Faculty of Information Technology,
Monash University

Minor Thesis

Self-organising Neural Network Hierarchy

This thesis is presented in partial fulfillment of the requirements for the degree of Master of Data Science at Monash University

By :
Satyabrat Borgohain

Supervisors :
Levin Kuhlmann
Christoph Bergmeir
Gideon Kowadlo
David Rawlinson

October 2019

Abstract

Artificial neural networks (ANNs) are extremely powerful tools when it comes to capturing complex, non-linear patterns in data. They have performed remarkably well on tasks which are trivial for humans but have been very difficult for traditional computer programs to carry out. However, despite the success in their widespread application and adoption, their architectures have to be manually designed and configured a priori and often tailored for very specific purposes. Although it augments the ability of the network to perform very well in domain specific tasks, it also simultaneously limits the ability to solve problems across domains.

A solution to this problem is the automated creation of robust, novel architectures that would be task agnostic. Subfields within deep learning such as Neural Architecture Search (NAS) have emerged to address the same. However, this project aims to take a different approach towards automated discovery of optimal network architectures by using meta-learning to learn a network's own architecture. The primary goal of the project was to create a variant of an ANN, with self-organising and adaptive network architecture properties, that exhibited both gradiental and hierarchical features for visual modality extracted through a meta-learning rule.

To that end, we devised a meta-rule based on entropy of the latent representations of the input which optimizes the receptive field placement of a network within a feature map. This led to learning progressively better topological configurations and yield higher classification performance overall and subsequently gave rise to emergent, self-organising and hierarchical topologies. As an initial step, this project demonstrates a proof-of-concept and a potential alternative for automated neural architecture creation.

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the work of others has been acknowledged.

Satyabrat Borgohain.

Signed by : ....

Name : Satyabrat Borgohain
Date : 25/10/2019

Acknowledgements

I would like to offer my deepest gratitude and thanks to my thesis supervisors Dr Levin Kuhlmann, Dr Christoph Bergmeir, Dr Gideon Kowadlo and Dr David Rawlinson for their unwavering support and guidance throughout the course of the project, all the while providing me with encouragement to try out new ideas.

I am profoundly grateful to have conducted my research in a joint collaboration between Monash University and Project AGI and it would not have been possible without them.

Thank you.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Deep learning is currently a tour de force behind the state-of-the-art performance on many tasks such as vision and natural language understanding. The fundamental ideas behind it like the perceptron or an artificial neural network (ANN), which are in turn are loosely based on biological neurons, have been around for quite sometime. However, it was only during early this decade that it has gained massive traction, primarily driven by an exponential increase in computational power and abundance of data coupled with novel algorithmic innovations.

One of the critical areas in applied deep learning is the design and configuration of the neural network architectures, which are fixed prior to the training of the network. Currently, designing neural architectures involves hand-crafting them which is an extremely time consuming and iterative process that requires very specialized knowledge about neural architectures, the domain of the problem, and usually involves some form of iterative search, empirical discovery, intuition or trial and error. For e.g. In Convolutional Neural Networks, the researcher needs to specify many hyperparameters such as number of convolutional and pooling layers, filters, strides and so on.

To alleviate this and for designing more robust architectures in an automated manner, there has been a recent emergence of a sub-field known as Neural Architecture Search (NAS) within deep learning (Zoph & Le, 2016) . However, the existing NAS methods employ complex strategies and algorithms to search the candidate space of possible architectures and evaluate their performance for finding the optimal structures. This makes it computationally very expensive.

An alternate and perhaps a more novel approach would be to let the network learn its own architecture during training. This would also be more feasible from a biological standpoint, as there is evidence of functionally self-organising hierarchy between the different neocortical regions in mammalian brains (Kandel et al., 2000). The driving criterion for learning such network hierarchies and connectivities would then be a meta-learning rule (i.e. a secondary learning algorithm). Emulating the Gradiental model, proposed by E. Goldberg (1989), should lead to better resource allocation for learning abstract representations and integration of the sensory modalities and receptive fields.

This project aims at creating a variant of an ANN, with a self-organising, adaptive network architecture which exhibits both gradiental and hierarchical features for visual modality extracted through a meta-learning rule (Project AGI & The Whole Brain Architecture Initiative, 2018). If successful, an extension of such implementation could be seen as a step towards generalized architectures which could potentially perform well over multiple tasks.

For the scope of this project we would primarily evaluate on one sensory modality, i.e. visual modality, for finding a meta-learning based approach to automatically learn network architectures for useful representation of inputs. Future extensions of the project would potentially build and expand across different modalities and achieve better generalization across them.

1.2 Background and context

1.2.1 Neural Network Architecture

A simple neural network, for e.g a multi-layer perceptron consists of the following architectural parameters (Goodfellow, Bengio, & Courville, 2016)

1. Number of hidden layers (depth of the network)
2. Number of neurons per layer

However in case of more complex and task specific architectures, like CNNs for visual processing, more parameters like (Goodfellow et al., 2016)

1. Receptive field size
2. Stride
3. Convolutional layers
4. Pooling layers

have to be specified. They are also referred to as hyperparameters of the network (parameters not learned, but set prior to training the model). There are a large number of task specific architectures like Recurrent Neural Network (RNN) (Mikolov, Karafiát, Burget, Černockỳ, & Khudanpur, 2010) and Long short-term memory (LSTM) (Gers, Schmidhuber, & Cummins, 1999) for sequence modeling and language tasks which have their own hyperparameters.

The performance of a network hence largely depends on the initialization of these parameters before the training. Considering a fixed pool of resources for our network, like number of units per hidden layer, it essentially becomes an optimization problem of how well these resources can be allocated to learn the best latent representations of the input in each consecutive layer and then perform a supervised task like image classification with a competitive accuracy.

We would begin with just a single layer, containing fixed number of hidden units, that would loosely form regions with a fixed size receptive field. These component units would be a sub-network in itself, as we discuss in the later chapters.

1.2.2 Neocortex and the Gradiental Model

The core inspiration for the project comes from the fact that the mammalian neocortex is physically a thin two dimensional sheet. The neocortical connectivities between the different regions of this sheet then gives rise to a functional virtual hierarchy (Kandel et al., 2000). This is akin to stacking of multiple virtual layers to form a hierarchy which arise out of a single physical layer through recurrent connections between its regions (Project AGI & The Whole Brain Architecture Initiative, 2018). Subsequent layers in the hierarchy can then combine information encoded in the preceding areas to build increasingly complex representations of the input i.e. the lower levels would combine information to form high level abstractions of the input.

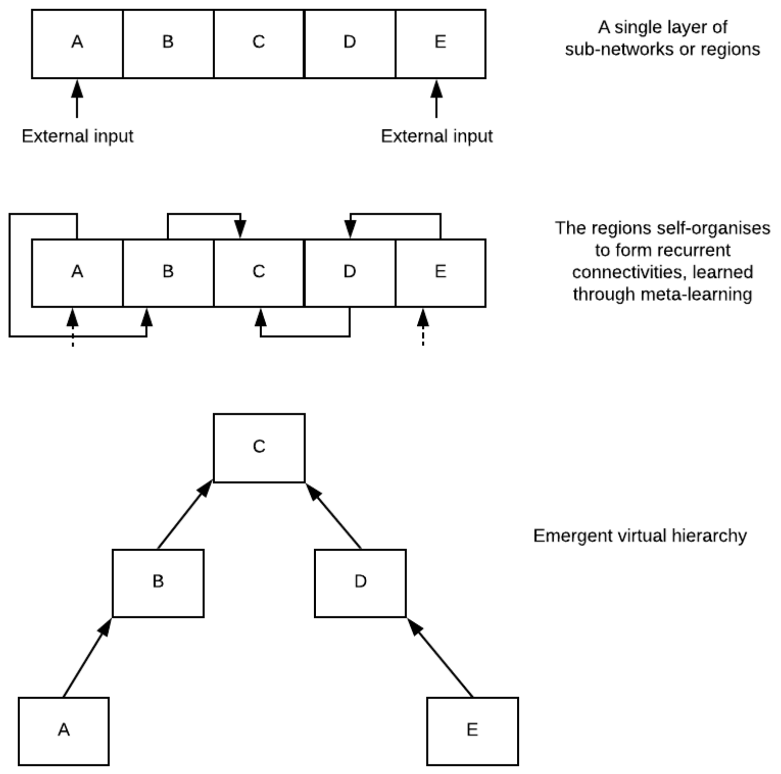

A
B
C
D
E
A single layer of sub-networks or regions
External input
External input
The regions self-organises to form recurrent connectivities, learned through meta-learning
Emergent virtual hierarchy

Figure 1.1: A simplified visual representation of the self-organising hierarchy. Adapted from Project AGI & The Whole Brain Architecture Initiative (2018).

E. Goldberg (1989) proposed the Gradiental Model according to which he implied that ”the functional neocortical organisation to a substantial degree is continuous, interactive and emergent, as opposed to mosiac, modular and prededicated” (p. 1). He also introduced the concept of a *cognitive gradient* which referred to gradual changes in encoded representations and integration between sensory modalities across the surface (E. Goldberg, 1989).

All these combined makes for an interesting case that if we could capture such flexibility in ANNs, it would not only be more biologically feasible but also impart a greater generalization power across a multitude of tasks. As the current state-of-the-art neural networks have been devised with very task specific objectives in mind, they perform extremely well in their domains. However, this can also be viewed as a fragmentation into specialized cases as opposed to being a more generalized problem solver.

1.2.3 Meta-learning

Meta-learning, loosely translated to ”learning to learn” (Vanschoren, 2018), in the context of this project would refer to minimizing the secondary loss function that would be used to learn the parameters for forming the recurrent connections of the virtual hierarchy, as depicted in Figure 1.1. This would run in conjunction with the base learning in the different sub-networks (i.e. the regions). To learn an effective architecture, the meta rule would have to make use of the underutilized resources within the sub-networks and redistribute them where required (Project AGI & The Whole Brain Architecture Initiative, 2018).

Meanwhile, the base learning would essentially minimize loss for learning the parameters (weights) of the sub-networks for an unsupervised or self-supervised task. Formulating a meta-learning objective function would be one of the core components of the project. Another key component would be parameterizing for the meta-learning. Possible candidate meta-parameters could be the *location* and *size* of the receptive fields. We can then relax other architectural constraints and try to parameterize them as well, which should lead to a much more adaptive and flexible architecture.

1.3 Research objectives

1.3.1 Research goals

As outlined above, the core aim of this project would be to demonstrate whether a self-organising architecture could be established with the help of meta-learning and if it optimizes the allocation and distribution of resources in a neural network. To that end, a successful algorithm design and implementation along with evaluation of its performance against fixed network architectures would be the primary goal of this project.

1.3.2 Research questions

The key research questions that would be the primary focus of this project could be broadly divided into the following (in order of exploration) :

1. Can we formulate a meta-learning rule for a self-organising hierarchy, emergent through recurrent connectivities between different regions, of a single layer neural network?

    (a) What types of objective functions and metrics could be used to evaluate the meta-learning?

    (b) What network parameters could be used to parameterize the meta objective function?

    (c) How much overhead would the meta-learning add on top of the base learning?

    Exploring potential meta-learning objective functions along with finding a metric which would be indicative of positive resource utilization and distribution in the network would be the key areas for discovery. As a starting point for the meta-learning objective function, Entropy or Free Energy Principle based schemes would be explored, along with receptive field size and its location for parameterizing the same (Project AGI & The Whole Brain Architecture Initiative, 2018). Also, to determine the excess overhead, code profiling could be done with and without the meta-learning.

2. How well does a single layer neural network with a self-organising architecture driven by meta-learning formulated above, perform when compared to a fixed network architecture given the same resources for image classification?

    To evaluate the performance between the two variants (fixed vs self-organising), they would both be used for a supervised task like image classification on MNIST (LeCun & Cortes, 2010) and would be compared based on their accuracy on the test set.

    Furthermore, building upon the results of the experiments we could extend our hypothesis in the future to investigate if the the Gradiental model proposed by E. Goldberg (1989) for the mammalian neocortex translates well algorithmically and if it leads to better optimization of resources over traditional fixed network architecture. As a stretch goal, we would be interested if such a self-organising neural network architecture generalizes well over different classification tasks i.e competitive performance on both MNIST and CIFAR-10.

1.4 Research methodology

The research methodology employed would be the *quantitative* experimental framework with evidence gathered in the form of performance indicators like accuracy and error measures to answer the key research questions and test the hypothesis. The approach would be broadly divided into the following two phases :

1. Unsupervised learning for feature encoding, and
2. Supervised Image Classification

They are discussed further in detail below.

1.4.1 Unsupervised learning for feature encoding

The preliminary phase of our approach would involve selecting and designing modular neural network units which would act as the sub-networks or sub-units (the regions) in the self-organising virtual hierarchy (Project AGI & The Whole Brain Architecture Initiative, 2018). These units can then be used for an unsupervised task, like learning useful features of an input image.

As a starting point, Autoencoders (AE) are a good choice to begin with since they are good for learning latent representations of the input in a lower dimensional feature space. The performance of an autoencoder can be easily evaluated with its reconstruction error i.e. a distance measure between the original input and the reconstructed output. Hence, the base learning would be driven by minimizing the loss function for the reconstruction error.

Briefly, the key components of an AE network are (Goodfellow et al., 2016) :

1. Encoder layer : Receives the external input
2. Hidden (or code) layer : Encodes the input into a reduced (undercomplete) or an extended (overcomplete) feature space
3. Code length : The number of units in the hidden layer
4. Decoder layer : Reconstructs the encoding to the same number of features as the input

We would initially constrain the code length, but later we could parameterize it for the meta learning or apply some heuristics which would adjust or scale itself depending on input dimensions. Similarly, receptive field size could also be fixed initially (Project AGI & The Whole Brain Architecture Initiative, 2018). Tied weights could be used for the encoder/decoder along with sparsity constraint and regularization for compactness and better generalization. A potential here is to explore different variants of autoencoders like Denoising AE, Variational AE (a generative model) and sparse AE to leverage their unique properties if required (Goodfellow et al., 2016). We could also introduce terms in the meta objective function for penalizing regions that are relatively far but have a high overlap of their receptive fields to encourage an even distribution of their placement in the input.

The goal here would be for the AE units to encode and integrate the most useful features of the input image. This would be accomplished with the self-organisation of the AE units into an effective hierarchy based on dynamic placement and size of their receptive fields learned via the meta-learning. As illustrated in Figure 1.2 below, different regions of the image are targeted by the autoencoder's receptive fields.

1.4.2 Supervised image classification

The next phase would be the evaluation of performance of the self-organising network, using a discriminative algorithm like logistic regression for image classification (Project AGI & The Whole Brain Architecture

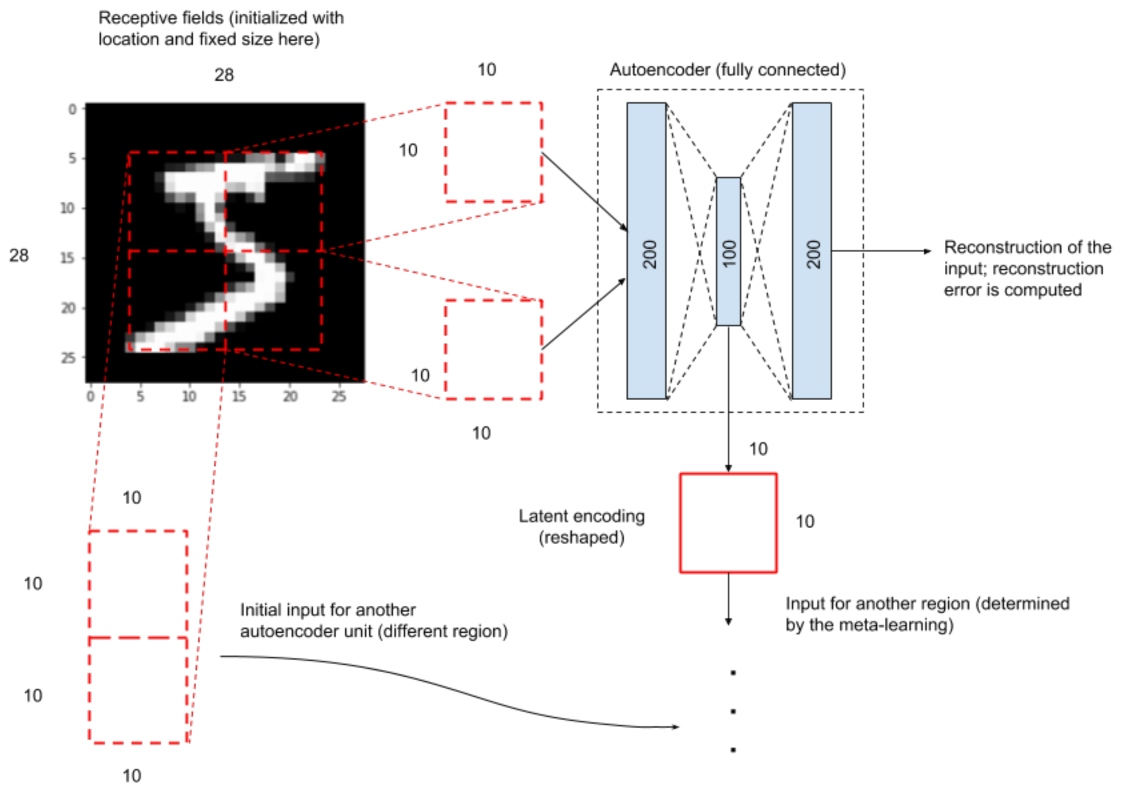

Receptive fields (initialized with location and fixed size here)
28
28
10
10
10
10
Autoencoder (fully connected)
200
100
200
Reconstruction of the input; reconstruction error is computed
10
10
Latent encoding (reshaped)
Input for another region (determined by the meta-learning)
10
10
10
10
Initial input for another autoencoder unit (different region)

Figure 1.2: Illustration of a single autoencoder unit and the proposed recurrent connections. Each AE unit would receive different inputs (a portion of the image) from each training example i.e. the spatial placement of the receptive field for the first level unit(s) would be different within a image (this could be randomly initialized for the first layer). Each unit would then be evaluated on its reconstruction error and its latent encoding would be the input for the next unit as shown.

Initiative, 2018). For obtaining the class probabilities in the final layer of the hierarchy, multinomial logistic (or softmax) regression could be used. This would essentially provide a measure of how well the receptive fields are integrated with each successive layer of the hierarchy, for learning lower to higher level features of the input image.

One key aspect of the project would be evaluation and comparison of performance between a fixed architecture network and a self-organising one. To setup an initial baseline, a few hand-crafted architectures with similar resources and hyperparameters, as the proposed self-organising architecture, would be evaluated for their performance on image classification (Project AGI & The Whole Brain Architecture Initiative, 2018).

Hence, one initial idea could be to arrange the modular AE units into a single layer (with no other layers or hierarchy beforehand), analogous to the single physical neocortical layer, and then allow the meta-learning algorithm to find the recurrent connectivities between them through the dynamic placement and size of receptive fields during training (giving rise to a hierarchy as a consequence) (Project AGI & The Whole Brain Architecture Initiative, 2018).

1.4.3 Experiment setup

For setting up a deep learning pipeline, the following options are considered :

Computing hardware

In terms of hardware, the following system specifications have been used successfully to train deep learning models using TensorFlow, PyTorch and Caffe. It should be able to provide sufficient compute capability for running the experiments.

Processor	Intel Core i7
GPU	Nvidia GeForce GTX 950M
RAM	8 GB

A standard autoencoder with two convolutional encoder/decoder layers and a batch size of 128 on MNIST (60000 training samples), took roughly 12-15 seconds per epoch to train with PyTorch with the above specifications. An alternative environment to train and evaluate the models would be on Google Colab which provides free cloud based GPUs for deep learning.

Deep learning framework

PyTorch would be used for the implementation as it is better suited for research and prototyping, and also provides dynamic computation graphs (Paszke et al., 2017). For the activation function, ReLU or leaky ReLU could be used. Other hyperparameters such as batch size, epochs, learning rate, regularization and sparsity constraints could be tweaked accordingly (Goodfellow et al., 2016).

With cuDNN and CUDA support, the GPU variant of PyTorch's data structures like CUDA tensors or the models provide better performance overall. Also TensorBoard could be used for visualization of loss and learning curves, as well as for viewing the reconstructed outputs.

1.4.4 Benchmark datasets

We would be training and evaluating our models on the MNIST dataset (LeCun & Cortes, 2010). Due to the simplicity of MNIST, the idea would be to start the implementation with the same and then later extend to CIFAR-10 (as a later extension or stretch goal). The benchmarks on these datasets are also readily available for performance comparison.

1.4.5 Evaluation metrics

For the base learning within the AE units, Mean Squared error (MSE) would be a good choice as the objective function to minimize the reconstruction error. A core challenge would then be to explore possible loss functions for the meta learning-rule (which is one of the key research questions). Entropy and other information theoretic measures along with energy based functions could also be potentially explored (Project AGI & The Whole Brain Architecture Initiative, 2018).

1.4.6 Potential roadblocks

Few of the issues anticipated are :

1. As this is relatively new approach, there is a lack of literature surrounding the area of research, with the closest areas being NAS, neocortical organization, CNNs and meta-learning (Project AGI & The Whole Brain Architecture Initiative, 2018)
2. Formulating the meta-learning rule
3. Figuring out ways to dynamically alter the receptive field size and its placement by the meta-learning while the base learning continues to operate in the background (Project AGI & The Whole Brain Architecture Initiative, 2018)
4. Preventing self-excitation cycles (output of an unit as its input in the next layer) in regions (Project AGI & The Whole Brain Architecture Initiative, 2018). An inhibitory constraint could be introduced that would discourage such recursive connections (taking inspiration from inhibitory inter-neurons in the cortex)
5. Optimization of the algorithm as the meta-learning has the potential to add a significant overhead
6. Insufficient computing resources

1.5 Expected outcomes and contributions

The expectation of this project is to showcase that self-organising neural architectures via meta-learning is possible and leads to better utilization of resources in a network. The exploration of whether meta-learning is feasible in ANN and if so, what kinds of objective functions, parameters and metrics of evaluation could be used is in itself a very interesting question. A successful design and implementation of the algorithm as a proof-of-concept could signal towards a promising new direction for automated crafting of deep neural network architectures (Project AGI & The Whole Brain Architecture Initiative, 2018).

The potential contribution of such research falls under the broader umbrella to democratize and make deep learning not just be accessible and usable by a handful few, who are well versed in neural networks and its architectures, but by everyone. Possibly, with more advanced self-organizing neural architectures, years of expertise and the time spent to tune neural networks for producing state-of-the-art results would hopefully no longer be required.

1.6 Thesis outline

The overall thesis is divided into the following chapters as per Figure 1.3 below.

In Chapter 2 we begin by evaluating and discussing the relevant literature in closely related fields and other works. In Chapter 3, we combine both research design and implementation together, where we discuss the different strategies and development details behind the overall experiment. In Chapter 4, we evaluate our experimental results and further discuss the outcomes of the project.

And finally in Chapter 5, we conclude our thesis by addressing the original research questions, highlighting the potential research contributions and the future scope of the project.

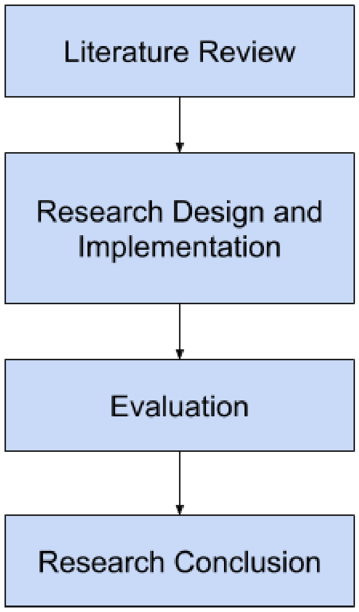


Figure 1.3: The overall thesis structure.

Chapter 2

Literature Review

2.1 Neocortical Organization

2.1.1 Neocortex

The fundamental ideas and numerous developments in deep learning have come from mimicking the human brain, we look at the mammalian neocortex and how it form complex representations and combines information.

It contains around 10 to 14 billion neurons among which are both excitatory neurons (80%) and inhibitory interneurons (20%). It consists of six layers numbered I-VI as follows, (Swenson & Gulledge, 2017)

1. Layer I : Molecular layer
2. Layer II : External Granualar layer
3. Layer III : External pyramidal layer
4. Layer IV : Internal granular layer
5. Layer V : Internal pyramidal layer
6. Layer VI : Multiform/Fusiform layer

2.1.2 Neocortical Hierarchy

The neocortex exhibits functional virtual hierarchy from a single thin two-dimensional physical sheet (Kandel et al., 2000). The connectivities between different neocortical regions functionally self-organize into a virtual hierarchy (as opposed to a physical hierarchy).

As we briefly discussed in Section 1.2.2, E. Goldberg (1989) proposed the Gradiental Model according to which he claimed that functional neocortical organisation is continuous, interactive and emergent (p. 1). He introduced the concept of a *cognitive gradient* which he defined as ”a continuous distribution of related functions along an axis defined at its extremes by a pair of sensory projection”. He implied that the encoded representations of the sensory inputs and the integration of different sensory modalities change gradually across the neocortical surface (E. Goldberg, 1989).

The basis of our sensory perception lies in the underlying neural architecture for each modality. There are cortical maps which contain continuous representations of sensory modalities associated with its corresponding sensory organ (Ranpura & Fitzgerald, 2019). For e.g.

- Auditory - Tonotopic map of frequencies
- Somatosensory (touch) - Somatotopic map of the body
- Vision - Retinotopic map of visual space

Features from different cortical maps then combine and give rise to complex receptive fields which incorporates information from multiple sensory modalities as a result (Ranpura & Fitzgerald, 2019).

2.1.3 Receptive Fields

Receptive field is a region of sensory space within which a stimulus could generate a reflex (Alonso & Chen, 2009). In terms of visual processing, a visual field would be the entire area a human eye can *see*, but a neuron's receptive field would be only a tiny portion in it from which it gets stimulated (Ranpura & Fitzgerald, 2019).

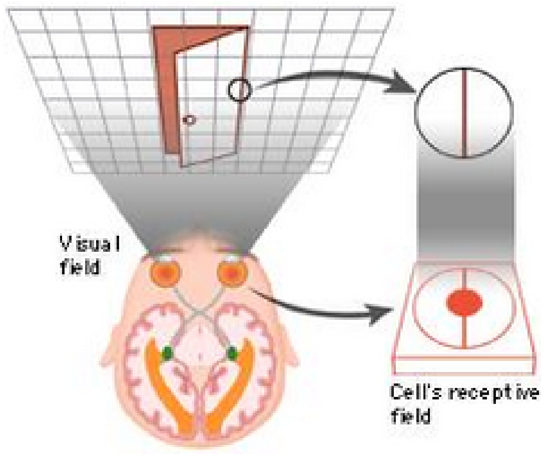


Figure 2.1: A schematic showing the receptive field of a neuron in the visual cortex. Here the neuron associated with this receptive field would be more and more *activated* as the edge of the door comes into the center of the field and vice versa. Adapted from Ranpura & Fitzgerald (2019).

The artificial neural network equivalent of this is quite analogous and describes the array of inputs (for e.g. pixels of an MNIST digit) an artificial neuron is receiving at any given time. For a deep neural network for e.g. a Convolutional Neural Network, receptive fields of neurons in lower layers (layers close to the input layer) can target a limited portion of the input image. As we move to high level layers of the network, the receptive field of these neurons are essentially a combination of receptive fields from the low level layers (LeCun et al., 1990).

As a result, subsequent layers in the network combine information encoded in the preceding layers to build increasingly abstract representations of the input which can then be used as features for a classifier. Receptive fields are one of the key components of this project, since their size and placement are the initial candidate parameters for the meta-learning algorithm (discussed further in the later sections) (Project AGI & The Whole Brain Architecture Initiative, 2018).

The emergent virtual hierarchy between the neocortical regions is *self-organizing* to facilitate the continuous allocation of resources for learning complex representations of the sensory inputs and integration of the receptive fields and modalities. Current neural network implementations in deep learning are based around fixed network architectures (their receptive fields, connectivities between them and their integration). In contrast, modeling the network architecture to be adaptive and evolving, by incorporating the flexibility and hierarchical self-organization exhibited by the neocortex should lead to better utilization of resources and well as generalization over different tasks.

Within the scope of this project, we would primarily focus on visual modality and tasks related to vision, specifically image classification.

2.2 Neural Network Architecture

Artificial neural networks (ANN) are often termed as *universal function approximators*, which can *approximate* any continuous, differentiable function (Nielsen, 2015). The ability of a network to derive this approximate function, from the observed data during training, is imparted in part to its architecture. Generalization becomes the key desirable property when it comes to ANN's ability to be applicable over new unseen data (assuming the data comes from similar distribution). In many cases for a neural network to perform well, feature pre-processing or extraction prior to (or during) training is required (LeCun et al., 1990).

2.2.1 Feedforward Architecture

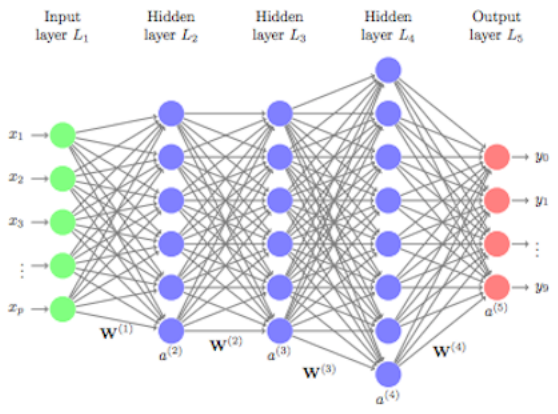


Figure 2.2: Illustration of a simple deep feedforward network architecture. The units are fully connected. *Feedforward* refers to the propagation of information in the forward direction. Adapted from Boehmke & Greenwell (2019).

A feedforward network architecture is the simplest setup of an ANN. They consists of the following elementary components :

1. Artificial neurons (units) : An artificial neuron is defined as a computation unit, whose activity is given by an affine transformation i.e. the weighted sum of its inputs, added with some bias and passed through a non-linearity like a Sigmoid or ReLU function. The activation via non-linearity allows the network to model complex non-linear relationships (Ruder, 2016).

$$Z = \sum_{i=1}^{n} x_i w_i = (x_1 w_1 + x_2 w_2 + x_3 w_3 + ..) + b = W^T x X + b \tag{2.1}$$

$$A = \sigma(Z) \tag{2.2}$$

$$\sigma(Z) = \frac{1}{1 - e^z} \tag{2.3}$$

where, Z is the weighted sum, A is the activation of Z and $\sigma(Z)$ is the sigmoid function (Goodfellow et al., 2016)

2. Hidden layers : The hidden layers of the network are all the intermediary layers between the input and output and contain the units.
3. Activation function : A non-linearity that every unit in a hidden layer adds to the weighted sum of all the inputs from the preceding layer.
4. Loss Function : The objective function which is to be minimized for finding the optimal parameters $(w_1, w_2, ..)$ of the network.
5. Optimizer : A optimization method to minimize the loss function and by finding the optimum parameters of the network . E.g. Stochastic Gradient Descent (SGD), RMSProp and Adam.

We consider a feedforward architecture, with a fixed number of resources such as number of hidden units, number of layers and receptive field size available for our network in order to estimate how the network performs with an adaptive architecture over a fixed architecture. The usage and allocation of these fixed resources would have to be optimized in a way, for learning relevant features of the input.

Also as in the case of CNNs, LeCun et al. (1990) demonstrated that designing network architectures with apriori knowledge about the problem domain and incorporating the geometric and topological properties of the input features has been critical for achieving greater performance. However this could make it too restrictive to a specific tasks and the architecture might not work as well for other tasks.

2.3 Autoencoders

In an unsupervised setting, an implicit secondary objective is to often learn useful *representations* of the input data, which could then be used as features, for e.g. in supervised classification. This should lead to better more meaningful representations and encoding of the input feature space and improve performance of the classifier.

The unsupervised model in our case would be an Autoencoder, where an input image could be compressed and encoded via the encoder-decoder architecture. The *distance* between the raw, non-encoded input and the decoded output could be computed as the reconstruction loss. It is somewhat indicative of how well a high dimensional input, such as an array of image pixels, is encoded into a low dimensional space (discussed further below) (Hinton & Salakhutdinov, 2006).

The objective of an Autoencoder is to encode the input, given the constraint of a *bottleneck* through which the input is forced through. The architecture of an Autoencoder is generally setup in a way that it inhibits its ability to learn the identity function (in which case $x = \hat{x}$), as a consequence of which encoding takes place (Goodfellow et al., 2016).

2.3.1 Architecture

Autoencoders have feedforward network architecture. A simple, shallow AE consists of (Goodfellow et al., 2016):

1. **Encoder** : Input (x), such as raw pixels of an image, is provided to this layer; encoder function $h = f(x)$

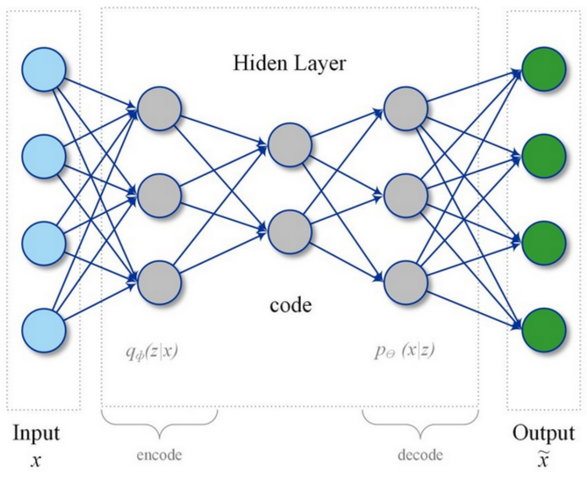


Figure 2.3: A Simple Autoencoder with two encoder and decoder layers. Adapated from Mohammadi et al. (2018).

2. **Hidden (code) Layer** : The *bottleneck* layer, where the number of units are lower (or higher, incase of overcomplete AE) than the encoder layer.
3. **Decoder** : The output layer that reconstructs the input using the encoding. A decoder function, $r = g(h)$

Stacked, deep AEs are an extension of this architecture wherein, there are several encoder and decoder layers stacked in succession.

The bottleneck forces the network to prioritize features which are the most representational of the input and encodes the same. This leads to an interesting property of feature representation in a low dimensional space with prioritized information content. We define a sub-network as the Autoencoder (AE) units with fixed sized receptive fields and initialize their spatial placement on the input image

From the high level architectural perspective The meta-learning algorithm should ideally modify the receptive fields of the Autoencoder units, to learn the best possible representations of the input training examples.

2.3.2 Loss Function

The loss function can be expressed as,

$$L(x, g(f(x))) \tag{2.4}$$

where L is the loss function that tries to minimize the error between the input x and its reconstruction $g(f(x))$. The following loss functions are generally used as the distance measure between input and output for Autoencoders :

1. Mean Squared Error (MSE) : Expressed as the mean of the squared distance between the estimated and actual values.

$$\frac{1}{n}\sum_{i=1}^{n}(\hat{y_i} - y_i)^2 \tag{2.5}$$

2. Binary Cross-entropy (BCE)(Goodfellow et al., 2016) : For $x \in$ [0,1],

$$-(y\log(p) + (1-y)\log(1-p)) \tag{2.6}$$

where, $y =$Actual label (ground truth), p = Predicted label

2.3.3 Types of Autoencoders

Here we briefly evaluate different variants of Autoencoders, for possible exploitation of their properties

1. Sparse Autoencoders : Sparse AE involve a regularization term, the sparsity penalty, on the hidden layer to restrict the activities of units to only very specific features of the training examples (Makhzani & Frey, 2013).
2. Denoising Autoencoders These AE try to undo corrupted or noisy training examples (Vincent, Larochelle, Bengio, & Manzagol, 2008).
3. Variational Autoencoders (VAE) These are generative models which try to learn a latent variable distribution (Kingma & Welling, 2013).

2.4 Meta-learning

The approaches for Meta-Learning, often described as *learning to learn*, first appeared during the 80s and 90s (Schmidhuber 2007; Bengio et al. 1990). The terminology has been used in a variety of context in literature (Vilalta & Drissi, 2002). Meta-learning in its current state is a type of learning which is adaptive to new tasks and environment without explicit training from the scratch (Finn, Abbeel, & Levine, 2017).

Bengio et al. (1990) proposed *learning* (more appropriately, finding) a learning rule through a synaptic modification rule, and estimating its parameters with gradient descent. This approach involves training a meta-learner which learns a parameter update rule of the learner. Similar approaches were applied to learn a parameter update rule (Hochreiter et al., 2001).

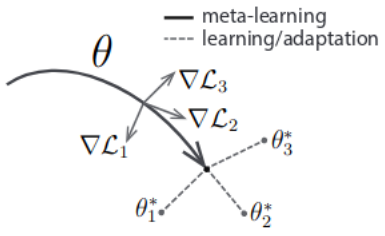


Figure 2.4: Schematic from MAML. Here θ is a representation, that is optimized for adapting to new tasks. We see the gradients and their directions along with new values of θ. Adapted from Finn et al. (2017).

Finn et al. (2017) proposed Model-Agnostic Meta-Learning (MAML), which provides good initializations for achieving few-shot learning i.e. based only on a few training samples. The model agnostic nature of the approach makes it applicable to a wide variety of existing architectures. In contrast to the work by Metz et al. (2019), MAML does not directly learn an update rule, but learns good parameter initializations W_init that leads to better generalization of tasks from similar domain very quickly into training. Another approach learns both optimizer as well as weight initialization (Ravi & Larochelle, 2016).

Meta-learning is done on a meta-training set which comprises of different tasks. An analogy would be to consider every task specific dataset as individual samples or training examples in the meta-training set (Finn et al., 2017). The base learner for e.g. would perform a supervised task like image classification and the meta-learner would in turn optimize and train the base learner (Finn et al., 2017). The learner model learns representations across all the tasks it was trained on and uses this prior knowledge or *meta-knowledge* to fine tune when it encounters a new task (Finn et al., 2017).

In the context of our project,

- The base or primary learning (f_θ) - Learning of base parameters i.e. the autoencoder feedforward weights using an optimizer.
- The meta or secondary learning (f_ϕ) - Learning of parameters that define the receptive field's spatial placement and size for best feature encoding in a lower dimensional space.

2.4.1 Types of Meta-Learning

The current approaches to meta-learning attempt to train meta-learning models on a large number of tasks and then expose the model to new tasks for evaluating how well they perform ((Finn et al., 2017); (Weng, n.d.)) :

1. **Metric Based Meta-learning** : The Metric Based approach involves learning an efficient distance or metric function over objects (Snell et al., 2017). This approach has been mainly used for few-shot classification such as in Convolutional Siamese Neural Network (Koch et al., 2015).
2. **Optimization based Meta-learning** : In this approach (Hochreiter et al. 2001; Snell et al. 2017) a meta-learner model learns an optimizer which is then used to train a learner neural network, to make it more efficient in performing the target task. Long short term memory (LSTM) networks are used as the meta-learners by Snell et al. (2017), since it can retain the *history* of the gradient updates in its hidden state (memory).
3. **Model based Meta-learning**: This approach involves designing a model for rapid parameter update with minimal training steps (Santoro et al., 2016)

A very closely related area of work was done by Metz et al. (2019), in which the authors propose a way to meta-learn an unsupervised learning (weight update) rule. They argued that the meta-learned algorithm produces representations useful for tasks performed subsequently such as supervised classification. They also did an extensive comparison between the past and current approaches to meta-learning (table below)

They considered a Multi-Layer Perceptron (MLP) as the base model with parameters θ which is trained by the inner loop of the meta-learning process via the learned weight update rule.

Metz et al. (2019) echoed a key idea of this project, that the quality and the utility of learned representations via unsupervised learning could also be evaluated by its performance on candidates tasks. Here we instead propose to meta-learn the architecture of a parent network composing of the aforementioned sub-networks of AEs and extract useful, latent representations of the input image.

Another crucial area during the meta-learning rule would be for optimizing the sub-network's usage of resources by redistributing them from the overutilized areas to the underutilized ones. Some useful metrics (discussed in a later section) would be explored through which we could directly (and indirectly) infer how *good* the learned representations are.

Meta-learning in this project would consist of *optimizing a secondary loss function* and learning the secondary or meta-parameters that would define an optimal topology of the network i.e. the recurrent connections of the emergent virtual hierarchy. The meta or secondary objective would be to learn parameters for e.g. the receptive field size and placement which would lead to better hierarchical feature extraction and encoding via an optimal network architecture during training, without any fixed apriori structure. The

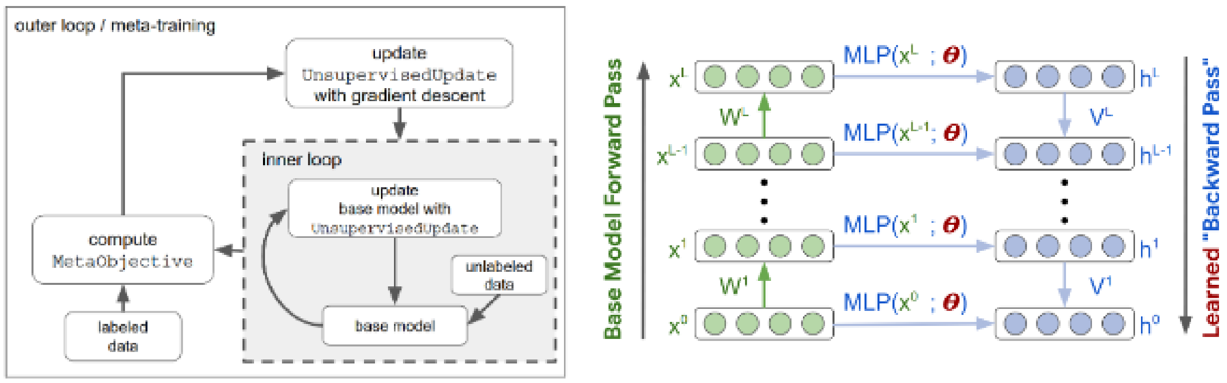


Figure 2.5: The method proposed by them consists of two loops, an outer and inner loop. Inside the inner loop an UnsupervisedUpdate is iteratively applied to the base learner or model. The UnsupervisedUpdated (parameterized by θ) is updated by SGD on the MetaObjective, during meta-training. Adapted from Metz et al. (2019).

network architecture could also be initialized for successive tasks based on MAML-like approach for rapid convergence (Finn et al., 2017).

2.5 Neural Architecture Search

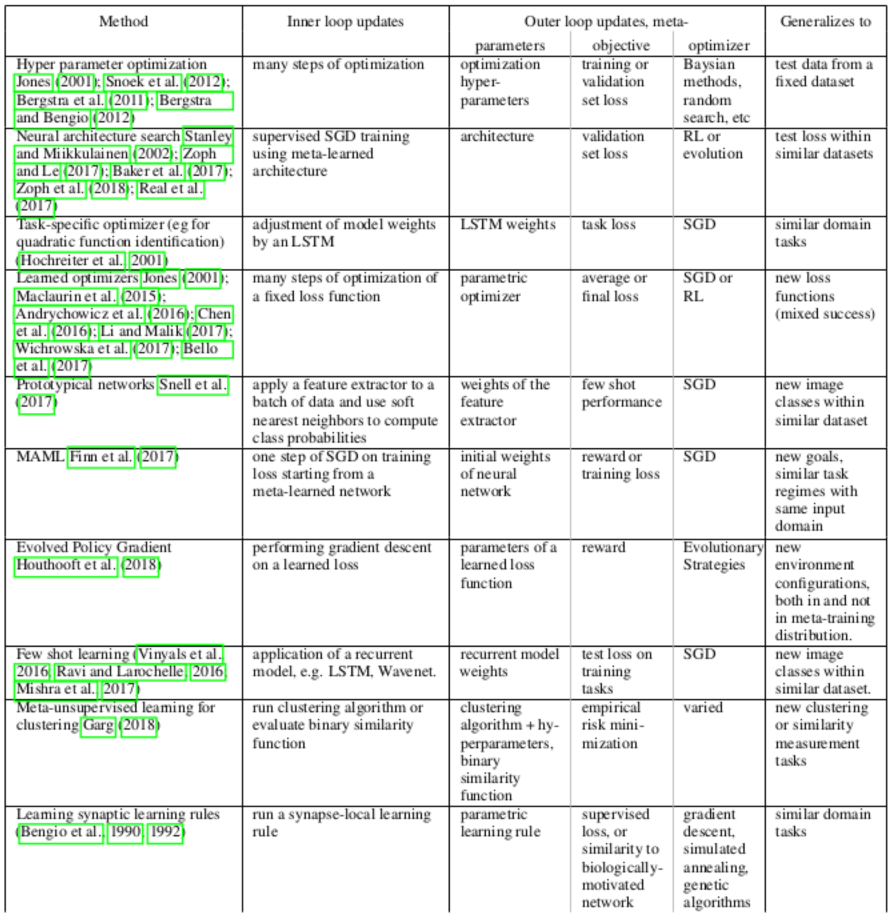
A emerging subfield collectively called *Neural Architecture Search* (NAS) is focused on automating the architecture engineering phase and finding optimal network architectures within a candidate search space of all possible architectures (bounded or unbounded) (Zoph & Le, 2016). NAS devised architectures have outperformed manual, hand-crafted architectures in several areas such as semantic segmentation, image classification and object detection (Elsken, Metzen, & Hutter, 2018). There are different strategies employed by different types of NAS algorithms in order to search and evaluate the architectures on training examples.

As Elsken et al. (2018) concisely sums up the different attributes of a typical NAS algorithm as follows :

- **Search space** : It involves pre-defining a candidate search space, by incorporating prior knowledge about the properties of architectures that would perform well for a given task.
- **Search strategy** : It involves exploration of the aforementioned search space for viable architectures. The search space itself could be exponentially large and hence a trade-off between exploration-vs-exploitation comes into play.
- **Architecture performance evaluation strategy** : It involves strategies for estimation of the feasible candidate architectures discovered in the search space instead of training $\rightarrow$ validation $\rightarrow$ evaluation of every possible candidate architecture on the training set.

2.5.1 Search Space

A representation of the search space is as structured chain networks with each node representing hidden layer of a neural network.



Method	Inner loop updates	Outer loop updates, meta-			Generalizes to
		parameters	objective	optimizer	
Hyper parameter optimization Jones (2001); Snoek et al. (2012); Bergstra et al. (2011); Bergstra and Bengio (2012)	many steps of optimization	optimization hyper-parameters	training or validation set loss	Baysian methods, random search, etc	test data from a fixed dataset
Neural architecture search Stanley and Miikkulainen (2002); Zoph and Le (2017); Baker et al. (2017); Zoph et al. (2018); Real et al. (2017)	supervised SGD training using meta-learned architecture	architecture	validation set loss	RL or evolution	test loss within similar datasets
Task-specific optimizer (eg for quadratic function identification) (Hochreiter et al., 2001)	adjustment of model weights by an LSTM	LSTM weights	task loss	SGD	similar domain tasks
Learned optimizers Jones (2001); Maclaurin et al. (2015); Andrychowicz et al. (2016); Chen et al. (2016); Li and Malik (2017); Wichrowska et al. (2017); Bello et al. (2017)	many steps of optimization of a fixed loss function	parametric optimizer	average or final loss	SGD or RL	new loss functions (mixed success)
Prototypical networks Snell et al. (2017)	apply a feature extractor to a batch of data and use soft nearest neighbors to compute class probabilities	weights of the feature extractor	few shot performance	SGD	new image classes within similar dataset
MAML Finn et al. (2017)	one step of SGD on training loss starting from a meta-learned network	initial weights of neural network	reward or training loss	SGD	new goals, similar task regimes with same input domain
Evolved Policy Gradient Houthooft et al. (2018)	performing gradient descent on a learned loss	parameters of a learned loss function	reward	Evolutionary Strategies	new environment configurations, both in and not in meta-training distribution.
Few shot learning (Vinyals et al., 2016; Ravi and Larochelle, 2016; Mishra et al., 2017)	application of a recurrent model, e.g. LSTM, Wavenet.	recurrent model weights	test loss on training tasks	SGD	new image classes within similar dataset.
Meta-unsupervised learning for clustering Garg (2018)	run clustering algorithm or evaluate binary similarity function	clustering algorithm + hy-perparameters, binary similarity function	empirical risk mini-mization	varied	new clustering or similarity measurement tasks
Learning synaptic learning rules (Bengio et al., 1990, 1992)	run a synapse-local learning rule	parametric learning rule	supervised loss, or similarity to biologically-motivated network	gradient descent, simulated annealing, genetic algorithms	similar domain tasks

Figure 2.6: Meta-learning approaches. Adapted from Metz et al. (2019).

The parameters for the search space are generally (Elsken et al., 2018):

- Number of layers possible (fixed or unbounded)
- Type of operation at each layer (convolutions or max pooling for e.g)
- Hyperparameters (related to the operation) such as number of hidden units incase of fully connected networks, number of filters, strides or kernel size.

Another idea proposed was searching for recurrent motifs or blocks instead of searching for the entire architecture (Zoph et al., 2018). These strategies like recurring motifs of the architecture for e.g. could be implemented in the scope of this project (exploring it as an architecture search problem) to potentially reduce the number of iterations for meta-learning to converge.

2.5.2 Search Strategy

A host of different strategies are employed for the architecture search such as Bayesian optimization based (Eggensperger et al., 2013), evolutionary algorithms based, reinforcement learning based gradient-based methods and random search (Elsken et al., 2018).

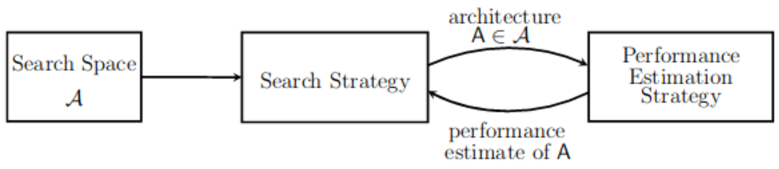

Search Space $\mathcal{A}$
Search Strategy
architecture $A \in \mathcal{A}$
Performance Estimation Strategy
performance estimate of A

Figure 2.7: Diagram depicting different components of NAS methods. Adapted from Elsken et al. (2018).

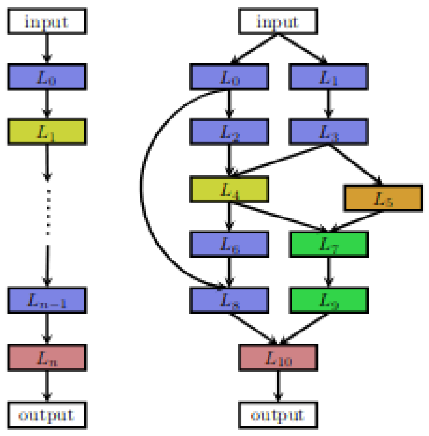


Figure 2.8: Different architecture search space. Every node represent a type of layer and the edges represent the output of layers as input for another. Adapted from Elsken et al. (2018).

Reinforcement Learning based

Zoph & Le (2016) showed a RNN-based approach for generating model descriptions or the architectural hyperparameters (expressed as a variable length string) and then training a reinforcement learning (RL) algorithm to iteratively discover or sample architectures that yield better accuracies on the validation set.

In their work, the RNN acted as the *controller* and sampled a variable length string which encodes all the information about an architecture. This candidate architecture is then trained on the validation set to evaluate its accuracy which is then used as a reward signal that updates the parameters of the controller via RL. The controller assigns high probabilities to architectures that would give better accuracies and as a consequence improves its search for viable architectures over time (Zoph & Le, 2016).

The benchmarks of the networks discovered by this approach were : For image recognition, ConvNet model achieving 3.65 test set error (1.05x faster) on CIFAR-10. For language modeling, Recurrent architecture achieving 62.4 test set perplexit on Penn Treebank at the time. There has been much subsequent work building incrementally on the idea of RL-based architecture search (Zoph & Le, 2016).

Evolutionary Algorithms based

An alternative to RL-based methods as a search strategy is to use evolutionary (genetic) algorithms for evolving network topologies. The evolutionary algorithms are heavily based on biological evolution. Neuroevolution (NE) led automated architecture discovery has shown great promise in this regard and seminal work was done with NEAT (Evolving Neural Networks through Augmenting Topologies) (Stanley & Mi-

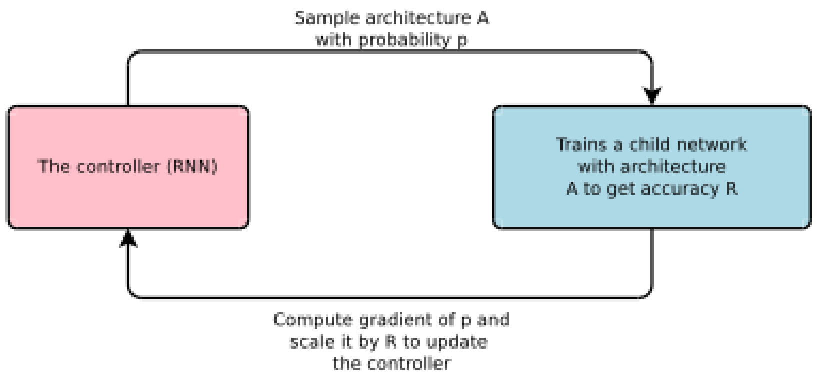

Sample architecture A with probability p
The controller (RNN)
Trains a child network with architecture A to get accuracy R
Compute gradient of p and scale it by R to update the controller

Figure 2.9: Schematic showing controller based NAS approach. Adapted from Zoph & Le (2016).

ikkulainen, 2002). Early work on neuroevolution was done to evolve a fixed network topology to optimize the network weights (Stanley & Miikkulainen, 2002). The central question addressed by NEAT of whether evolving topology of a network lead to better performance or provide an advantage over fixed-topology network somewhat mirrors our project (Stanley & Miikkulainen, 2002).

The authors used direct encodings and an additive approach to their evolutionary framework wherein they initialized it with minimal topological network structure. The population is split into species (based on similarity between their topology) that restricts organisms to only compete with organisms in their own species so that more topologies could be explored before they would be eliminated as a result of the evolutionary process. They also used the idea of *explicit fitness sharing* i.e. sharing of fitness between organisms in the same species, to prevent any one species from dominating the population (D. E. Goldberg et al., 1987).

Recent work by Liu et al. (2017) also used evolutionary algorithms and introduced hierarchical representations for expressing neural networks and performed evolutionary search on the representations as genotypes (genetic representation of an organism). They also focused on the importance of construction of search space, claiming that architectures discovered via random search in such a space give competitive performance on CIFAR-10 (Liu et al., 2017).

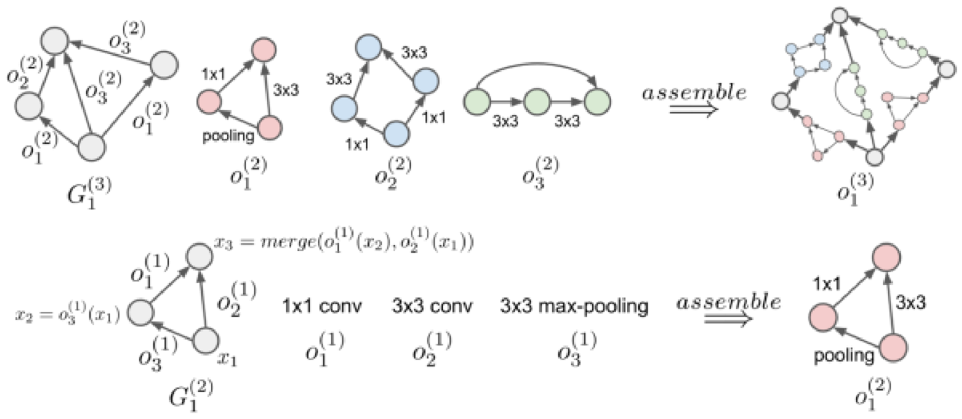


Figure 2.10: Diagram of a 3-level hierarchical topology. Low level primitives form higher level motifs. Adapted from Liu et al. (2017).

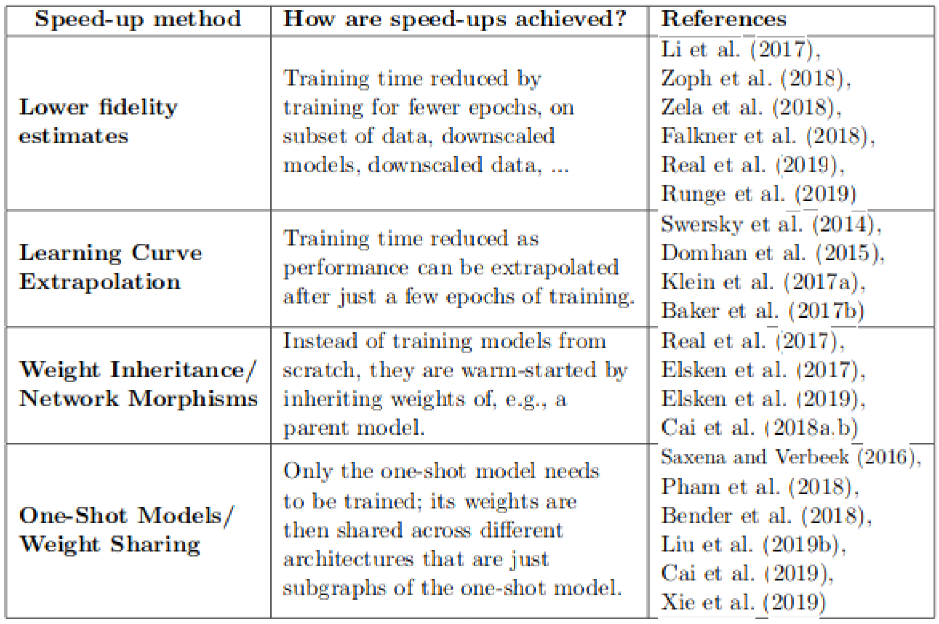
The scope of this project does not directly include the usage of any evolutionary algorithm. However, the one recurring characteristic that could be used is propagating the *fitness* i.e. better quality topographical features, across similar topographies using a metric analogous to the fitness metric. Such properties should be assigned higher probability in subsequent iterations.

2.5.3 Architecture Performance Estimation Strategy

The naive performance estimation of the architectures discovered through NAS would be to train on a dataset and test on validation set for accuracy (incase of a classifier). However this has proved to be computationally restrictive (Zoph & Le 2016; Real et al. 2017; Zoph et al. 2018; Real et al. 2019). Hence, rapid performance estimation strategies were devised to reduce the time for evaluating each candidate architecture.

Few of the strategies for performance estimation are :

- Estimation done on *lower fidelities* which provides a *approximate* metric for performance of the architecture. Includes reduced training time, downsampled training images or training on a subset of data). Potentially introduces bias in the estimate (Elsken et al., 2018).
- Learning curve extrapolation in which initial learning curves on the candidate architectures are extrapolated to decide whether to terminate or not depending on their predicted performance (Rawal & Miikkulainen, 2018).
- Training a surrogate model (Liu, Simonyan, Vinyals, Fernando, & Kavukcuoglu, 2017)
- *Network morphisms* which allows for architectures to be modified without affecting the function approximated by the network. Based on the weights of other previously trained architectures, weights of new novel architectures are initialized (Wei, Wang, Rui, & Chen, 2016).



Speed-up method	**How are speed-ups achieved?**	**References**
Lower fidelity estimates	Training time reduced by training for fewer epochs, on subset of data, downscaled models, downscaled data, ...	Li et al. (2017), Zoph et al. (2018), Zela et al. (2018), Falkner et al. (2018), Real et al. (2019), Runge et al. (2019)
Learning Curve Extrapolation	Training time reduced as performance can be extrapolated after just a few epochs of training.	Swersky et al. (2014), Domhan et al. (2015), Klein et al. (2017a), Baker et al. (2017b)
Weight Inheritance/ Network Morphisms	Instead of training models from scratch, they are warm-started by inheriting weights of, e.g., a parent model.	Real et al. (2017), Elsken et al. (2017), Elsken et al. (2019), Cai et al. (2018a,b)
One-Shot Models/ Weight Sharing	Only the one-shot model needs to be trained; its weights are then shared across different architectures that are just subgraphs of the one-shot model.	Saxena and Verbeek (2016), Pham et al. (2018), Bender et al. (2018), Liu et al. (2019b), Cai et al. (2019), Xie et al. (2019)

Figure 2.11: Tabular overview of different performance estimation strategies. Adapted from Elsken et al. (2018).

In evaluating the performance of the meta-learned, emergent architecture, the *learning curve extrapolation* as well as estimation on *lower fidelities* for rapid performance estimation could be employed. In particular, if the meta-learning update rule adds a significant processing overhead to the training process. Overall, NAS provides a great wealth of resources and strategies to optimize the searching phase for discovery of novel neural network architecture which could be implemented.

2.6 Metrics for Evaluation

In order to evaluate the learned representations via unsupervised (or self-supervised) learning and the performance of the emergent architecture via the meta-learning process, we divide and explore two different types of evaluations - intrinsic and extrinsic metrics, below.

2.6.1 Intrinsic Metrics

We define intrinsic metrics for our task, as measuring the quality of the learned encodings and through which we can directly infer about the fitness of the network. Quality of encodings would be proportional to the content of relevant information to solve a given problem (Tishby, Pereira, & Bialek, 2000). This would ideally be correlated with the accuracy which we derive later from training the encodings on the supervised system. We approached the intrinsic metrics primarily from an information theoretic point of view below.

Tishby & Zaslavsky (2015) used information theoretic concepts to highlight the trade-off between *compression* and *prediction* in neural networks. Our problem can be formulated as effective compression of the input training examples in the unsupervised phase and prediction in the supervised phase.

Entropy

Shannon entropy is defined as the uncertainty in an entire probability distribution (or the average rate of information) which can be quantified mathematically as follows Goodfellow et al. (2016) :

$$H(x) = E_{x \sim P}[I(x)] \tag{2.7}$$

where, x is a discrete random variable and $I(x)$ is the self information given by

$$I(x) = -P\ (log(x)) \tag{2.8}$$

Alternatively,

$$H(x) = -\sum_i P_i\ log(P_i) \tag{2.9}$$

Here the choice of the base for the log determines if the computed entropy is in *bits* (base 2) or *nats* (natural log). This formulation for Shannon entropy above was described for discrete random variables, however when extended to continuous random variables it is called *continuous or differential entropy* (Goodfellow et al., 2016).

We discuss three different types of entropy estimators as follows :

1. **Histogram (or binning) based entropy estimation** : For a given continuous variable c, we can discretize the same into a discrete variable D, by computing a histogram and binning all the values. This allows for a rough estimation of the probability distribution of c which could be used in turn to calculate the entropy (Shwartz-Ziv & Tishby, 2017).

As the encodings or activations of the hidden layers in a neural network are continuous, the discrete representation of the activations could serve as a rough approximation for the same. One caveat to this approach as the authors noted, is that the estimated entropy is sensitive to the choice of binning as it yields different discrete representations for c. However, it has been shown that entropy estimated using a binning-based approach falls within the theoretical limits for small and large bins (Purwani, Nahar, & Twining, 2017). For the purposes of our experiments, we would restrict the bins of the computed histogram to a constant number of evenly spaced bins, irrespective of the topology.

2. **Non-parametric Kernel Density Estimator (KDE)** : Kolchinsky & Tracey (2017) provided an approach to estimating entropy of a mixture which gives both the upper and lower bound estimate. It provides a very good continuous entropy estimation with Chernoff-α-divergence (or Bhattacharyya distance) taken as a distance metric for lower bound and the KL-divergence for the upper bound Kolchinsky & Tracey (2017).

3. **K-Nearest Neighbour (KNN) based Kraskov Estimator** : Another state-of-the-art non-parametric entropy estimator is the Kraskov estimator (Kraskov, Stögbauer, & Grassberger, 2004) where entropy of a continuous random variable is estimated by using nearest neighbor distances between samples, which in our case would be the hidden layer activity of the network (Saxe et al., 2018). Mathematically it is given by,

$$\frac{d}{P}\sum_{i=1}^{P} log(r_i+\epsilon)+\frac{d}{2}log(\pi)log\Gamma(d/2+1)+\psi(P)-\psi(k) \tag{2.10}$$

where d is the dimension of the hidden representation, P is the number of samples, r_i is the distance to the k^{th} nearest neighbor of sample i, Γ is the Gamma function, ψ is the digamma function and ϵ is a small constant (Saxe et al., 2018).

Information Bottleneck

A quantitative approach to what consists as *relevant information* was studied by Tishby et al. (2000) where they formulated a variational principle for representation of relevant information. They argued that although rate distortion theory provided distortion functions for *lossy compression*, it however constrained the discovery of relevant features by the choice of the distortion function itself. Tishby et al. (2000) proposed the Information Bottleneck (IB) theory for deep neural networks for an attempt to explain the representational aspects of the hidden network layers.

A *minimal sufficient statistic* t is defined as the relevant information about X with respect to Y and maps the *mutual information*, $I(X;Y)$ (Tishby et al., 2000).

They introduced a Langrange multiplier, with the mutual information term, for finding an effective trade-off between compressing and encoding the representations and simultaneously preserving the relevant information. For joint probability distribution $p(x,y)$ and $(X,Y)\sim p(x,y)$, minimizing the following :

$$L[p(t|x)] = I(X;T) - \beta I(Y;T) \tag{2.11}$$

where, T is the Information Bottleneck, yields $I(T;X)$ being *minimized* while the term $I(T;Y)$, representing sufficiency and accuracy, being *maximized* ($T \leftrightarrow X \leftrightarrow Y$ forms a Markov Chain). With an optimal, finite value of β most of the relevant and meaningful information could be encoded (Tishby & Zaslavsky, 2015). One good feature about the IB theory is that it works for both supervised as well as unsupervised (Tishby & Zaslavsky, 2015).

The self-organizing topology in our case would be constrained on a fixed set of resources (number of hidden units, layers and so forth) for comparison between a vanilla supervised classifier and one trained via the features extracted through meta-learning in the unsupervised setting. For the Autoencoder units this would be finding the optimal placement of its receptive fields for capturing all the relevant features of the input, MNIST digits for e.g. given a fixed code length (number of hidden units). An indication or a positive (or negative) signal to the meta-learning process, whether the minimum sufficient information is encoded in the representations to successfully predict an instance could be helpful.

KL-Divergence

If $P(x)$ and $Q(x)$ are two separate probability distributions over a random variable x, **KL Divergence** (Kullback-Leibler divergence) or *relative entropy* measures the similarity (or disimilarity) between the distributions (Goodfellow et al., 2016).

$$D_K L(P \| Q) = H(P,\ Q) - H(P) = E_{x \sim P}[log \frac{P(x)}{Q(x)}] = E_{x \sim P}[log P(x) log Q(x)] \tag{2.12}$$

where D_{KL} is the KL-Divergence and $H(P)$ and $H(P,\ Q)$ are the entropy and cross-entropy, respectively.

One way to formulate the problem of estimating the quality of encodings could be, how well we can predict Y from original X vs how well we can predict Y from a compression of X. A key property of KL-Divergence is that it is asymmetric i.e.

$$D_{KL}(P \| Q) \neq D_{KL}(Q \| P) \tag{2.13}$$

and hence can not be used as a true distance metric.

However, using a particular direction of KL-Divergence (since we only care about the input $\rightarrow$ encodings) we can estimate parameters to closely model the probability distribution of the latent encodings with that of the inputs. KL-Divergence has found use in VAEs (Kingma & Welling, 2013) and could potentially be introduced in our meta-objective function (Goodfellow et al., 2016).

Mahalanobis distance

Apart from minimizing only the reconstruction loss (such as MSE) during training for learning an effective compressed representation of the input data, Denouden et al. (2018) used *Mahalanobis distance* as a novelty measure and found better anomaly detection rates on out-of-distribution (OOD) test samples than using only the reconstruction error. Mahalanobis distance is a distance metric, in a multidimensional setting, which measures the distance between a vector and a given distribution (Mahalanobis, 1936).

Denouden et al. (2018) defined the novelty feature as a composite of,

$$\alpha D_M(E(x)) + \beta l(x, D(E(x))) \tag{2.14}$$

where, D_M is the Mahalanobis distance and l is the reconstruction loss.

They trained only on a single digit class from MNIST and tested on OOD test samples and also experimented with variable bottleneck sizes.

Incorporating the Mahalanobis latent distance led to an improvement in performance over when only reconstruction error was used. We could potentially use this distance metric (or something similar) to be able to infer about the encodings as a proxy objective (since reconstruction loss would primarily drive the base learning).

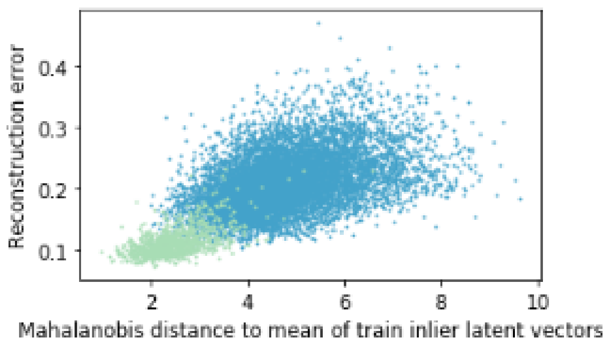


Figure 2.12: The diagram shows reconstructed error plotted against Mahalanobis distance. Inliers are indicated as light green and the out-of-distribution test samples as blue. Adpated from Denouden et al. (2018).

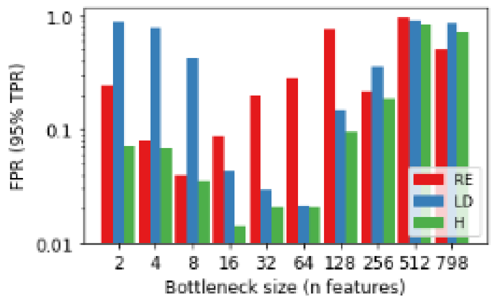


Figure 2.13: The diagram shows the comparison between reconstruction error (RE), latent space distance and hybrid as the score when used for OOD detection with varying bottleneck sizes. Adapated from Denouden et al. (2018).

Predictive Coding

Free energy minimisation theorizes that self-organizing, adaptive systems have a tendency to resist disorder and maintain equilibrium by minimizing free energy (Friston, 2010). Using Predictive coding (Rao & Ballard, 1999), which comes under the Free energy Principle, as the learning algorithm, Dora, Pennartz, & Bohte (2018) trained a generative models that could infer latent representations in subsequent layers of the network. More accurately, given an input image x_i, learned latent representations for layer (l) could be accurately used for predicting the latent representation in the layer directly below it (l-1) and this learned latent representation acts as the target for the next layer (l+1) to learn the latent representations from (Dora et al., 2018).

The error in prediction at each subsequent layer is minimized by appropriately learning the the latent representations and the parameters of the network. They employed an architecture similar to a Convolutional neural network (CNN) with the key distinction being the direction of gradient propagation which is opposite to that of CNNs (i.e. from n^{th} layer to 0^{th} layer) (Dora et al., 2018). The learning of subsequent latent representations at each layer is crucial for our project and predictive coding could provide another viable alternative.

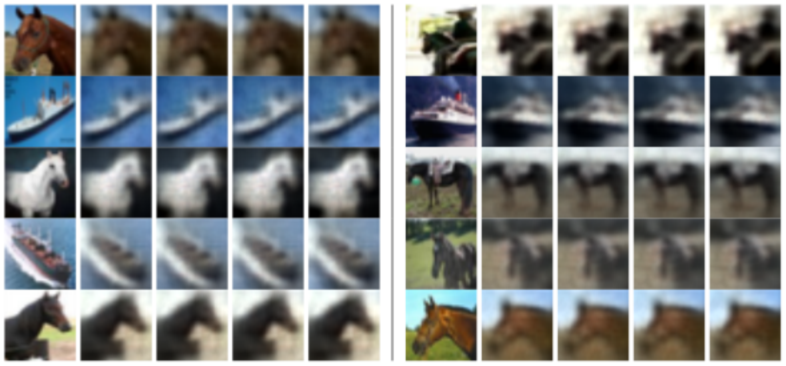


Figure 2.14: Reconstructed images using latent representations at layers 1,2,3 and 4 of the network. Adapted from Dora et al. (2018).

2.6.2 Extrinsic Metrics

The key extrinsic measure we would be primarily focusing on is the classification accuracy or the error rate of supervised learning system trained using the latent encodings from the unsupervised system.

Extracting the encodings of each training example x from the trained autoencoders, a supervised classifier, like a multinomial logistic regression would be trained. For evaluation purposes, the baselines considered could be the performance (accuracy) of a multinomial logistic regression on some fixed architecture vs the one on a meta-learned architecture.

2.6.3 Summary

We started with the neuroscientific exploration of the mammalian neocotex, its functional hierarchy and the integration of receptive fields of different sensory modalities. The importance of the receptive fields in the context of this project was highlighted and how we could parameterize its size and spatial placement for a meta-learning rule. We explored the artificial neural network architecture and its fundamental elements, followed by autoencoders and its properties.

We then discussed about meta-learning in current literature and the different approaches for few-shot learning and generalization across a greater domain of tasks. The current developments in NAS algorithms were traced thereafter with certain searching strategies and performance estimation strategies explored for possible inspiration.

Finally we explored possible metrics (intrinsic and extrinsic) for evaluation of the latent encodings and the formulation of meta-learning rule including information theoretic approaches, predictive coding (free-energy principle) and alternative distance metric.

Chapter 3

Research Design and Implementation

In this chapter, we discuss our research design along with the details of implementation and approaches taken during the set up of our experiment.

Broadly, we identify the three most critical steps of our project as follows :

- Creating the meta-network
- Identifying some intrinsic metric correlated with our extrinsic metric (accuracy), and
- Formulating a meta-learning rule based on the same

Furthermore, we explore and develop a variety of handcrafted architectures to train and test their performances on MNIST and generate data for investigating potential intrinsic metrics correlated with the final accuracy of the classifier. We also discuss the strategies and implementation of different meta-heuristics for a viable meta-learning rule.

The figure below gives a high-level breakdown of the overarching strategy behind our experiments.

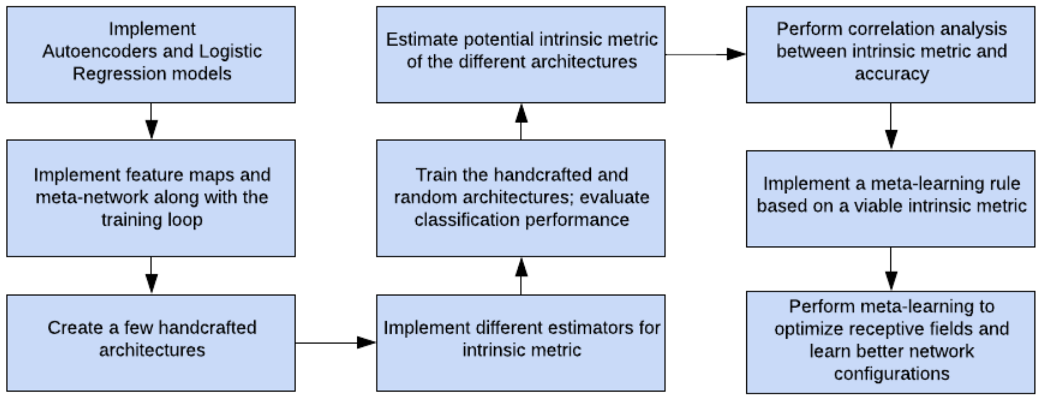

Implement Autoencoders and Logistic Regression models
Implement feature maps and meta-network along with the training loop
Create a few handcrafted architectures
Implement different estimators for intrinsic metric
Train the handcrafted and random architectures; evaluate classification performance
Estimate potential intrinsic metric of the different architectures
Perform correlation analysis between intrinsic metric and accuracy
Implement a meta-learning rule based on a viable intrinsic metric
Perform meta-learning to optimize receptive fields and learn better network configurations

Figure 3.1: Overall design of the experiment.

3.1 Environment

The core development was done on *Python v3.6* with *PyTorch v1.0* (Paszke et al., 2017) as the back-end deep learning framework. All the models were implemented on CUDA based, GPU-accelerated hardware, notably with Nvidia's GTX 950M (6GB) and Tesla K80 (12GB).

Jupyter notebooks were used for code development on a Google Colab environment, to leverage faster, cloud-based hardware and to preserve maximum reproducibility of the experimental results. The following libraries/packages were identified as the core dependencies for the project : PyTorch, scikit-learn, SciPy, pandas, matplotlib, seaborn and NPEET (Non-parametric Entropy Estimation Toolbox) (Ver Steeg, 2000).

3.2 Dataset

We performed our experiments on **MNIST** (LeCun & Cortes, 2010) which is a well established and widely studied data set, generally used for benchmarking deep learning models on computer vision tasks. It consists of 10 classes of handwritten digits, from 0 to 9, with $60,000$ instances as training set and $10,000$ as the test set. The images are 28×28, single channel (grayscale) with pixels values ranging between $[0, 255]$. For our experiments, the images were imported in standard PIL (Python Imaging Library) format.

3.3 Data Preprocessing

Minimal pre-processing was done on the MNIST data set, with the following steps (in order) :

1. **Centre Cropping** : This was done to simplify the handling of dimensions of the receptive fields as multiples of 10. The dimensions of the original MNIST images were reduced from 28×28 to 20×20 (the extra 8×8 pixels along the edges mostly acted as padding).
2. **Data Normalization** : Min-Max normalization was applied to normalize the pixels values of the images from $[0, 255]$ to be between $[0, 1]$. Mathematically,

$$X_{sc} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{3.1}$$

To ensure that the center cropping did not affect the classifier's performance in any significant way, we evaluated the performance of the logistic regression model for both the cases. On reducing the input dimensions from 784 to 400 per image, there was an increase of only ~ 0.003 in the test error rate. Hence, we concluded that these 8×8 regions for the most part acted as extra padding and did not carry any significant additional information.

3.4 Sub-network(s) : Autoencoders

In order to demonstrate the effectiveness of hierarchical encoding of information with a minimal and simple autoencoder architecture, we chose an *undercomplete autoencoder*. These autoencoders formed the **sub-units** or **sub-networks** of our meta-network.

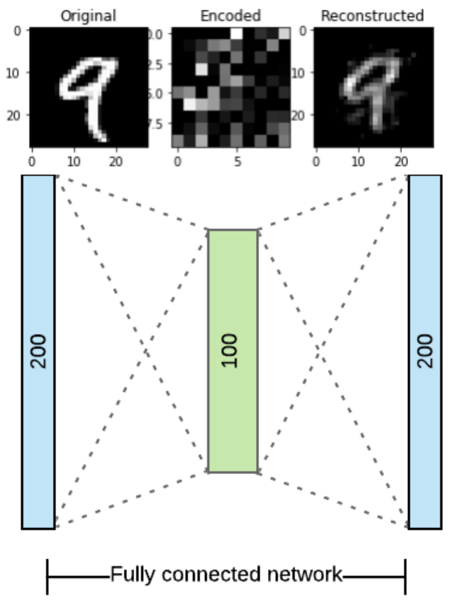


Figure 3.2: The autoencoder (or sub-unit) architecture. The input/output layers are represented by blue and the hidden layer is represented by green. Each image on the top represents the corresponding layer below it and the numbers represent the number of units for the same.

3.4.1 Model Architecture

The network consists of three layers : an input, output and a single hidden layer with 100 latent units (n_{hidden}). The input and output layers have a fixed dimension of 200 to allow the network to target two receptive fields (r_k), each 10×10 dimensions, making it $2 \times 10 \times 10$ per autoencoder. The choice of 100 dimensions for the latent unit was made after cross-validation with 50, 200 and 300 respectively. This particular size along with the MNIST images, cropped to be 20×20, collectively allowed for easier manipulation of the receptive fields within the feature map which we discuss in Section 3.5. These dimensions were however purely chosen for the sake of our experiments and the meta-network would work with any arbitrary dimension of the layers.

We selected **ReLU** (Rectified Linear Unit) as the non-linear activation function for this network. It produces activations which have a lower bound of zero with no fixed upper bound. It can be formulated as,

$$f(x) = max(0, x) \tag{3.2}$$

Additionally, we used **dropout** for regularization in the networks. It renders the activations of a certain proportion of hidden units within a layer to become zero (at random) and induces sparsity as a side-effect during training (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014). This ensured that the network did not overly rely on a handful of hidden units when it is learning the parameters (Srivastava et al., 2014).

After some experimentation, a value of 0.2 was chosen which translated to randomly dropping the activations of 20% of the units in the hidden layers, using samples from a Bernoulli distribution (Hinton, Srivastava, Krizhevsky, Sutskever, & Salakhutdinov, 2012). Dropout is only activated during training and is turned off during evaluation to fully utilize the network.

3.4.2 Loss Function

Mean squared error (MSE) was selected as the loss function for the autoencoder sub-units. In the context of images, it represents how *far off* the reconstructed image of the input is, which is computed between the normalized pixel values (input vector) and the reconstruction of the same after passing through the latent bottleneck.

Mathematically,

$$L(x, \hat{x}) = \frac{1}{n} \sum_{i=1}^{n} (x - \hat{x})^2 \tag{3.3}$$

where, x is the input and $\hat{x}$ is the reconstructed output.

This essentially serves as our *primary objective function* which is used for learning the best latent representation of the local regions within the feature map.

3.4.3 Optimizer

Adaptive Moment Estimation or Adam (Kingma & Ba, 2014) is used as the optimizer and was found to be more robust than vanilla Stochastic Gradient Descent (SGD) for training. A learning rate of $1e^{-02}$ was chosen for the same. For additional regularization, a weight decay of $1e^{-04}$ was applied to the optimizer which penalized the overall MSE loss with the *L2* norm.

3.5 Feature Map of Encodings

A simple, high level abstraction for the cortical map was conceived, which stored all the inputs and the subsequent hidden layer activations of the autoencoder sub-units at any given time. All the read/write operations for the inputs and outputs was done through it during training and testing. We refer to this as the *feature map* of encodings, analogous to the terminology used in the context of CNNs and how they map the intermediate activations during convolutions. A key distinction from the standard CNN feature map is that here the input is stored along with the activations from the hidden layers of different sub-units.

Since a single channel image data set was considered for the experiments, the feature map of encodings was a three dimensional tensor which composed of multiple training examples (or minibatches) along its depth and the hidden layer activations expanded along the other two dimensions. This setup was implemented in a modular way to accommodate multi-channel inputs, if required.

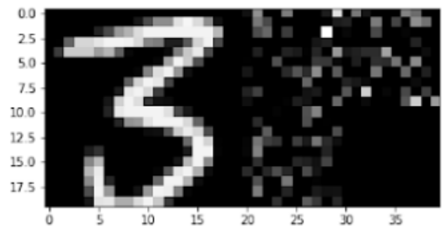


Figure 3.3: Feature map for a single input image. The pixelated, noise-like section to the right is the hidden layer activity or the encoded representations in the different sub-units for this input.

The size of the feature map depends on the following :

1. Size of the inputs (fixed in our case i.e. 2×100)
2. The number of sub-units (K), and
3. Size of the hidden layer of the sub-units (n_{hidden}), which is 100 in our case

No of sub-units	h	w
3	20	40
5	20	50
7	20	60

Table 3.1: Height and Width of the feature map for different number of sub-units

For our experiment, we considered the number of sub-units $K \in \{3, 5, 7\}$. We increased the dimension of the feature map only its width, as per Figure 4.4 and Table 4.1, to accommodate for greater number of sub-units. However, both the height and width could be altered. Since the feature map would later serve as the search space for the meta-learning rule, different dimensional considerations could potentially lead to faster/slower convergence.

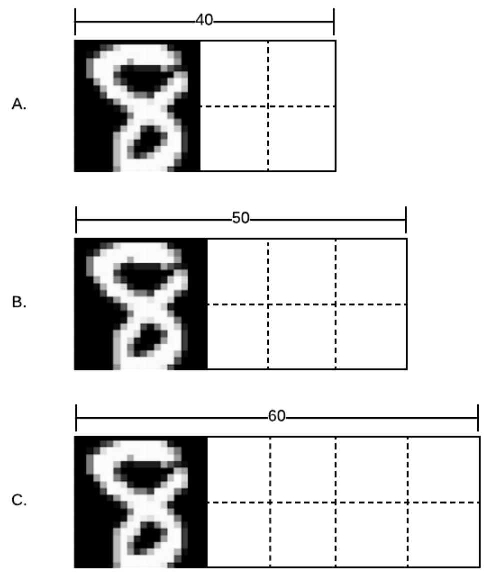


Figure 3.4: Depiction of the different sizes of the feature map, along with the same input, used for different number of sub-units i.e. 3, 5 and 7. The patches separated by dashed lines are the location of the hidden layer activity of each sub-unit.

The hidden layer activations are reshaped to be 10×10 dimensions before storing and updating the feature map. Also since all the hidden layer activations during training are stored, the final feature map has a depth equal to $epochs \times N$. An intermediate feature map of depth equal to N is also used for computing the intrinsic metrics per epoch during training.

3.5.1 Receptive fields and hidden layer activity

Each sub-unit targets a fixed sized receptive field to receive its inputs from the feature map. More specifically, each sub-unit was set to target *two* receptive fields and subsequently receive inputs from these regions. This design consideration was made to have effective hierarchies where different sub-units would have composites of the input image and learn high level features. The *total* effective receptive field size of a sub-unit was then $2 \times r_k$. It also meant that a given sub-unit would receive only part of the input and have incomplete information about all its features. For a hierarchical topology to perform well in this case, it would have to combine the information from each sub-unit to learn a good representation of the input.

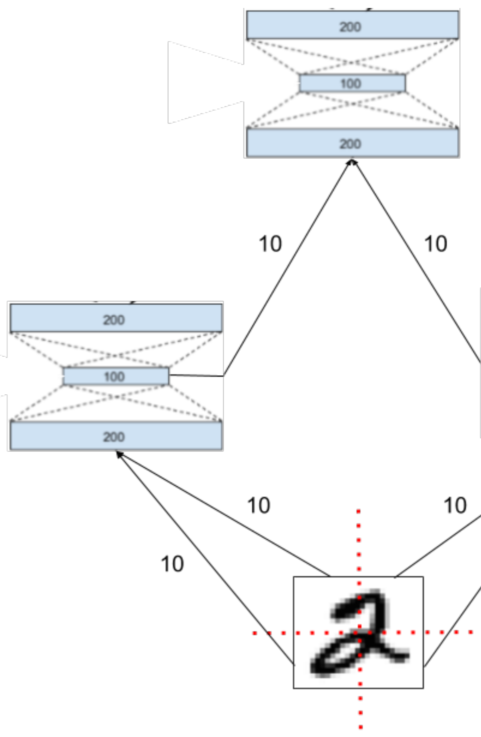


Figure 3.5: A simple meta-network with 3 sub-units. Each sub-unit targets two receptive fields and receives $2 \times 10 \times 10$ dimensional input. The top sub-unit subsequently receives the hidden layer activations from the two below in this case.

Allocation of the sub-unit activations within the feature map was completely arbitrary and was specified prior to training which remain fixed throughout. The hidden layer activations from all the sub-units hence formed an abstraction of the gradual, changing, encoded representations across the cortical map which we briefly discussed in Section 1.2.2.

In order to denote the location of the receptive fields as well as the hidden layer activations of the sub-units within the feature map, the following representation was used :

$$R = \{r_k = \{(x_{start}^{(k)}, x_{end}^{(k)}), (y_{start}^{(k)}, y_{end}^{(k)})\} \mid k \in \{1, \dots, K\}\} \tag{3.4}$$

where,

- R represents the set of all the receptive fields of a topology
- $(x_{start}^{(k)}, x_{end}^{(k)})$ are the beginning and ending *row* indices, and

- $(y_{start}^{(k)}, y_{end}^{(k)})$ are the beginning and ending *column* indices respectively

for the hidden layer activity of the k^{th} sub-unit within the feature map. Indexing based on the same returns K 3-dimensional tensors, each with size $(h \times w \times d)$ where,

- $h = x_{end}^{(k)} - x_{start}^{(k)}$
- $w = y_{end}^{(k)} - y_{start}^{(k)}$, and
- $d = epoch \times N$

We considered a single receptive field to have a size of 10×10 (matching that of a single hidden layer). Also in case the *total* receptive field size provided for a particular sub-unit was smaller than its input dimensions (for e.g. 10×10 instead of $2 \times 10 \times 10$), this implicitly indicated that the sub-unit was targeting both its receptive fields onto a single 10×10 patch and would receive redundant inputs by expanding the same to match its input dimensions of $2 \times 10 \times 10$. As explored further in Section 3.9, this was the case with the columnar architectures where the hidden layer activity of a sub-unit was directly targeted to be the receptive field for the next sub-unit up the hierarchy.

3.6 Meta-network

The meta-network is a combination of the feature map and the sub-units. Training in the context of this meta-network was then to learn the K meta-parameters i.e. $\{r_k \mid k \in \{1, \ldots, K\}\}$, where r_k is the receptive field location for the k^{th} sub-unit. This network would have to essentially evolve and adapt its topology to capture the best latent representations of the input. We also avoided over-engineering any architectural component to allow the meta-learning rule to *discover* better configurations of the meta-network, which was the key recurring idea.

The topological structure of the meta-networks is a directed acyclic graph (DAG) and is defined by :

1. The number of sub-units (K), and
2. The location of their receptive fields (r_k)

Training of this meta-network essentially involved the simultaneous training of the sub-units *locally*, which were independently training of one another.

3.6.1 Training

For each training minibatch, the feature map was initialized with the input images and zeros for rest of the regions which represented non-activation. This was followed by performing $K \times K$ forward passes. For each iteration of this $K \times K$ loop, the feature map was updated with the hidden layer activations of the k^{th} sub-unit. The intuition behind the $K \times K$ forward passes was to make the training of the sub-units order agnostic. This overcame scenarios where a sub-unit would receive zeros as input when it targeted its receptive field onto the hidden layer of another sub-unit that did not have had its forward pass yet. Hence it ensured that the feature map was fully saturated and all the sub-units had some activity in their hidden layer.

At this stage, all gradient tracking operations were turned off since we wanted the meta-network to stabilize before doing back-propagation and actually training the sub-units. It also reduced the overhead at the same time. Once the hidden layer activations were fully stabilized, K final forward passes were performed for each sub-unit, with the gradients now allowed to accumulate. Thereafter, K backward passes were performed and a single training iteration for the meta-network completed. At the end of each epoch, the average training losses for each sub-unit were computed along with the estimation of entropy per sub-unit on the sample of hidden layer activity. Below is the pseudocode for the training :

Algorithm 1: Pseudocode for Meta-network training

```
1  Initialize : $K$ subUnits;
2  while not maxEpoch do
3      for minibatch do
4          initialize : featureMap;
           /* $K \times K$ forward passes to fill the feature map with hidden layer activity */
           /* Gradients are not accumulated */
5          for $k \in K$ do
6              for $k \in K$ do
7                  forwardPass;
8                  update featureMap;
9              end
10         end
           /* Actual $K$ training passes (with backpropagation) */
           /* Gradients are accumulated */
11         for $k \in K$ do
12             forwardPass;
13             update featureMap;
14             backwardPass;
15         end
16     end
17     compute IntrinsicMetric
18 end
```

3.7 Estimating Entropy

We selected entropy of the hidden activity as the intrinsic metric to test for a possible correlation with accuracy. As briefly discussed in Section 4.10, we implemented three different entropy estimators for the experiments. We refer to the entropy of the hidden layer activity for a single sub-unit as the *local entropy* here, since it is localized to a specific region of the feature map. We estimate all entropies in bits (base 2) for our experiments.

To compute the binning based local entropy, we employed a similar strategy as Saxe et al. (2018), where the sub-units within the meta-network were allowed to be fully trained to capture the maximum activation value. This ensured that the resulting histogram represented the full range of activations (as outputs from ReLUs do not have an upper bound).

Thereafter, the activations were discretized by computing a histogram, using a bin size of 100, which gave a rough estimation of the probability distribution of the hidden layer activations. Using this resulting histogram, the entropy of the hidden layers was estimated using the shannon entropy formula from Section 2.6.1. One advantage of this approach is that the estimation of entropy is independent of the choice of non-linearity in our networks.

The other two state-of-the-art estimators were implemented from NPEET (Non-Parametric Entropy Estimator Toolbox) (Ver Steeg, 2000) for continuous entropy estimation. However, both the KNN based Kraskov estimator (Kraskov et al., 2004) as well as the KDE based estimator (Kolchinsky & Tracey, 2017) had a much higher overhead than the first approach, primarily due to a combination of computing pairwise-distance matrices for higher dimensions along with exploring nearest neighbours via a tree based approach.

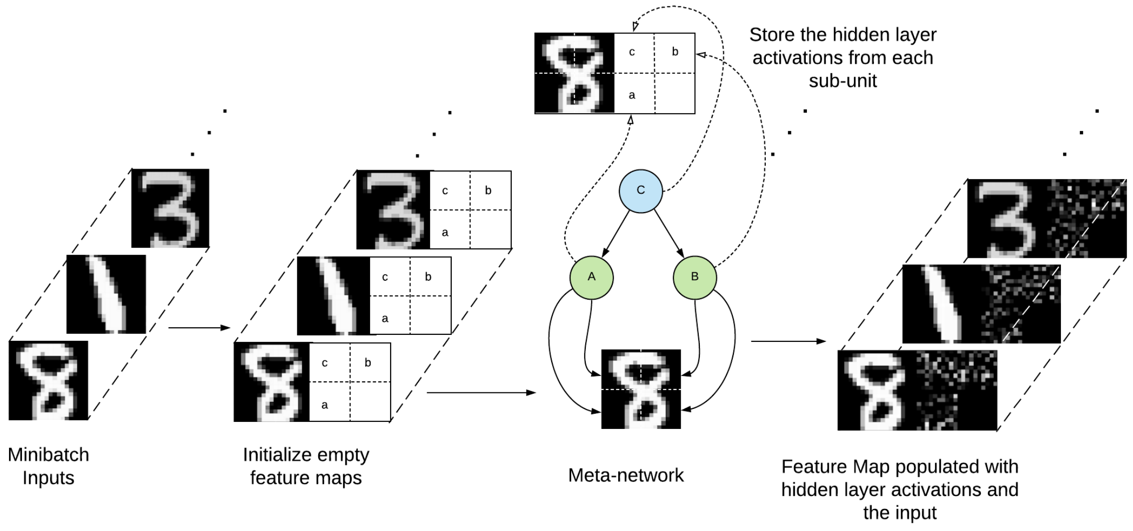

Store the hidden layer activations from each sub-unit
c
b
a
C
A
B
Minibatch Inputs
Initialize empty feature maps
Meta-network
Feature Map populated with hidden layer activations and the input

Figure 3.6: A schematic depiction of a forward pass in the meta-network. Here A, B and C are the AE sub-units and a, b and c are the location of their hidden layer activity in the feature map. Receptive fields of A and B : r_A and r_B, respectively, target the image whereas r_C targets the hidden layer activations of both A and B. The sub-units train locally to reconstruct their inputs as they get progressively better at compressing relevant features of the input. The dotted arrows represent the updating of the hidden layer activations of the corresponding sub-units in the feature map.

A workaround around the same was to estimate the entropy based on a sample of around $12,000$ final hidden layer activations per epoch. This provided a reasonable trade-off between getting a good enough estimate and performance overhead of the network. In contrast to both these estimators, the histogram based estimator had an extremely low overhead on training and hence we used all the hidden layer activations for the same.

3.8 Classifier : Multinomial Logistic Regression Model

3.8.1 Model Architecture

A simple logistic regression model with two linear layers : the input and the output, was selected. Due to the classification problem being multi-class, the output layer had 10 units followed by a softmax (instead of the standard sigmoid) on the logits to obtain the normalized probability scores for each class (Goodfellow et al., 2016).

Softmax can be formulated as,

$$\sigma(z^{(i)}) = \frac{e^{z^{(i)}}}{\sum_{j=1}^{C} e^{z_j^{(i)}}} \tag{3.5}$$

where, $\{1, \dots, C\}$ are the class labels and $z^{(i)}$ is the i^{th} output (Goodfellow et al., 2016).

The dimensions of the input layer of the classifier depends on :

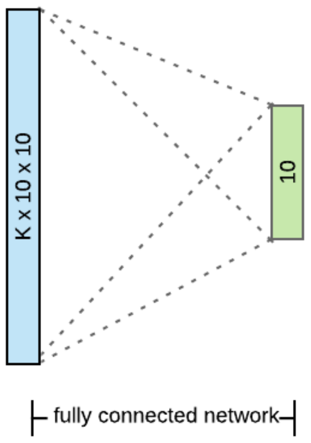


Figure 3.7: The logistic regression model architecture. The input layer is represented by blue and the output layer by green. K is the number of AE sub-units.

1. The number of sub-units (K), and
2. The dimensions of the hidden layer of these units (n_{hidden})

3.8.2 Loss Function

Cross Entropy was used as the loss function for the classifier. Mathematically,

$$L(y, \hat{y}) = -\sum_{i} y_i * log(\hat{y_i})$$

where, y is the probability vector of true class labels and $\hat{y}$ is the probability vector of the predicted class labels (obtained after applying softmax on the raw logits) (Goodfellow et al., 2016).

3.8.3 Optimizer

We once again used Adam as the optimizer with a learning rate of $1e-02$ for the same. A value of $1e-04$ was used as weight decay for L2 regularization.

3.8.4 Training

For every minibatch during the training of the classifier, the inputs were first propagated through the fully trained meta-network to obtain the hidden layer activations of the inputs via the feature map. After the feature extraction process, the hidden layer activity of all the sub-units were *concatenated* into a $K \times n_{hidden}$ dimensional tensor which was then passed as the input to the classifier. Hence, the classifier was trained on the combined encoded representations of the input from all the sub-units of the meta-network.

3.9 Handcrafted Architectures

A total of 17 handcrafted architectures for the meta-network were implemented during the initial phase to benchmark their performances and compute the intrinsic metrics for correlation analysis. All the receptive fields along with the encoding locations were explicitly specified before training these handcrafted meta-networks.

The core motivation behind this manual setup of architectures was to find some intrinsic metric, computed over the sub-units of a meta-network, which was correlated or indicative of the final performance of the classifier. Establishing a correlation would therefore lead to the intrinsic metric being a candidate term for the meta or secondary learning algorithm.

The handcrafted topologies were divided into the following categories :

1. **Type A** : topologies with 3 AE sub-units
2. **Type B** : topologies with 5 AE sub-units
3. **Type C** : topologies with 7 AE sub-units

The different number of sub-units were necessary to allow for a variety of meta-network architectures, both optimal and sub-optimal, to be created. In order to contrast whether hierarchical topologies led to better representations (and eventually better performance) some trivial and non-hierarchical architectures were also conceived for e.g. *flat* architectures such as A1, B1, C1 and *columnar* architectures such as A2, B3, C2.

Topology	Name	No of sub-units
A1	Flat	3
A2	Columnar	3
A3	Pyramidal	3
B1	Flat	5
B2	X-Shaped	5
B3	Columnar	5
B4	Inverted-Y	5
B5	Asymmetric Pyramidal	5
C1	Flat	7
C2	Columnar	7
C3	Symmetric Pyramidal	7
C4	Asymmetric Pyramidal	7
C5	Semi-X	7
C6	Inverted-Y	7
C7	Inverted Tuning Fork	7
C8	Inverted-U	7
C9	Semi-Columnar	7

Table 3.2: Different handcrafted topologies

The notion of perfect hierarchy was also taken into account and highly hierarchical architectures for e.g. A3, B5 and C3, were expected to perform better than their *less* hierarchical counterparts due to better compositional properties imparted by its structure.

Corominas-Murtra, Goñi, Solé, & Rodríguez-Caso (2013) stated that a *perfectly* hierarchical systems exhibits the following fundamental properties :

1. Order

2. Reversibility, and
3. Pyramidal structure

Hence, a few topologies were selectively imparted with these properties while some were made to deviate from the same for e.g. C3 (Symmetric Pyramidal) vs C4 (Asymmetric Pyramidal).

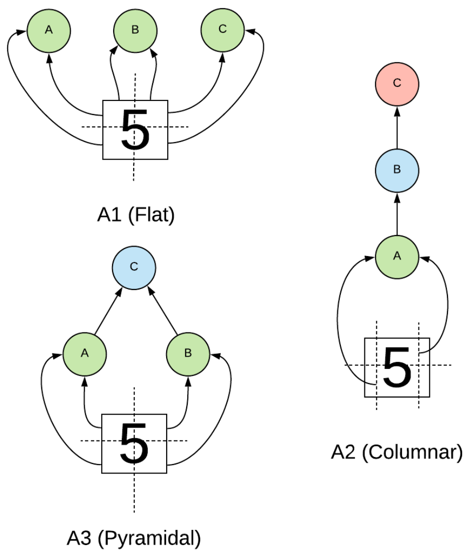


Figure 3.8: Type A handcrafted architectures. Each sub-unit is designated an alphabet and the different colors of the sub-units represent their level within the hierarchy. The arrows represent the input to the sub-units and the targeted receptive fields. The dotted lines divides the input MNIST image (a handwritten 5 in this case) into regions of 10×10 each which serve as the inputs.

3.9.1 Hyperparameters

Minimal hyperparameter tuning was done and similar values were used for both the sub-units and the classifier, as follows :

- Epochs for training the sub-units = 5
- Epochs for training the classifier = 10
- Mini batch size (for both phases) = 128
- Weight decay for L2 regularization (for both) = $1e^{-04}$
- Learning rate (for both) = $1e^{-02}$

3.9.2 Training

In order to train the meta-network and evaluate the classification performance for the handcrafted architectures, the following steps were implemented :

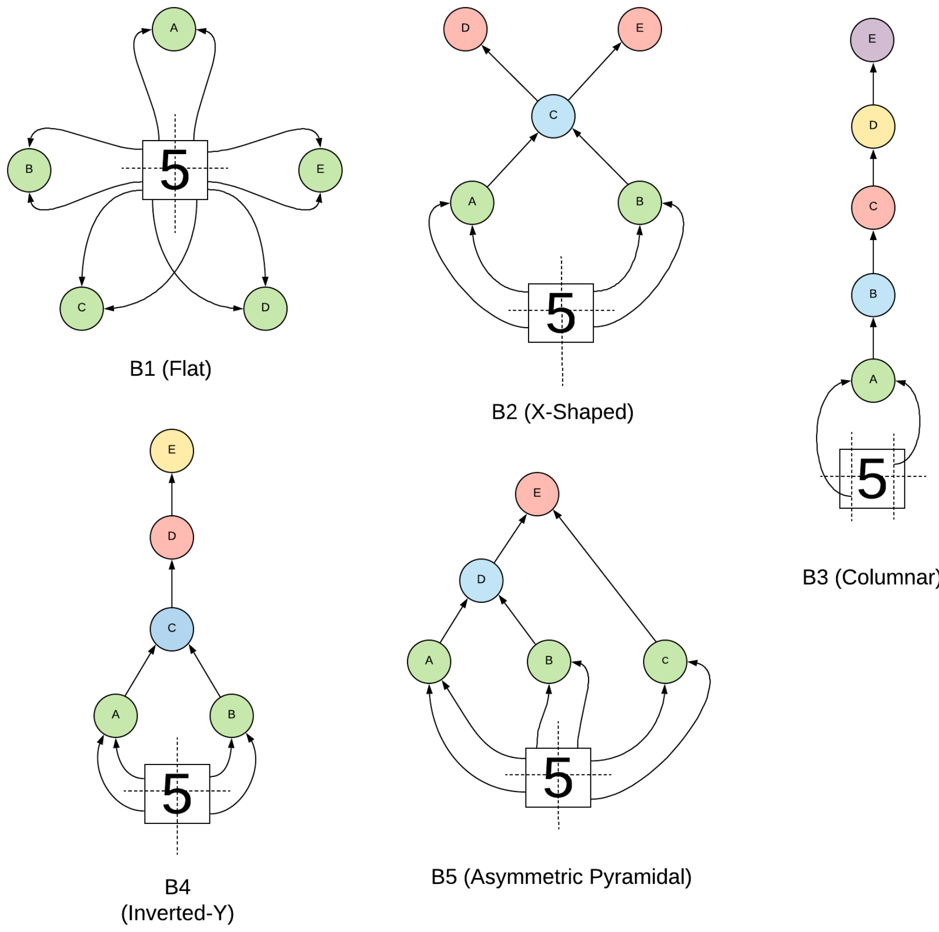


Figure 3.9: Type B handcrafted architectures.

1. Initialize the K sub-units (as per the handcrafted architecture) with random weights along with their corresponding receptive fields and encoding locations for the feature map
2. Train the meta-network for a few epochs as per Section 3.6.1
3. Obtain the final feature map and the estimated entropies (per epoch) using all three entropy estimators
4. Obtain the histogram-based entropy estimate for all hidden layer activations using the final feature map
5. Initialize the multinomial logistic regression model and a new feature map
6. Use the fully trained meta-network to obtain the hidden activations for each input x and update the new feature map
7. Train the logistic regression model by concatenating the hidden layer activity of each sub-unit from the feature map for a few epochs
8. Evaluate the test accuracy per epoch of training and obtain the final accuracy of the classifier

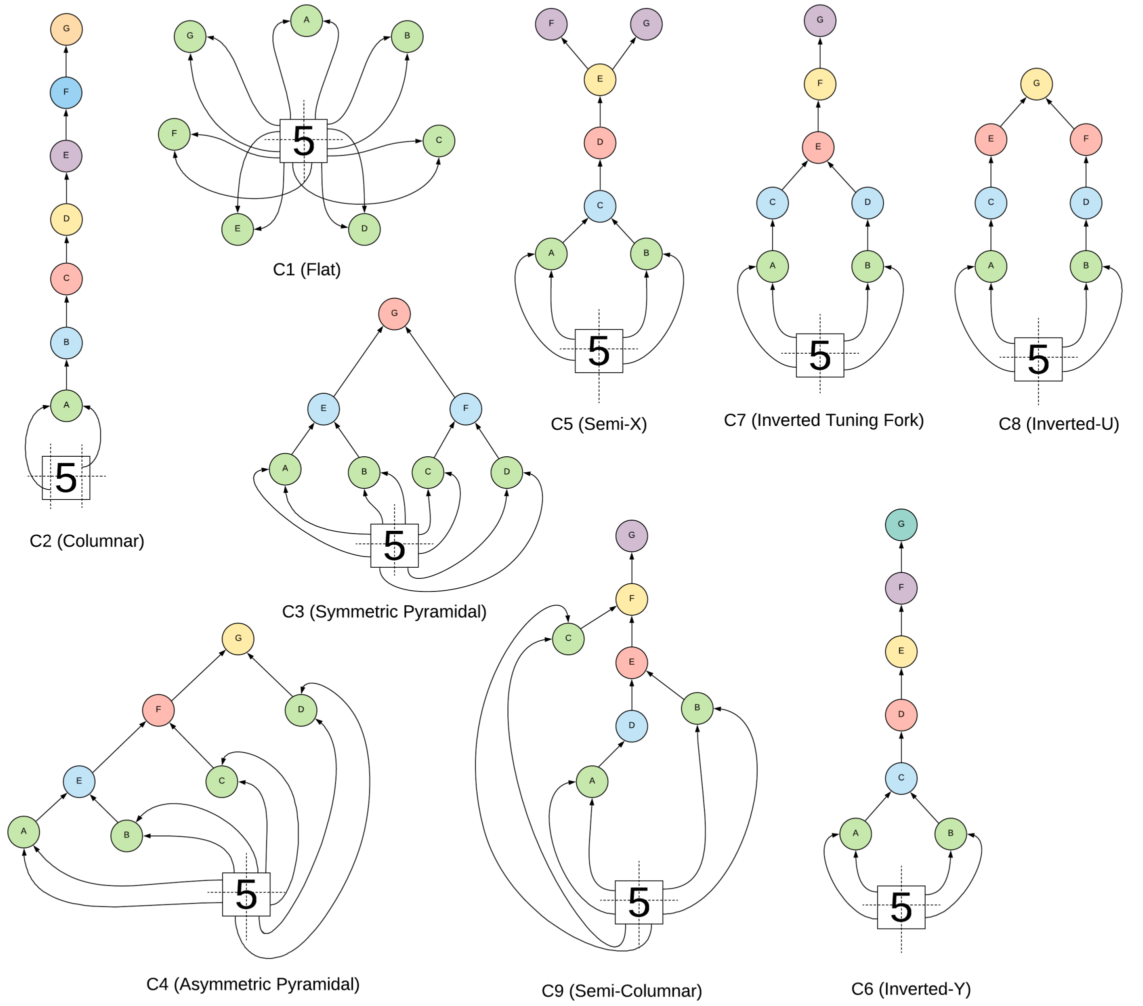


Figure 3.10: Type C handcrafted architectures.

3.10 Random Architectures

Performance of the classifier was also evaluated on randomly initialized architectures. Depending on the location of the receptive fields and if they captured enough discriminative or relevant information through the encoded representations, the results were expected to be vary greatly. A total of 9 randomly generated topologies were to be trained and evaluated using the same training loop as Section 3.9.2.

The inclusion of random architectures was an important step to illustrate that carefully designed handcrafted architecture should in most cases easily outperform any architecture with randomly initialized receptive fields. In essence, the meta-learning rule would need to leverage and capture all the *good design elements* that come with the handcrafted ones without ever being explicitly encoded with the same.

3.11 Meta-learning Rule

As previously stated in Section 2.4 the objective of a viable meta-rule would be that it is able to effectively distribute its fixed, allocated resources to learn or extract the best possible compressed representations of the inputs.

Based on the correlation analysis below in Section 4.4 of the final classification accuracy with the summary statistics of all the local entropies, we found a few candidate terms to be used for the meta learning rule. We selected the strongly correlated of these intrinsic metrics, the *median* entropy of the meta-network to form a meta-learning rule.

As we discuss further in Section 4.4, a strong *positive correlation* with accuracy of the classifier meant that if the receptive fields targeted regions on the feature map which led to an increase in the median of all the hidden layer activations, then the resulting topological configuration would yield a higher classification performance. The meta-objective function was then expressed in terms of the median entropy of the meta-network and maximizing the same would amount to finding the best meta-parameters i.e the receptive field locations.

Hence, to formulate this as an optimization problem, we started by exploring *meta-heuristics* such as *Local Search* and its variants which we discuss below. The meta-objective function in our case would be the *fitness function* for our optimization problem. We focused on single solution meta-heuristics rather than population-based approaches for our experiments (Brownlee, 2011).

Local Search is a direct search technique, since it does not require derivatives of the search space (Brownlee, 2011). They also have very small space complexity since only the current state is examined before making a move and previous states could be discarded. Consequently, since it only keeps a single current state we do not need to maintain or handle a extensive search tree or graph (Russell & Norvig, 2016). In terms of time complexity, it varies depending on the type of local search which we discuss for each variant below. Another aspect which we discuss below was that *all* the receptive field locations were perturbed during each iteration.

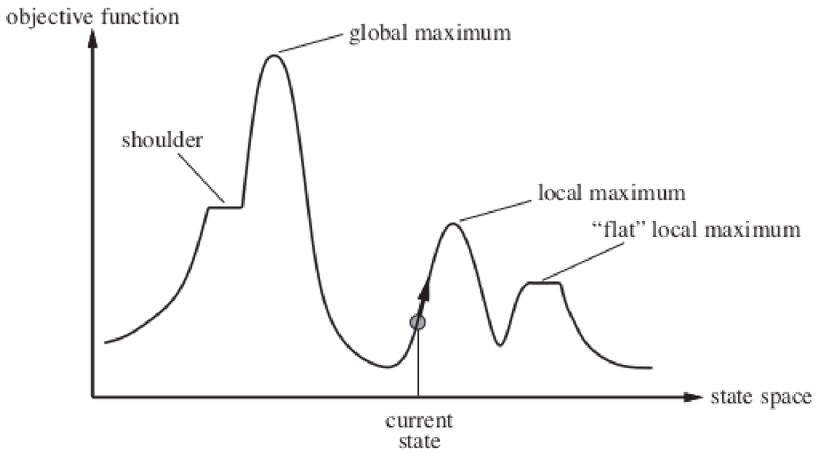


Figure 3.11: A diagram of 1-dimensional state space vs an objective function along with the different topological features. The peaks correspond to the maximum values with the highest peak being the global optima. Adpated from Russell & Norvig (2016).

For a given topology, let X_k be a *random variable* representing the hidden layer activity and $H(X_k)_t$ be the local entropy of the k^{th} sub-unit at time t. Hence we can define our intrinsic metric as,

$$\alpha_t = \textit{median}\{H(X_k)_t : k \in \{1, .., K\}\} \tag{3.6}$$

where, K is total number of sub-units and α_t is the median of all the local entropies at time t.

3.11.1 Random Search

We started with a simple random search in which a random receptive field configuration was selected and evaluated at every iteration of the optimization process. It sampled the receptive field locations randomly using a uniform probability distribution across the entire search space (Brownlee, 2011). Although, random search in quite inefficient and generally yields non-optimal solutions, the motivation behind exploring this meta-heuristic was to validate the correlation and demonstrate that a simple local search based meta-rule with an intrinsic metric, like the median entropy, could lead to better feature extraction and compressed representations of the input for downstream tasks like classification.

We define the random search with intrinsic metric α as follows (Brownlee, 2011),

Algorithm 2: Random Search

```
   Input: metaSteps
   Output: $r_k$
 1 Initialization:
 2 $r_{best} \leftarrow \emptyset$
 3 for $m \in metaSteps$ do
 4     $r_t \leftarrow$ randomTopology()
 5     $\alpha_t \leftarrow$ computeMetric($r_t$)
 6     if $\alpha_t > \alpha_{t-1}$ then
 7         $r_{best} \leftarrow r_t$
 8     else
 9         continue
```

3.11.2 Stochastic Hill Climbing

Next we explored the *stochastic hill climbing* algorithm from the class of local search algorithms. Hill climbing algorithms in general are *greedy* and chooses the nearest best node iteratively (Russell & Norvig, 2016). Although it does not guarantee a global minima or maxima in case of non-convex optimization problems, it provides a reasonably optimal solution within a time constrained setting.

Within the family of hill climbers, we explored the *stochastic hill climbing* algorithm as a potential candidate for the meta-rule. It is more robust than a simple hill climber and could alleviate the problem of being stuck at local optimas.

The main difference from a simple hill climber is that the next best step is accepted with some probability of p based on the improvement in the cost evaluated by the cost function, rather than with 1 every time (Russell & Norvig, 2016). Hence, candidate solutions are explored probabilistically in this case, rather than deterministically (Brownlee, 2011). One caveat to this approach is that it converges much slower than a simple hill climber, since it occasionally heads in a random direction to maximize the exploration of the search space (Russell & Norvig, 2016).

At each hill climbing step, a small random perturbation in the receptive field location, constrained to a range between $[-1, 1]$, was introduced for allowing incremental change, which is a characteristic of the same. Hence it only accepted a randomly selected neighbour as a candidate solution if it led to an improvement

Algorithm 3: Stochastic Hill Climbing

```
    Input: metaSteps
    Output: $r_k$
1   Initialization:
2   $r_{best} \leftarrow$ randomTopology()
3   $\alpha_0 \leftarrow$ computeMetric($r_{best}$)
4   for $m \in$ metaSteps do
5       $r_t \leftarrow$ randomNeighbor($r_{best}$)
6       $\alpha_t \leftarrow$ computeMetric($r_t$)
7       if $\alpha_t > \alpha_{t-1}$ then
8           $r_{best} \leftarrow r_t$
9       else
10          continue
```

over the current state. Although this led to gradual shift in the receptive field locations towards better configurations, there were still chances of getting stuck in a local optima.

3.11.3 Random-restart hill climbing

To encourage a greater exploration of the search space, we also explored the *random-restart hill climbing* algorithm. It tries to find the global optima by restarting the search R times with a random initialization of the receptive field location each time (Russell & Norvig, 2016). With a different initial condition r_0 every time, it performs a series of R different hill climbing searches and retains only the best r_i. If p is the probability of success of each hill-climbing search, then the expected number of restarts required is $1/p$ (Russell & Norvig, 2016).

The search space was constrained in two different ways :

1. For each new random initialization/restart, the dimensions of the feature map were chosen as the range of values for the indices of new receptive fields, while preserving their original size, and
2. For each hill climbing iteration, a small, discrete range of values $[-1, 1]$ were allowed for the indices of the receptive field location to shift

In terms of time complexity, the average cost of hill climbing was multiplied by a constant factor of R and was not as expensive, considering the ability to explore the search space.

3.11.4 Training

A meta-learner in the form of a *class* was defined to maintain the state of the hill climber and the computed intrinsic metrics as class variables. A similar training loop as the handcrafted topologies in Section 3.9.2 was employed, with the main difference now being the random initialization of the receptive fields over the feature map at the start of the hill climbing.

For the weights of the sub-units through the course of the training, we considered the following cases :

- **Case 1** : *Resetting* all the autoencoder weights for each random perturbation in the receptive fields and retraining from scratch.
- **Case 2** : *Retaining* all the autoencoder weights learned from training on previous receptive field configurations.

We experimented with both the cases and observed that retaining the sub-unit weights learned from previous receptive field positions did not offer any advantage and in fact led to poor quality of encodings. The hidden layer activity was much sparser and had fewer activations over time which led to lower entropy estimates. Re-initializing the weights before each random perturbation led to stable pool of activations in hidden layers of the sub-units from which entropy could be properly estimated.

In the context of parameter updates, the meta-learning rule ran slower than the primary local learning in the sub-units. This ensured that the sub-units were fully trained and that they converged to a reasonable degree, before estimating the entropy of their hidden layer activity and evaluating α for the acceptance criterion of the inner loop.

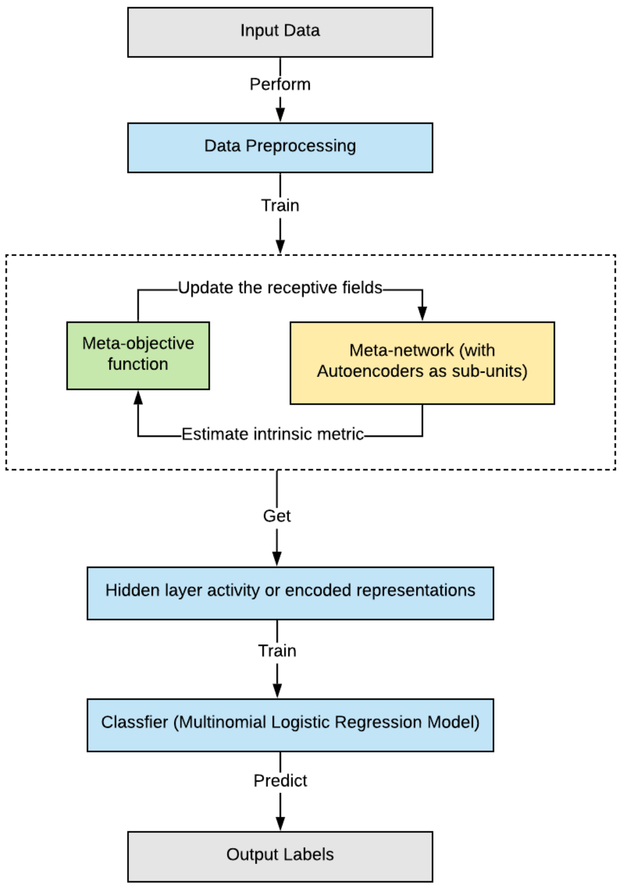

Input Data
Perform
Data Preprocessing
Train
Update the receptive fields
Meta-objective function
Meta-network (with Autoencoders as sub-units)
Estimate intrinsic metric
Get
Hidden layer activity or encoded representations
Train
Classfier (Multinomial Logistic Regression Model)
Predict
Output Labels

Figure 3.12: Schematic overview of the training pipeline

3.12 Summary

We implemented both the unsupervised and supervised models : the autoencoder and multinomial logistic regression models respectively, for learning effective representations of the input and subsequent classification based on the same. We further defined a high-level abstraction for the cortical map i.e. the feature map, that stored the encoded representations or the hidden layer activations of the autoencoders along with their inputs. We also developed a *meta-network* in which the autoencoders acted as the *sub-units* and its overall topology could configured based on the receptive field locations of these sub-units.

We also implemented three different entropy estimators, namely histogram-based, KNN-based and KDE-based estimators (Kraskov et al. 2004; Kolchinsky & Tracey 2017). For the first approach we used all the hidden layer activations whereas for the latter two methods, a sample of 12000 hidden layer activations per epoch was used for estimation of the entropy, due to a high overhead of the network.

We conceived a total of 17 handcrafted architectures along the 9 randomly generated ones for the meta-network with 3, 5 and 7 sub-units, to compute the proposed intrinsic metric(s) and find potential correlation with classification accuracy. Different design considerations like the properties of perfect hierarchies were taken into account to yield both optimal and sub-optimal architectures and to also have a greater variance in the data generated through the experiments.

We considered several variants of the local search algorithm as candidates for the meta-learning rule. Initially starting with a simple hill climber we slowly moved towards more robust optimization methods to counter getting stuck in local maxima, with stochastic and random-restart hill climbers. Combining all the above we created a pipeline for an adaptive, self-organising topology.

Chapter 4

Evaluation

In the previous chapter, we discussed and developed the overall framework for training a meta-network, along with a few candidates for a meta-learning rule. Here we look at the results from our experiments with the same.

Specifically, we discuss the :

- Performance of the handcrafted architectures
- Performance of the randomly initialized architectures
- Feature maps and estimated entropy during training
- Correlation between entropy and accuracy, and
- Performance of the meta-learned architectures

We begin by evaluating all the handcrafted and random architectures and then show in Section 4.4 below that both the *median* and *maximum* of all the local entropies could be used as a viable intrinsic metric for a meta-learning rule that could lead to better receptive fields locations within the feature map.

Our experiments also show in Section 4.5 that using an approach where a network maximizes the *median* of all local entropies, it could learn to optimize the receptive fields of the autoencoder units in a hierarchical manner and generate better encoded representations of the inputs. We verify this through supervised classification of the meta-learned architectures on the MNIST data set. The topologies learned via the meta-rule also provide a competitive performance with the best performing handcrafted architectures which we tested.

4.1 Handcrafted Architectures

We trained the 17 handcrafted architectures discussed in Section 3.9 in a series of 3 runs each (51 runs in total). We subsequently performed image classification on MNIST for each run per architecture. We compile our results below in Table 4.1, where we report the *mean* of the accuracy for all 3 runs per topology. We observed an increase in training time with the addition of more sub-units, with a total of approximately ~ 0.65 GPU days taken to run the training for all 51 runs.

As expected per Section 3.9, we observed that the topologies with properties of a *perfect* hierarchy, namely A3, B5 and C3, performed the best for each category. An interesting contrast in the performances within pyramidal structures was that between C3 (Symmetric Pyramidal) and C4 (Asymmetric Pyramidal). Although both the topologies fulfilled the criteria to be perfect hierarchies, *symmetry* seemed to be playing an

Topology	Name	Accuracy
A1	Flat	94.88
A2	Columnar	91.88
A3	Pyramidal	95.11
B1	Flat	95.46
B2	X-Shaped	95.74
B3	Columnar	92.20
B4	Inverted-Y	95.52
B5	Asymmetric Pyramidal	96.00
C1	Flat	95.53
C2	Columnar	92.16
C3	Symmetric Pyramidal	96.30
C4	Asymmetric Pyramidal	95.94
C5	Semi-X	95.41
C6	Inverted-Y	95.31
C7	Inverted Tuning Fork	95.55
C8	Inverted-U	95.96
C9	Semi-Columnar	95.93

Table 4.1: Accuracy of all the handcrafted architectures. The architectures with the highest accuracy of each topology type (A, B and C) are highlighted in gray.

important role in this case. It was evident from this that given exactly the same resources, a hierarchical topology with a symmetric and pyramidal structure is able to combine low level features into high level representations slightly more effectively. Also, we observed that although increasing the number of sub-units led to an overall improvement in performance of these perfectly hierarchical architectures, the difference in improvement from $5 \rightarrow 7$ was not as high as from $3 \rightarrow 5$. It could potentially indicate that only increasing the number of sub-units would not guarantee the same margin of improvement in performance and might have diminishing returns when compared to the training overhead for meta-networks with a large number of them.

On the other hand, columnar architectures had the lowest accuracy scores. It could be attributed to the fact that they completely deviated from a perfectly hierarchical topology, received partial information overall and had very poor integration of receptive fields as each sub-unit only *looked* or targeted its receptive field at the unit directly below it.

The other non-hierarchical flat architectures namely A1, B1 and C1 yielded accuracy scores somewhere in between. Among these, type C topologies had the highest accuracy followed by type B and type A. This also seemed to indicate that increasing the number of sub-units did lead to better encoded representations coupled with the fact that the feature map had a more hidden layer activations. In general, architectures which deviated from the properties of a perfect hierarchy, seemed to exhibit lower performance relative to the degree of deviation.

4.2 Random Architectures

We also trained the 9 randomly initialized architectures for each type (A, B and C) and measured their classification performance. Similar to the handcrafted ones, we report the mean accuracy of the classifier from 3 different runs per architecture in Table 4.2, trained using randomly initialized receptive fields.

As discussed in Section 3.10, we observed some variation in their performances. We saw that the topologies in which the receptive fields of the sub-units targeted regions of high activity had better performance in general than the ones targeting less active regions. We also observe from Table 4.2 that a meta-network with

Units	Type	Name	Accuracy
3	A	R1	90.72
		R2	90.91
		R3	87.16
5	B	R1	92.88
		R2	93.11
		R3	86.46
7	C	R1	90.76
		R2	93.50
		R3	76.64

Table 4.2: Accuracy of the randomly initialized architectures.

greater number of sub-units does not guarantee better performing random architectures due to an increase in the number of receptive fields (since the feature map scales as well).

4.3 Feature Maps and Estimated Entropy

During the training of the meta-network, we computed all the local entropies using the estimators described in Section 3.7. The following key observations were made about entropy of the hidden activity and the feature map :

1. The sub-units higher up the hierarchy had lower estimated entropy of their hidden layer activations than the lower level ones, and
2. The upper-bound (or maximum activation) of the hidden layer activations tended to increase with the number of epochs of training. As a result, the entropy of all the hidden layer activations also increased with the number of epochs.

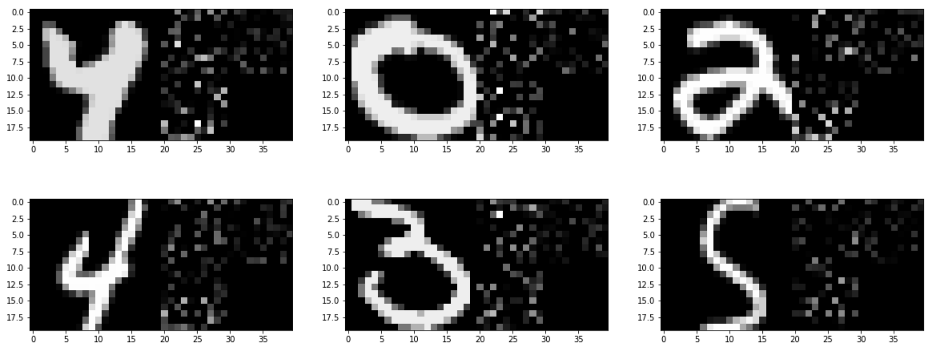


Figure 4.1: Feature maps from the early training epochs of a A3 (Pyramidal) topology. The upper right corner of each feature map is encoded by the top level sub-unit in the hierarchy.

From Figure 4.1 above, we could see that in case of a hierarchical topology (such as A3) the sub-units which encode information on a higher level had lower activations when compared to the sub-units that were receiving the raw image as inputs. A possible explanation could be that as we are compressing the original inputs at each level of the hierarchy, fewer number of neurons get fired or have lower activations in the hidden layers of the sub-units at each successive level up the hierarchy.

The different levels of activations through the epochs had an influence on the estimated entropy as we can see an increasing trend for the most part in the Figure 4.2 below (for more refer to Appendix A). As the receptive fields of the sub-units for the different architectures have access to different quality of inputs, a few deviations from this trend are also observed. We can also observe from the figure that the lower level sub-units receiving the raw inputs from the image had much higher entropic value than their higher level counterparts. We observed a similar trend between the local entropies and the number of epochs of training with the Kraskov (KNN) estimator as well.

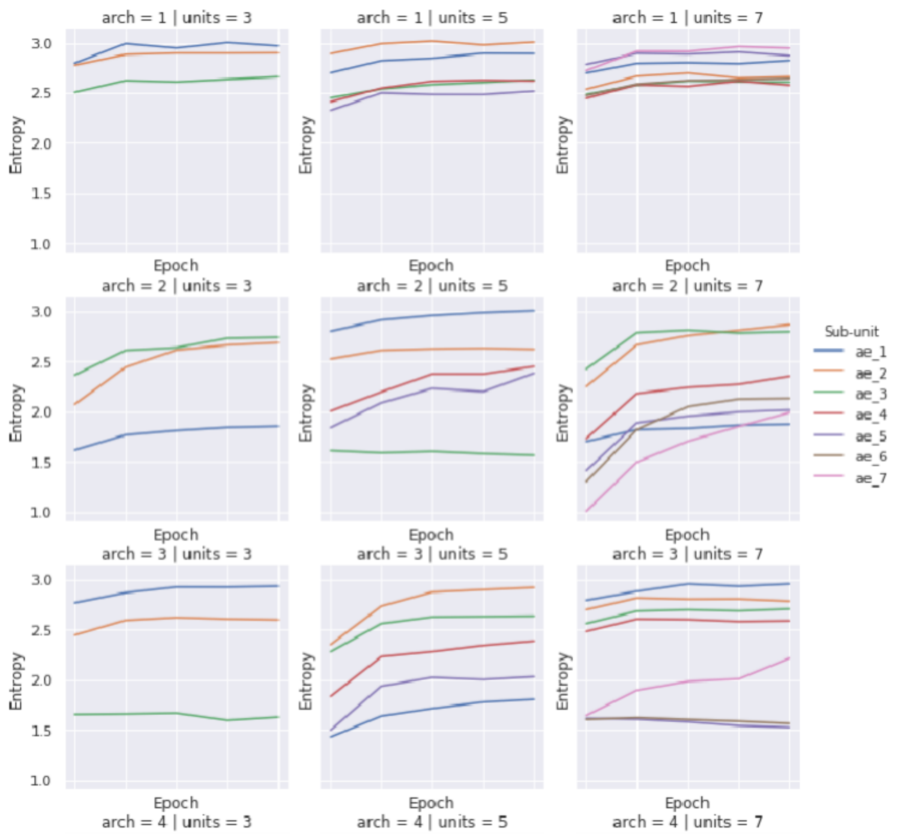


Figure 4.2: Estimated entropy of the sub-units through the epochs. Here the the sub-units are ordered in the legend by their level within the hierarchy.

Figure 4.3 below also shows the contrast between the hidden layer activations through the epochs, as the feature maps during the early epochs of training had much lower values of activations when compared to the feature map from towards the end of training (final epoch).

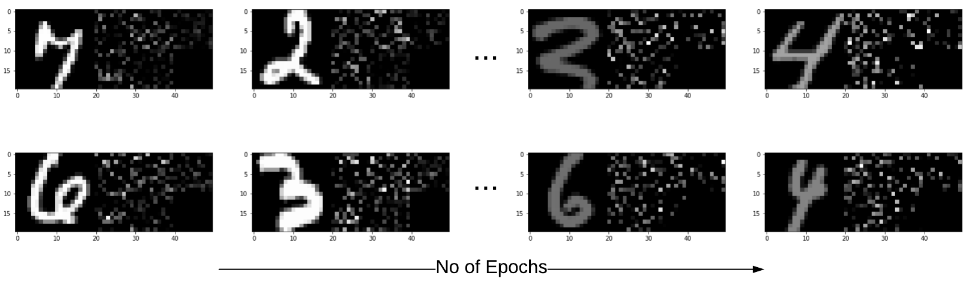


Figure 4.3: Feature maps of a B5 (Asymmetric Pyramidal) topology through the epochs during training. The first two columns are from the first epoch and the latter two are from the final epoch. We can notice the magnitude of activations by the relative *lightness* of the input image with the encoded regions (the input values remain the same within the range $[0, 1]$).

4.4 Correlation Analysis

The data generated from training the handcrafted and the randomly initialized architectures above were used for performing a correlation analysis between the estimated entropy of the hidden layer activations in the sub-units and the final accuracy.

In order to capture the distribution of the entropy of all the sub-units within each topology, we derived the summary statistics for the same, such as the mean, median, range, maximum and minimum of all the local entropies for each run. Thereafter, we computed the *Pearson's correlation coefficients* (ρ) to get the strength of the correlation between the derived summary statistics of the entropy and accuracy of the classifier trained on the corresponding architecture. For the purposes of our analysis, we considered the histogram based entropy estimates and got the results as per Table 4.3 below.

Intrinsic metric (entropy)	ρ
Median	0.709904
Maximum	0.755751
Mean	0.706967

Table 4.3: Top 3 most strongly correlated intrinsic metrics with classification accuracy.

Hence, we observed that the median, maximum and mean of all local entropies, estimated over the hidden layer activations in the sub-units, were most strongly and positively correlated with the final accuracy of the classifier. All three indicated that greater the local entropies of the sub-units, more useful information was captured in the encoded representation. Hence, seeking out regions of high activity on the feature map by the sub-units via their receptive fields should lead to better encoding and subsequently higher supervised classification accuracy. We also observed a similar correlation with accuracy using the KNN-based entropy estimates.

Figure 4.4 below shows all the datapoints which were used in the correlation analysis. We also fitted a regression line to the data points in Figure 4.5 below to explore a linear relationship between median entropy and accuracy.

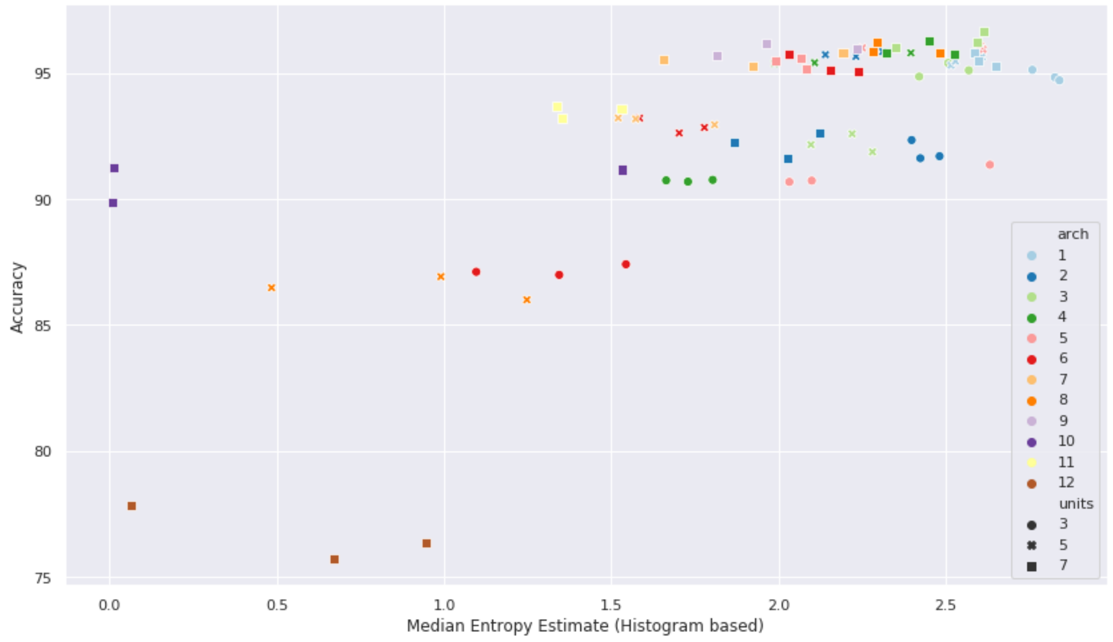


Figure 4.4: Plot showing Median Entropy Estimate vs Accuracy. The type of topology is indicated by the three different shapes and each architecture is depicted by a color.

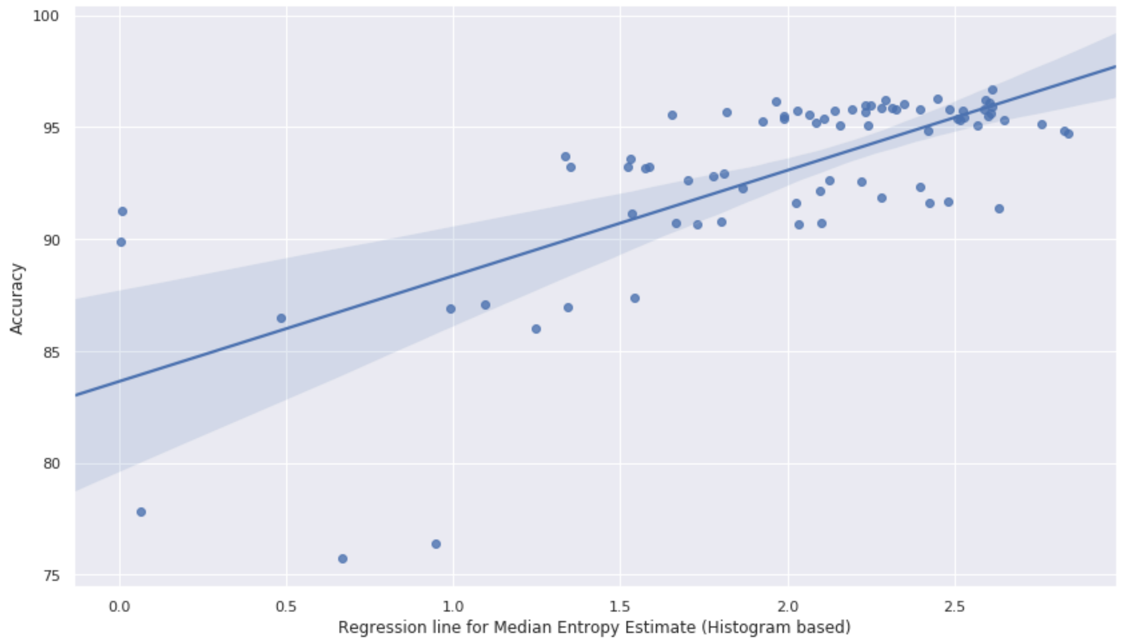


Figure 4.5: A Regression line fitted between accuracy and median entropy for all types of topologies.

Similar trends were observed for each type of topology as per Figure 4.6 below. Hence, we concluded that

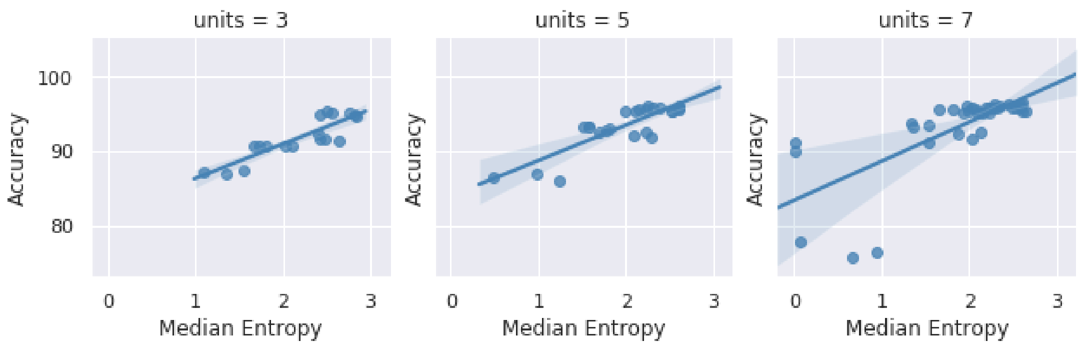


Figure 4.6: Regression lines fitted between accuracy and median entropy for each type of topology. For type C topologies, there was a higher uncertainty towards the lower entropy values due to fewer datapoints.

by using either the median or maximum entropy estimate for α in the meta-rule formulated in Section 3.11, we could potentially create self-organising, meta-learned architectures from randomly initialized ones.

4.5 Meta-learned Architectures

Using the meta-rule(s) implemented in Section 3.11 with α as the *median* entropy estimate, we performed a total of 12 runs and recorded how the topology evolved over the time as the receptive fields were optimized during the training of the meta-network. For the experiments, we *tied* the two receptive fields of each sub-unit together, forming a continuous patch of $2 \times 10 \times 10$ dimensions per sub-unit. Also no change in the orientation and size was allowed during the meta-learning as this would introduce additional meta-parameters. Hence, the resulting system had only two degrees of freedom i.e. the horizontal and vertical direction to shift the receptive field (r_k) for the k^{th} sub-unit within the feature map.

For the initial evaluation, we started with the random search based meta-rule and then modified the same to be a stochastic hill climber. We report the performances of both the randomly initialized architecture and the meta-learned architecture of the same topology discovered through the above strategies in Table 4.4 below.

Topology	Units	Meta-steps	Initial accuracy	Final accuracy
M1	3	3	89.32	91.90
M2	3	5	80.69	92.13
M3	3	3	85.79	93.70
M4	3	1	93.64	93.81
M5	3	5	43.87	81.91
M6	3	4	85.96	85.61
M7	3	5	65.03	83.27
M8	5	1	89.11	94.75
M9	5	5	93.31	94.77
M10	5	3	95.43	95.33
M11	7	3	92.90	95.85
M12	7	1	95.73	95.20

Table 4.4: The different performances of meta-learned architectures. The accuracy of the randomly initialized architecture for each case along with the final accuracy of the meta-learned architecture are given. Meta-steps indicates the number of receptive field updates before termination of the meta-learning loop.

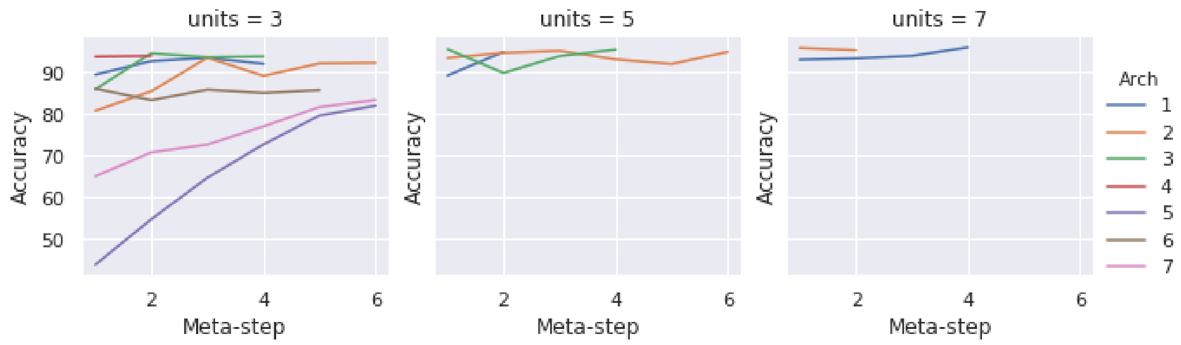


Figure 4.7: Plot showing the change in accuracy of the architecture with each meta-step. Different randomly initialized architectures are represented by different colors.

We observed an increase in the accuracy of the architecture after just a few hill climbing steps (referred here as meta-steps). Hence, the meta-rule based on entropy was effective in optimizing the receptive field locations within the feature map. We show the performances of the different intermediate architectures during meta-learning in Figure 4.7 above.

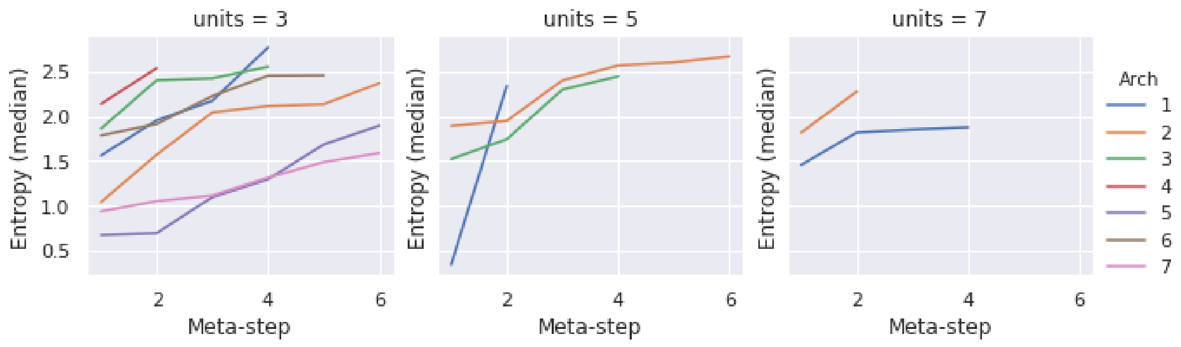


Figure 4.8: Plot showing the change in α (median entropy) of the different topologies with each meta-step.

For the convergence condition of the hill climber, we used a maximum number of meta-steps M as 5 along with a maximum number of random perturbations P as 10 that was allowed before continuing the next hill climbing iteration. This helped impart robustness to the search algorithm and terminate it in a reasonable number of iterations. Also, since the convergence of the meta-learning is sensitive to initialization of the receptive fields, random-restarts helped in the wider exploration of the search space.

From Table 4.4 above, we can also note that most of the meta-learned architectures such as M1, M2, M3, M5, M7, M8 and M11 have optimized their receptive field configuration given their original random topologies as evident from the boost in their performances. It is interesting to note however that random configurations which already yielded a very high accuracy to begin with such as M4 and M9 (and hence had good encoded representations) had comparatively smaller increase in their performances. We can also observe these trends reflected in Figure 4.7. Similarly in some cases such as M6, M10 and M12 we observe a slight decrease (≤ 0.5) in performance. This is likely due to their initial configurations being more prone to getting stuck to a local maxima than the others. As we noted before, both the initial placement and size of the receptive fields play a vital role in the same (for visualizations of feature map with the receptive fields refer to Appendix A).

We also observed that the meta-learning led to the sub-units targeting areas in the feature map that yielded higher activity in their hidden layers and hence captured more information about the inputs. Hence it essentially led to the *redistribution* of the resources to the underutilized areas. By increasing the number of meta-steps and random perturbations we could potentially obtain even greater performances, close to the perfectly hierarchical handcrafted topologies.

4.6 Summary

We started by evaluating the performance of the 17 different handcrafted architectures along with 9 randomly initialized ones. We found that architectures with properties of a perfect hierarchy yielded the best results closely followed by ones which slightly deviated from the same. We also found that symmetry played a crucial role in effective integration of information in the resulting hierarchy.

We observed that as the number of epochs increased during training, estimated entropy of the hidden layer activity increased as well. We also observed from the feature maps during training that higher level sub-units had lower estimated entropy than the lower level ones.

Based on the correlation analysis between different summary statistics of entropy with accuracy, we observed that by maximizing the median of all the local entropies of a meta-network via the meta-learning rule, we were able to progressively obtain better architectures which captured more relevant features of the input images. Hence simple meta-heuristic based optimization like stochastic hill climbing and even random search provided a good meta-rule and yielded better performing meta-learned architectures when compared to their completely random initial architectures.

Chapter 5

Research Conclusion

The current trends in deep learning signal a paradigm shift towards the direction of automated architecture search methods like NAS (Zoph & Le, 2016) along with meta-learning based approaches to rely less upon explicit *tailoring* of neural network architectures for a given task. This is an extremely important step towards achieving general purpose deep learning algorithms which can adapt and generalize well over multiple domains as we briefly discussed in Chapter 1.

We conclude with a discussion on the implications of our experimental results from the last chapter. We showed that by taking inspiration from the biological brain which we outlined in Section 1.2.2, a self-organising hierarchical architecture which progressively adapts and improves its topology could provide a potential alternative solution along with NAS for the problem of automated architecture search.

5.1 Research Summary

We begin by addressing the original research questions formulated in Section 1.3.2. In particular, on (1):

- We developed a meta-learning rule based on *entropy* of the encoded representations of the input. To facilitate the same, we developed an abstraction of the cortical map (feature map) and implemented a meta-network in which single layer, undercomplete autoencoders acted as the sub-units and its topology completely defined by the receptive fields of the same. We also conceived several different handcrafted and random architectures to investigate for any potential intrinsic metric correlated with accuracy and also to study the effects of the hierarchy on the learned representations. From Section 4.1, we saw topologies with symmetric, pyramidal structures performed the best, followed closely by the slightly hierarchical ones and concluded that hierarchy played a crucial role in integration and learning of high level information overall. From Section 4.5, we observed that randomly initialized architectures driven by an entropy-based meta-learning rule, self-organised itself into better performing architectures.
- With regards to (a), we found a strong positive correlation between accuracy and the *median* of all the local entropies in a meta-network, which allowed us to infer about the latent representations across the feature map and subsequently its performance, without needing to train and evaluate every single architecture in the search space. We also explored different meta-heuristics, in particular random search and variants of the hill climbing algorithm, such as stochastic hill climber and random-restart hill climber, to maximize the median entropy of the meta-network. This entropy maximization scheme became the objective function for our meta-learning rule to learn optimal configurations of the receptive fields.

- For (b) we parameterized the meta-learning rule with the *receptive field location* (r_k). The number of meta-parameters depended on the number of sub-units (K) within the meta-network. Other potential candidates which we considered were the receptive field size and its orientation. For the purposes of our experiments, we simplified the optimization problem by only focusing on a single meta-parameter to drive the learning.
- For (c), the meta-rule updated the receptive fields at a much slower rate than the base learning. This was due to the fact the sub-units were trained for every perturbation of the receptive fields (required for estimation of the entropy), whereas the meta-updates only took place *if* a perturbation led to better entropic values. The number of times the sub-units were trained depended, to a large extent, on the meta-heuristic's ability to quickly find good optimas along with the intial conditions. Hence, the overhead added by the meta-rule was proportional to the total number of perturbations of the receptive fields.
- For (2), we observed from both Section 4.1 and Section 4.5, that the meta-learned architectures provide competitive performance on MNIST as the handcrafted ones, given the same amount of resources.

We also observed that the meta-rule led to receptive fields targeting regions which yielded higher hidden layer activity in the sub-units. This led to an emergent hierarchy from the meta-learning and the integration of the receptive fields became the recurrent connectivities between the self-organising units.

5.2 Research contributions

The potential contributions of our research outcomes are two-fold :

1. Formulation of an entropy based meta-learning rule and,
2. Self-organising hierarchical architectures based on the same

As we discussed initially in Section 1.1, designing of neural network architectures, hyper-parameter tuning and other forms of architectural engineering are the de facto standards for driving state-of-the-art results. These approaches however require a lot of resources in terms of manual effort, compute and time between iterations of neural architectures to push the performance in some domain. We argue that a self-organising, meta-learning based architecture has the potential to drastically reduce both time and effort for the same.

We drew inspiration from the neocortical organisation and using visual modality, in particular, highlighted this different approach to solving the manual architectural problem. We showed that by identifying intrinsic metrics which inform us about the quality of the representations learned by a network during training, we can effectively optimize the architecture of the network. The adaptive and evolving nature of the topology could also potentially lead to generalizing well over to other tasks and modalities. We also observed that the meta-learning process led to emergent properties like self-organisation and better integration of receptive fields in the meta-learned topologies.

One of the main contrasting features between NAS and our method is that the latter does not make use of an extrinsic metric, like accuracy, to directly inform the architecture search process. As we discussed in Section 2.5, NAS methods (Zoph & Le, 2016) employ novel strategies to evaluate candidate architectures on a test set which remains a huge computational challenge. Using an information theoretic approach, we demonstrated that it is possible to leverage intrinsic metrics that are indicative of the learned latent representations, optimizing which leads towards better feature extraction, without evaluating every candidate architecture in the search space.

Another potential contribution as we briefly mentioned in Section 1.5 could be towards the democratization of deep learning as a tool and greater adoption across different industries.

5.3 Future scope

Future work could build upon the project by exploring other sensory modalities, in addition to visual modality, with a similar self-organising and hierarchical architecture (Project AGI & The Whole Brain Architecture Initiative, 2018). Much like how different sensory modalities integrate in the human brain, we could further explore the integration of receptive fields of the different modalities with a similar framework. It would be also extremely interesting to see if such an approach would translate well over to sequence based tasks.

We could also carry out more experiments with just a single modality and vary the termination criteria for the meta-learning rule to explore how far the overall performance improves and if it eventually convergences. Experiments on data sets such as CIFAR-10 and CIFAR-100 should also provide better insights about its performance on a multi-channel, larger scale setting.

We could also investigate gradient based approaches with differential entropy and potentially improve the efficiency of the meta-learning algorithm. Meta-heuristics like local search are a great place to start with but the meta-learning rule would need to be more robust and efficient to converge in a minimal amount of time. Other advanced meta-heuristics such as Simulated Annealing could also be explored.

A future extension of our project could potentially incorporate some of the above and create an end-to-end automated neural architecture pipeline for ease of experimentation and testing the self-organising architectures on different benchmarks.

References

Alonso, J.-M., & Chen, Y. (2009). Receptive field. *Scholarpedia*, *4*(1), 5393.

Bengio, Y., Bengio, S., & Cloutier, J. (1990). *Learning a synaptic learning rule*. Université de Montréal, Département d'informatique et de recherche .

Boehmke, B., & Greenwell, B. M. (2019). Hands-on machine learning with r.

Brownlee, J. (2011). *Clever algorithms: nature-inspired programming recipes*. Jason Brownlee.

Corominas-Murtra, B., Goñi, J., Solé, R. V., & Rodríguez-Caso, C. (2013). On the origins of hierarchy in complex networks. *Proceedings of the National Academy of Sciences*, *110*(33), 13316–13321.

Denouden, T., Salay, R., Czarnecki, K., Abdelzad, V., Phan, B., & Vernekar, S. (2018). *Improving reconstruction autoencoder out-of-distribution detection with mahalanobis distance.*

Dora, S., Pennartz, C., & Bohte, S. (2018). A deep predictive coding network for learning latent representations.

Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., & Leyton-Brown, K. (2013). Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *Nips workshop on bayesian optimization in theory and practice* (Vol. 10, p. 3).

Elsken, T., Metzen, J. H., & Hutter, F. (2018). Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*.

Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th international conference on machine learning-volume 70* (pp. 1126–1135).

Friston, K. (2010). The free-energy principle: a unified brain theory? *Nature reviews neuroscience*, *11*(2), 127.

Gers, F. A., Schmidhuber, J., & Cummins, F. (1999). Learning to forget: Continual prediction with lstm.

Goldberg, D. E., Richardson, J., et al. (1987). Genetic algorithms with sharing for multimodal function optimization. In *Genetic algorithms and their applications: Proceedings of the second international conference on genetic algorithms* (pp. 41–49).

Goldberg, E. (1989). Gradiental approach to neocortical functional organization. *Journal of Clinical and Experimental Neuropsychology*, *11*(4), 489–517.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, *313*(5786), 504–507.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Hochreiter, S., Younger, A. S., & Conwell, P. R. (2001). Learning to learn using gradient descent. In *International conference on artificial neural networks* (pp. 87–94).

Kandel, E. R., Schwartz, J. H., Jessell, T. M., of Biochemistry, D., Jessell, M. B. T., Siegelbaum, S., & Hudspeth, A. (2000). *Principles of neural science* (Vol. 4). McGraw-hill New York.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Koch, G., Zemel, R., & Salakhutdinov, R. (2015). Siamese neural networks for one-shot image recognition. In *Icml deep learning workshop* (Vol. 2).

Kolchinsky, A., & Tracey, B. (2017). Estimating mixture entropy with pairwise distances. *Entropy*, *19*(7), 361.

Kraskov, A., Stögbauer, H., & Grassberger, P. (2004). Estimating mutual information. *Physical review E*, *69*(6), 066138.

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., & Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems* (pp. 396–404).

LeCun, Y., & Cortes, C. (2010). MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/. Retrieved 2016-01-14 14:24:11, from `http://yann.lecun.com/exdb/mnist/`

Liu, H., Simonyan, K., Vinyals, O., Fernando, C., & Kavukcuoglu, K. (2017). Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*.

Mahalanobis, P. C. (1936). On the generalized distance in statistics..

Makhzani, A., & Frey, B. (2013). K-sparse autoencoders. *arXiv preprint arXiv:1312.5663*.

Metz, L., Maheswaranathan, N., Cheung, B., & Sohl-dickstein, J. (2019). Meta-learning update rules for unsupervised representation learning.

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., & Khudanpur, S. (2010). Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association.*

Mohammadi, M., Al-Fuqaha, A., Sorour, S., & Guizani, M. (2018). Deep learning for iot big data and streaming analytics: A survey. *IEEE Communications Surveys & Tutorials*, *20*(4), 2923–2960.

Nielsen, M. A. (2015). *Neural networks and deep learning* (Vol. 25). Determination press San Francisco, CA, USA:.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., ... Lerer, A. (2017). Automatic differentiation in pytorch.

Project AGI, & The Whole Brain Architecture Initiative. (2018, Nov). *Rfr: Self-organising architectures.* Retrieved from `https://wba-initiative.org/en/research/rfr/rfr-self-organising-architectures/`

Purwani, S., Nahar, J., & Twining, C. (2017). Analyzing bin-width effect on the computed entropy. In *Aip conference proceedings* (Vol. 1868, p. 040008).

Ranpura, A., & Fitzgerald, M. (2019, Oct). *Overview of receptive fields.* Retrieved from `https://brainconnection.brainhq.com/2004/03/06/overview-of-receptive-fields/`

Rao, R. P., & Ballard, D. H. (1999). Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature neuroscience*, *2*(1), 79.

Ravi, S., & Larochelle, H. (2016). Optimization as a model for few-shot learning.

Rawal, A., & Miikkulainen, R. (2018). From nodes to networks: Evolving recurrent neural networks. *arXiv preprint arXiv:1803.04439*.

Real, E., Aggarwal, A., Huang, Y., & Le, Q. V. (2019). Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 33, pp. 4780–4789).

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., … Kurakin, A. (2017). Large-scale evolution of image classifiers. In *Proceedings of the 34th international conference on machine learning-volume 70* (pp. 2902–2911).

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Russell, S. J., & Norvig, P. (2016). *Artificial intelligence: a modern approach.* Malaysia; Pearson Education Limited,.

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., & Lillicrap, T. (2016). Meta-learning with memory-augmented neural networks. In *International conference on machine learning* (pp. 1842–1850).

Saxe, A. M., Bansal, Y., Dapello, J., Advani, M., Kolchinsky, A., Tracey, B. D., & Cox, D. D. (2018). On the information bottleneck theory of deep learning.

Schmidhuber, J. (2007). Gödel machines: Fully self-referential optimal universal self-improvers. In *Artificial general intelligence* (pp. 199–226). Springer.

Shwartz-Ziv, R., & Tishby, N. (2017). Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*.

Snell, J., Swersky, K., & Zemel, R. (2017). Prototypical networks for few-shot learning. In *Advances in neural information processing systems* (pp. 4077–4087).

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929–1958.

Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, *10*(2), 99–127.

Swenson, R. S., & Gulledge, A. (2017). The cerebral cortex. In *Conn's translational neuroscience* (pp. 263–288). Elsevier.

Tishby, N., Pereira, F. C., & Bialek, W. (2000). The information bottleneck method. *arXiv preprint physics/0004057*.

Tishby, N., & Zaslavsky, N. (2015). Deep learning and the information bottleneck principle. In *2015 ieee information theory workshop (itw)* (pp. 1–5).

Vanschoren, J. (2018). Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*.

Ver Steeg, G. (2000). Non-parametric entropy estimation toolbox (npeet).

Vilalta, R., & Drissi, Y. (2002). A perspective view and survey of meta-learning. *Artificial intelligence review*, *18*(2), 77–95.

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on machine learning* (pp. 1096–1103).

Wei, T., Wang, C., Rui, Y., & Chen, C. W. (2016). Network morphism. In *International conference on machine learning* (pp. 564–572).

Weng, L. (n.d.). *Meta-learning: Learning to learn fast.* Retrieved from `https://lilianweng.github.io/lil-log/2018/11/30/meta-learning.html`

Zoph, B., & Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 8697–8710).

Appendix A

Appendix

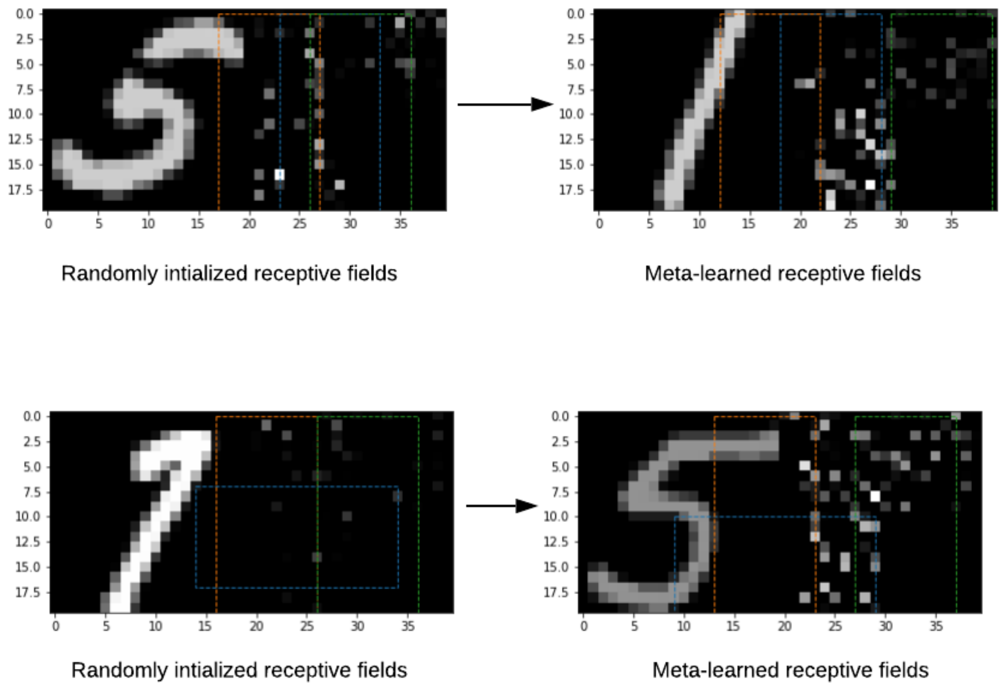


Figure A.1: The receptive fields during meta-learning (Type A topology setup). The left column shows the initial configuration and the right column shows the updated one after a few meta-steps. The colors indicate the receptive fields of the different sub-units. Stochastic hill climbing was used as the optimization algorithm with α as the median entropy.

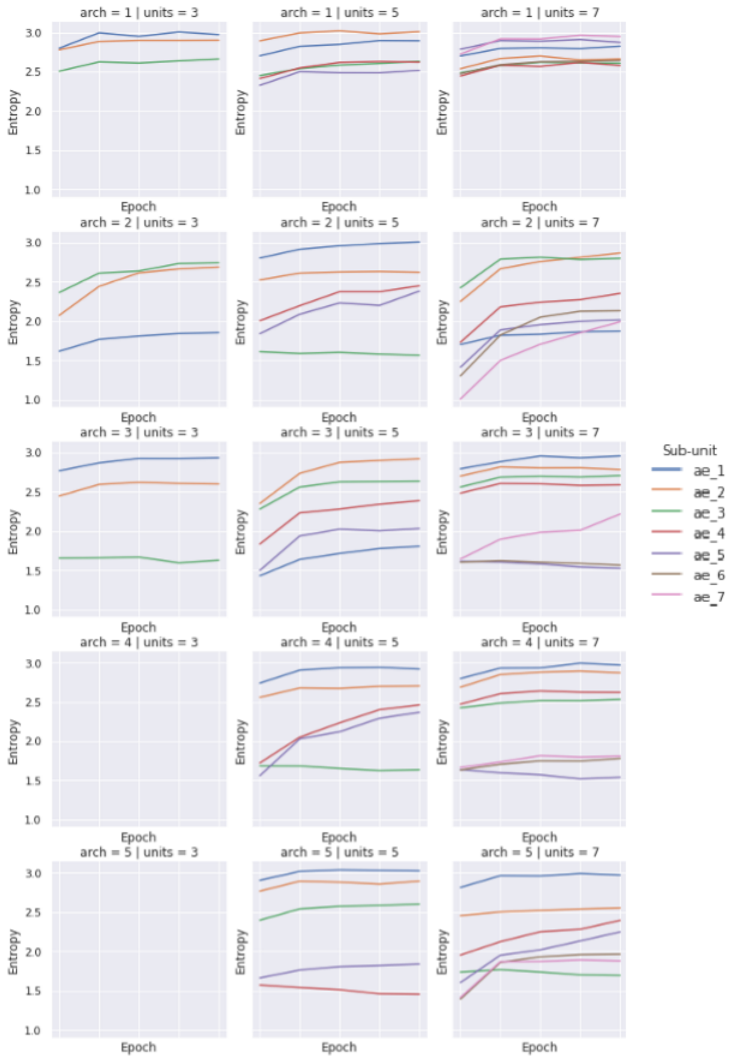


Figure A.2: Entropy vs Epochs of training in different topologies (part a)

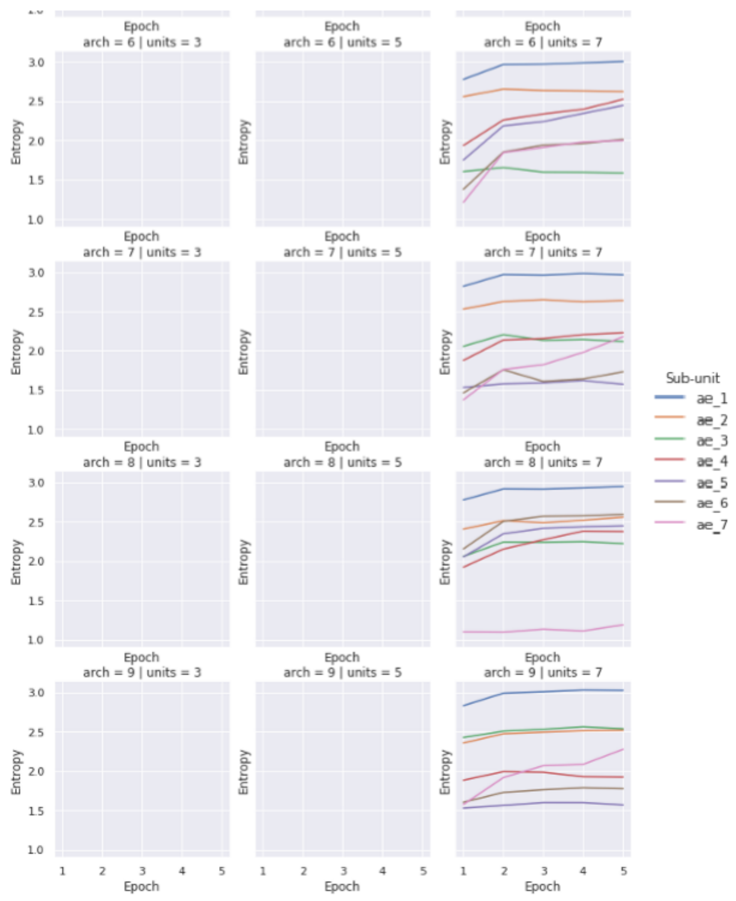


Figure A.3: Entropy vs Epochs of training in different topologies (part b)

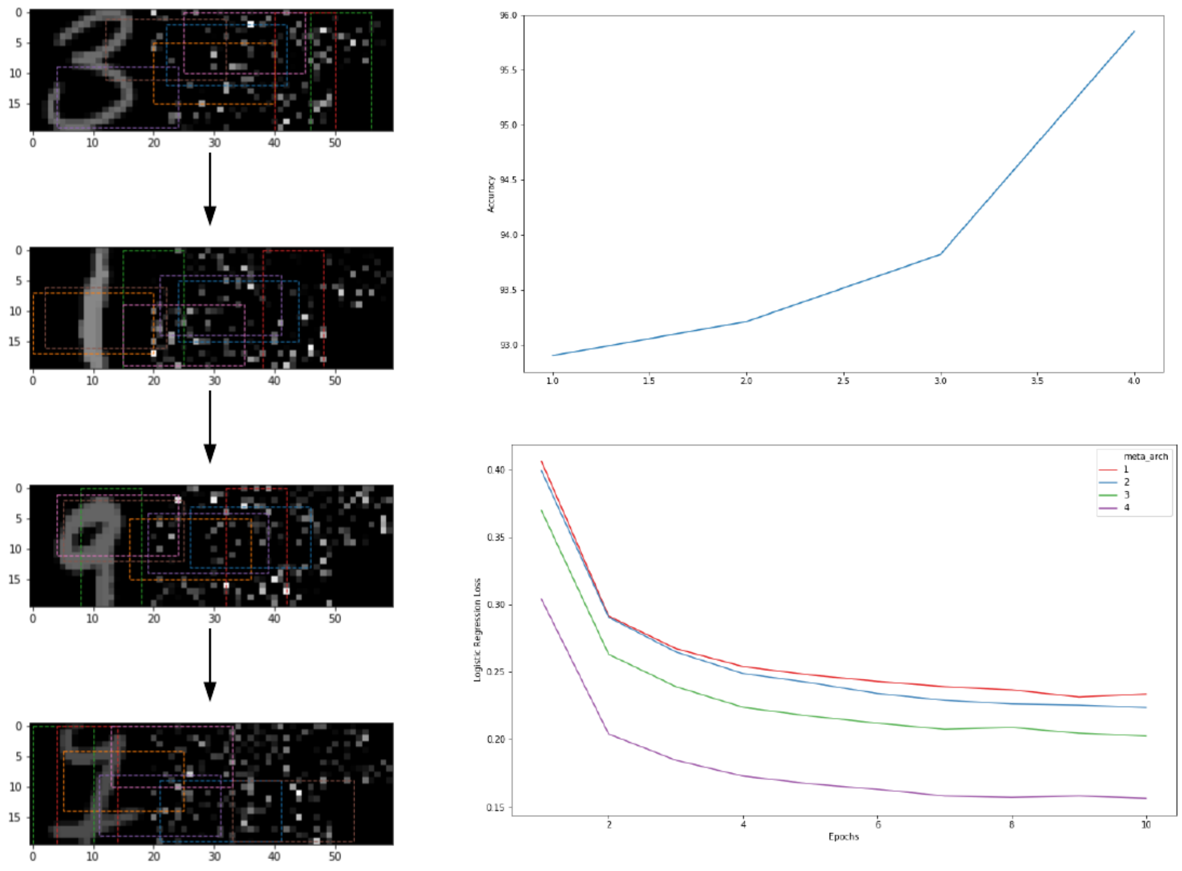


Figure A.4: The feature maps on the left shows the different iterations of the architecture through the meta-steps (Type C setting). On the top right corner, the accuracy of the classifier at each meta-step i.e. trained on m^{th} architecture is shown. On the bottom right corner the logistic regression losses for the same is displayed. Each box in the feature map represents one sub-unit's receptive field. Random search was used as the optimization algorithm with α as the median entropy.