assessment," in *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, Pacific Grove, CA, USA, 2003, pp. 1398–1402, doi: 10.1109/ACSSC.2003.1292216.
[47] Zhou Wang and A. C. Bovik, "A universal image quality index," *IEEE Signal Process. Lett.*, vol. 9, no. 3, pp. 81–84, Mar. 2002, doi: 10.1109/97.995823.
[48] "yjn870/FSRCNN-pytorch: PyTorch implementation of Accelerating the Super-Resolution Convolutional Neural Network (ECCV 2016)." https://github.com/yjn870/FSRCNN-pytorch (accessed Sep. 06, 2020).
[49] "motokimura/capsnet_pytorch: PyTorch implementation of Geoffrey Hinton's Dynamic Routing Between Capsules." https://github.com/motokimura/capsnet_pytorch (accessed Sep. 06, 2020).
[50] Y. Meng, *YuxianMeng/Matrix-Capsules-pytorch*. 2020.
[51] Y.-H. H. Tsai, *yaohungt/Capsules-Inverted-Attention-Routing*. 2020.
[52] "sewar · PyPI." https://pypi.org/project/sewar/ (accessed Oct. 26, 2020).

Appendix

Appendix I: Inverted Dot Product Based Attention Routing

Concurrent Routing Pseudo Code

procedure INFERENCE($\mathbf{I}; \boldsymbol{\theta}, \mathbf{W}^{1:N-1}$)
    / Pre-Capsules Layers: backbone features extraction */*
    $\mathbf{F} \leftarrow \text{backbone}(\mathbf{I}; \boldsymbol{\theta})$ ▷ backbone feature
    / Capsules Layers: initialization */*
    $\mathbf{P}^1 \leftarrow \text{LayerNorm}\big(\text{convolution}(\mathbf{F}; \boldsymbol{\theta})\big)$ ▷ primary capsules
    for L in layers 2 to N: $\mathbf{P}^L \leftarrow \mathbf{0}s$ ▷ non-primary capsules
    / Capsules Layers (1st Iteration): sequential routing */*
    for L in layers 1 to $(N-1)$ **do**
        $\mathbf{P}^{L+1} \leftarrow \text{Routing}\ (\mathbf{P}^L, \mathbf{P}^{L+1}; \mathbf{W}^L)$ ▷ non-primary capsules
    / Capsules Layers (2nd to tth Iteration): concurrent routing */*
    for $(t-1)$ iterations **do**
        for L in layers 1 to $(N-1)$: $\bar{\mathbf{P}}^{L+1} \leftarrow \text{Routing}\ (\mathbf{P}^L, \mathbf{P}^{L+1}; \mathbf{W}^L)$
        for L in layers 2 to N: $\mathbf{P}^L \leftarrow \bar{\mathbf{P}}^L$ ▷ non-primary capsules
    / Post-Capsules Layers: classification */*
    for all capsule i in layer N: $\hat{y}_i \leftarrow \text{classifier}(\mathbf{p}_i^N; \boldsymbol{\theta})$ ▷ class logits
    return $\hat{\mathbf{y}}$

Appendix II: Data Generator

```
import argparse
import glob
import h5py
import numpy as np
import PIL.Image as pil_image
from utils import calc_patch_size, convert_rgb_to_y


def train(args):
    h5_file = h5py.File(args.output_path, 'w')

    lr_patches = []
    hr_patches = []

    for image_path in sorted(glob.glob('{}/*'.format(args.images_dir))):
        hr = pil_image.open(image_path).convert('RGB')
        hr_images = []

        if args.with_aug:
            for s in [1.0, 0.9, 0.8, 0.7, 0.6]:
                for r in [0, 90, 180, 270]:
                    tmp = hr.resize((int(hr.width * s), int(hr.height * s)), resample=pil_image.BICUBIC)
                    tmp = tmp.rotate(r, expand=True)
                    hr_images.append(tmp)
        else:
            hr_images.append(hr)

        for hr in hr_images:
```

```
            hr_width = (hr.width // args.scale) * args.scale
            hr_height = (hr.height // args.scale) * args.scale
            hr = hr.resize((hr_width, hr_height), resample=pil_image.BICUBIC)
            lr = hr.resize((hr.width // args.scale, hr_height // args.scale), resample=pil_image.BICUBIC)
            hr = np.array(hr).astype(np.float32)
            lr = np.array(lr).astype(np.float32)
            hr = convert_rgb_to_y(hr)
            lr = convert_rgb_to_y(lr)

            for i in range(0, lr.shape[0] - args.patch_size + 1, args.scale):
                for j in range(0, lr.shape[1] - args.patch_size + 1, args.scale):
                    lr_patches.append(lr[i:i + args.patch_size, j:j + args.patch_size])
                    hr_patches.append(hr[i * args.scale:i * args.scale + args.patch_size * args.scale,
                                      j * args.scale:j * args.scale + args.patch_size * args.scale])

    lr_patches = np.array(lr_patches)
    hr_patches = np.array(hr_patches)

    h5_file.create_dataset('lr', data=lr_patches)
    h5_file.create_dataset('hr', data=hr_patches)

    h5_file.close()


def eval(args):
    h5_file = h5py.File(args.output_path, 'w')

    lr_group = h5_file.create_group('lr')
    hr_group = h5_file.create_group('hr')

    for i, image_path in enumerate(sorted(glob.glob('{}/*'.format(args.images_dir)))):
        hr = pil_image.open(image_path).convert('RGB')
        hr_width = (hr.width // args.scale) * args.scale
        hr_height = (hr.height // args.scale) * args.scale
        hr = hr.resize((hr_width, hr_height), resample=pil_image.BICUBIC)
        lr = hr.resize((hr.width // args.scale, hr_height // args.scale), resample=pil_image.BICUBIC)
        hr = np.array(hr).astype(np.float32)
        lr = np.array(lr).astype(np.float32)
        hr = convert_rgb_to_y(hr)
        lr = convert_rgb_to_y(lr)

        lr_group.create_dataset(str(i), data=lr)
        hr_group.create_dataset(str(i), data=hr)

    h5_file.close()


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--images-dir', type=str, required=True)
    parser.add_argument('--output-path', type=str, required=True)
    parser.add_argument('--scale', type=int, default=2)
    parser.add_argument('--patch-size', type=int, default=20)
    parser.add_argument('--with-aug', action='store_true')
    parser.add_argument('--eval', action='store_true')
    args = parser.parse_args()

    if not args.eval:
        train(args)
```

```
    else:
        eval(args)
```

```
    print(out.size())


if __name__ == "__main__":
    main()
```

Appendix III – Dynamic Routing

DR Implementation

```
class RoutingCapsules(nn.Module):
    def __init__(self, in_dim, in_caps, num_caps, dim_caps, num_routing, device: torch.device):
        """
        Initialize the layer.
        Args:
            in_dim:       Dimensionality (i.e. length) of each capsule vector.
            in_caps:      Number of input capsules if digits layer.
            num_caps:     Number of capsules in the capsule layer
            dim_caps:     Dimensionality, i.e. length, of the output capsule vector.
            num_routing:   Number of iterations during routing algorithm
        """
        super(RoutingCapsules, self).__init__()
        self.in_dim = in_dim
        self.in_caps = in_caps
        self.num_caps = num_caps
        self.dim_caps = dim_caps
        self.num_routing = num_routing
        self.device = device

        self.W = nn.Parameter(0.01 * torch.randn(1, num_caps, in_caps, dim_caps, in_dim))

    def __repr__(self):
        tab = '  '
        line = '\n'
        next = ' -> '
        res = self.__class__.__name__ + '('
        res = res + line + tab + '(' + str(0) + '): ' + 'CapsuleLinear('
        res = res + str(self.in_dim) + ', ' + str(self.dim_caps) + ')'
        res = res + line + tab + '(' + str(1) + '): ' + 'Routing('
        res = res + 'num_routing=' + str(self.num_routing) + ')'
        res = res + line + ')'
        return res

    def forward(self, x):
        batch_size = x.size(0)
        # (batch_size, in_caps, in_dim) -> (batch_size, 1, in_caps, in_dim, 1)
        x = x.unsqueeze(1).unsqueeze(4)
        #
        # W @ x =
        # (1, num_caps, in_caps, dim_caps, in_dim) @ (batch_size, 1, in_caps, in_dim, 1) =
```

```
        # (batch_size, num_caps, in_caps, dim_caps, 1)
        u_hat = torch.matmul(self.W, x)
        # (batch_size, num_caps, in_caps, dim_caps)
        u_hat = u_hat.squeeze(-1)
        # detach u_hat during routing iterations to prevent gradients from flowing
        temp_u_hat = u_hat.detach()

        '''
        Procedure 1: Routing algorithm
        '''
        b = torch.zeros(batch_size, self.num_caps, self.in_caps, 1).to(self.device)

        for route_iter in range(self.num_routing - 1):
            # (batch_size, num_caps, in_caps, 1) -> Softmax along num_caps
            c = F.softmax(b, dim=1)

            # element-wise multiplication
            # (batch_size, num_caps, in_caps, 1) * (batch_size, in_caps, num_caps, dim_caps) ->
            # (batch_size, num_caps, in_caps, dim_caps) sum across in_caps ->
            # (batch_size, num_caps, dim_caps)
            s = (c * temp_u_hat).sum(dim=2)
            # apply "squashing" non-linearity along dim_caps
            v = squash(s)
            # dot product agreement between the current output vj and the prediction uj|i
            # (batch_size, num_caps, in_caps, dim_caps) @ (batch_size, num_caps, dim_caps, 1)
            # -> (batch_size, num_caps, in_caps, 1)
            uv = torch.matmul(temp_u_hat, v.unsqueeze(-1))
            b += uv

        # last iteration is done on the original u_hat, without the routing weights update
        c = F.softmax(b, dim=1)
        s = (c * u_hat).sum(dim=2)
        # apply "squashing" non-linearity along dim_caps
        v = squash(s)

        return v
```

DR - SR Model

```
class DynamicRoutingModel(nn.Module):
    def __init__(self, scale_factor=4, num_channels=1, out_channels=56, s=12, m=4, batch_size=16,
image_size=10):
        super(DynamicRoutingModel, self).__init__()
        device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

        self.batch_size = batch_size

        primary_caps_total_out_channel = 12
        per_channel_capsules = 4
        capsule_channels = primary_caps_total_out_channel // per_channel_capsules
        routing_caps_in_caps = capsule_channels * image_size * image_size
        routing_capsule_channel_dimensionality = image_size * image_size
```

```
        self.first_part = nn.Sequential(
            nn.Conv2d(num_channels, out_channels=out_channels, kernel_size=3, stride=1, padding=3 //
2),
            nn.PReLU(out_channels)
        )
        self.primary_caps = PrimaryCapsules(in_channels=56, out_channels=12, dim_caps=4,
                                    kernel_size=1, stride=1, padding=0)

        self.routing_capsule = RoutingCapsules(in_dim=4, in_caps=routing_caps_in_caps,
                                       num_caps=12, num_routing=3,
                                       dim_caps=routing_capsule_channel_dimensionality,
                                       device=device)

        self.last_part = nn.ConvTranspose2d(in_channels=12, out_channels=1, kernel_size=9,
stride=scale_factor,
                                    padding=9 // 2,
                                    output_padding=scale_factor - 1)
```

Appendix IV – EM Routing

EM Routing Implementation

```
class ConvCaps(nn.Module):
    r"""Create a convolutional capsule layer
    that transfer capsule layer L to capsule layer L+1
    by EM routing.
    Args:
        input_channels: input number of types of capsules
        output_channels: output number on types of capsules
        kernel_size: kernel size of convolution
        pose_matrix_size: size of pose matrix is pose_matrix_shape*pose_matrix_shape
        stride: stride of convolution
        iters: number of EM iterations
        coor_add: use scaled coordinate addition or not
        w_shared: share transformation matrix across w*h.
    Shape:
        input:  (*, h,  w, out_channels*(pose_matrix_shape*pose_matrix_shape+1))
        output: (*, h', w', C*(pose_matrix_shape*pose_matrix_shape+1))
        h', w' is computed the same way as convolution layer
        parameter size is:
kernel_size*kernel_size*out_channels*C*pose_matrix_shape*pose_matrix_shape +
out_channels*pose_matrix_shape*pose_matrix_shape
    """

    def __init__(self, input_channels=32, output_channels=32, kernel_size=3, pose_matrix_size=4,
stride=2, iters=3,
                 coor_add=False, w_shared=False):
        super(ConvCaps, self).__init__()
        # TODO: lambda scheduler
        # Note that .contiguous() for 3+ dimensional tensors is very slow
        self.B = input_channels
        self.C = output_channels
        self.K = kernel_size
```

```
        self.P = pose_matrix_size
        self.psize = pose_matrix_size * pose_matrix_size
        self.stride = stride
        self.iters = iters
        self.coor_add = coor_add
        self.w_shared = w_shared
        # constant
        self.eps = 1e-8
        self._lambda = 1e-03
        self.ln_2pi = torch.cuda.FloatTensor(1).fill_(math.log(2 * math.pi))
        # params
        # Note that \beta_u and \beta_a are per capsule type,
        # which are stated at https://openreview.net/forum?id=HJWLfGWRb&noteId=rJUY2VdbM
        self.beta_u = nn.Parameter(torch.zeros(output_channels))
        self.beta_a = nn.Parameter(torch.zeros(output_channels))
        # Note that the total number of trainable parameters between
        # two convolutional capsule layer types is 4*4*k*k
        # and for the whole layer is 4*4*k*k*out_channels*output_channels,
        # which are stated at https://openreview.net/forum?id=HJWLfGWRb&noteId=r17t2UIgf
        self.weights = nn.Parameter(
            torch.randn(1, kernel_size * kernel_size * input_channels, output_channels, pose_matrix_size,
                        pose_matrix_size))
        # op
        self.sigmoid = nn.Sigmoid()
        self.softmax = nn.Softmax(dim=2)

    def m_step(self, a_in, r, v, eps, b, B, C, psize):
        """
            \mu^h_j = \dfrac{\sum_i r_{ij} V^h_{ij}}{\sum_i r_{ij}}
            (\sigma^h_j)^2 = \dfrac{\sum_i r_{ij} (V^h_{ij} - mu^h_j)^2}{\sum_i r_{ij}}
            cost_h = (\beta_u + log \sigma^h_j) * \sum_i r_{ij}
            a_j = logistic(\lambda * (\beta_a - \sum_h cost_h))
            Input:
                a_in:     (b, output_channels, 1)
                r:        (b, out_channels, output_channels, 1)
                v:        (b, out_channels, output_channels, pose_matrix_shape*pose_matrix_shape)
            Local:
                cost_h:   (b, output_channels, pose_matrix_shape*pose_matrix_shape)
                r_sum:    (b, output_channels, 1)
            Output:
                a_out:    (b, output_channels, 1)
                mu:       (b, 1, output_channels, pose_matrix_shape*pose_matrix_shape)
                sigma_sq: (b, 1, output_channels, pose_matrix_shape*pose_matrix_shape)
        """
        r = r * a_in
        r = r / (r.sum(dim=2, keepdim=True) + eps)
        r_sum = r.sum(dim=1, keepdim=True)
        coeff = r / (r_sum + eps)
        coeff = coeff.view(b, B, C, 1)

        mu = torch.sum(coeff * v, dim=1, keepdim=True)
        sigma_sq = torch.sum(coeff * (v - mu) ** 2, dim=1, keepdim=True) + eps
```

```
    r_sum = r_sum.view(b, C, 1)
    sigma_sq = sigma_sq.view(b, C, psize)
    cost_h = (self.beta_u.view(C, 1) + torch.log(sigma_sq.sqrt())) * r_sum

    a_out = self.sigmoid(self._lambda * (self.beta_a - cost_h.sum(dim=2)))
    sigma_sq = sigma_sq.view(b, 1, C, psize)

    return a_out, mu, sigma_sq

def e_step(self, mu, sigma_sq, a_out, v, eps, b, C):
    """
        ln_p_j = sum_h \dfrac{(\V^h_{ij} - \mu^h_j)^2}{2 \sigma^h_j}
                - sum_h ln(\sigma^h_j) - 0.5*\sum_h ln(2*\pi)
        r = softmax(ln(a_j*p_j))
          = softmax(ln(a_j) + ln(p_j))
        Input:
            mu:       (b, 1, output_channels, pose_matrix_shape*pose_matrix_shape)
            sigma:    (b, 1, output_channels, pose_matrix_shape*pose_matrix_shape)
            a_out:    (b, output_channels, 1)
            v:        (b, out_channels, output_channels, pose_matrix_shape*pose_matrix_shape)
        Local:
            ln_p_j_h: (b, out_channels, output_channels, pose_matrix_shape*pose_matrix_shape)
            ln_ap:    (b, out_channels, output_channels, 1)
        Output:
            r:        (b, out_channels, output_channels, 1)
    """
    ln_p_j_h = -1. * (v - mu) ** 2 / (2 * sigma_sq) \
               - torch.log(sigma_sq.sqrt()) \
               - 0.5 * self.ln_2pi

    ln_ap = ln_p_j_h.sum(dim=3) + torch.log(a_out.view(b, 1, C))
    r = self.softmax(ln_ap)
    return r

def caps_em_routing(self, v, a_in, C, eps):
    """
        Input:
            v:        (b, out_channels, output_channels, pose_matrix_shape*pose_matrix_shape)
            a_in:     (b, output_channels, 1)
        Output:
            mu:       (b, 1, output_channels, pose_matrix_shape*pose_matrix_shape)
            a_out:    (b, output_channels, 1)
        Note that some dimensions are merged
        for computation convenient, that is
        `b == batch_size*oh*ow`,
        `out_channels == self.kernel_size*self.kernel_size*self.out_channels`,
        `psize == self.pose_matrix_shape*self.pose_matrix_shape`
    """
    b, B, c, psize = v.shape
    assert c == C
    assert (b, B, 1) == a_in.shape

    r = torch.cuda.FloatTensor(b, B, C).fill_(1. / C)
```

```
        for iter_ in range(self.iters):
            a_out, mu, sigma_sq = self.m_step(a_in, r, v, eps, b, B, C, psize)
            if iter_ < self.iters - 1:
                r = self.e_step(mu, sigma_sq, a_out, v, eps, b, C)

        return mu, a_out
```

SR – EM Routing Model

```
class EMRouting(nn.Module):
    def __init__(self, scale_factor=2, num_channels=1, out_channels=56, s=12, m=4, batch_size=16,
image_size=10):

        super(EMRouting, self).__init__()
        device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
        self.batch_size = batch_size

        self.first_part = nn.Sequential(
            nn.Conv2d(num_channels, out_channels=out_channels, kernel_size=5, stride=1, padding=5 //
2),
            nn.PReLU(out_channels)
        )
        self.primary_caps = PrimaryCaps(in_channels=56, out_channels=3, kernel_size=1,
pose_matrix_shape=4, stride=1)
        # self.conv_caps1 = ConvCaps(input_channels=3, output_channels=56, kernel_size=1,
pose_matrix_size=4, stride=1,
        #                            iters=5)
        self.conv_caps1 = ConvCaps(input_channels=3, output_channels=4, kernel_size=1,
pose_matrix_size=4, stride=1,
                                   iters=3)
        # self.last_part = nn.ConvTranspose2d(in_channels=952, out_channels=1, kernel_size=9,
stride=(2, 2),
        #                                     padding=9 // 2,
        #                                     output_padding=2 - 1)
        self.last_part = nn.ConvTranspose2d(in_channels=68, out_channels=1, kernel_size=9, stride=(2,
2),
                                            padding=9 // 2,
                                            output_padding=2 - 1)
```

Appendix V - Attention Routing

Implementation – Network Parameters

```
{
  "backbone": {
    "kernel_size": 3,
    "output_dim": 56,
    "input_dim": 1,
    "stride": 1,
    "padding": 1,
    "out_img_size": 16
    },
  "primary_capsules": {
```

```
    "kernel_size": 1,
    "stride": 1,
    "input_dim": 56,
    "caps_dim": 4,
    "num_caps": 3,
    "padding": 0,
    "out_img_size": 10
    },
  "capsules": [
    {
      "type" : "CONV",
      "num_caps": 4,
      "caps_dim": 4,
      "kernel_size": 1,
      "stride": 1,
      "matrix_pose": true,
      "out_img_size": 10
    }
  ]
}
```

Attention Routing – Model

Note: Here, the variable ‘params’ is correspond to the above parameter file

```
## Primary Capsule Layer
self.pc_num_caps = params['primary_capsules']['num_caps']
self.pc_caps_dim = params['primary_capsules']['caps_dim']
self.pc_output_dim = params['primary_capsules']['out_img_size']
## General
self.num_routing = num_routing  # >3 may cause slow converging

#### Building Networks
## Backbone (before capsule)
if backbone == 'simple':
  self.pre_caps = layers.simple_backbone(params['backbone']['input_dim'],
                                         params['backbone']['output_dim'],
                                         params['backbone']['kernel_size'],
                                         params['backbone']['stride'],
                                         params['backbone']['padding'])
elif backbone == 'resnet':
  self.pre_caps = layers.resnet_backbone(params['backbone']['input_dim'],
                                         params['backbone']['output_dim'],
                                         params['backbone']['stride'])

## Primary Capsule Layer (a single CNN)
self.pc_layer = nn.Conv2d(in_channels=params['primary_capsules']['input_dim'],
                          out_channels=params['primary_capsules']['num_caps'] * \
                                       params['primary_capsules']['caps_dim'],
                          kernel_size=params['primary_capsules']['kernel_size'],
                          stride=params['primary_capsules']['stride'],
                          padding=params['primary_capsules']['padding'],
                          bias=False)

# self.pc_layer = nn.Sequential()

self.nonlinear_act = nn.LayerNorm(params['primary_capsules']['caps_dim'])

## Main Capsule Layers
```

```
self.capsule_layers = nn.ModuleList([])
for i in range(len(params['capsules'])):
    if params['capsules'][i]['type'] == 'CONV':
        in_n_caps = params['primary_capsules']['num_caps'] if i == 0 else \
            params['capsules'][i - 1]['num_caps']
        in_d_caps = params['primary_capsules']['caps_dim'] if i == 0 else \
            params['capsules'][i - 1]['caps_dim']
        self.capsule_layers.append(
            layers.CapsuleCONV(in_n_capsules=in_n_caps,
                               in_d_capsules=in_d_caps,
                               out_n_capsules=params['capsules'][i]['num_caps'],
                               out_d_capsules=params['capsules'][i]['caps_dim'],
                               kernel_size=params['capsules'][i]['kernel_size'],
                               stride=params['capsules'][i]['stride'],
                               matrix_pose=params['capsules'][i]['matrix_pose'],
                               dp=dp,
                               coordinate_add=False
                               )
        )
    elif params['capsules'][i]['type'] == 'FC':
        if i == 0:
            in_n_caps = params['primary_capsules']['num_caps'] * params['primary_capsules']['out_img_size'] * \
                        params['primary_capsules']['out_img_size']
            in_d_caps = params['primary_capsules']['caps_dim']
        elif params['capsules'][i - 1]['type'] == 'FC':
            in_n_caps = params['capsules'][i - 1]['num_caps']
            in_d_caps = params['capsules'][i - 1]['caps_dim']
        elif params['capsules'][i - 1]['type'] == 'CONV':
            in_n_caps = params['capsules'][i - 1]['num_caps'] * params['capsules'][i - 1]['out_img_size'] * \
                        params['capsules'][i - 1]['out_img_size']
            in_d_caps = params['capsules'][i - 1]['caps_dim']
        self.capsule_layers.append(
            layers.CapsuleFC(in_n_capsules=in_n_caps,
                             in_d_capsules=in_d_caps,
                             out_n_capsules=params['capsules'][i]['num_caps'],
                             out_d_capsules=params['capsules'][i]['caps_dim'],
                             matrix_pose=params['capsules'][i]['matrix_pose'],
                             dp=dp
                             )
        )
```

Attention Routing Implementation – Forward Pass

```
def forward(self, x, lbl_1=None, lbl_2=None):
    #### Forward Pass
    ## Backbone (before capsule)
    c = self.pre_caps(x)
    # print(c.size())

    ## Primary Capsule Layer (a single CNN)
    u = self.pc_layer(c)
    u = u.permute(0, 2, 3, 1)
    u = u.view(u.shape[0], self.pc_output_dim, self.pc_output_dim, self.pc_num_caps,
               self.pc_caps_dim)
    u = u.permute(0, 3, 1, 2, 4)
    init_capsule_value = self.nonlinear_act(u)  # capsule_utils.squash(u)
```

```
    ## Main Capsule Layers
    # concurrent routing
    if not self.sequential_routing:
        # first iteration
        # perform initilialization for the capsule values as single forward passing
        capsule_values, _val = [init_capsule_value], init_capsule_value
        for i in range(len(self.capsule_layers)):
            _val = self.capsule_layers[i].forward(_val, 0)
            capsule_values.append(_val)  # get the capsule value for next layer

        # second to t iterations
        # perform the routing between capsule layers
        for n in range(self.num_routing - 1):
            _capsule_values = [init_capsule_value]
            for i in range(len(self.capsule_layers)):
                _val = self.capsule_layers[i].forward(capsule_values[i], n,
                                                      capsule_values[i + 1])
                _capsule_values.append(_val)
            capsule_values = _capsule_values
    # sequential routing
    else:
        capsule_values, _val = [init_capsule_value], init_capsule_value
        for i in range(len(self.capsule_layers)):
            # first iteration
            __val = self.capsule_layers[i].forward(_val, 0)
            # second to t iterations
            # perform the routing between capsule layers
            for n in range(self.num_routing - 1):
                __val = self.capsule_layers[i].forward(_val, n, __val)
            _val = __val
            capsule_values.append(_val)

out = capsule_values[-1]
out = out.reshape(out.shape[0], out.shape[1] * out.shape[4], out.shape[2], out.shape[3])

out = self.last_part(out)
```

Appendix VI: PSNR Implementation

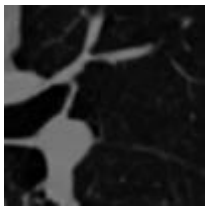
```
def calc_psnr(img1, img2):
    return 10. * torch.log10(1. / torch.mean((img1 - img2) ** 2))
```

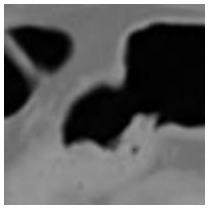
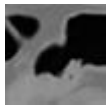
Evaluation at training phase

```
for data in eval_dataloader:
    inputs, labels = data
    # print(labels)

    inputs = inputs.to(device)
    labels = labels.to(device)

    model.set_batch_size(batch_size=16)
```

```
    with torch.no_grad():
        preds = model(inputs).clamp(0.0, 1.0)

    epoch_psnr.update(calc_psnr(preds, labels), len(inputs))
```

Appendix VII – Sample 100x100 (HR) and 50x50 (LR) Image Pairs For Evaluation



Appendix VIII - Image Zooming

```
class ImageZoomerApplication:

    def __init__(self):
        print('Initializing')

    def image_zoomer(self, input_image, scaling_factor, sub_image_size, model, weights_file,
save_path):
        # calculating o/p size
        w, h = input_image.shape
        w_new, h_new = w * scaling_factor, h * scaling_factor
        new_image_array = np.empty([w_new, h_new, 3])

        # splitting image, scaling and merging
        for i in range(0, w, sub_image_size):
            for j in range(0, h, sub_image_size):
                sub_image = input_image[i:i + sub_image_size, j: j + sub_image_size]

                state_dict = model.state_dict()
                for n, p in torch.load(weights_file, map_location=lambda storage, loc: storage).items():
                    if n in state_dict.keys():
                        state_dict[n].copy_(p)
                    else:
                        raise KeyError(n)

                model.eval()
```

```
            sub_image = pil_image.fromarray(sub_image).convert('RGB')

            bicubic = sub_image.resize((sub_image.width * scaling_factor, sub_image.height *
scaling_factor),
                                       resample=pil_image.BICUBIC)
            _, ycbcr = preprocess(bicubic, device)

            lr, _ = preprocess(sub_image, device)

            with torch.no_grad():
                preds = model(lr).clamp(0.0, 1.0)

            # print(preds.shape)
            preds = preds.mul(255.0).cpu().numpy().squeeze(0).squeeze(0)
            output = np.array([preds, ycbcr[..., 1], ycbcr[..., 2]]).transpose([1, 2, 0])
            output = np.clip(convert_ycbcr_to_rgb(output), 0.0, 255.0).astype(np.uint8)

            new_image_array[i * scaling_factor: (i + sub_image_size) * scaling_factor,
            j * scaling_factor: (j + sub_image_size) * scaling_factor] = output

    input_image = pil_image.fromarray(input_image)
    image_size = w * scaling_factor
    bicubic_image = input_image.resize((image_size, image_size))

    new_image_array = new_image_array.astype(int)
    upscaled_image = pil_image.fromarray(new_image_array[:, :, 2] * 255)
    upscaled_image.save(save_path)
    return upscaled_image, bicubic_image
```

Appendix IX – Image Evaluator

```
import numpy as np
from skimage.measure import compare_ssim
from sewar.full_ref import uqi, msssim, psnr


def calculate_evaluation(original_image, upsampled_image):
    psnr = calculate_psnr(original_image, upsampled_image)
    ssim = calculate_ssim(original_image, upsampled_image)
    uqi_val = calculate_uqi(original_image, upsampled_image)
    msssim_val = np.absolute(calculate_msssim(original_image, upsampled_image))

    return psnr, ssim, uqi_val, msssim_val


def calculate_psnr(original_image, upsampled_image):
    return psnr(original_image, upsampled_image, 255)


def calculate_ssim(original_image, upsampled_image):
    (score, diff) = compare_ssim(original_image, upsampled_image, full=True)
    return score
```

```
def calculate_uqi(original_image, upsampled_image):
    return uqi(original_image, upsampled_image)


def calculate_msssim(original_image, upsampled_image):
    return msssim(original_image, upsampled_image)
```

Appendix X – Attached (SR _Result_Verification.pdf)