

On Collaborative Intelligence

Ramasubramanian Balasubramanian

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2019-37

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-37.html>

May 14, 2019



Copyright © 2019, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

Please refer to the acknowledgment page in the thesis.

On Collaborative Intelligence

by

Ramasubramanian Balasubramanian

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Gerald Friedland, Chair
Professor Kannan Ramchandran, Second Reader

Spring 2019

On Collaborative Intelligence

by

Ramasubramanian Balasubramanian

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Sciences

in the

Graduate Division


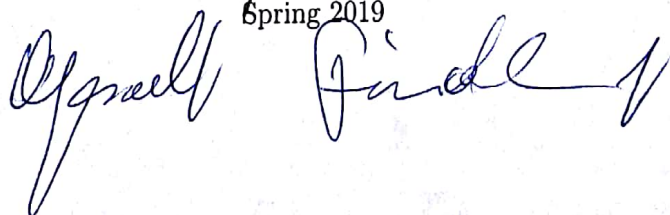
of the

University of California, Berkeley

Committee in charge:

Professor Gerald Friedland, Chair

Professor Kannan Ramchandran, Second Reader


Spring 2019


On Collaborative Intelligence

Copyright 2019

by

Ramasubramanian Balasubramanian

Abstract

On Collaborative Intelligence

by

Ramasubramanian Balasubramanian

Master of Science in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Gerald Friedland, Chair

We first explore the behavior of self-assembling wires, ball bearings that form emergent structures, under an electric field. We model this behavior using the laws of physics as well as some heuristics, with the aim to implement this self-organizing behavior on to a group of drones covering a given region optimally, communicating with each other in a peer-to-peer manner, and collating multimedia information demonstrating collaborative intelligence. After looking at how these ball bearings turn ‘intelligent’ even with simple adaptation rules, we explore frameworks with more theoretically grounded rules: can neural networks collaborate to improve performance? We realize that we first need a way to understand what goes on behind-the-scenes when a neural network makes a particular decision to discuss its behavior, and hence we turn to heatmapping. We use it to figure out how much does each pixel contribute to the final output in an image classification task, or in other words what parts of an image led a network to classify an image into a particular class. We demonstrate that the network looks for characteristic features belonging to each class while trying to classify and how this could lead to errors. We then ask the important question of whether we can use heatmaps obtained from one neural network to filter out background noise from the images by blackening out the pixels that don’t contribute much to the final output, and if so, can these filtered images, when passed through another neural network, lead to a higher classification accuracy. We see that the data processing inequality holds and the accuracy does not increase, but given that the images are simpler now, they need fewer bits to be encoded and hence networks with simpler architectures, having a smaller capacity, are enough for the task without much drop in the accuracy.

Contents

Contents	ii
List of Figures	iii
1 Introduction and Motivation	1
2 Related Work	4
2.1 Heatmapping for classification	5
2.2 Self-organizing agents	6
3 Self-Organising Agents	8
3.1 The problem statement	8
3.2 Models and approaches	9
4 Heatmapping	27
4.1 Theory	27
4.2 Preliminary Experiments - MNIST dataset	29
4.3 Centrifuge classification	33
4.4 Other attempts	38
5 The data processing inequality	41
5.1 Experimental procedure	41
5.2 Results	44
5.3 Discussion	45
6 Conclusion and Future Work	48
Bibliography	51

List of Figures

3.1	Initial set-up	11
3.2	Ball bearings under the effect of the Coulomb force alone	12
3.3	Ball bearings with linked positive and negative subagents under Coulomb force	13
3.4	Ball bearings as shooter agents with leader and follower attributes	15
3.5	Shooter agents but with only certain positions allowed	17
3.6	Ball bearings in the neighbor model move based off of local neighbor statistics	19
3.7	Potential model: Initial position of the ball bearings in a square grid	21
3.8	Iterations from the potential model with $N = 20$ and $m = 30$: the white squares show the location of the balls.	23
3.9	Iterations from the potential model with $N = 40$ and $m = 100$: the white squares show the location of the balls.	25
3.10	A few iterations from phase 2 of the shooter agent and the discretized grid approach.	26
4.1	Layerwise relevance propagation procedure	27
4.2	MLP - Deep Taylor	31
4.3	CNN - Deep Taylor	31
4.4	CNN - Alpha Beta	32
4.5	The original VGG16 architecture. The block marked in red is the one we played around with, in terms of the number of layers, number of neurons in each layer, activations, dropouts etc. The layers in this block were retrained on our data to fit our task. The weights in all the convolutional layers prior to this block were retained and those layers were used as feature extractors.	35
4.6	An example heatmap belonging to the centrifuge class	37
4.7	An example containing characteristic features of multiple classes, thus making it harder for the network to classify	38
4.8	Averaging to obtain a template heatmap	39
4.9	Architecture of the RMAC feature extractor	40
5.1	Training phase: an example of the not-centrifuge (cats/dogs) class is shown here.	43

5.2 Testing phase: an example of the centrifuge class is shown here. In the top pipeline, we assume that the class of the image is 0 (centrifuge) and we get a heatmap with that assumption. The histogram has been omitted here for brevity. The filtered image obtained after blackening the pixels with low relevance is then fed to the second network, which outputs the probability of the image belonging to class 0 and to class 1. p_{ij} here corresponds to the probability of the label being j for assumed label i . In the bottom pipeline, we assume that the class of the image is 1 (not-centrifuge). This has to be done since the label is needed for generating heatmaps and the true label is not known for a test image. We see that the heatmap shows a lot of negative relevance (blue color). The filtered image obtained here doesn't blacken out most of the background like is the case when the label is correct (top pipeline) because here we are telling it to find relevance that it is not a centrifuge. The final decision is made this way: if $p_{00} > p_{11}$, we declare that it is class 0; if $p_{00} < p_{11}$, then class 1, if $p_{00} = p_{11}$ and $p_{01} < p_{10}$, class 0; else class 1.

Acknowledgments

I would first and foremost like to thank my adviser Prof. Gerald Friedland, with whom I have had many interesting and stimulating discussions, both in one-on-one meetings and in discussion groups. He always had new and interesting approaches to try out and assisted me whenever I got stuck somewhere. I would also like to thank my co-adviser Prof. Kannan Ramchandran, whose valuable insights have helped me in my projects. Dr. Alfredo Metere was very helpful with the physics parts of the collaborative intelligence project as well as parts involving ordinary differential equations, fractals, cellular automata etc. His work on sizing neural networks, along with Dr. Mario Krell and Prof. Friedland, as well as his inputs at various times during my masters were very helpful. Dr. Jaeyoung Choi had some good ideas to try out w.r.t. the heatmapping project. Brainstorming sessions with him on a couple of multimedia related projects as well as the heatmapping project were very useful.

Some of the initial approaches in the collaborative intelligence project were a collaborative effort between two undergraduate students, Rishi Veerapaneni and Michael Weymouth, and myself. Their valuable contributions helped kickstart the project and led to newer ideas for later parts of the project. I would also like to thank Erickson Nascimento, who provided us with data from his project on images captured by drones in mines in Brazil. He also assisted us with the 3d modelling methodologies for aerial images. I would also like to thank Prof. Zettl, who explained a few of the concepts at play in the self-assembling wires experiment, and the people in Prof. Hubler's lab as well as the Stanford Complexity group, who responded to our emails and updated us on what they had tried so that we don't reinvent the wheel. I would also like to thank the Lawrence Livermore National Lab and the International Computer Science Institute for sponsoring the various projects that I was a part of.

Chapter 1

Introduction and Motivation

Collaborative intelligence characterizes a large number of agents having enough local autonomy in a system to intelligently communicate with each other and self-organize, distribute the load, and optimize the group's combined output for a task. We started by asking the question: can we find an optimal path for a group of drones acting in a self-organizing setting, flying around in say flood affected areas, for a search-and-rescue operation, with the added constraint that each drone can only communicate with drones within a certain radius and not directly with the central station? Can we collate multimedia information, say the images from one drone and the audio from another to increase the confidence of detection of people in danger so that rescue teams could be sent to that location? How can we cover the maximum area with as few drones as possible while also ensuring that all important information reaches the central station via a chain of drones within a reasonable time?

We needed an approach that works such that we wouldn't have to come up with a path for each drone and such that it works in several different scenarios, in terms of the shape of the area to be covered. Hence, we wanted the drones to self-organize and find optimal paths. We turned to nature for this and realized that there are many processes in nature that are self-organizing. One very interesting experiment was the self-assembling wires experiment [10], in which a group of ball bearings form emergent structures under an electric field. This experiment mirrored what we wanted to achieve, since the ball bearings first form chains and pull other ball bearings with them till one of the chains touches the rim of the petridish that they are in. This is similar to how we imagined a group of drones would all start off from a central base station and keep moving outwards, while maintaining a distance such that it can communicate with another drone to send its information back to the central station, till one of the drones reaches the boundary of the region we want to cover. Then, we see in the ball bearings experiment mentioned that the other bearings just snake around trying to occupy as much area as they can while still maintaining contact with their chains. This is again in line with what we thought should happen in our case, where the drones spread around to cover as much area as possible, while still maintaining contact with nearby drones.

The other place we were taking inspiration from was fractal theory, as we could see those structures too, in the ball-bearing experiment. Some of the chains going towards the rim would branch and new chains would start with similar properties as the original bigger chains and now those two would move towards the rim. We also decided to approach the problem from different angles: in one approach, we tried to model the complex physical interactions between the ball bearings and their surrounding environment, in another we came up with heuristics for how the balls should move and treated the ball bearings as solutions to an ordinary differential equation, which led us to look at the theory of cellular automata. The applications of this could be far reaching: we already discussed the possibility of search and rescue operations, but any similar situation wherein one needs to look for something in a confined space, communicate that information to a central station efficiently, and cover the entire space in a reasonable time could use this idea, such as in battlefields to detect enemy soldiers etc.

After looking at how simple adaptation rules are sometimes enough to make agents act in a self-organizing manner and collaborate ‘intelligently’, we turned our attention to collaboration between neural networks and more theoretically grounded frameworks. But to understand the behavior of neural networks, we needed to understand the inner workings first. It is useful to understand how networks reach a particular decision in order to understand how to make them self-optimizing. A simple example of that would be to drop connections between neurons that are not contributing much to final output for a particular task.

Interpreting a nonlinear classifier, especially the ones that are seen as ‘black-boxes’, is gaining traction in the AI community as it is important to gain trust into the predictions made and to identify potential data selection biases or artefacts. Heatmapping [6] is a technique to decompose the prediction of a neural network in terms of contributions of individual input variables such that the produced decomposition can be visualized in the same way as the input data. Previous works using heatmapping have used the corresponding explanations obtained to compare classifiers [21], measure context use in text documents [13], analyze facial expressions [4], increase interpretability in the sciences etc.

We worked on a project to find out whether images, obtained through satellite imagery or otherwise, had centrifuges in them. Such projects highlight the need for interpretability because centrifuges are used in nuclear weapon programs, so if a network labels an image as having a centrifuge, we need to understand why it thought so, what were the characteristic features in the image that led to this conclusion, is the network able to see some patterns or features that we humans might miss etc.

Given that a heatmap gives the contribution of each pixel in predicting the output, a natural question that arose in our minds was about whether we could use heatmaps to improve the accuracy of classification. Our initial attempts involved passing the heatmap obtained from the neural network through another neural network but that led to the loss of

a large amount of information, as the heatmap is just an RGB representation of the relevance scores; hence it didn't perform well. Finally, we decided to blacken out those pixels that didn't contribute much to the prediction, thus 'filtering' the image, and use these filtered images for classification. Here, we see that the heatmap produced by one neural network is used to educate the second neural network in some sense, and the neural networks collaborate and work towards improving the performance, as measured by accuracy. We then discuss why this idea did not work, and what other insights we drew from this negative result. The major roadblock was the Data processing inequality, an information theoretic concept which states that the information content of a signal cannot be increased via a local physical operation. In other words, post-processing cannot increase information. [16]

Chapter 2

Related Work

Explainability in AI has been around for some time now, but the use of layerwise relevance propagation provided a much-needed thrust to this field. A joint project at Fraunhofer HHI, TU Berlin and SUTD Singapore on developing new method to understand nonlinear predictions of state-of-the-art machine learning models led to some very interesting and impactful results and publications, paving way for other researchers to build on it. The work by Montovan et. al. on using deep Taylor decomposition to explain nonlinear classification decisions [7] in terms of input variables proved to be a foundation stone throughout our experiments. The theoretical connections between the Taylor decomposition of a function, and rule-based relevance propagation techniques, provide good starting point for further research. They showed that the model was stable under different architectures and datasets, and it does away with the need for hyperparameter tuning. The work on Layerwise relevance propagation put the Deep Taylor method to use. Layer-wise relevance propagation is a method to compute scores for image pixels and image regions denoting the impact of the particular image region on the prediction of the classifier for one particular test image [1]. They explore the impact of different parameter settings on the resulting explanation.

Another work that came out around the same time is the one by Samek et. al.[23] in which they present a general methodology based on region perturbation for evaluating ordered collections of pixels such as heatmaps as an objective quality measure for the usefulness of heatmaps. They support the aforementioned work and arrive at the conclusion that the recently proposed layer-wise relevance propagation algorithm qualitatively and quantitatively provides a better explanation of what made a DNN arrive at a particular classification decision than a sensitivity-based approach or a deconvolution method. They also explore the use of heatmaps for unsupervised assessment of the neural network performance.

A few slightly older works in this domain were important theoretical milestones as well. A paper from 2015 [19] talks about a general solution to the problem of understanding classification decisions by pixel-wise decomposition of nonlinear classifiers. They talk about a methodology that allows to visualize the contributions of single pixels to predictions for

kernel-based classifiers over Bag-of-Words features and for multilayered neural networks. Simonyan et.al. [11] try to address the visualisation of image classification models, learnt using deep Convolutional Networks. They compute a class saliency map, specific to a given image and class, and show that such maps can be employed for weakly supervised object segmentation using classification ConvNets. They also establish the connection between the gradient-based ConvNet visualisation methods and deconvolutional networks.

2.1 Heatmapping for classification

Heatmapping has mostly been used to explain the predictions made by a network, as opposed to making or improving the predictions. Analyzing and comparing the properties and strategies of different models beyond classification accuracy often opens up new ways for model improvements. Some of these works on LRP and understanding how a model makes a decision formed the motivation for us to try and improve the classification accuracy based on these understandings. Becker et. al. [20] trained two neural network architectures on a spectrogram and raw waveform data for audio classification tasks on the AudioMNIST dataset and applied layer-wise relevance propagation (LRP) to investigate the models' feature selection and decision making. They demonstrate that the networks are highly reliant on features marked as relevant by LRP through systematic manipulation of the input data. Based on the relevance scores, they hypothesized that gender classification is based on the fundamental frequency and its immediate harmonics. Apart from being a discriminant feature for gender, it also opens up avenues for rule-based classification methods where a basic prediction could be made based on the fundamental frequency. Ideas like these help improve the classification accuracy. This work also shows that such attempts have their fair share of difficulties: the derivation of higher-order concepts such as phonemes or certain frequency ranges proved to be more difficult than for gender classification.

Works on pixel flipping drove home the idea that LRP identifies pixels, in image-related tasks, in accordance to their relevance in predicting the output, thus motivating our idea that the low-relevance pixels are nothing but noise. Bach et. al. [19] propose a novel way to judge objectively the quality measure of the usefulness of heatmaps. They present a general methodology based on region perturbation for evaluating ordered collections of pixels such as heatmaps. Samek et. al. [23] propose a general solution to the problem of understanding classification decisions by pixel-wise decomposition of nonlinear classifiers. They introduce a methodology that allows to visualize the contributions of single pixels to predictions for kernel-based classifiers over Bag-of-Words features and for multilayered neural networks. These pixel contributions can be visualized as heatmaps and are provided to a human expert who can intuitively not only verify the validity of the classification decision, but also focus further analysis on regions of potential interest. Although they concede that there are still some issues that need to be addressed, such as the greediness of the layer-wise relevance propagation procedure and the analytical justification of the distribution of the relevance

messages being proportional to the weighted neuron activations, they propose a pixel-flipping method that allows to discriminate between two heatmapping methods that may otherwise look of similar quality to the human. They show that if the heatmap is random, the decrease in the classification function is much slower than when the heatmap is informative and pixels in the informative region are destroyed. They compare the selectivity of different heatmap approaches and show that removing input features that are deemed relevant reduces the evidence at the output. Though none of these works directly address the task of trying to improve classification accuracy based on the insights gained about how a network makes predictions, that seemed like the logical next step for us to pursue.

2.2 Self-organizing agents

One of the main driving experiments behind our simulations related to self-organizing agents was the self-assembling wires experiment by the Stanford Complexity group. The experiment is based on [10]. In this paper, the authors work with stationary ramified transportation networks in a macroscopic nonbiological system. They describe the dynamical formation of the network which consists of three growth stages, called by them as strand formation, boundary formation, and geometric expansion. The strand formation phase is the one with shooters as described in the video and as has been described later in our simulations. We combine the strand formation and boundary formation phases and called it phase 1, and called the phase where the ball bearings spread to occupy maximum area as phase 2. The authors also found that the system forms statistically robust network features, which wasn't very useful to our simulation. What was useful though was their finding that the networks are usually trees, meaning that they lack closed loops since those are unstable. They also establish that the final topology of the network is sensitive to the initial conditions of the particles, in particular to its geometry, which again was useful to us, as the drones in our bigger picture and the ball bearings in the simulation need to have a starting formation, so to speak.

Given that our main objective was to implement the behavior observed in the ball bearings onto drones, we also needed data from drones as well as 3d model estimation methodologies for aerial and ground images. The work by Potje et.al. [8] proposes an efficient approach for automatic generation of 3D models from images based on structure from motion (SfM) and multi-view stereo reconstruction techniques. They use meta-data information such as GPS, keypoint filtering and multiple local bundle adjustment refinement instead of global optimization in a novel scheme to speed up the incremental SfM process. Given that we didn't want to reinvent the wheel and that our main focus wasn't to improve upon image processing algorithms, we decided to use this work to stitch the images that the drone captures. Though the accuracy of the GPS was an issue, this was a good starting point as our idea involved collating multimedia information, such as image, audio etc. from different drones to make decisions in search-and-rescue operations.

Closely mirroring the experiment was the work by Marani et.al. [15] where they study the stationary state of a Poisson problem for a system of N perfectly conducting metal balls driven by electric forces to move within a medium of very low electrical conductivity onto which charges are sprayed from outside. When grounded at a confining boundary, the system of metal balls is experimentally known to self-organize into stable fractal aggregates. They simulate the dynamical conditions leading to the formation of such aggregated patterns and analyze the fractal properties. They talk about minimum total energy dissipation (and potential energy of the system as a whole), and suggest rules for the emergence of scale-free structures in nature. This work was pivotal in our potential model as well as for the updates that the balls receive in each iteration based on rules. Other works on self-assembling objects in nature and fractals were also helpful, such as [3], which develops a new model of the self assembly process starting from the underlying physical interaction between conductors.

Works such as [14] have also explored self-constructing and self-repairing electrical connections built by agglomeration of metallic particles between two electrodes. They show that self-assembling electrical connections grow by building a chain of particles between two electrodes immersed in a dielectric liquid. They establish that the growth time for the self-assembling process is a linear function of the initial average spacing of metallic particles and a linear function of the distance between the electrodes. They also demonstrate that the agglomeration process occurs in such a way as to minimize the overall resistance of the system, which corroborates some of the other works mentioned before.

Chapter 3

Self-Organising Agents

3.1 The problem statement

The big picture we had in mind was this: imagine a group of drones and a central station. In the simplest setting of the problem, we can have as many drones as we want, and all of them can contact the central base station. The objective is for the drones to go around the central station and cover a particular area and report back certain things. One example would be employing these drones in flood affected areas to look for people who need help, either because they are stranded or are drowning etc. The other use-case could be in battlefields, for the drones to go around and detect enemy soldiers, tanks etc.

In a slightly advanced version of the problem, the drones can only communicate within a certain radius, so for a drone to send some information back to the central base station, it would possibly have to send it through a chain of other drones, in a peer-to-peer communication sort of manner. The issues that could arise here are things like how to prevent cross-talk, what if multiple drones want to send information at the same time through the same set of drones, what is the best path for the drones to fly to cover maximum area while remaining in contact with at least one other drone so that the information could be transmitted back, what is the best path for the drones to transmit the information if there are multiple paths etc.

We also wanted drones to be a bit smart and collate multimedia information, something along the lines of collaborative intelligence, consider this: let us say one drone is able to see from a high up position that a person is sinking but is not able to hear that person because of the distance; another drone is able to hear someone crying for help but is not able to see the person because of a wall; can we somehow combine the information from these two drones and increase the confidence of detection so that rescue forces are sent only to those places where there's a high chance that someone actually needs help. Reducing false positives is important here since the resources (time, personnel etc.) are limited, but missing positives (a

positive here being a person in danger, for instance) is also expensive as that would cost lives.

3.2 Models and approaches

The main motivator behind our logic for the path of the drones was the self-assembling wires experiment [10] and the corresponding video by the Stanford Complexity Group. In the video, they take metal ball bearings in a petridish to study self-assembling wires. These ball bearings form emergent structures. The ball bearings are placed in a petridish containing castor oil. Castor oil is used to keep the ball bearings from moving around too fast. They apply a negative charge to a metal ring placed at the edge of the petridish. They then hang a wire connected to a power supply above the petridish, and spray electrons onto the petridish by applying a large voltage ($\sim 20kV$).

In the first set-up, they start with all the ball bearings at the center of the petridish. The negative electrode, as was mentioned, is attached to the metal ring at the edge and the wire that's hanging above the petridish serves as the positive electrode. When the voltage is applied, nothing happens in the beginning. But after some time, they observed that some balls start moving and they also interestingly observe that the balls moving out pull other balls near them with them, thus forming chains (which they call 'shooters'). Hence, several shooters emerge from the mass of balls at the center, all of them pulling other balls thus elongating the chain, and moving towards the outer ring. The chain ensures that the balls are in contact with this central mass of balls. This process of moving towards the metal ring continues till one such chain touches the rim. Then, that becomes a stem and all the other shooters die, and a new structure starts to emerge. All the other shooters stop going towards the rim and start spreading out in a snake-like fashion, to cover as much area as possible, all while recruiting more and more balls into their chain. Finally, all the balls that were initially at the center are part of some chain and the chains have all spread nicely all over the petridish, except the one shooter that touched the rim first.

In another set-up, they start by having all the ball bearings spread near the edge of the petridish. In this case, a lot of stems/chains form, and all these chains go on pulling as many balls from nearby positions as they can into their chain. In some sense, the different chains compete to get balls, and in this case, the motion is not in a straight line, rather they twist and turn in an effort to recruit more balls. This experiment demonstrates some of the complex dynamics arising from local interactions and the different structures that arise as a result of different starting points.

The second set-up wasn't very relevant to our case, but the first set-up is a good starting point for our problems. Let us think about that scenario by replacing the balls with the drones, all of them starting from a central base station. In the scenario where drones are only allowed peer-to-peer communication, the situation boils down to what was observed

with the balls where they had to recruit other balls to form chains and move towards the boundary. The communication that the leaders of the shooters had with the central mass of balls was through this chain, just like how drones that fly far enough will have to rely on other drones to send their message to the central station. After the first chain touches the rim, we observed that all the other shooters just died and the balls started spreading evenly, still maintaining contact with their chains. Similarly, we have a ring/boundary in our problem and once one drone reaches there, the other drones just need to spread out optimally to cover as much area as possible, while still maintaining contact with a few other drones. The balls do follow laws of physics, whereas there are no such restrictions when it comes to the drones, since we can assign a path to each of them. But we made the assumption that the path followed by drones was the most optimal one since humans and science have learnt a lot from nature and natural processes.

The first logical step after this was to simulate the ball bearings experiment. After coming up with some method to assign paths to the drones, either by following the laws of physics or heuristically but still ensuring that the behavior seen with the ball bearings is replicated, the idea was to implement this on drones used to survey underground gold mines in Brazil. The issue with that was that the location determined by the GPS on the drones was within 10 meters of the accurate location, and that was a slightly large error margin to work with. The idea was this: we were planning to use pixels as images and a circle as the boundary. In each iteration, we would go over each pixel and assign a new position to it (which could be same as the previous position) based on how that ball bearing should move.

3.2.1 Simple agent

We started off by defining functions to convert Cartesian coordinates to polar, as it would be easy to work with polar coordinates especially when things moving radially outwards from a center to the circumference of a circle are involved. We also defined functions to get the L2 norm as well as the distance between two points. We started by getting N number of points randomly within a circle of very small radius, which served as the balls starting off as a central mass. We varied N to see how that affected the results and we'll discuss that later. The initial set up looked like the one shown in Fig.3.1.

We then defined a class called the agent, and each ball bearing would then be an instance of this class. The inherent properties include a radius, a charge, a mass, and a position. There were functions to find the distance between two ball bearings and to maintain the size of a ball bearing bearing if it suffers a collision with another ball bearing. Such measures were needed since pixels could easily overlap during a collision, a case that won't happen with the ball bearings. For each pair of agents, the position was updated based on the force acting on them in agreement with the Coulomb's law. Agents' position was also updated based

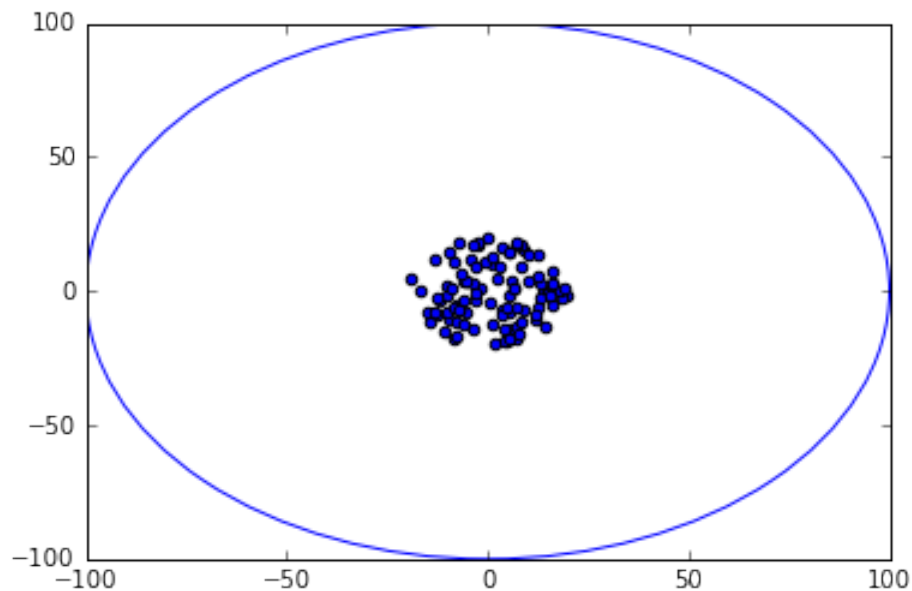


Figure 3.1: Initial set-up

on the force they experienced from the rim of the petridish, which was also charged. Hence the forces considered were the gradient force, which was a function of charge and distance from the outer ring, and the pair potential, which was a function of two agents (charges and distance). Whenever there's a collision and two agents touch, they share the charge and hence have equal charge.

We defined another class for the rest of the world, where the circular rim was modelled as a point mass with charge equivalent to that of the entire rim, at the center of the petridish. All the location updates for the agents were carried out and the behavior was monitored by constantly plotting the petridish and its contents. We realized that the model didn't behave the way we expected it to. Since the Coulombic forces between like charges just push the balls apart, all the balls just started going away from each other and no chains were formed as the balls with like charges couldn't come close to each other. We realized that there were other forces at play here. What we got by just implementing the Coulombic forces is shown in Fig.3.2.

We called this attempt the simple agent. To recap, each agent has a position, charge, radius, and mass. Agents have two forces acting on them, local (between other agents) and global (between the agent and the outer ring). The outer ring was simulated as a point charge in the center that repels charges. In every iteration, we apply the local forces and global forces, both computed by Coulomb's law.

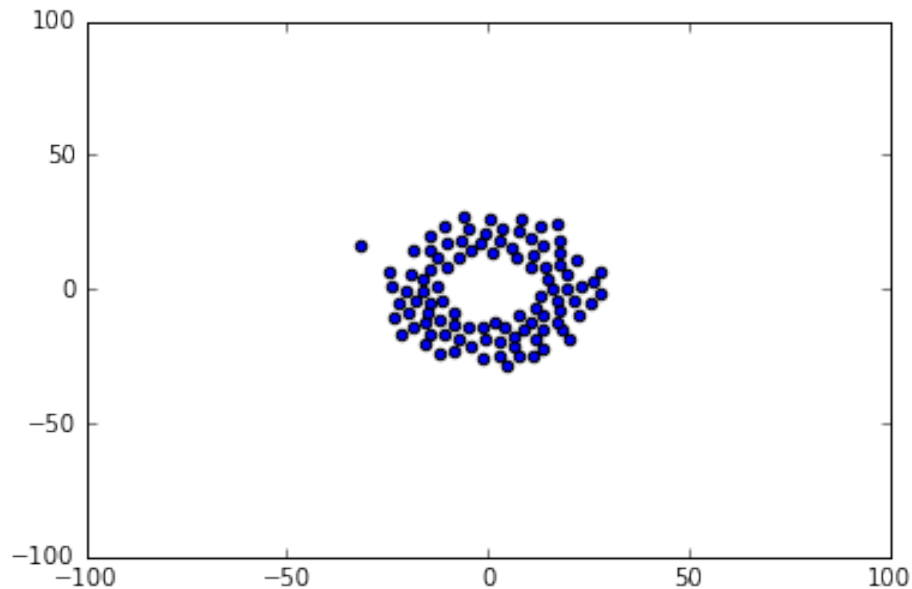


Figure 3.2: Ball bearings under the effect of the Coulomb force alone

3.2.2 Double agent

We called our next approach the double agent approach. One of the issues that we faced with the previous approach was that given that all the ball bearings had the same sign of charge on them, no chains were being formed as like charges repel. To mitigate this issue, we came up with ball bearings that have both a positive and a negative side. Each Double Agent is made of a linked positive and negative subagent. This allows some small linking between Double Agents as one's positive subagent can be attracted to another's negative subagent. We were able to observe the linking behavior in the simulation. Now, each agent had a top subagent and a bottom subagent and we had to ensure that all the sizes were maintained upon collision and that the locations of the top and the bottom subagents were such that they were still together, after each update etc. To be consistent with the ball bearings seen in the actual experiment, we still had to maintain the overall charge on a ball bearing to be -1. Hence we assigned a charge of -3 to the negative half and +2 to the positive half.

The updates were pretty similar to the first case, where Coulomb's law was used for pairwise updates of agents as well as for the forces acting between the rim and the ball bearings. The only difference here being that each agent is now comprised of two subagents, so we had to find the forces acting on the top subagent, find the forces acting on the bottom subagent, and move the ball in such a way that these forces are taken into consideration while maintaining the size of the ball and keeping the subagents together. The obtained result is shown in Fig.3.3. As mentioned, we did observe some linking behavior between the

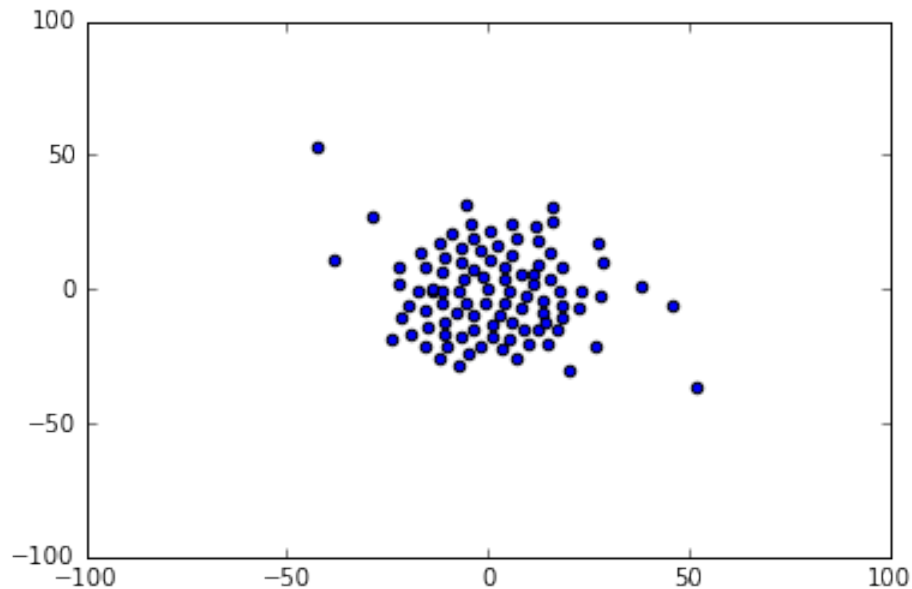


Figure 3.3: Ball bearings with linked positive and negative subagents under Coulomb force

balls and we could see some chains being formed but since like charges repel each other and since the rim attracts the balls from the outside (or repels from the center), the balls did move out in general.

3.2.3 Shooter agent

Our next approach was the shooter agent approach. Shooter was the name given to the leading ball bearing in each chain by the original authors. We decided to use some heuristics to simulate the behavior seen. We start off by choosing a few agents as the leaders. We do this by finding all those agents that are farther than a given radius from the center. The logic behind choosing these outermost agents as leaders was that the role of the leaders is to lead their chains to the outer rim and the agents farthest from the centers have the best chance of doing that. This was also in agreement with what we observed in the ball bearings video. In this set-up, every agent has a leader and a follower attribute. So, in some sense, every agent knows who its leader is and who its follower is.

This is what happens in every iteration: every leader updates its position radially outwards, every leader's follower goes to its corresponding leader's old position, their followers similarly move into their original position and so on. This is continued down the chain until we reach an agent without a follower. This is where recruitment happens. The free agent closest to this agent without a follower is recruited into this chain and is assigned as the follower of

the last ball in the chain. Here, a free agent is a ball bearing that is not a part of any chain. Hence a free agent is a ball bearing that neither has a follower nor a leader. Followers are the ones that have a leader and leaders are the ones that don't have a leader but only a follower. These actions are repeated then in subsequent iterations. The sizes are maintained in case of collisions as in the previous approaches.

One thing we ensured while trying to find followers was that the closest free agent shouldn't be farther than a threshold distance away, which was set to a default value of 10. This ensured that ball bearings didn't randomly jump really far from one iteration to the next. This is also done when initially choosing the leaders, because if we choose a ball lying really far away from the central mass of balls as a leader, it wouldn't be able to recruit any other balls since they would be very far away and that defeats the purpose of the shooter. The outer rim was still considered as a point charge at the center with a very high negative charge. Coulomb's force between this point charge and all the ball bearings was also considered. The pairwise Coulomb's forces weren't considered as the movement and locations of the ball bearings were described by the heuristics explained. Fig 3.4 shows a few iterations of this approach. The leaders are marked in red, the followers in green and the free agents in blue. We can see how in the first iteration, the ball bearings farthest from the center, but still connected to the central mass, are chosen as leaders. We see some green balls in the second iteration and it becomes clearer in the third iteration, these are the followers being recruited by the leaders and their followers. We then see in subsequent iterations how the leader moves radially outwards and the followers occupy their leaders' positions. We also see the number of free agents reducing and more and more balls becoming part of some chain. The race is on and this phase will end when one of the chains reaches the rim (not shown in the figure).

3.2.4 Shooter agent: discretized version

We tried a variant of the shooter agent approach described before. We discretized the petridish and hence the ball bearings are allowed to take only certain positions now. Hence, on updating the radial position, if the new position is one of the allowed ones, we just let it be. If not, the closest position that is allowed is found and the ball is moved there. This is inefficient as we have to go through all the allowed positions to get the closest one. This would be more inefficient when Y-shaped branches are introduced into the simulation, to mimic the real life experiment better. The size of balls is maintained as before in the event of a collision.

The process of choosing leaders was also changed. The idea here was this: imagine an arc of a circle with center at any given agent, radius being the distance between that agent and the center of the petridish, and angle defined as a hyperparameter. Now imagine the line segment from the agent to the center of the petridish sweeping angle/2 both to its right and left with the location of the agent being a pivotal point. If all the other agents lie in this imaginary arc, then that agent is a leader. To put things into perspective, let's say the

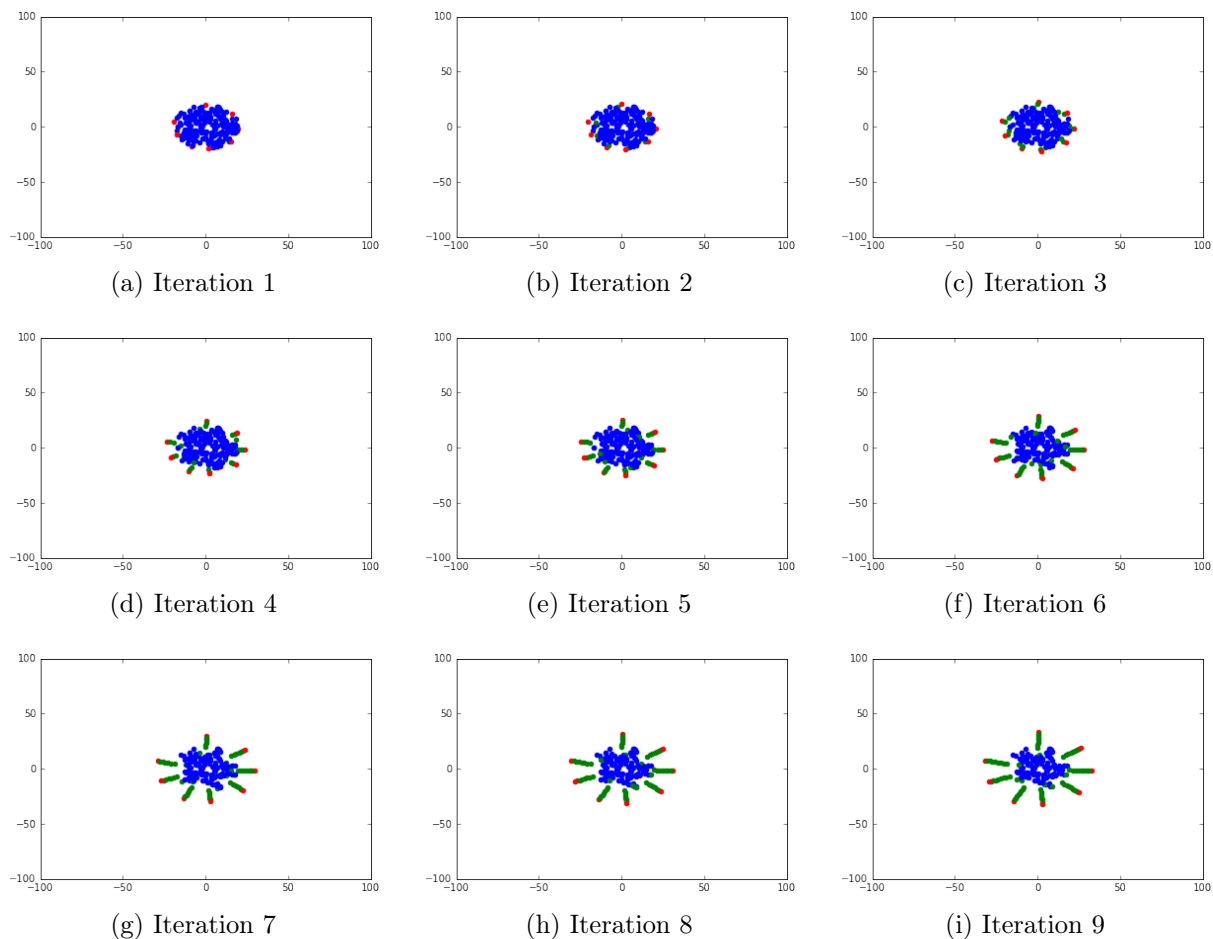


Figure 3.4: Ball bearings as shooter agents with leader and follower attributes

chosen angle is 180° . So we are asking the question whether all the agents lie in a half plane which is defined by the line perpendicular to the line joining the agent and the center of the petridish and passing through the agent. If all agents lie in this half-plane, then we say that that agent is a leader. The idea is that followers or free agents will be more surrounded as compared to leaders. Leaders will always have an arc where they would no agents, the angle covered by this arc is a hyperparameter.

The way that is done is like this: we find the slope of the line perpendicular to the line joining a given agent and the center of the petridish, by using the logic that the product of slopes of perpendicular lines is -1 . With this slope, we find the intercept of the line passing through the agent. Thus, we have the hyperplane that divides the plan into two half-planes, in the 180° angle case. Then, we looked at two cases: if the desired angle is less than 180° and if it's greater than 180° . One question that comes to mind is whether we can reduce the

dependence on having an idea about the location of the center of the petridish. We realized that we needed to know where the center was because otherwise it could happen that some agent has no agents in the arc that mentioned before but that arc might include the center of the petridish, as in, the agent under consideration might be way inside the central mass and have neighbors surrounding it in the direction facing the petridish wall but not towards the inside; such an agent can not be a leader and we wouldn't know this if we didn't know the location of the center.

In the first case, where the desired arc covers an angle of less than a 180° , we first check to see if there's any agent in the half plane not containing the origin. If there is, then the agent under consideration can not be a leader and we move on to the next agent. If not, then we find the slope of line joining the agent under consideration and every other agent. We then find the acute angle between these two lines, since we know their slopes. The angle between two lines with slopes m_1 and m_2 is $\tan^{-1} \frac{m_1 - m_2}{1 + m_1 * m_2}$. Using this, we can figure out whether every agent lies within a large arc of a circle swept by the line passing through the agent under consideration and the center of the petridish and covering an angle determined by us on either side, with the agent under consideration being the center. All the cases with infinite slopes, zero slopes etc. were taken care of. If so, the agent under consideration is assigned the role of a leader. If the angle is greater than 180° , the only points that could be the ones not lying in the desired arc are the ones not in the half-plane containing the origin. For such points, we just use the analysis used above for the case when the angle was less than 180° .

We found the angles between agents over several simulations and tried to manually look at the angles made by what we thought could be called leaders with all the other agents and we realized that the best angle was a 170° . Another change in this particular approach to more realistically model the behavior seen in the self-assembling wires experiment was that the recruitment happened before the chain moved. If a chain could not recruit a member, it wasn't allowed to move radially outward in that iteration. The way to recruit as well as to update the position of agents based on forces acting on them was the same as was used in the previous approaches. Also, to make the recruitment easier, we made the free agents repel each other away from the central mass using Coulombic forces, taking care to see that they weren't repelled very far away. The sizes were also maintained as before upon collision. A few iterations from this approach are shown in Fig 3.5. The allowed locations are marked in cyan. The leaders are chosen with the modified approach mentioned here, and then they recruit, and move their chain ahead only if they are successfully able to recruit in an iteration. We see that this approach replicates the original experiment really well, preventing some balls from going astray, since there are only a fixed number of positions that the balls can take. The only thing missing from the first phase seen in the experiment is the Y-shaped branch. The issue with that is that the number of allowed spots becomes less dense as one goes radially outward but most of the branches were seen later, when the leaders moved sufficiently away from the central mass. In that case, we would have to define new allowed

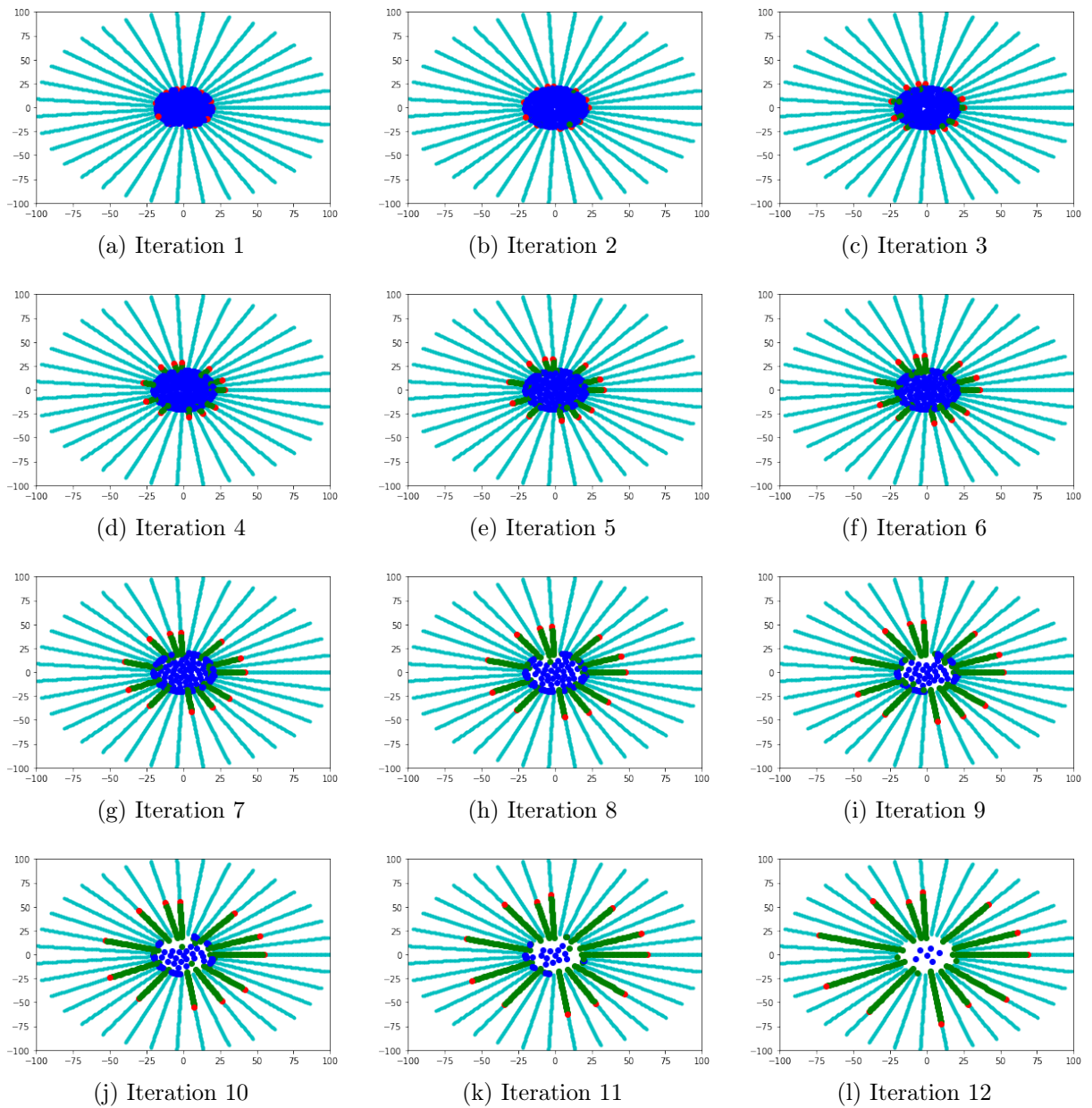


Figure 3.5: Shooter agents but with only certain positions allowed

spots that move radially outward starting from each branching point.

3.2.5 Neighbor Model

In this model, the ball bearings move based off of local neighbor statistics. The shooter logic was still kept the same: the furthest agent from the center would be the one to move out. But now we tried to pose a question asking what if the center itself is not known? In such a case, what exactly does one mean by the furthest agent? So instead of the furthest ball bearing, we started looking for the least connected ball bearing. Now, by connected, we not only mean touching another ball bearing, but the number of neighbors around a ball bearing within a given radius. It was observed that the shooters, which were mostly on the edge, had only as much as half the number of neighbors as compared to their counterparts that were not at the edge of the central mass of ball bearings.

We also needed to figure out how would the ball bearings form chains in this setup. The underlying concept here is that each ball bearing needs to be within a delta of another, and if that's not the case, we would heuristically move it towards the closest neighbor. We tried another approach, where instead of moving it towards its closest neighbor, we pulled the closest neighbor towards the ball bearing under consideration, using it as an anchor and this replicated the experiment seen better. We finally decided to pull each ball a distance of $\text{radius}/2$ towards the other. Thus, in each coordinate, we pulled it by $[\text{distance in that coordinate}] \times \text{radius} / [2 \times \text{Euclidean distance between the balls}]$.

Just like in the previous approaches, the sizes of the balls were maintained in the event of a collision. For each agent, the number of neighbors within a radius of $5 \times \text{radius}$ of the agent was found out. Then, this is what we did for each agent: we iteratively went through all the neighbors of an agent, hence we looked at all the balls that were within 5 radii away from the agent under consideration. The number of neighbors for each of these was found out and the minimum number was noted. Now, if the agent under consideration had zero neighbors, that would mean that it is really far away from all the other balls and hence can't serve as a leader since it would have to pull another ball to be a leader but then there are no balls in its vicinity. So, for such balls, we just found the closest agent, and moved the two close to each other by a distance equal to the radius.

The other case was when the number of neighbors of the agent under consideration was the less than that of each of its neighbors, but not zero. So, in its circle of agents, this ball is the one that is least connected, but it has neighbors that it can pull with itself. Such balls are perfect candidates to be leaders. Hence, we move them radially outward by a distance equal to 1.5 times the radius, just like in the shooter case, then we find the nearest agent, and then we move the two close to each other by a distance equal to the radius to simulate the Coulombic forces. The last case was when the number of neighbors of the agent under consideration wasn't lesser than each of its neighbors, which just means that it's a follower. After each iteration, the number of neighbors was updated for each agent. The results of a

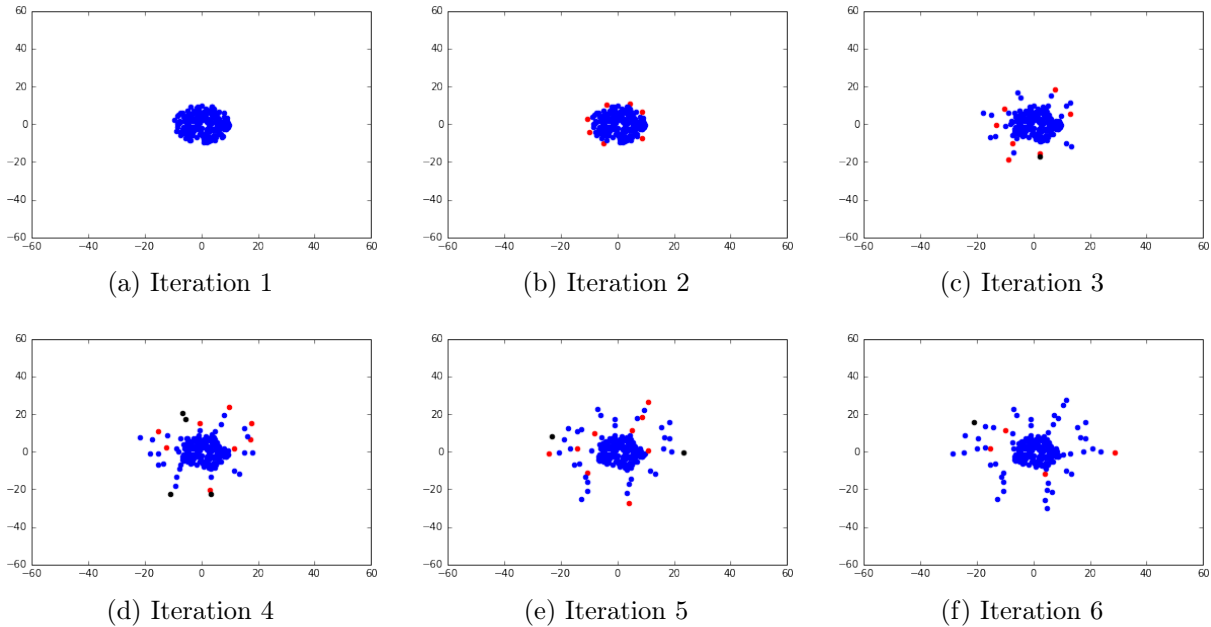


Figure 3.6: Ball bearings in the neighbor model move based off of local neighbor statistics

few iteration are shown in Fig 3.6.

This model didn't work as expected. The selection of leaders at the beginning was fine, we do see that the ball bearings at the edge with lesser number of neighbors as compared to their neighbors are selected as leaders. But at times, two balls would sort of pair up and move away from the central mass. Both of them would then only have one neighbor, which would be the other ball, and hence their number of neighbors wouldn't be lower as compared to their neighbors', and it would be non zero as well. So, they would act as followers but given that they are not part of any chain and given that their closest neighbor would be their pair, they are in some sense perpetually stuck with the other ball. This was a pretty prevalent problem as can be seen from the figure: there are several balls in pairs. We do see some leaders leading their chains well towards the outer rim. In some cases, we see that the leader gets converted to a follower since once balls start moving out in chains, their number of neighbors reduces and hence the leader doesn't have much issue with matching the number of neighbors with a follower, hence not satisfying the criteria defined in this approach to be a leader. Thus as time goes on, the number of leaders decrease, which was not something that we observed. We tried implementing a criteria that once a ball becomes a leader, it can't become a follower, but that came with its own set of problems. Another issue observed due to the decrease in the number of neighbors as balls are pulled out of the central mass was that some of these followers start becoming leaders as they have the least number of neighbors as compared to their neighbors, which are mostly balls in the central mass. Hence

the last follower in a chain, closer to the central mass, starts acting like a leader and starts a new chain, thus breaking the connection that the original chain had with the central mass. Due to these inherent issues with the model, we decided to discard it.

3.2.6 Potential model

The other attempt to simulate the self assembling wires experiment was to leave out the heuristics and model the entire behavior based on laws of physics. To this end, we started off by initializing a square grid of size $N \times N$ with 1s everywhere and 0s along the boundary. We placed m balls on the grid, drawn from a 2d Gaussian. The iterations, which will be described later, were run till convergence. Then, the direction of maximum gradient for each ball was found and the ball was moved. Similar to the collisions that we have described before, we took care of slots with double occupancy by moving in the direction of the second highest gradient. This entire process was continued till some ball became a part of the boundary, which marked the end of phase 1, after which the potential was computed again.

We started off with a $N \times N$ grid, where N was chosen as 20, and 30 balls. As mentioned before, the grid was initialized with ones everywhere, except the boundary slots, which were initialized to zero. We created another grid to keep track of the balls in the square grid. We then drew m , which was 30 in this case, locations for the m balls from a 2d Gaussian distribution whose mean was at the center of the square grid, with an identity covariance matrix. Given that these were independent draws, it could happen that a ball is picked out to be at the same location as another one, in which case that draw was performed again. Before we settled on the Gaussian multivariate distribution, we also tried picking locations at random but that led to a couple of issues: there was no central mass as was seen in the experiment; this was resolved by restricting the locations that the balls can choose to start at to only those around the center; even then some balls were really far away from each other, so they couldn't be a part of any chain, so we went ahead with the Gaussian model. We also tried assigning fixed positions to all the balls, in and around the center, this will be described later. One such arrangement of the balls, drawn from the 2d Gaussian is shown in Fig. 3.7. The white squares are the locations in the grid where balls start at. As we see, they are pretty concentrated around the center and hence this model was a good starting point.

Keeping in mind that the potential values need to be recomputed if any ball reaches the boundary, we start off by checking whether any ball is already touching the boundary. This could happen as the locations are drawn from a distribution, though the probability is very low since that would be somewhere in the tail of the Gaussian distribution. We initialized another grid, apart from the square potential grid and the grid to track the balls, called boundary points with zeros everywhere but ones at the boundary. Initially, only the points at the boundary can be boundary points, but as the iterations go on, if a ball touches the boundary, its entire chain will be considered boundary point for our purposes since they'll all

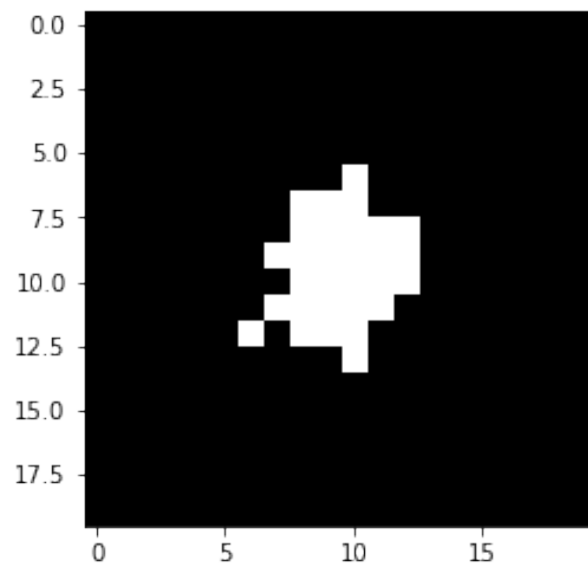


Figure 3.7: Potential model: Initial position of the ball bearings in a square grid

have the same potential value of zero. Hence, this grid is to figure out if any ball is touching the boundary as part of a chain. We go to each location that has a ball and look at this grid to see if that's touching already. If not, we look at its neighbors in all eight directions. If any of them are touching the boundary, information that we would get from the boundary points grid, and by touching we mean through other balls, then we update the boundary points grid to say that this location is a boundary point as well and change the potential of that location to zero. If this happens, then we need to go through all the balls again to see whether any of them are now a part of the boundary. Hence, we keep doing this till in an iteration, there are no updates to the boundary point grid at all.

Then, we start by recomputing the potential values at all the locations. This is done as follows: the first thing to keep in mind is that the potential of the boundary points is zero and that can not change. We look at each point in the potential grid and see if the potential value at a point is the average of its eight neighbors. If so, we move to the next location. If not, we assign the potential value at a location to be the average of its eight neighbors. We have to go through all the locations again even if one potential value changes, and this is what we meant by iterations till convergence. Once this is done, we need to now update the location of the balls.

For this, we initialize another grid to keep track of which balls have received an update in the current iteration. Also, balls will receive an update only if they haven't become a part of the boundary, as has been described before. For each ball that hasn't received an update in the current iteration, we compute the direction of the maximum gradient. We considered

several approaches for this. The first approach was to calculate the difference in the potential values in the first directions: vertical, horizontal, and the two diagonal directions, given that each ball has eight neighbors. So, in the vertical direction, we would just find the difference in the values of potential between the location above and below a ball's location. We abandoned this approach as it didn't consider the potential of the ball's position.

In our next approach, we went ahead with the same logic as before that the ball would move from a location of higher potential to that of lower potential, but this time we considered the potential at the ball's location as well. We found out the difference in potential between that of the ball's location and each of its eight neighbors. We then move the ball to the location with maximum positive difference, if that location is empty. If that location is not empty, we find the location with the second highest positive difference and move the ball there. Otherwise, the ball is left untouched. Once this direction that the agent is supposed to be moved in is found out, we update the grid that was being used to track the balls, we add this ball to the list of balls that have received an update in this iteration, and we also check to see whether any of the eight new neighbors of the ball are boundary points, and if so, we update the potential of the ball's location to zero, we add it to the list of locations that count as boundary points, and we recompute the potential at all the locations in the next iteration.

A few iterations of the potential model with a 20 X 20 square grid and 30 balls has been shown in Fig. 3.8. We see that all the balls start somewhere near the center, the locations were drawn from a 2d Gaussian distribution. We initially see the entire mass shifting towards the top left corner, that is because that's a region of lower potential, but we also observe some chaining. Some chains completely break away from the central mass, thus not replicating what we saw in the experiment. This behavior is needed since in our bigger picture, we want the drones to communicate back to the central base station in a peer-to-peer manner and hence it needs other drones to form a chain till the central station.

We tried increasing the number of balls to resemble the original ball bearing experiment. We took a 40 X 40 grid and 100 balls. The iterations are shown in Fig. 3.9. We again see the mass of balls moving towards the rim, again because of the fact that the boundary points start off with zero potential, which is the lowest and by the averaging of the neighbors logic we used to determine the potential of a location, the gradient is towards the rim. We again see similar behavior where some chains are formed but the chains lose contact with the central mass. We discarded this approach as it wasn't replicating the behavior seen in the experiment. The physics and calculus involved were much more nuanced and we couldn't model in all the laws. Electrons are discrete entities and one of the first deviations from the actual experiment is that we didn't account for how many electrons land on each ball per step. We also didn't account for charge lost automatically, as in due to some electrons travelling the whole way through the oil to the outer ring. This will be proportional to the squared distance to the ring. Also, to get to the attracting outer rim, the balls need to move, which costs energy. In other words, when they move, they lose some of their charge. We

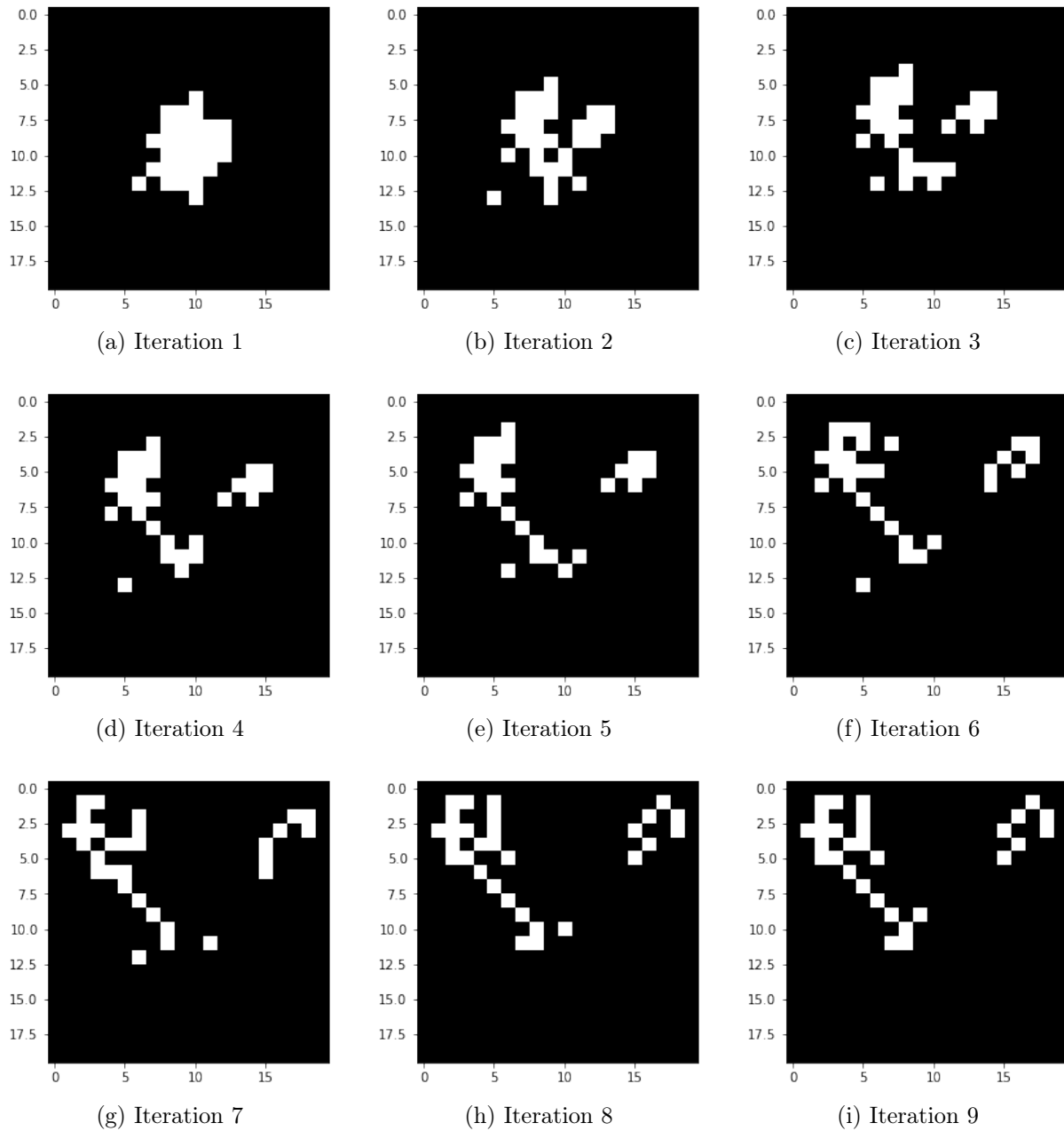


Figure 3.8: Iterations from the potential model with $N = 20$ and $m = 30$: the white squares show the location of the balls.

didn't experiment with how much charge is lost per distance.

3.2.7 Visualization and phase 2

We also created visualizations as each of the aforementioned approaches went through the different iterations, by putting together the snapshots pertaining to each iteration and converting that to a video. We wrote a utility code called the visualizer for this purpose. It has functions that generate animations of the generated plots over different iterations. It runs the plotter with specified engines and saves figures as well. This helps in easily following the path of different ball bearings and in seeing how similar or different each approach is as compared to the original experiment. One of the initial issues we faced with our visualization was that all the particles were in the entire field of view all the time. So, for instance in the simple case with only Coulombic forces in action, the video started off showing the entire central mass in full screen and as particles moved away from each other, the axes were correspondingly scaled but one couldn't clearly see the particles move outwards. Similarly, in the shooter agent case, instead of looking like the agents were shooting out of the central mass, it looked like the the central mass was contracting and shrinking away from the shooters which were just at their place. This issued was fixed simply by preventing the scaling of the axes.

We went ahead with the shooter model and the discretized grid as that was the approach that most closely resembled the original experiment. The first leader touching the rim of the petridish marks the end of phase 1. In phase 2, as has been described before, the ball bearings sort of spread around trying to cover as much area as they can while still maintaining contact with their chains. The chains start snaking and twist and twirl, and this also helps with our big picture of implementing this approach to determine an ideal path for a swarm of drones to cover as much area as possible in a specific region. To this end, apart from updating the radial position, we realized that we also needed to update the angular position of the ball bearings. Given a certain angle to update the position of an agent, the ball bearing was moved by that many radians in the required direction, and then if the new position was not one of the allowed positions, then just like with the position update case, we found the slot closest to the current location that was an allowed slot and moved the ball there.

To replicate the snaky motion, we initially just decided to move every odd numbered ball, starting from the leader, in one direction, say left; and every even numbered ball to the right. On running the simulation, we realized that some degree of randomness was needed as this made all the chains, except the one that has made a connection with the rim, behave similarly. We then flipped a coin for each agent and let it decide whether it wanted to move left or right by a radian. This ensured that there was enough randomness in the system. But, the connections between agents in a chain was getting broken. We then used alternate agents as anchors, not letting them move in a given iteration, and using the ones that moved in this

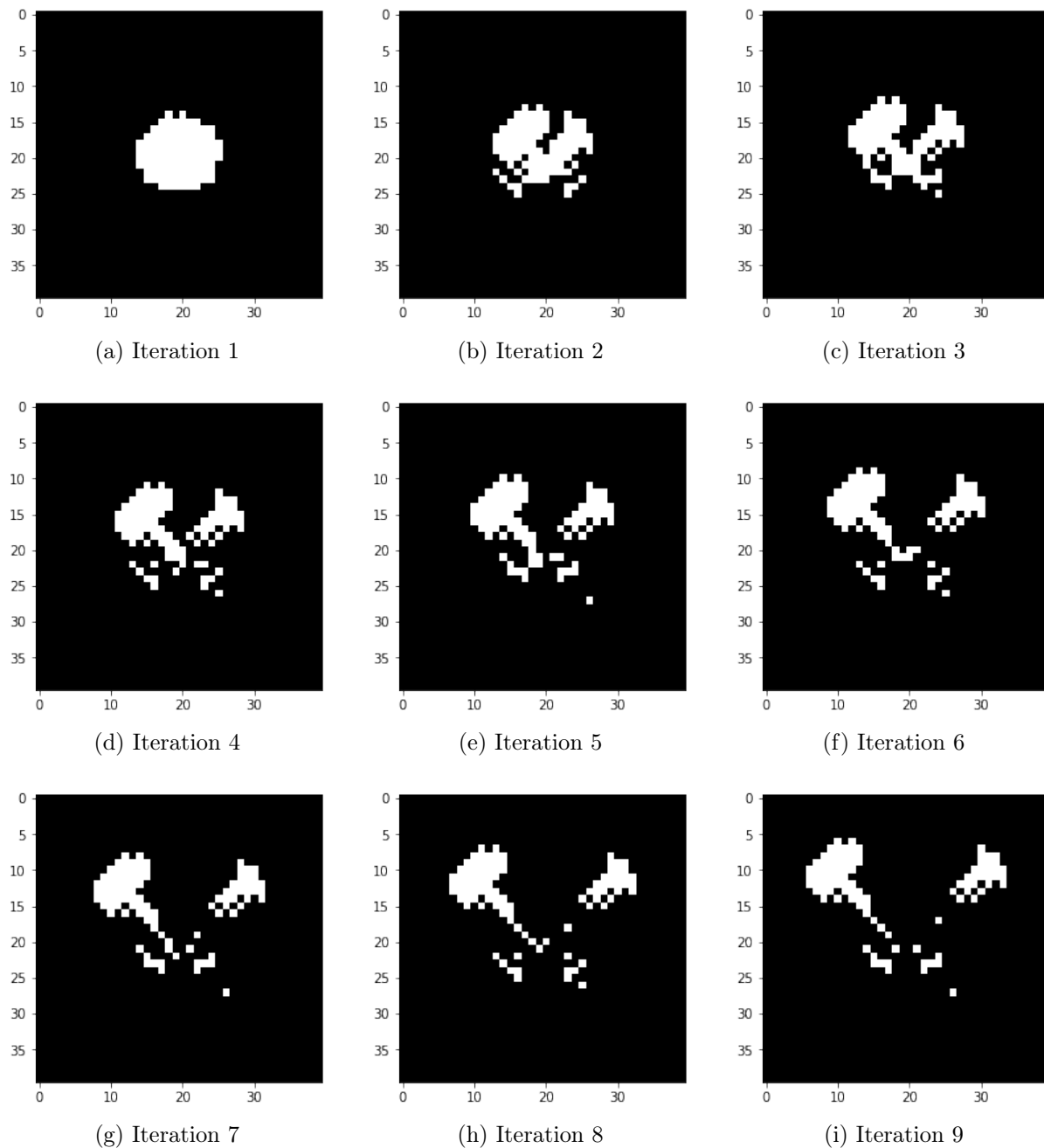


Figure 3.9: Iterations from the potential model with $N = 40$ and $m = 100$: the white squares show the location of the balls.

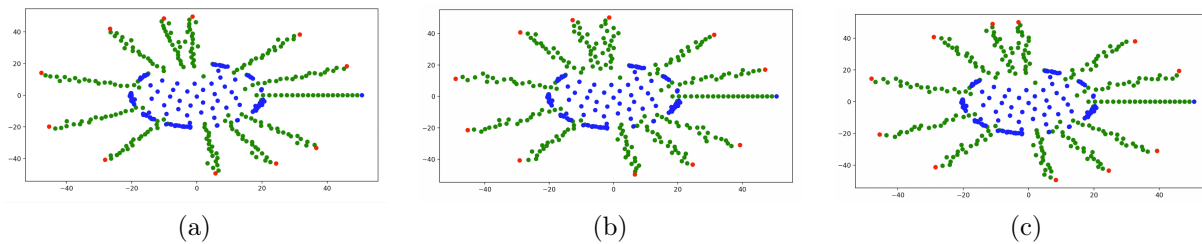


Figure 3.10: A few iterations from phase 2 of the shooter agent and the discretized grid approach.

iteration as the anchors for the next iteration. This idea helped keep the connections intact. We also imposed a threshold on how far an agent can move away from another agent it's in contact with to prevent large deviations. Sizes of the agents were also maintained, which was especially important when two chains snaked and came close to each other, situations pretty similar to collisions described and dealt with before.

When we looked at our simulations again, we realized that the free agents were not getting any updates in phase 2. Coulombic forces are still in action and even though they don't contribute much covering maximum area directly, they still help indirectly because by moving closer to an existing chain, they provide a path for the agents in that chain to transmit the information back to the central mass, in the case where the agents are replaced by drones. So we incorporated those updates as well. Again, collisions were taken care of. A couple of iterations of phase 2 are shown in Fig. 3.10.

Chapter 4

Heatmapping

4.1 Theory

Layer-wise Relevance Propagation (LRP) is a method that identifies important pixels by running a backward pass in the neural network. The backward pass is a conservative relevance redistribution procedure, where neurons that contribute the most to the higher-layer receive most relevance from it [19]. A pictorial representation, from the heatmapping.org website, is shown in Fig. 4.1

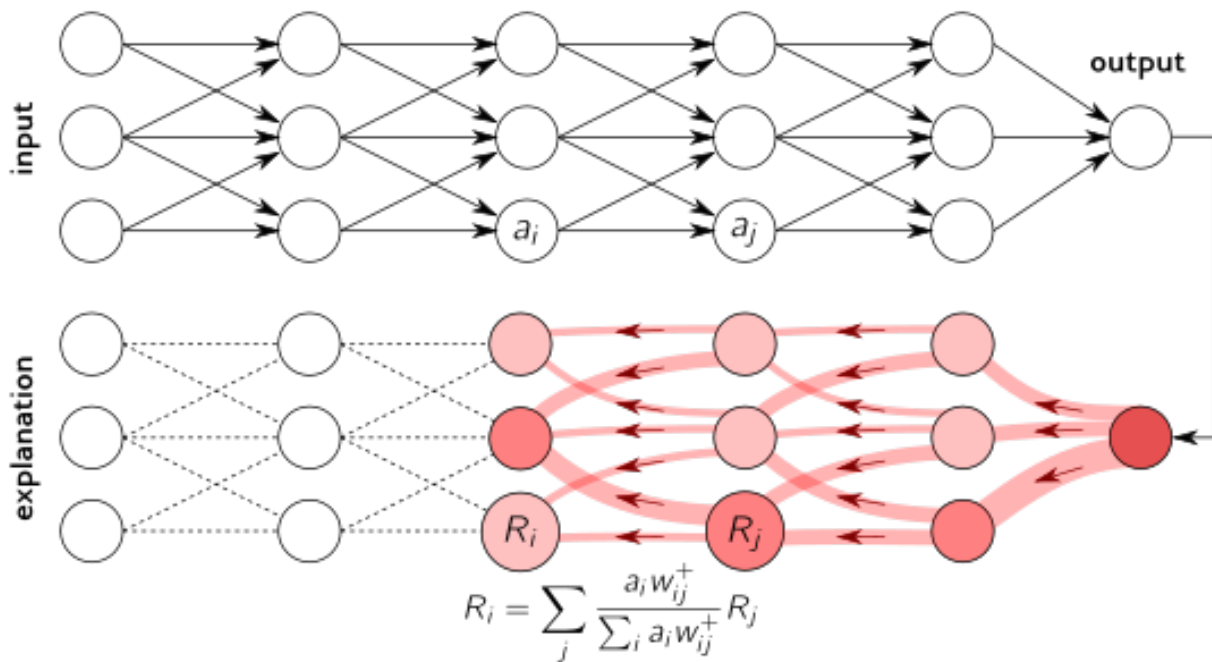


Figure 4.1: Layerwise relevance propagation procedure

Deep Taylor decomposition is a method to explain individual neural network predictions in terms of input variables [7]. This provided a much-needed push in the AI community towards not only the need for, but also a method to achieve, explainability. It operates by running a backward pass on the network using a predefined set of rules. It produces as a result a decomposition of the neural network output on the input variables. A heatmap is basically an assignment of these relevance scores onto pixels/input features. Deep Taylor Decomposition can be used as a visualization tool, or as part of a more complex analysis pipeline. The inspiration for Deep Taylor decomposition, as described by the authors [7], was derived from the way error backpropagation operates: dissociating the overall computation into a set of localized neuron computations, and recombining these local computations in an appropriate manner, similar to the chain rule for error backpropagation.

A decomposition seeks to redistribute the whole predicted output onto inputs, which in our case are the pixels of the input image. Decomposition techniques, as opposed to sensitivity etc. which only measure a differential effect, seek to explain the prediction in its entirety. It operates on a more global scale. The authors did introduce some positivity constraints as well, to prevent the possibility of a decomposition where a really large positive and a really large negative relevance values are assigned such that their sum is a small desired value. Consider x_f to be the final output and $[x_f]_p$ to be the contribution of x_p to the output. As we just discussed, we enforce the positivity constraint:

$$[x_f]_p \geq 0, \forall p$$

And then, the main idea behind decomposition being:

$$\sum_p [x_f]_p = x_f$$

As has been discussed, we'll start with the last layer and find the contribution of each neuron there to the output and iteratively keep coming back and redistributing the relevance till we reach the input layer. Now consider a neuron x_j and let's say we have found out that it contributes $[x_f]_j$ to the output. Now the question is: how to redistribute this $[x_f]_j$ to all the neurons in the previous layer that are connected to x_j . Consider any one such neuron and let's call it $(x_i)_i$, where it's the i^{th} neuron in the i^{th} layer, which is the previous layer, say. We want to find something like $[[x_f]_j]_i$. To solve this issue, the authors went ahead with an assumption that we can write $[x_f]_j = c_j x_j$, where x_j is the neuron activation and c_j is a factor by which the neuron activation affects the contribution, with the added constraint that $c_j \geq 0$. This is a very convenient assumption that leads to distributions that satisfy all the properties and constraints mentioned till now.

The interested reader may refer to the original paper [7] which explains the concepts of root points, search directions, and the use of Taylor decomposition to arrive at the final result. We'll take a more intuitive approach to it. The i^{th} neuron in the i^{th} layer has an

activation x_i . Now, out of all the neurons in the i^{th} layer, which interact with the neuron with activation x_j through weights $w_{i'j}$, where i' is any neuron in the i^{th} layer, it intuitively makes sense that the fraction of the relevance that should be redistributed on the i^{th} neuron should be something like $\frac{p_{ij}w_{ij}}{\sum_{i'} p_{i'j}w_{i'j}}$. Now p_{ij} should depend on the neuron activation x_i , and given that we are using ReLU activations, we should only consider those weights w_{ij} that are non-negative. The authors reach the same conclusion in the paper [7]. There's one additional thing, what we get from doing whatever has been described till now is the contribution of the i^{th} neuron in the i^{th} layer that it gets from x_j , but this i^{th} neuron of the i^{th} layer also interacts with other neurons in the next layer, so we need to sum up the contributions from all such neurons that this neuron interacts with:

$$[x_f]_i = \sum_j \frac{x_i w_{ij}^+}{\sum_{i'} x_{i'} w_{i'j}^+} [x_f]_j$$

4.2 Preliminary Experiments - MNIST dataset

The first task was to reproduce the task of predicting handwritten digits by training a neural network on the MNIST data set and the associated Layerwise Relevance Propagation (LRP) demo which results in a heatmap that highlights the regions of the image, basically the pixels, in proportion to how relevant they are in predicting the output. A couple of different architectures were tried out. The first one was a fully connected neural network with two hidden layers having 250 and 100 neurons respectively, with ReLU activations. The network was trained for 75 epochs, at which point the test error was at 1.12%. Following the steps outlined in the heatmapping tutorial at heatmapping.org and the corresponding paper on Deep Taylor Decomposition [7], a global propagation procedure from the top to the bottom of the network was implemented. The method described there iteratively visits layers of the network in reverse order, calls the propagation function of each individual layers, and feeds the output of each called function to the function of the previous layer, pretty much like how gradient propagation is implemented. As is described by the authors, relevance propagation operates at a higher level of granularity than standard gradient backpropagation. In particular, it considers as a layer the pairing of a linear and ReLU layer. Instead gradient propagation implementations typically treat them as two separate layers.

Next, I am going to describe the propagation rules for a linear-detection layer. This is affected by the input domain. Both of these rules are from the aforementioned paper, and as is explained there, the z^+ -rule applies to layers with positive input domain (e.g. hidden layers receiving as input the ReLU features of a previous layer) and the z^β -rule, applies to box-constrained domains (e.g. input layer receiving as input pixel values). The rules are being replicated here, from the tutorial at heatmapping.org and from [7]. The z^+ -rule is defined by the equation:

$$R_i = \sum_j \frac{x_i w_{ij}^+}{\sum_i x_i w_{ij}^+} R_j$$

where R_i is the relevance of the i^{th} neuron of some layer in predicting the output, w_{ij} is the weight assigned to the edge connecting the i^{th} and the j^{th} neurons. This is sort of intuitive actually, just by looking at the equation (and forgetting the non-linearity for a moment), we see that the procedure starts from the output and moves backwards. So, at any stage, we have the relevance of all the neurons in the next layer. To find the contribution of the i^{th} neuron in the current layer, we simply visualize it as contributing something to each neuron that it is connected to in the next layer (the outer summation), and how much is it contributing to a particular neuron in the next layer: that is simply in proportion to its output times the weight as compared to that of all the neurons connected to this specific neuron in the next layer under consideration. Now, to account for the ReLU nonlinearity, we only take those weights that are positive. This method is very intuitive and does satisfy certain constraints like positivity and summation, though the correctness of just saying that the relevance is proportional to the function value needs more justification. Certain small things had to be taken care of such as what happens when none of the inputs of a layer contribute positively to the neuron activation etc. To make the computations faster, it's often a good practice to vectorize the operations. We can look at the above rule as the following sequence of operations [7]:

$$\forall_j : z_j = \sum_i x_i w_{ij}^+ \forall_j : s_j = R_j \cdot z_j^{-1} \forall_i : c_i = \sum_j w_{ij}^+ s_j \forall_i : R_i = x_i \cdot c_i$$

For layers whose input activations belong to some closed interval $[l_i, h_i]$, the z^β -rule is defined as follows [7]:

$$R_i = \sum_j \frac{x_i w_{ij} - l_i w_{ij}^+ - h_i w_{ij}^-}{\sum_i x_i w_{ij} - l_i w_{ij}^+ - h_i w_{ij}^-} R_j$$

As is known, MNIST images are 28X28, are often converted to a 1X784 vector. We stored all the images in this format, and scaled them so that the values lied between -1 and 1. The target labels were one-hot vectors of length 10. We ran the classic feedforward pass on the network. This helps in getting the neuron activations at each layer and completes the beaten-to-death MNIST classification task. Only retaining the part of the output corresponding to the correct class, the output was backpropagated. The heatmaps we obtained were similar to the ones obtained by the original authors, and it is seen that the heat is spread more exhaustively on the contour of the digit. The heatmaps obtained are shown in Fig 4.2. We also needed to understand how to implement the relevance propagation procedures w.r.t. CNNs, and there were certain modelling artefacts in the MLP implementation, so we looked at a CNN implementation as well, for this MNIST classification task, again from the same tutorial.

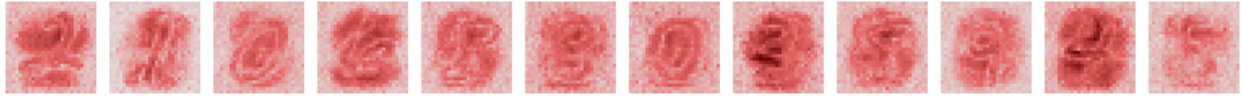


Figure 4.2: MLP - Deep Taylor



Figure 4.3: CNN - Deep Taylor

Since most of our main task (to be described later) was dealing with images, we had to deal with CNNs, and especially with propagating the relevance between a Conv/pool layer and a fully connected layer. This was one of the few things missing from the tutorial, but we were able to leverage the previously applied techniques by simply reshaping the outputs of different layers so that the shapes matched. Next, we describe the relevance propagation for CNNs. We used the LeNet-5 architecture described by LeCun et. al. in [24], with ReLU nonlinearities. One top-level pooling layer was added to further increase global translation invariance, as this was suggested by the authors of the tutorial. A test error of 0.81% was achieved after training for 15 epochs. Now we have to deal with convolution and pooling layers, for the decomposition of the output, and the rules for these can also be expressed in terms by feedforward activation and gradient propagation (both of them performing efficient multiplications by the weight matrix). The multiplication-type operations of both the z^+ -rule and the z^β -rule can be expressed as forward computation and gradient propagation of cloned layers with modified parameters. The proportional redistribution occurring in the sum-pooling layer given by the equation [7]:

$$R_i = \sum_j \frac{x_i \delta_{ij}}{\sum_i x_i \delta_{ij}} R_j$$

, where δ_{ij} is 1 if neuron x_i is in the pool of x_j .

Sensitivity analysis experiments were also reproduced from the original tutorial, both for the multilayer perceptron case and this CNN case, but details about that have been omitted for brevity. By inputting all the data as images of size 32X32X1, we obtain the heatmaps. The heatmaps obtained are shown in Fig 4.3. Comparing these heatmaps with the ones obtained from the MLP classifier, we see that the images are much more crisp here, in the sense that the heat (the red spots) is more concentrated on where the digits actually are and the background is pretty much completely white (less relevance). I guess this was expected because the task, at its core, is an image classification task. MLP gives good results because the task is a relatively easy one, one that requires very little capacity, but with other more complex tasks, the difference will be pronounced.

What we finally used in our main task, which is described in the next subsection, is

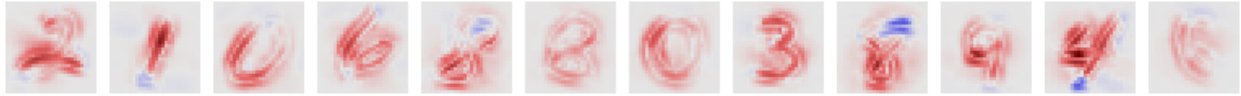


Figure 4.4: CNN - Alpha Beta

something called the $LRP - \alpha\beta$ rule, which helps increase the focus of the heatmaps and leverages the spread of heat very effectively [7]. One obvious question that came to our minds, and to the minds of the authors of the tutorial I believe, is whether negative relevance can be leveraged somehow. Given that we know the total number of classes in advance, we can think about why an image is not classified as something else (negative relevance) and whether that will help. We delve deeper into this in our discussion about our attempts to make the testing phase of classification using heatmaps, where the true label is not known yet is needed for generating heatmaps, faster and more efficient without having to try out all the classes. At this point, we just describe the implementation and the rule as described by the authors, and the application to MNIST data.

With the above objective of increasing the focus of the heatmaps in mind, it was found that injecting some negative relevance into the redistribution process helped. The LRP rule generalizes the z^+ -rule mentioned before and we reach the z^+ rule simply by plugging in $\alpha = 1$ and $\beta = 0$ in the following rule [7]:

$$R_i = \sum_j \left(\alpha \frac{x_i w_{ij}^+}{\sum_i x_i w_{ij}^+} - \beta \frac{x_i w_{ij}^-}{\sum_i x_i w_{ij}^-} \right) R_j$$

This negative relevance that is being injected in a way that is controlled by us has an inhibitory effect on neurons in the higher layers. This rule also satisfies relevance conservation between layers, as it should. The case where all the weights are of the same sign are also handled well. In the absence of negative weights, this implementation redistributes counter-relevance uniformly to all input neurons. The heatmaps obtained using this rule, with the value of alpha set to 2 (recommended), for the set of MNIST digits we selected, have been shown in Fig.4.4.

We see a few blue regions in the images here, those are the ones depicting negative relevance. If we look at some of the 8s shown, the loop isn't complete, and that's what the blue region shows. It says that that's a local irregularity, 8 is supposed to have complete loops. We see that the heatmap isn't as dense as seen earlier. Again, the heatmaps concentrate the heat on the region where the digit is present and assign very little relevance to the background. This is the rule we used in our main task, as we felt that this gives more information as we learn about what a template for each class should look like according to the machine or the presence/absence of certain characteristic features (such as a cat's ears or whiskers when it comes to a cat/dog classifier) that distinguish one class from another. This will not only inform humans about what features does the machine look for, it might also lead to

characteristics that humans might have never thought would be distinguishing (obviously, not every group of pixels seen by the computer as distinguishing would make sense to a human).

4.3 Centrifuge classification

This project was part of a larger effort to detect centrifuges in images. Centrifuges are used in nuclear power and nuclear weapon programs to enrich rare isotopes relevant to nuclear energy (such as Uranium-235) out of the mixture of isotopes found in naturally occurring compounds. Finding whether an image, obtained using satellite imagery or otherwise, has a centrifuge can be useful for national security purposes. Going manually through each image is not feasible and in scenarios like this, it's often important to understand why a machine thought that there was a centrifuge in an image. Interpretability and explainability are important and sometimes also help us see and understand things that human eyes might miss.

The first task was to classify a given set of images as being/having a centrifuge or not. We had a few images which had centrifuges in them, and hence served as ground truth (NFC imagedata). Given that manual annotation is both expensive and time consuming, this corpus was rather small, with about 300 different centrifuge images, classified into subcategories such as BWR Fuel, Magnox Fuel, Cooling Tower, Magnox Fuel, Reactor fuel pools, Reactor core fuel matrix, UF6 cylinders 30, and UF6 cylinders 48. For our purposes, all of these were under the broader umbrella (and hence the single class) of centrifuges. We got images of cats and dogs from the internet and used that as the not-centrifuge class. We scraped about 300 images of the not-centrifuge class as well. The dataset was roughly divided in a 5:1:1 ratio as training, validation, and test set. By stratified sampling, we ensured that each set had images from all the subclasses so that the distribution from which the images are drawn is similar for training, validation, and test sets.

We read each image in each of the folders (train and test, and 2 classes), resized them to a standard size of 150X150X3, reshaped them to #channels X height X width, and appended them to a list, which was then stacked. All of these were then typecasted to uint8. We skipped images that didn't have three channels, as the number of such images was so little that taking care of that issue didn't seem to be very useful. Labels were assigned to all the images, and then the labels were converted to categorical variables. We went ahead with a CNN architecture as the task was that of image classification. We tried filters of sizes 16, 32, 64, and 128; kernels of sizes 3X3, 5X5, and 7X7; pooling area for max pooling of sizes 2X2, 3X3, and 4X4; stride of 1 and 2 in both directions etc. in a grid-search manner. There were two conv layers with ReLU activations, each followed by a pooling layer. The images were not padded, hence the output size was smaller than the input size. Glorot normal initialization was used (also called Xavier normal initialization). From the documentation, it draws samples from a truncated normal distribution centered on 0 with $stddev = \sqrt{\frac{2}{(fan_in+fan_out)}}$, where

`fan_in` is the number of input units in the weight tensor and `fan_out` is the number of output units in the weight tensor.

We got the best result with 64 filters, a kernel size of 3X3, pooling size of 4X4, and a stride of 1 in each direction. The output of the final pooling layer was flattened and passed through a fully connected layer consisting of 10 times the number of neurons as there were classes, with ReLU activation. This was followed by dropout. We tried playing around with the number of fully connected layers and the number of neurons and this produced the best results. This was followed by one more fully connected layer with 2 neurons and a softmax activation function. We also tried different optimizers, such as Adam ($learningrate = 1e - 3, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e - 06$), Nesterov Adam Optimizer ($learningrate = 1e - 3, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e - 06, scheduledelay = 0.0001$), and SGD (learning rate = 0.01). Categorical crossentropy was used as the loss function and the metric to be optimized for was accuracy. We tried another model with a very similar architecture, but with 64 neurons in the fully connected layer, binary crossentropy as the loss function, a sigmoid activation function after the last layer (which had only 1 neuron), and rmsprop as the optimizer. After training for 50 epochs, the best accuracy from either of these models was 58.16%, which is only slightly better than a coin flip. We realized that the major issue here was the lack of enough training data, so we decided to try something different.

Since the amount of training data was less, we turned to using a pretrained model. We went ahead with VGG16 trained on Imagenet[12]. Researchers at Oxford came up with this model, which won the 2014 ImageNet images classification challenge, and the ease with which one can simply load and use the model weights made it a good choice. VGG16 has since then been used in a lot of transfer learning settings, where these pre-trained models are used with minor modification on wholly new predictive modeling tasks, harnessing the state-of-the-art feature extraction capabilities of these proven models. The final few layers often need to be retrained to fit to the task under consideration. We used the Keras deep learning library as it provides an application interface for loading and using pretrained models. There are two versions of the VGG model that the authors put out, a 19 layer model and a 16 layer model. We went ahead with the 16 layer model. The model expects inputs of a particular shape, specifically 224X224X3, so we had to reshape all our input images, which were of several different shapes. The architecture of VGG16 is shown in Fig.4.5.

Given that we are using this pretrained model for our own problem in a transfer learning framework, we chose the option to not include the top output layers, as those are very specific to the problem it was trained on. We added a couple of fully connected layers with softmax activation and trained these on our dataset. We did retain the weights of the convolutional layers from the original model, using those layers as feature extractors, as it was obviously not desirable to just retain the architecture and train it from scratch due to the lack of enough training data (and hence to avoid overfitting) and the required computational power. Most of these models learn hierarchical representations, that's why transfer learning works well.

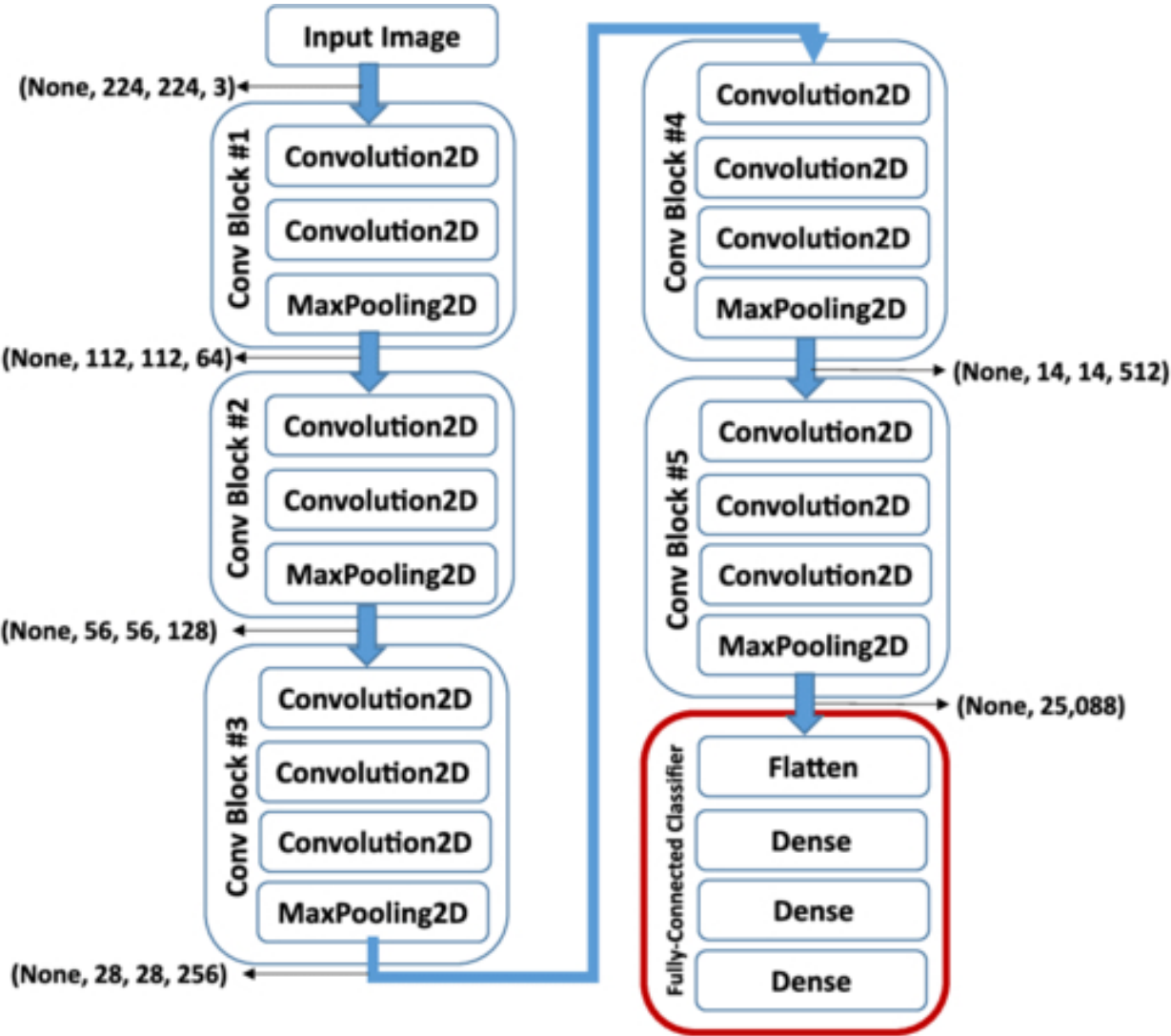


Figure 4.5: The original VGG16 architecture. The block marked in red is the one we played around with, in terms of the number of layers, number of neurons in each layer, activations, dropouts etc. The layers in this block were retrained on our data to fit our task. The weights in all the convolutional layers prior to this block were retained and those layers were used as feature extractors.

The first few layers learn general features (such as edge detection etc.) which can be reused across datasets and tasks, whereas the final few layers (which we retrained) are specific to the problem at hand. Also, the thumb-rule for using transfer learning over something like fine tuning is that the dataset should be small and the task should be similar, both of which hold to a good extent in our case.

The images are loaded using the `load_img()` function. Keras by default uses the PIL format, so the images are width X height X channels. Keras works well mostly with NumPy arrays, so the images had to be converted from pixels to arrays. One of the preprocessings done in the original paper [12] was that they subtracted the mean RGB value, computed on the training set, from each pixel. We did so as well, since the pretrained network is used to such inputs. Also, we removed all the images which were grayscale or had 4 channels as these were in some sense, outliers. Finally we stack all the images up as the input that Keras expects is a 4 dimensional tensor, samples X width X height X channels, this is called batching. There are a lot of convenient functions and classes which help us load images and batch them, from their respective folders. We used the `ImageDataGenerator` class to load the images and `flow_from_directory` function to generate batches of images and labels. One of the things that we observed was that the output was a 7X7X512 shaped 3-dimensional tensor. We added a sequential model on top, with a fully connected layer consisting of 250 neurons and a ReLU activation, followed by dropout, followed by another fully connected layer with 2 neurons (equal to the number of output classes) and a softmax activation.

After loading the images and generating batches of features and labels, for both the train and the validation set, the top layers were trained using categorical crossentropy as the loss function, and RMS Prop as the optimizer. Accuracies of around 90% were obtained on the validation set for this binary classification task. The model weights were stored for future use. Then we ran some basic checks such as visualising which images were wrongly classified etc. We observed that more images from the not-centrifuge class were wrongly classified. Given the higher variance of the images in the not-centrifuge class, this wasn't surprising as that meant that a network with a higher capacity would be needed to accurately classify the images in that class.

The complete architecture of both the VGG16 model and the FC layers was defined, and the weights and bias parameters of each layer were instantiated, as this was needed for the heatmapping part. Heatmapping is a technique to decompose the prediction of a neural network in terms of contributions of individual input variables such that the produced decomposition can be visualized in the same way as the input data. A heatmap, in our case, is thus an assignment of the scores onto pixels, based on their relevance in predicting the outcome [6]. Layer-wise Relevance Propagation (LRP) identifies important pixels by running a backward pass in the neural network. Firstly, the global propagation procedure from the top to the bottom of the network was defined. Deeptaylor propagation [7] rules (LRP is a deeptaylor decomposition of the prediction function), which are just a combination of the

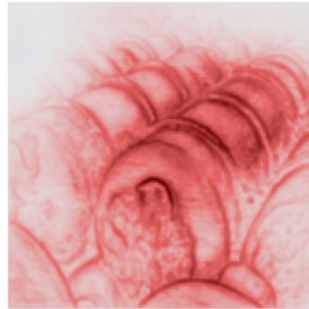
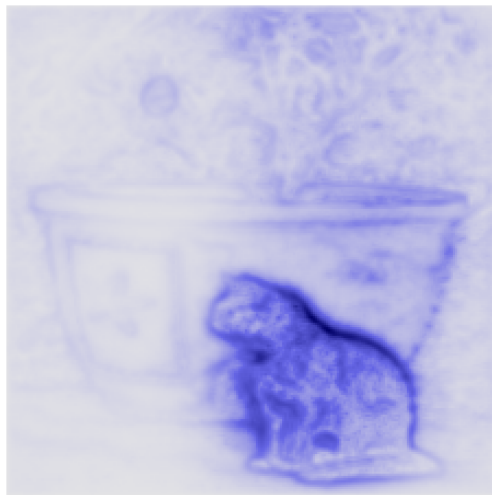


Figure 4.6: An example heatmap belonging to the centrifuge class

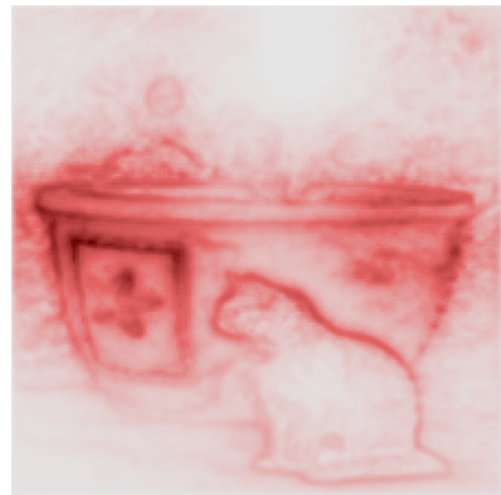
feedforward activations and gradient propagation, were implemented in the convolutional and pooling layers, as well as the FC layers. For the FC layers, a feed-forward pass was run to collect neuron activations at each layer, and only the part of the output corresponding to the correct class was kept. The backward pass was a conservative relevance redistribution procedure, where neurons that contribute the most to the higher-layer receive most relevance from it. So basically, layers of the network were iteratively visited in reverse order, propagation function of each individual layer was called, and output of each called function was fed to the function of the previous layer.

All the images were resized to $1 \times 224 \times 224 \times 3$, which was the standard input size we chose. A forward pass was run through the convolutional and pooling layers, and the output was collected, reshaped appropriately, and fed into the fully connected layers. Then, the output of that was sent to the backward pass and once we reached the first fully connected layer (last fully connected layer, coming from the back), which acted as a boundary between the fully connected and the convolutional parts of the network, the relevances there were redistributed accordingly taking into consideration the output at the convolutional layers. Then, these relevances were further propagated backwards through the convolutional and pooling layers till we reached the input layer. We converted RGB to grayscale and used helper functions to visualize the heatmap. An example heatmap is shown in Fig.4.6.

We did look at the heatmaps of a few images that were incorrectly classified. The regions highlighted in red (the regions with higher relevance, according to the network) did resemble eyes and other features from other classes, even though as humans, we could see barrels and cooling towers. Similarly, in certain images with cats and dogs, the background had a barrel or a pot and that was the region that was highlighted, and hence the image was wrongly classified as a centrifuge. There were interesting outcomes when we passed these images as part of the training set with labels. For instance, we took an example image which had a cat in the foreground and a huge flower vase in the background, as shown in Fig 4.7. When we told the network that the label was a centrifuge, it highlighted the vase, and when we said it was a cat, it highlighted the cat, both with probabilities very close to 1.



(a) Heatmap with 'cat' as the label



(b) Heatmap with 'centrifuge' as the label

Figure 4.7: An example containing characteristic features of multiple classes, thus making it harder for the network to classify

4.4 Other attempts

We also experimented to see if heatmapping could be used to correct labels in geotagging. We basically had a geolocation model inspired by the ideas presented in [22]. The data we used is from the geotagged images in the YFCC100M Multimedia Commons dataset. Training, validation, and test images were split so that images uploaded by the same person did not appear in multiple sets. The classes were created with the training data using Google's S2 Geometry Library as described in the [22]. This data did not have GPS labels though. We also looked at the Google landmark dataset released on Kaggle [18] for a recognition task. Our basic aim was this: all these images had tags which would give more information about the image. But some of the tags were either wrong or couldn't be seen in the image. We wanted to clean up the tags using heatmapping. The basic idea was to come up with a template heatmap for each class, by looking at the heatmaps for all the examples in the training set, and then removing a label from an image if some metric of the image under consideration was greater than a threshold value from the template.

We ran into several issues with this approach. Firstly, none of the approaches we tried served as promising candidates for what a template heatmap would look like for a class. Approaches such as averaging all the heatmaps, which worked for simpler datasets such as MNIST, didn't work with noisy images as seen in Fig.4.8. Some approaches didn't work in lieu of the fact that the main object in each image was present in different locations and the picture was taken at different angles. The definitions of the metric as well as the threshold were fuzzy as well. We used the sum of the absolute differences of the pixel values as the

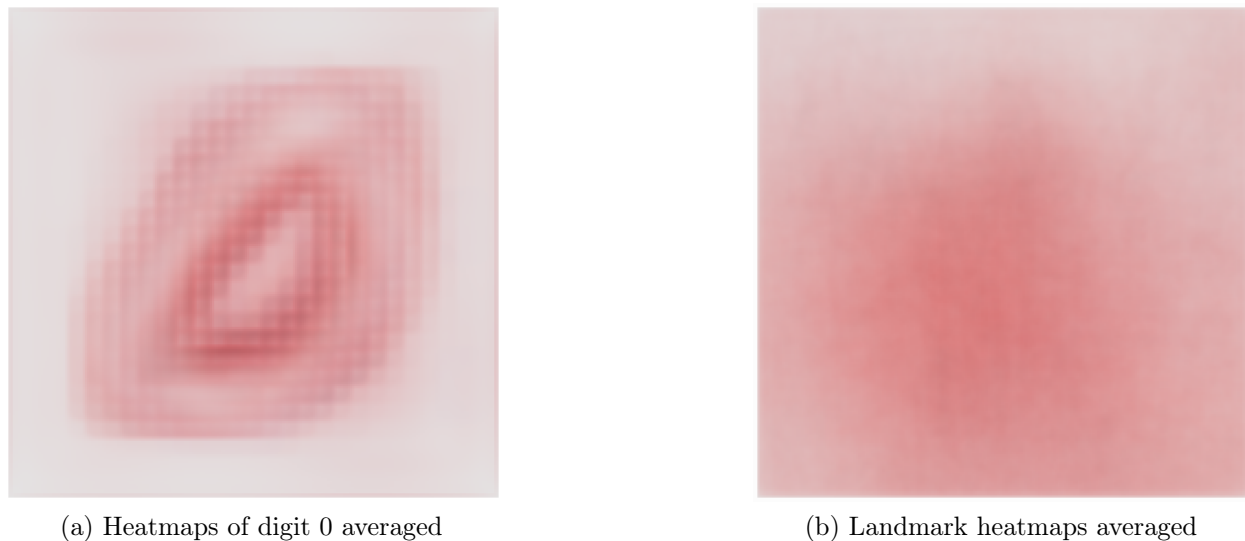


Figure 4.8: Averaging to obtain a template heatmap

metric which again ran into issues due to noise. We then only looked at the regions with high relevance but there was no quantitative measure to say that this was a good metric. As for the threshold, we just used the maximum difference between the template and any given image in the training set.

We then turned to Regional Maximum Activation of Convolutions and used the Keras implementation of that, as described in [9], [2], and [17]. The architecture of the model is as shown in Fig.4.9. In the original paper, the authors re-visit both filtering and re-ranking retrieval stages by employing CNN activations of convolutional layers. The compact vector representation encodes several image regions with simple aggregation method and is shown to outperform state-of-the-art competitors. The localization increases the performance of the retrieval system that is initially based on a compact representation. The same CNN information adopted during the filtering stage is employed for re-ranking as well. We employed this to find where the object of interest is in the image, and then use the heatmap of only this region of interest for our purposes of creating a template. This approach also suffered from some of the issues described above.

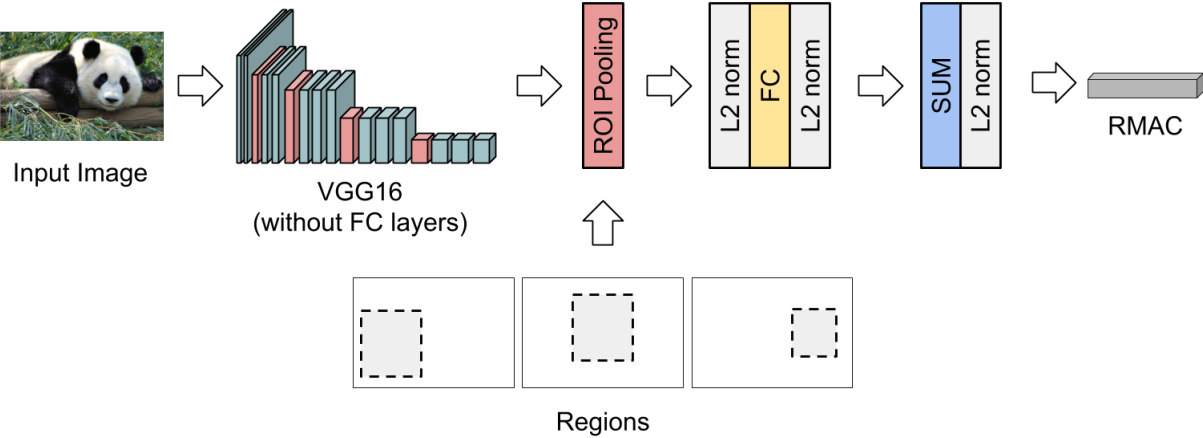


Figure 4.9: Architecture of the RMAC feature extractor

Chapter 5

The data processing inequality

5.1 Experimental procedure

The task was to classify a given set of images as being/having a centrifuge or not. The centrifuge images were obtained from NFC imagedata ¹ and the images for the other class were images of cats and dogs scraped from the internet. Since the amount of training data was less, a pretrained model, VGG16 trained on Imagenet [12], was used. The weights of the convolutional layers were retained from the original model, thus using those layers as a feature extractor, and the top fully connected layers were trained on our dataset for our classification task. After loading the images and generating batches of features and labels, for both the train and the validation set, the top layers were trained using categorical crossentropy as the loss function, and RMS Prop as the optimizer.

The complete architecture of both the VGG16 model and the FC layers was defined, and the weights and bias parameters of each layer were instantiated, as this was needed for the heatmapping part. A heatmap, in our case, is an assignment of the scores onto pixels, based on their relevance in predicting the outcome. Layer-wise Relevance Propagation (LRP) [6] identifies important pixels by running a backward pass in the neural network. Firstly, the global propagation procedure from the top to the bottom of the network was defined. Deeptaylor propagation rules [7] (LRP is a deeptaylor decomposition of the prediction function), which are just a combination of the feedforward activations and gradient propagation, were implemented in the convolutional and pooling layers, as well as the Fully Connected (FC) layers. For the FC layers, a feed-forward pass was run to collect neuron activations at each layer, and only the part of the output corresponding to the correct class was kept. The backward pass was a conservative relevance redistribution procedure, where neurons that contribute the most to the higher-layer receive most relevance from it. So basically, layers of the network were iteratively visited in reverse order, propagation function of each individual layer was called, and output of each called function was fed to the function of the previous

¹<http://www.icsi.berkeley.edu/fractor/NFC-imagedata.zip>

layer.

We tried a couple of things from here. In our first approach, we employed two neural networks: the first neural network was trained on the given images, thus generating the heatmaps. These heatmaps were then used as the input for the second neural network, which did the final classification. The idea here was that given that the heatmap highlights important regions in an image, training a new neural network on these regions of interest would help the second neural network learn the characteristic features of that class. We would then do the same thing for the test images. Given that the test images don't have a label, and that a label is necessary to generate the heatmap, we would first assign a label of 'centrifuge' to the test image and ask the neural network to highlight those parts of the image that it thinks were characteristic for the image to be classified as a centrifuge. We then assign a label of 'not-centrifuge' to the test image and again ask the network to highlight regions of high relevance. We then pass these two heatmaps through the second neural network, which was trained on heatmaps and hence has an idea of what the characteristic features of each class would be, which then gives probabilities that the heatmap that came from the part of the pipeline labelled 'centrifuge' is a centrifuge and the one from the other pipeline is not a centrifuge. We simply compare these probabilities and make the final prediction. The pipeline described here for testing is very similar to the pipeline we used for our approach with 'filtered' images described slightly later, which has been illustrated in Fig.5.2.

As mentioned in the discussion section, we observed that the second network was overturning some of the right predictions made by the first network. So, in our next approach, we looked at the prediction made by the first network and turned to the second network only if the confidence of the first prediction was low. In another approach we tried to come up with a weighted combination of the confidences of the predictions made by the two networks for our final prediction. Given that we are using the LRP - $\alpha\beta$ rule, there were blue regions indicating negative relevance. In another approach, we incorporated that as well because when the image was clearly that of a cat and we told the network to find the characteristics of a centrifuge in that, it simply marked several regions blue showing why that image is not that of a centrifuge.

For our next approach, we looked at the prediction made by the first network, generated heatmaps for the case when the class was not the one predicted by the first network, passed that through the second network trained on heatmaps, and looked at the confidence that that image belonged to the class not predicted by the first network. To give an example, if the first network predicts an image to be of class 0, we generated the corresponding heatmap with the label of class 1, passed that heatmap through the second network and looked at the confidence of it being class 1. If this confidence was high, we overturned the decision. Firstly, we concede that this is harder to generalize for multiple classes, but that in some sense holds for all our approaches because a heatmap needs a label and given that a test image does not have a label, we have to generate a heatmap corresponding to each class in

the dataset assuming in that pipeline that that class is the correct one. To tackle this, we did another experiment where the goal was to look at the confidences of all the classes obtained by passing the heatmap obtained by assuming any one class through the second network and seeing if classes with really low confidences could be eliminated from consideration.

The next logical thing was to think about what would happen if we just looked at all the pixels whose contribution was low and set those to zero in the original image, and retrained the network with these images. The first approach here was to look at the three channels in the heatmaps. We know that the relevant regions are highlighted in red, hence we looked at whether the value in the red channel was considerably higher than those in the other two, and if not (with the threshold being a difference of 0.1, after some tuning) we would blacken out that pixel in the original image because those are not regions of high relevance. This approach suffered a lot due to it being not dependant on the data itself and a lot of images where the most relevant regions were only slightly red got affected. Also, images with blue regions were not being focused on since we were looking only at the red channels and negative relevance was indeed a big factor to consider. We wanted an approach that filters based on the relevance values in each image. Hence, we employed two neural networks: the first neural network was trained on the given images, thus generating the heatmaps. We plotted a histogram of the relevance scores obtained from the heatmaps. We observed that if we took the absolute values of the relevances, then those bins with lower absolute values (low relevance, irrespective of positive or negative) had lots of pixels, and were noise. The bins with high absolute relevance values had only a few pixels, showing that both the positive and negative relevance pixels are important indicators. After adjusting for positive and negative relevance, we heuristically blackened out the pixels in the lowest two bins, using the logic that these pixels are mostly background noise as they are not contributing much to the final prediction, and that their removal would be akin to passing the image through a filter, which would result in better classification accuracies. A second neural network was trained on these ‘filtered’ images (Figure 5.1).

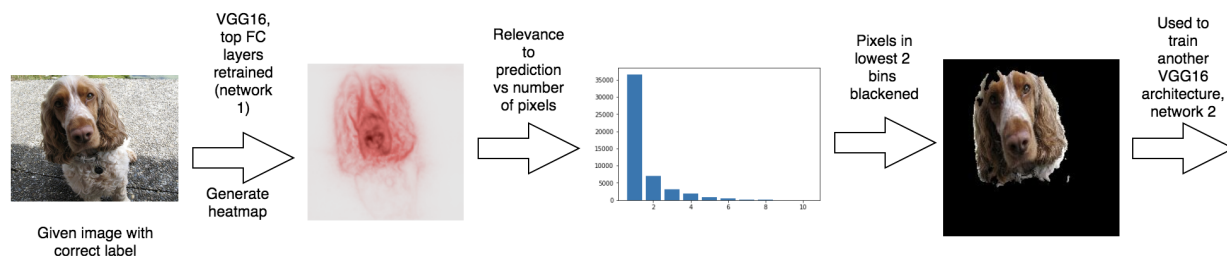


Figure 5.1: Training phase: an example of the not-centrifuge (cats/dogs) class is shown here.

During the testing phase, since the true label is needed for generation of the heatmaps, we passed the given test image through two pipelines. In the first, we assumed that it belonged to the centrifuge class, generated the corresponding heatmap, filtered it as described above,

passed the filtered image through the second network, and obtained the probability that it belonged to the centrifuge class. In the second pipeline, we did the same thing, except that we assumed that the image belonged to the ‘not-centrifuge’ class. Finally, these probabilities were compared, and if the probability that the given test image belongs to the centrifuge class from the first pipeline was higher than the probability that it belongs to the ‘not-centrifuge’ class from the second one, we predict its class as centrifuge, and so on. (Figure 5.2). All our work can be reproduced using the attached Jupyter notebook.

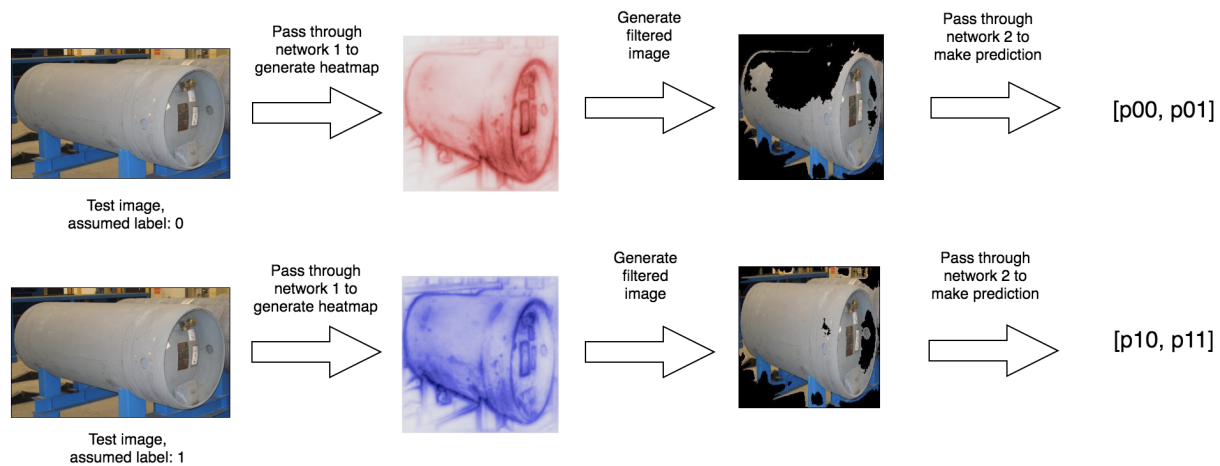


Figure 5.2: Testing phase: an example of the centrifuge class is shown here. In the top pipeline, we assume that the class of the image is 0 (centrifuge) and we get a heatmap with that assumption. The histogram has been omitted here for brevity. The filtered image obtained after blackening the pixels with low relevance is then fed to the second network, which outputs the probability of the image belonging to class 0 and to class 1. p_{ij} here corresponds to the probability of the label being j for assumed label i . In the bottom pipeline, we assume that the class of the image is 1 (not-centrifuge). This has to be done since the label is needed for generating heatmaps and the true label is not known for a test image. We see that the heatmap shows a lot of negative relevance (blue color). The filtered image obtained here doesn’t blacken out most of the background like is the case when the label is correct (top pipeline) because here we are telling it to find relevance that it is not a centrifuge. The final decision is made this way: if $p_{00} > p_{11}$, we declare that it is class 0; if $p_{00} < p_{11}$, then class 1, if $p_{00} = p_{11}$ and $p_{01} < p_{10}$, class 0; else class 1.

5.2 Results

87.36% accuracy was obtained on the validation set for the binary classification task, during retraining of the top layers of the VGG16 model. The accuracy obtained with the approach

where we fed heatmaps as the inputs to the second network was 82.66%. The accuracy with the approach where the second network was called into action only when the confidence of prediction was low for the first network gave an accuracy of 82.92% and the accuracy didn't change much when we incorporated the negative relevance. The accuracy dropped a bit when we tried looking at heatmaps corresponding to the class not predicted by the original network. The accuracy obtained with the filtered images was 85.26%, which was lesser than that obtained with just the original architecture without the heatmapping part, but higher than all the other approaches. The experiment was repeated multiple times and the same results were observed every time. Most of the misclassified images belonged to the not-centrifuge class, which was understandable as that class had higher variance among the images and hence required more bits/higher capacity for encoding. Another interesting observation was that all the images that were misclassified by the original architecture were misclassified by our entire pipeline as well. Hence, the heatmapping and filtering couldn't overturn any of the wrong decisions made by the first neural network.

5.3 Discussion

Talking about the approach where the heatmaps of the images were used to train the second network and predictions were made using that, the accuracy went down by a good amount. The idea in this approach was that given that the heatmap would highlight the parts of the image relevant to predicting that class, we could use those as characteristic features of a class, and when a test image comes in, its characteristic parts could be compared with the ones that the network has learnt, thus making classification better. That did not work because a heatmap can only explain why a network made a particular prediction and not overturn wrong predictions. Let's consider an example. Let's consider an image that has a cat and a flower vase in the background. Now, if the first network predicts that it's a centrifuge, then when we pass it through the second network, the vase would be highlighted more (when it is sent in with a label of 'centrifuge') than what the cat would be highlighted (when it is sent in with the label of 'not-centrifuge'). This is so because the heatmap is just explaining the network's predictions and given that the network predicted the original image to be a centrifuge, the characteristic features of that would be highlighted much more strongly (because that's what the network sees) than the features to say that the image was that of a cat. This explains why the accuracy would not go up. Talking about why it went down, a lot of information is lost when an image is converted to a heatmap, a simple example being the fact that it is converted from rgb to grayscale. Some of these play an important role in the classification, so sometimes the second network was overturning some of the correct decisions made by the first network, and that led to a decrease in the accuracy. Interestingly, not even a single wrong prediction made by the first network was being overturned.

The second approach where we look at the predictions made by the network trained on heatmaps only if the confidence of the first network was low lead to a slight improvement in

the accuracy, but that can not be attributed to the issue of the second network overturning the first one's right predictions having been tackled. We clearly observed a lot of cases where that was still happening, and the definition of what a 'low' confidence meant was a heuristic. It still suffered from all the issues that the first approach did and hence the accuracy was still relatively low as compared to the approach with just the original network. The approach where we explicitly included the negative relevance as a feature didn't work for a couple of reasons. Firstly, the network was already taking that into consideration while making a prediction. Second, to train for that particular feature, we fed the same images labelled as centrifuge once and as not-centrifuge the other time. The idea was this: if the image was originally that of a centrifuge, the heatmap would have regions highlighted in red corresponding to those pixels that help the network in making the prediction that it is a centrifuge, and so when we tell the network that it is not a centrifuge, it highlights the same regions in blue to say why it thinks it is not a 'not-centrifuge' image. So essentially we are just feeding two images to the second network for training, with the only difference being the color of the highlighted regions. This didn't help the network learn anything more than what it was learning already and hence there was no improvement seen in the accuracy.

There was a drop in the accuracy seen when we passed only the heatmap of the class that was not the one that was predicted by the first network to second network. This was because we are telling the network to look for regions that might show that it belongs to the class not predicted by the first network and in certain cases, it was able to do that effectively. Given that we didn't have another heatmap, corresponding to the class predicted by the first network, to compare with the confidences seen for the heatmap corresponding to the class not predicted by the first network, that again was just a heuristic and despite some tuning, it didn't yield any promising results. Talking about our approach to tackle the issue of having to generate as many heatmaps as there are classes for each test image, we observed that the confidences corresponding to the other classes was not very informative. As in, if we generate the heatmap for an image with an assumed label of class 0, and then pass that through the second network and look at the probabilities that it belongs to classes 1, 2, 3, and so on, those probabilities don't tell you anything about whether there's a need to look at the heatmaps with those classes assumed to be the correct one. More often than not, the probabilities were equally distributed on all the other classes, apart from the one assumed to be correct in that round. In other cases, given that we had the ground truth, we observed that there were several cases where the correct class had lower probability than most of the other classes and disregarding that would harm us. This made sense because we are looking at the pixels that contribute to the output being predicted as class 0, say (when the correct label is class 3, say). So, the network will highlight in red those pixels which it thinks are the reason why it is class 0, and these don't help us in anyway to predict that it is class 3; and will highlight those regions in blue which it thinks show that it doesn't belong to class 0, some irregularities, some features which are expected to be present were it class 0 but are not present, again, this doesn't say anything about it being class 3. There were cases where the correct class had higher probability and those were when the network was able to highlight

features not characteristic of class 0, but of class 3.

Finally, coming to the approach with the filtered images, we set out to improve the classification accuracy by filtering out the noise in images using heatmapping. We observed that the classification accuracy didn't increase, rather there was a slight decrease. This reiterates the data processing inequality, which states that information is never gained (usually lost) when transmitted through a noisy channel. The heatmaps, which are used for filtering, are obtained by passing the image through a neural network. This neural network determines which pixels are important and which ones are not for the prediction, and hence, by zeroing out the pixels that have very low relevance, we will never be able to increase the information available. The neural network already uses this information to make the prediction, hence it acts as a filter itself. But then, something should be different given that the 'filtered' images have lesser pixels to deal with. We realized that fewer bits were required to encode the information now, and hence an architecture with lesser capacity was sufficient. Leveraging the work on capacity scaling laws [5], we are now working on dynamic network optimization. The idea is to iteratively prune the network to obtain the network with the smallest capacity (weights) that can still classify the given images without much loss in the accuracy.

Chapter 6

Conclusion and Future Work

We tried several models to replicate the self-organizing behavior seen in the self-assembling wires experiment, where ball bearings placed in castor oil in a petridish under an electric field form emergent structures. We discussed our approaches like the simple agent, the double agent, the neighbor model, the potential model, the shooter agent, the shooter agent with the discretized grid etc. The shooter agent with the discretized grid approach was the one that resembled the strand formation and boundary formation phases the most. We then added phase 2 into the model as well, but I guess this is where some more work could be done. As of now, we heuristically move the ball bearings either right or left by flipping a coin while ensuring that no collisions happen and that the peer-to-peer communication or the chains are maintained. This doesn't necessarily lead to them covering the maximum area.

The other direction that one could look at is how to collate the multimedia information from different drones to increase the accuracy of object detection. Also, we did implement Y-shaped branches in our approach, but that was based on heuristics and not any laws of physics or fractals. It would be interesting to see what laws govern the formation of those structures and in such a case what happens if multiple branches want to send information back to the base station: how does one handle the case such that the messages don't get mixed up. I think the other interesting aspect to look at would be the 3d version of the problem: right now, we are looking at the petridish problem which is essentially 2d. But drones can be flying at different heights as well, and in that case, the peer-to-peer communication would be possible within a sphere around each drone. The other interesting question is whether a drone should only pass information through its chain: let's say the leader of a drone is close to some member of another chain and the path to the base station is shorter through that chain than its own chain; can the leader then transmit information using that or should that be prohibited as that might then involve having to figure out whether that chain itself is trying to send some information of its own and if so how to route the two messages etc. The accuracy of a drone's GPS location adds additional complexity to the problem, especially in cases where accurate objection detection is needed, and the errors might compound as the information travels through different drones.

We realized that simple adaptation rules were themselves doing a good job of making the ball bearings ‘intelligent’, in such a way that they could self-organize. Then, to understand how theoretically grounded adaptation rules would help other agents, such as neural networks, to collaborate and self-optimize, we looked first at heatmapping and explainable AI to understand how neural networks make predictions. We ran Layerwise Relevance Propagation (LRP) to identify the important pixels by running a backward pass in the neural network. We did some preliminary experiments on the widely used MNIST dataset and implemented LRP for both a multi-layer perceptron and a CNN (and a combination of the two). We clearly demonstrated the parts of the images that the network was looking at, which were mostly just the outlines of the digits, to make predictions. We also showed the use of negative relevance, wherein the network looked at some image and predicted that it does not belong to a particular class as it was missing some characteristic feature of that class. We then moved on to the actual task of interpreting a network’s decisions as to why certain images were classified as containing a centrifuge and others as not. We used a pretrained model for this task and characterised some features that the network looks for. We also tried to see if heatmapping could be used to correct labels in geotagging, but that was an unsuccessful attempt.

One case that is hard to distinguish is when characteristic features of both the classes exist in an image. Depending on what label is assigned at training time, the network is able to find parts of the image to justify that label with a very high confidence. The issue is when such images show up in the test set. This is something to think about in the future. The other problematic case was when certain images were very clear to the human eye as belonging to a particular class but the network classified it as some other class and highlighted some part of the image that didn’t seem like the characteristic feature of any class. That makes one wonder why did the network miss something so glaringly obvious to the human eye. The other direction one could explore is the geotagging task: to find a template heatmap for different types of landscapes and given a newer image, using heatmaps, figure out whether the image has the landscape mentioned in its labels or not.

Finally, we tried to explore the question of whether filtered images, obtained by blackening out those parts of the image that don’t contribute much to the final output as determined by the heatmaps obtained from one neural network, can help improve classification accuracy when passed through another neural network. We realized that a heatmap can only explain a decision made by the network and can not produce any new information in the process. As the data processing inequality states, no information is gained by post-processing. Hence, this didn’t lead to any increase in the accuracy. We have discussed a couple of different approaches that we tried and how the accuracy varied with these approaches. We are repeating the experiment for the CIFAR-10 dataset and expect similar results.

We realized in the process that though the classification accuracy can not be improved, the system could be made more efficient. Given that fewer bits are needed to encode the

newer images, an architecture with lesser capacity would be enough for the task now, and that leads to a lot of savings in cost, time etc. A future direction of work would be to figure out an ideal way to prune the network. Currently, works exist where people have tried to come up with the most optimal architecture for a given task and dataset by using techniques such as parameter sharing etc., but that involves brute force search over large solution spaces, guessing and checking in a massively parallel way. Given that we have an exact way to quantify how much each neuron contributes to the output, we can drop entire neurons or some of the weights associated with it. And this could happen in an iterative manner such that the network dynamically self-optimizes to reach an efficient architecture.

We finally conclude that the information content present in the observations/initial conditions dictates accuracy, not the amount of collaboration. Collaboration therefore does not improve accuracy once a single intelligence has enough capacity to tackle the problem. This is often seen in multimedia communities where projects report that accuracy did not increase by much, or in some cases didn't increase at all, when other modalities were incorporated. And that would be the case unless the new modality has some information complementary to the information already captured by the existing modalities. Just having more modalities does not guarantee an improvement in terms of accuracy. That does help make the system robust though, as collaboration can be used to distribute the computational load.

Bibliography

- [1] Binder A. et al. “Layer-Wise Relevance Propagation for Deep Neural Network Architectures”. In: *Information Science and Applications (ICISA) 2016 pp 913-922* (2016).
- [2] Gordo A. et al. “Deep image retrieval: Learning global representations for image search”. In: (2016).
- [3] Stephenson C., Lyon D., and Hübler A. “Topological properties of a self-assembled electrical network via ab initio calculation”. In: *Scientific Reports volume 7, Article number: 41621* (2017).
- [4] Arbabzadeh F. et al. “Identifying Individual Facial Expressions by Deconstructing a Neural Network”. In: *Pattern Recognition - 38th German Conference, GCPR 2016, Lecture Notes in Computer Science, 9796:344-54, Springer International Publishing* (2016).
- [5] Friedland G. and Krell M. “A Capacity Scaling Law for Artificial Neural Networks”. In: (2018).
- [6] Montavon G., Samek W., and Müller K.R. “Methods for Interpreting and Understanding Deep Neural Networks”. In: *Digital Signal Processing, 73:1-15* (2018).
- [7] Montavon G. et al. “Explaining NonLinear Classification Decisions with Deep Taylor Decomposition”. In: *Pattern Recognition, 65:211-222* (2017).
- [8] Potje G. et al. “Towards an efficient 3D model estimation methodology for aerial and ground images”. In: *Machine Vision and Applications, November 2017, Volume 28, Issue 8, pp 937-952* (2017).
- [9] Tolias G., Sire R., and Jégou H. “Particular object retrieval with integral max-pooling of CNN activations”. In: (2016).
- [10] Jun J.K. and Hübler A.H. “Formation and structure of ramified charge transportation networks in an electromechanical system”. In: *PNAS January 18, 2005 102 (3) 536-540;* (2005).
- [11] Simonyan K., Vedaldi A., and Zisserman A. “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”. In: *cs.CV* (2014).
- [12] Simonyan K. and Zisserman A. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: (2014).

- [13] Arras L. et al. "What is Relevant in a Text Document?": An Interpretable Machine Learning Approach". In: (2016).
- [14] Dueweke M., Dierker U., and Hübner A. "Self-assembling electrical connections based on the principle of minimum resistance". In: *Phys. Rev. E* 54, 496 (1996).
- [15] Marani M. et al. "Stationary self-organized fractal structures in an open, dissipative electrical system". In: *Journal of Physics A: Mathematical and General, Volume 31, Number 18* (1998).
- [16] Beaudry N. and Renner R. "An intuitive proof of the data processing inequality". In: *Quantum Information & Computation*, 12 (5-6): 432-441 (2012).
- [17] Garcia N. and Vogiatzis G. "Asymmetric Spatio-Temporal Embeddings for Large-Scale Image-to-Video Retrieval". In: (2018).
- [18] Araujo A. Noh H. et al. "Large-Scale Image Retrieval with Attentive Deep Local Features". In: *Proc. ICCV'17* (2017).
- [19] Bach S. et al. "On Pixel-wise Explanations for Non-Linear Classifier Decisions by Layer-wise Relevance Propagation". In: *PLOS ONE*, 10(7):e0130140 (2015).
- [20] Becker S. et al. "Interpreting and Explaining Deep Neural Networks for Classification of Audio Signals". In: (2018).
- [21] Lapuschkin S. et al. "Analyzing Classifiers: Fisher Vectors and Deep Neural Networks". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2912-20 (2016).
- [22] Weyand T., Kostrikov I., and Philbin J. "PlaNet - Photo Geolocation with Convolutional Neural Networks". In: *European Conference on Computer Vision (ECCV)* (2016).
- [23] Samek W. et al. "Evaluating the Visualization of What a Deep Neural Network has Learned". In: *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, 28(11):2660-2673 (2017).
- [24] LeCun Y. et al. "Gradient-Based Learning Applied to Document Recognition". In: *Proc. of the IEEE, Nov. 1998* (1998).