- Image Categorisation Using latent representations Obtained From Predictive Coding Models -

By: Louis Mitchener

Supervisor: Dr Shirin Dora

Department of Computer Science

Loughborough University

June 2022

Computer Science and Artificial Intelligence

Thesis Project

21COD290

B821780

Abstract

The field of research surrounding predictive coding has expanded greatly since the first examples of computational predictive models produced over 20 years ago. However, the property of deep neurons within these models achieving successful encoding of categorically identifiable information has not been researched until now.

By encoding images to be categorised by means of a predictive coding model, a latent representation of each of these images can be obtained with a lower size than the original image, while still retaining enough information about the unique features of each image, such that the original can be reconstructed.

In this project, a predictive coding model is used to summarise the features of an image with 784 points of data, in a representation with 10 points of data. A set of these representations is used to train a support vector machine for a categorisation task, and the resulting categorisations are compared with the categorisations of a support vector machine trained on the original images.

Although the accuracy of categorisation saw a slight drop when compared to the categorisation of original images, the validity of performing categorisations using latent representations of images produced via predictive coding was proven. Further research could find ways in which to optimise this problem, resulting in even more accurate categorisations.

This project shows that the categorisation of the latent representation of these images is possible and results in a high degree of categorisation accuracy.

Acknowledgements

I would like to thank Dr. Shirin Dora for his assistance and theoretical expertise with predictive coding throughout the duration of this project.

Contents

1 Introduction **5**

1.1 Project Overview . . . 5
1.2 Introduction to Predictive Coding . . . 5
1.3 Real-World Applications . . . 6
1.4 Hypothesis . . . 6
1.5 Project Outcomes . . . 6

2 Literature Review **6**

2.1 Predictive Coding in Context . . . 6
2.1.1 The Neuroscience Behind Predictive Coding . . . 7
2.1.2 From Neuroscience to Neural Networks . . . 7
2.1.3 Recent Developments in Predictive Coding . . . 8
2.2 Image Classification . . . 9
2.2.1 Support Vector Machines . . . 9
2.3 Literature Review Conclusion . . . 10

3 Methodology **10**

3.1 Development Platform . . . 10
3.2 Proposed System Structure . . . 11
3.2.1 Predictive Coding Model . . . 11
3.2.2 Classifier . . . 11
3.3 Dataset . . . 11
3.4 Development Method . . . 11
3.5 Experimental Process . . . 12

4 Design **12**

4.1 Complete System Structure . . . 12
4.2 Predictive Model . . . 14
4.3 Logic of the Training Loop . . . 15
4.3.1 Forward Pass . . . 15
4.3.2 Gradient Descent on Model Weights . . . 17
4.3.3 Gradient Descent on Nodes in Layer Y . . . 19
4.3.4 Testing Loop . . . 20
4.3.5 Additional consideration of the predictive model . . . 20
4.4 SVM classifiers . . . 21
4.5 User Interface Considerations . . . 21

5 Implementation **21**

5.1 Dataset Preprocessing . . . 22
5.2 Implementation of the Predictive Model . . . 22

5.2.1 Training Function and Loop . . . 23
5.2.2 Testing Function and Loop . . . 25
5.2.3 Perfecting the Model Parameters . . . 27
5.3 Encoding the Dataset for the SVM . . . 27
5.4 Training the Support Vector Machines . . . 30
5.5 User Interface . . . 31

6 Results **33**
6.1 Results of the Classifications . . . 33
6.1.1 MNIST Classifications . . . 33
6.1.2 Latent Representation Dataset Classifications . . . 33
6.2 Evaluation of Results . . . 34

7 Project Evaluation **35**
7.1 Future Additions to the Project . . . 35
7.2 Directions for Future Research . . . 35
7.3 Conclusion . . . 36

List of Figures

1 The architecture of a predictive model as described by R.P Rao [1] . . . 5
2 Wireframe of the relationships between each section of the proposed system . . . 13
3 Wireframe of the Predictive Model implemented in this project, where Y indicates the deeper neural layer, X indicates the output of the network and W indicates the weights of each neural connection between the deep layer and output layer. . . 14
4 Wireframe to demonstrate how the value at a single node in X is calculated, as well as a prediction error. . . . 16
5 A single neural connection between a neuron in the deep layer and output layer. 18
6 An example of the neural pathways that must be considered when updating the value of a single node Y. . . . 19
7 Retrieving the MNIST data . . . 22
8 Preprocessing of the dataset . . . 22
9 The decleration for the structure of the Predictive model . . . 22
10 Functions used to calculate output, error, and derivatives. . . . 23
11 The training function of the Predictive Model . . . 24
12 The main training loop of the Predictive Model . . . 25
13 The testing function of the Predictive Model . . . 26
14 The testing loop of the Predictive Model . . . 26
15 Demonstration of the trained predictive model reconstructing MNIST images from the MNIST test set . . . 28
16 Loop for converting a selected number of images from the MNIST test set into their encoded representation using the trained predictive model . . . 29

17 Example of an encoded representation used to generate a reconstruction of an MNIST image. 29
18 Preprocessing of MNIST testing set for SVM classifier. 30
19 Preprocessing of Latent Representation Dataset testing set for SVM classifier. . . 30
20 Training and testing the SVM on one of the datasets, then saving then producing a confusion matrix of the results. 31
21 Code for the command line interface of the Predictive Model python file 32
22 Code for the command line interface of the EncodeYvalues python file 32
23 Code for the command line interface of the SVM python file 33

List of Tables

1 Confusion matrix showing the predictions produced by the SVM on MNIST . . . 34
2 Confusion matrix showing the predictions produced by the SVM on the Latent Representation Dataset 34

1 Introduction

1.1 Project Overview

Predictive coding[1] has been a highly researched topic within artificial intelligence over the past 20 years. One property of predictive coding models that has seen little research is of their ability to encode categorical information about image data. In this project, this property of predictive coding models, as well as the properties and performance of encoded data obtained from such models is evaluated. If this property can be measured, it could open the door to new ideas about the ways in which data is encoded in both predictive coding models and the visual cortex of the brain.

In order to test this property, predictive coding is used to obtain latent representations of a set of images. This is achieved by training a model using theories presented by predictive coding, in order to obtain a set of values of a much smaller size than the original image.

This set of images is then used to train a support vector machine in order to validate that each representation of an image retains an acceptable degree of information, such that it can be categorised accurately.

1.2 Introduction to Predictive Coding

Predictive coding, as described by R.P. Rao [1], outlines a model by which visual processing in the brain can be summarised in a computational model by a series of high to low-level connections, where deeper layers in a network describe neuron activity that in turn can predict the activity of higher levels. These networks attempt to create an internal model of the input presented to it by means of carrying a prediction error between layers, that allow local predictive models to predict the activity of their connected layer. Figure 1 shows the architecture of the model presented in Rao's paper, in which high-level predictors predict the activity of lower levels, and the error of these predictions is carried from low to high-levels.

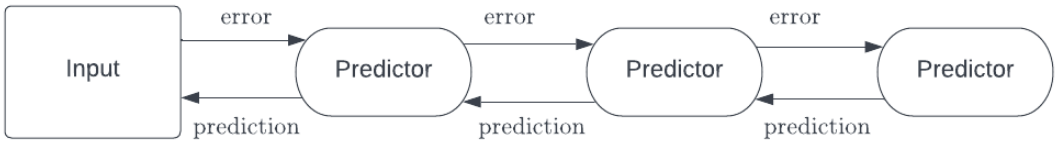

Figure 1: The architecture of a predictive model as described by R.P Rao [1]

When a given predictor makes a prediction of the activity at a lower level in the model, a prediction error is generated, which is used to adjust the predictor to make more accurate predictions. Through iterative training, each of these high-level predictors can be trained to make more accurate predictions of the activity in lower layers, creating a model that can predict the activity of a given input.

In practice, simple predictive coding models differ from traditional Neural Networks, in that instead of having an input and output in the traditional sense, they are trained to converge to

an input by means of training weights and neurons on deeper levels of a network to generate the data presented by this input.

1.3 Real-World Applications

This project provides an alternate means by which to perform image categorisation tasks. This extends to real-world applications such as dataset analysis, traffic analysis, and medical imaging; any task requiring the discrete categorisation of images, based on a predefined set of categories. Image Categorization in general requires a neural network with a large number of weights and neurons. By utilising predictive coding, this can be minimised to some extent. This lends this method to applications that require the use of less powerful lightweight hardware with reduced memory, as it minimises the overhead on the amount of simultaneous memory required to perform the algorithms required to obtain latent representations.

As well as these practical uses, the results of this project confirm a previously unmeasured property of predictive models, which could lead to new and further research in this area of artificial intelligence research.

1.4 Hypothesis

Deep neurons in predictive coding models have been shown to respond differently to different stimuli [1]. By extension, the responses of neurons to stimuli of the same category should be measurably similar. By training a neural network for categorisation on values obtained from these deeper neurons, the network should be able to successfully perform categorisation.

1.5 Project Outcomes

This project aims to produce a method to train a predictive coding network in order to obtain a set of latent representations from a dataset of images for use in image categorisation.

The goal of the predictive model is to obtain a set of weights that can reliably reconstruct an image from a dataset of images. A number of configurations for the predictive coding network will be tested in order to obtain an optimally performing trained network while minimizing the dimensions of the deep layer of the network. Once this configuration is obtained, the final model produced will be used to encode a dataset into a new dataset of latent representations.

This new dataset will then be trained on an image classification network, and compared to the performance of the classification network when trained on the original dataset. This comparison will be performed in order to confirm the validity of the latent representations as a means to quantify abstract details of an image, while maintaining a suitable degree of categorical uniqueness, such that they can be successfully classified with an acceptable degree of accuracy.

2 Literature Review

In order to fulfil the objectives set out by this project, research into past studies of both predictive coding as well as image categorisation has been undertaken.

2.1 Predictive Coding in Context

Predictive coding emerged as a result of studies that observe the way in which the visual cortex of the brain processes visual information.

2.1.1 The Neuroscience Behind Predictive Coding

The initial theories behind predictive coding originate from studies that aimed to find models that describe the way in which the brain propagates information based on retinal stimulus. Preliminary observations within the visual cortex of the cat by David H. Hubel, and Torsten N. Wiesel [6] proved that neural responses undergo modification within neural pathways when the retina is exposed to specific stimuli. This study measured the activation of specific neural areas within the brains of cats when exposed to different patterns of light, and found evidence that the propagation of neural signals between spatial regions of the brain had unique responses to different shapes of light. Specifically, the study highlights that the activations did not just measure direct illuminations of an area, but also the gradient of illuminations between an area and its surroundings. A further study also carried out by David H. Hubel, and Torsten N. Wiesel [7] continues to validate these observations by measuring similar responses in the visual cortex of spider monkeys, which have an altered structure to that of cats while carrying similar signal propagations in the visual cortex of their brains.

These initial observations were described in more detail by Stephen Grossberg, Ennio Mingolla, and William D Ross [8] who observed with greater precision the behavior of neurons at different levels of the cerebral cortex. Through this research, Grossberg, Mingolla, and Ross produced a model which describes the specific function and relationships between cortical cells: "Cortical Columns" compete via both long-range signaling and short-range competition to learn patterns in visual signals. Importantly, this model confirms that visual signaling cannot be modeled accurately by a purely feed-forward hierarchy.

2.1.2 From Neuroscience to Neural Networks

The first description of a Neural Network based on the previously mentioned studies into the visual cortex was put forward by R.P Rao and D.H Ballard [1]. This was the first study to propose the hierarchical architecture that would go on to form the basis of most of the subsequent studies into predictive coding. The model outlined by this paper suggests a bidirectional propagation of information, where feed-forward error signals carry errors between predictions and deeper neural activity and signaling in the opposite direction carry predictions of previous neural layers. Note that in neuroscience, feed-forward signaling refers to the direction in which signals propagate through area of the brain. Areas of the brain are usually connected both ways and as such the direction of feed-forward signaling is contextual. Whereas feed-forward signaling in computing refers to the direction in which a given type of information propagates through a neural network, and is usually fixed. Rao's study validated that visual cortical neurons can be represented by residual error detectors between an input and its predictions from lower neural activity. This resulted in a model in which higher-level estimator layers attempt to predict lower-level activities, and the error in these predictions is calculated and used to correct the estimate on higher layers. This architecture allowed for multiple higher layers that each predict the activity of lower layers, forming a predictive hierarchy.

The receptive fields of each layer in the network increase in size further up the network's hierarchy. After training, by observing the properties of the receptive fields at each layer,

similarities can be drawn between the behaviours of these artificial receptive fields and the behaviours of real neural activity during the studies on cats and monkeys brains [6, 7] that this architecture was derived from. New properties of the training algorithm of the predictive coding network presented by Rao's research were also put forward. Due to the unique generative nature of this model, layers only have dependencies on their connected layers. This phenomenon was initially theorised by D. Mumford [9]. Using this model, training of neurons in each layer can be performed in parallel with other layers, with the most recent result of each layer used to inform updates within connected layers. In larger networks, this can greatly decrease training time, as training can be distributed across multiple pieces of hardware.

Rao and Ballard's paper described the input layer as the lowest layer in the network. However, this design is slightly changed for the implementation in this project. As described later in the implementation, an input image is presented at the highest layer of the network as opposed to being used as an input in the lowest layer. Their theory about the propagation of errors and predictions, however, is directly applied within the implementation of this project.

2.1.3 Recent Developments in Predictive Coding

The idea of the predictive model was built upon by Yan Karklin, and Eero Simoncelli [10], with a new type of model, "Efficient Coding". They introduce a number of other elements into the predictive coding framework. Whereas before the structure of a generic predictive coding model for visual signaling was fairly linear, the model proposed in efficient coding aims to incorporate more biologically realistic constraints and modifiers to elements of the network in order to increase the overall performance observed during testing. Karklin and Simoncelli's algorithm incorporates a factor described as a metabolic cost [11, 12] for firing spikes within the network. The underlying theory of this model is that the firing of certain neurons within a model carries more importance than other neurons, and as such, by eliminating the firing of irrelevant neurons, a more accurate and efficient model can be achieved. Furthermore, they aimed to optimise both the "linear receptive field and the nonlinear response properties" within their proposed network architecture. By applying a cost to the firing of neurons at each level in the network, they could obtain optimal neural pathways throughout the network, that minimise the cost-benefit trade-off of activating any particular spike.

In another paper, Bruno Olshausen and David Field [13] propose a model by which encoded inputs are measured along a scale spanning from negative to positive values. The model takes advantage of the concept of sparseness, which describes a model in which only a limited number of neurons are firing at a given time. This method maximises efficiency within a predictive network as it reduces "cross talk" [14] and better utilises available memory. This study also showed that when networks are trained using this framework of activity storage, the deep activity of the network more closely resembles that of simple neural cells.

All the previously discussed theories were brought together by Matthew Chalk, Olivier Marre, and Gasper Tkačik [15] who successfully produced a mathematical framework by which predictive, sparse, and efficient coding can be defined. One interesting observation which could have major consequences for the way in which these types of networks are evaluated is that of

their analysis of single neuron activity as opposed to multiple neuron responses. Chalk, Marre, and Tkacik found that by examining single neurons, the information encoded does not include motion for future predictions, whereas when analysing populations of multiple neurons, the opposite seems to be the case. For the analytical procedure when evaluating networks that fall within the discussed categories, this concept will need to be taken into consideration.

One of the most recent studies in predictive coding was undertaken by JCR Whittington and R Bogacz [16]. In the study, they explore a number of theories as to how the brain handles back propagation tasks, or rather how the conventional approach of back propagation cannot be applied to the brain. This is because conventional back propogation suggests a purely feed forward architecture of neural connections. They also make the observation that the one to one relationship between error calculations and their corresponding prediction neurons described in predictive coding, is not representative of neural activity within the visual cortex of the brain. The paper then puts forward an architecture expanding on predictive coding and incorporating the Dendritic Error Model, initially proposed by João Sacramento, Rui Ponte Costa, Yoshua Bengio and Walter Senn [17]. This new dendritic architecture describes excitatory pyramidal neurons that make predictions, that are connected to their corresponding dendrite which handles the prediction error for that neuron. The pyramidal neuron and dendrite are connected to an inhibitory interneuron by a new set of weights. Pyramidal, dendritic, and interneurons replace typical neurons from the classic predictive coding model and aim to more accurately model the way in which the brain handles prediction and approximations of input features.

While each of these theories analyse a different approach to predictive coding, a common thread is that they all describe unique neural activity within low-level neurons when exposed to different stimuli. While the exact methods of representing activities described in the above papers are not directly implemented in this project, the concepts describing the way in which information is encoded by these networks, led to the hypothesis researched in this project: that categorisation of the described encoded information is possible.

2.2 Image Classification

The architectures considered for the encoding task of this project are derived from predictive coding models, while the architectures for the categorisation task for this project are purely of a typical feed forward design. Due to the simplistic nature of the dataset used within this project, support vector machines are used for categorisation.

2.2.1 Support Vector Machines

Support vector machines present a means to classify binary categories of data [20]. They do this by optimising a function that bisects two labelled categories of data and once trained, uses the function to make predictions as to the category which a piece of data presented to the network falls into. For multi-class problems on a higher order than binary, multiple SVMs can be used to make multiple binary classifications which when evaluated, will allow for accurate multi-class classification. Chih-Wei Hsu and Chih-Jen Lin [21] evaluate an alternative means of performing multi-class categorisations using models derived from the classic SVM. They discuss

methods by which data can be evaluated in a single function into multiple categories, solving the scalability issue of binary classification. By the nature of the function required to perform multi-class separation, the training time for such SVMs is generally high as datasets are scaled up. As such, SVMs lend themselves to less complex categorisation tasks working on smaller data.

The dataset used within the implementation of this project, MNIST[2], although an image dataset, is simplistic in nature, leading to SVMs being the first consideration for classification of this data.

2.3 Literature Review Conclusion

Multiple iterations and variations of the predictive coding model have been explored in the past 20 years, but they all use the same core idea of deep neurons predicting the activity of connecting layers derived from R.P Rao and D.H Ballard's original paper [1]. The papers discussed in this literature review each lead to optimisations of the original training algorithm, as well as models that more accurately mimic the structure of the visual cortex, and neural activity in the brain [16].

Means of performing image categorisation tasks have also changed over the years and preexisting theories have benefited greatly from increasingly powerful hardware[18], as well as more refined training processes and model designs. However, there does not seem to be much, if any research into the categorical properties of deep neural activity in predictive coding networks. While deep neurons in such models have been shown to approximate areas of activity in a given input [1], it has not been seen as to whether these models encode sufficient information in deep neural layers for activity in these layers to remain categorically unique.

In this project, this property of deeper neurons is explored in more detail, and their ability to encode categorically identifiable information is validated.

3 Methodology

Based on the research discussed in the literature review, a system by which the hypothesis posed by this project can be tested is possible to produce. In this section, the considerations made that enable the development of each section of the system are discussed, as well as the requirements of each section of the system that need to be fulfilled in order to prove the hypothesis. The system design itself, as well as the specific functionality of each section of the system, are discussed in greater detail in the Design and Implementation sections.

3.1 Development Platform

This project uses an Anaconda [23] environment, for development using python. The specific python packages used as well as their purpose will be explored in more detail in the implementation section of this report. All development, training, and testing for this project was completed from within the PyCharm IDE [24] enabling rapid prototyping of system iterations.

3.2 Proposed System Structure

The system produced in this project is split into two halves, the predictive coding model and the categorisation model.

3.2.1 Predictive Coding Model

This project uses a predictive coding model with a single fully connected layer between a set of deep neurons and an output layer. The core architecture of the model is based on that which was outlined by R.P Rao [1]. The activity in the deep neurons of the model will be able to generate a reconstruction in the output of an image presented to the network. It will achieve this through the iterative application of gradient descent to the deep neurons based on the error between the predicted output image and the actual image presented to the network. A more in-depth description of the design of the model, as well as the mechanics of each element within it, will be discussed in greater detail in the design section of this report.

In terms of the functionality required to be achieved by the model, it must be able to reproduce an image presented to it once the weights within the model have been trained. These generated images must be accurate across not only the training set of images, but also a testing set. This would confirm that the trained model is working as intended and is not overfitting to the training data. Such properties are validated by analysing reconstructions from the model of images from the testing set.

This predictive coding section of the project does not make use of any machine learning libraries in order to enable more in-depth manipulation of all parameters within the model.

3.2.2 Classifier

The classifier this project opted to use is a Support Vector Classifier, using the scikit-learn python Library [22]. This will be responsible for the classification of both the raw images from the dataset, as well as the classification of the encoded images from the predictive model. Had this classifier proved insufficient for the task, a Convolutional Neural Network[18] would have been implemented.

3.3 Dataset

The dataset used throughout this project is the MNIST dataset[2]. MNIST is a dataset specifically designed for image categorisation tasks. The full dataset consists of 70'000 hand-written digits, each with a corresponding 0-9 label denoting the digit represented by an image. This project aims to act as a proof of concept for classifying deep neural representations from a predictive network, as such, a more simplistic dataset has been selected as it should show more defined boundaries between categories of data during testing, if the hypothesis presented is correct. By reducing the complexity of the dataset, the risk that errors stemming from the dataset itself could skew the results of the testing of this project is minimised.

3.4 Development Method

The method applied to the development of the model is derived from the agile approach of development[25]. While not identical to the agile method, the iterative design process is heavily

implemented in this project, primarily for the development of the predictive model. Through experimentation, iterative updates to the parameters of the Predictive model, as well as the parameters used to train it, were performed. This allowed the performance of the model to improve over each iteration, in order to fulfil the requirements outlined by the Experimental Process.

3.5 Experimental Process

In order to test the hypothesis presented by this project. There are three main experimental results that are required to be produced. These are as follows:

- The predictive coding model must accurately reproduce representations of MNIST images presented to it, such that there is high confidence that the model has been trained correctly. This involves adjusting both the parameters of the network and training algorithm and will be measured by visual comparison and error analysis. This will be discussed in more detail in the implementation section of this report.

- The classifier must successfully perform classification for the data in MNIST. This is to ensure that the model is capable of classifying the MNIST dataset. The will be measured by a prediction error rate, as well as an analysis of a confusion matrix obtained after testing.

- The classifier must successfully perform classification of the latent representations of the MNIST dataset obtained from the predictive coding model. The accuracy of these categorisations must be sufficiently high, which will once again be tested by prediction error rate and confusion matrix analysis. Even if the prediction rate is not exactly the same as on the original dataset, so long as there is clear evidence of categorisations occurring across both the original dataset and the latent representations, the hypothesis of this paper can be proven.

4 Design

Before moving to a functional implementation, considerations for the structure of each of the models as well as the overall system needed to be taken into consideration.

4.1 Complete System Structure

As shown in Figure 2. The design proposed by this model primarily centers around the predictive model itself. It is important to focus primarily on the design of this section of the system, as producing accurate deep representations is important to obtaining accurate results. The predictive coding model itself consists of a deep layer, which is fully connected to an output layer. The neural activity in the deep layer generates output images in the output layer, which are then compared to an MNIST[2] image presented to the network. The error in this prediction is used to perform gradient descent[3] on the weights of the fully connected layer in the network, as well as values at each neuron within the deep layer for each given piece of MNIST Data. It

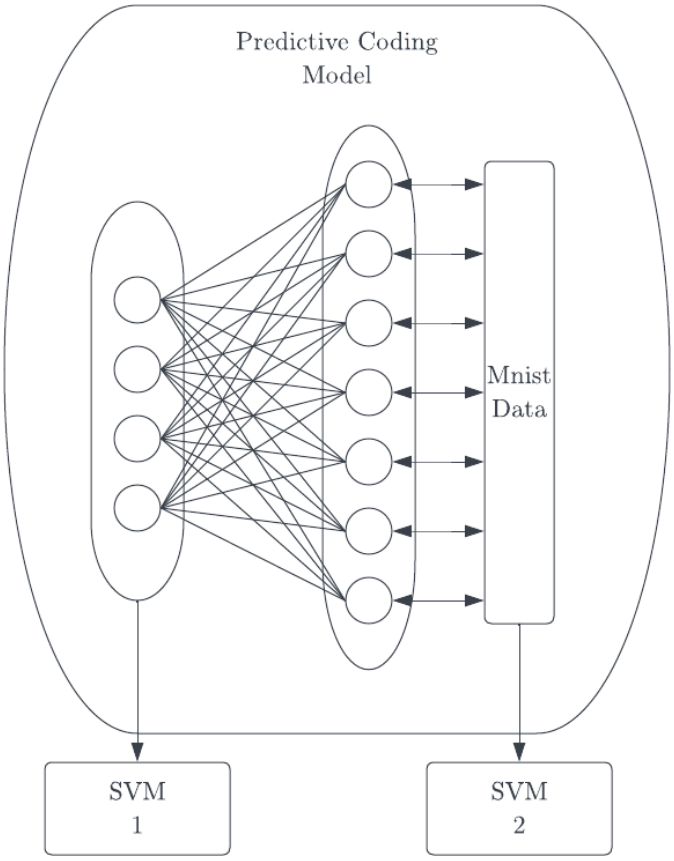

Figure 2: Wireframe of the relationships between each section of the proposed system

should be stated that the predictive model depicted by Figure 2 does not fully represent the dimensions of each layer in the Predictive Coding model of the final system.

Once trained, the predictive Model is be used to encode images from the MNIST test set, to the deeper layer in the network. This set of trained images will be split into a new training and testing set, then passed to the first SVM along with their corresponding labels from the original MNIST images used to obtain the encoded values. SVM 1 will then be trained on the new training set before being tested on the testing set.

The Second SVM will be trained on the same MNIST test set used to obtain the encoded representations in the Predictive Model. This set will be split in the same manner as the encoded testing and training set used in SVM 1 such that the labels and order of the labels across both the encoded set for SVM 1 and the non-encoded set for SVM 2 are identical. This ensures that the only variable across both SVMs are the representations of the data itself. The results from this testing set will either prove or disprove the hypothesis put forward by this paper.

4.2 Predictive Model

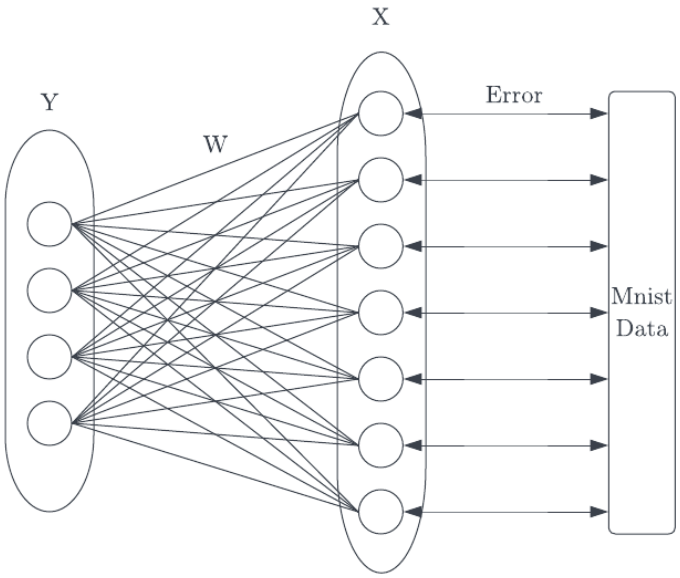

Figure 3: Wireframe of the Predictive Model implemented in this project, where Y indicates the deeper neural layer, X indicates the output of the network and W indicates the weights of each neural connection between the deep layer and output layer.

The predictive model of this project is derived from that which was proposed by R.P Rao[1]. The main difference between between this project's implementation and that which is described in Rao's paper, is that instead of making use of 'error neurons', error is applied directly during training on both the weights and deep neurons of the network.

The model shown in Figure 3 describes a predictive coding network with a single fully connected layer. The set of neurons labelled as Y describe the deep neurons whose activity are

responsible for the generation of images in the output layer X. The fully connected nature of the network describes that each of the neurons in Y are connected to all of the neurons in X by means of a set of weights in W. Just as in Figure 2, the number of neurons and weights in Figure 3 are not representative of the scale of the network in the implementation of this project. However, the logic applied to each component of the Network is the same.

From this point forward, the deep layer of the network will be referred to as Y, the output layer will be referred to as X and the weights connecting these layers will be referred to W.

In the final system, The number of neurons in the output X will be equivalent to the dimensions of the data within the MNIST dataset, which is to say that there will be 784 neurons, each corresponding to a single pixel on a generated output image. The number of neurons in Y was not static going into the development of the model, and testing was performed to find the lower limit for the number of nodes in Y that could accurately generate images in X.

The way in which image reconstruction is achieved within this predictive coding model is by producing an output by using the values at Y, multiplying them with the values in W, in order to produce an output in X. Initially these predictions are entirely random, but by comparing this output with a piece of data presented to the network from the MNIST dataset, an error at each neuron in X can be obtained, which can be used to perform gradient descent on both W and Y. By repeating this process, a set of values for both Y and W can be obtained which are capable of accurately reproducing an image presented to the network.

4.3 Logic of the Training Loop

In predictive coding, both weights and node values are trainable. Training of this network will therefore require a modified training loop to that of a conventional neural network. While each forward pass in the training loop will remain the similar to traditional neural networks, the updating of the values in W and Y will be slightly different. This method does not use conventional backpropagation[4], but rather performs gradient descent on both Y and W. The number of times this occurs for both Y and W and the order that it occurs in will be explained in the implementation. During training, W is initialized only once, whereas Y is reinitialised to a set of zeros each time a new image is presented to the network. This is because a unique set of values for Y is obtained for each image presented to the network. However, the values of W are never reset during training as the aim is to obtain a static set of weights that that is capable of performing well across all categories in the dataset. Once trained, these weights should allow the Y values we obtain to generate any handwritten digit presented to the network. Pseudocode for the training of the model is as follows:

Note that in the final implementation, W was moved inside this nested for loop due to an optimisation discovered for training. This is further discussed in the implementation section.

4.3.1 Forward Pass

The forward pass describes how the network produces an output image, and calculates the error of that prediction. The updating of both the weights and values in the deep layer Y is performed using gradient descent after each forward pass.

Algorithm 1 Pseudocode for Training the Model

for each sample c **do**
    $Y \leftarrow zeros$ ▷ reinitialise Y after each sample
    for times trained per sample **do**
        $X \leftarrow f(YW)$ ▷ generating an output
        $E \leftarrow (X - c)^2$ ▷ calculating error in the output
        $Y \leftarrow Y - dE/dY$ ▷ gradient descent on Y
    end for
    $W \leftarrow W - dE/dW$ ▷ gradient descent on W
end for

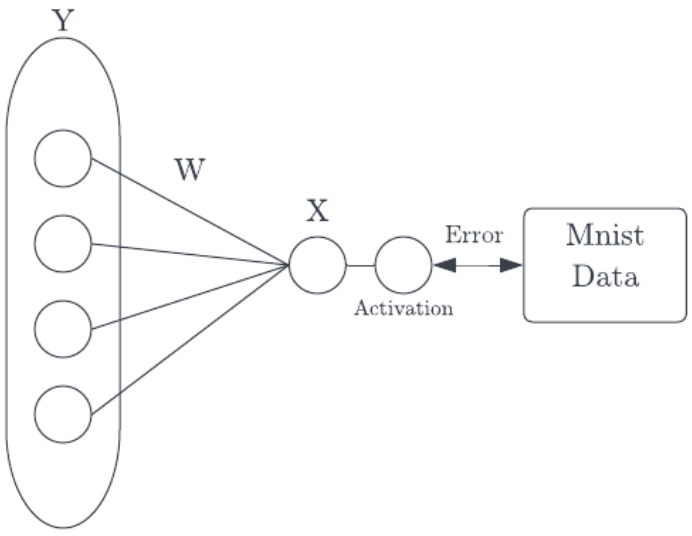

Figure 4: Wireframe to demonstrate how the value at a single node in X is calculated, as well as a prediction error.

During the forward pass, an output is calculated by multiplying the values at the deep layer Y, with the values in their connected weights W, and summing these values into the connected neuron in X. Figure 4 shows the connections to an output node in X. The same calculation can be applied across all elements in the output layer, giving the value of the output before an activation function is applied. For each output neuron this is given as:

$$X = \sum_{i=1}^{n} Y_i W_i$$

where n denotes the number of connected weights and Y values.

An activation function is then applied to the output. In this project, the sigmoid activation function[26] has been chosen. The sigmoid function is useful as all possible values of X from negative to positive infinity, are encoded to the range 0-1, which allows the trained values to converge towards the normalised dataset values, which are also between 0 and 1. Dataset normalisation will be explained in the implementation section. After applying the sigmoid function to a node in X, the output of the network is obtained for a single neuron. By applying the sigmoid function across the whole set of X values, the full prediction for an image is obtained. The sigmoid function in this network can be given as:

$$X_{out} = \frac{1}{1 + e^{X}}$$

The final calculation on the forward pass is working out the error in the prediction. The error used by this project is the mean squared error[28]. This is calculated using the value given by the target piece of data from MNIST and the output of the sigmoid function at X. The output layer contains 784 nodes, and the shape of an image from MNIST after preprocessing is 784. We assign a single output node in X to a pixel value in an MNIST Image. This target pixel value is given as 'target' in the following equation for finding the mean squared error for the prediction of a single neuron in X:

$$MSE = (target - X_{out})^2$$

This equation is applied across all of the output neurons X, giving a new set containing the prediction error of each output neuron for the current MNIST image.

No more information is required to be calculated on the forward pass.

4.3.2 Gradient Descent on Model Weights

In order to train the weights in the network to make better predictions, gradient descent[4, 3] must be performed on both the Weights W, and the deep layer Y. The process for performing this on the weights W is common in conventional neural networks and will be explained first.

Figure 5 shows a single weight connecting a neuron in Y to a neuron in X. In order to update W, the values of Y, X, the activation function, and the error are required. The activation is the sigmoid function and will be denoted as 'sig' and the mean squared error is denoted as E.

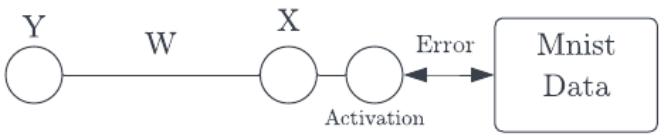

Figure 5: A single neural connection between a neuron in the deep layer and output layer.

In order to update the weight W, the rate of change of the error with respect to the weights must be found, then appended to the current value of the weight W. This can be denoted as the derivative of the error with respect to W. A learning rate is also applied that determines the scale of the adjustment to the weight denoted as LR. Also note that X_{out} refers to the value of X after being passed through the sigmoid function.

$$W_{new} = W - LR(\frac{\partial E}{\partial W})$$

However, as the Error is not a direct function of the weights, the chain rule[27] must be applied to dE/dW and expanded to a series of terms, each of which can be evaluated as a mathematical function. This derivative can be expanded as:

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial sig(X)} \frac{\partial sig(X)}{\partial X} \frac{\partial X}{\partial W}$$

Breaking this down further, each of these derivatives resolves to a mathematical formula. dE/dSig(X) resolves to the derivative of the mean squared error function:

$$\frac{\partial E}{\partial sig(X)} = -2(target - X_{out})$$

dSig(X)/dX resolves to the derivative of the sigmoid function[26]:

$$\frac{\partial sig(X)}{\partial X} = X_{out}(1 - X_{out})$$

Finally, in dX/dW, dX can be written as d(YW) as the value of X is equivalent to the operation YW:

$$\frac{\partial X}{\partial W} = \frac{\partial YW}{\partial W} = Y$$

Recombining these formulas, the formula for updating a single value for weight W is found:

$$W_{new} = W - LR(-2(target - X_{out}))(X_{out}(1 - X_{out}))(Y)$$

This gives us the means to update a single weight in the network. When the weights are

updated in the training loop, this logic is applied across all weights, taking into consideration the values at their connected nodes in the deep layer Y and output layer X.

4.3.3 Gradient Descent on Nodes in Layer Y

The property of trainable nodes within a neural network is unique to predictive coding. A similar gradient descent algorithm is applied to the values in deep layer Y, as is applied to the weights W. However, due to the many-to-one relationship that each Y has with its corresponding weights, the formula for updating weights must be modified slightly. Firstly, we are now aiming to find the gradient of the error with respect to y, in order to update each Y value.

Consider Figure 5. If there is a single weight connected to a neuron Y, then we would see the formula for updating Y as:

$$Y_{new} = Y - LR(\frac{\partial E}{\partial Y})$$

Using the same logic with the chain rule as applied to the formula for the weights, this formula expands to:

$$\frac{\partial E}{\partial Y} = \frac{\partial E}{\partial sig(X)} \frac{\partial sig(X)}{\partial X} \frac{\partial X}{\partial Y}$$

$$Y_{new} = Y - LR(-2(target - X_{out}))(X_{out}(1 - X_{out}))(W)$$

However, as the relationship between Y and W is not one-to-one, we must consider the values produced from the other neural connections. Figure 6 shows the possible connections of a single neuron Y. Due to this one-to-many relationship of each neuron Y, the gradient obtained across each neural pathway must be considered.

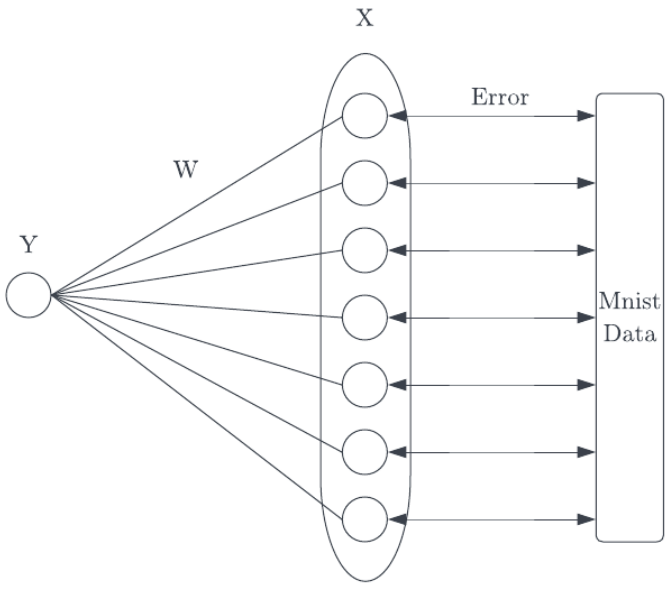

Figure 6: An example of the neural pathways that must be considered when updating the value of a single node Y.

This is achieved by simply summing the dE/dY terms calculated from all connected neural pathways before applying the learning rate and updating the Y value and is given as the following:

$$Y_{new} = Y - LR \sum_{i=1}^{n} \frac{\partial E_i}{\partial Y_i}$$

where n denotes the number of connected weights W to a given node Y.

4.3.4 Testing Loop

The testing loop is not overly dissimilar from the training loop. The primary difference is that the weights W in the network are not updated during testing. Training is responsible for adjusting the weights, and testing validates that the weights obtained are valid and capable of reproducing images from the deep neurons Y.

During the testing loop, images from the MNIST testing set are presented to the network. Note that these images, while still representative of digits from 0-9, are not identical to the data in the training set. Gradient descent is performed a number of times on the nodes at Y in order to converge on a set of deep representations that are capable of reconstructing the current image being presented from the MNIST testing set.

Pseudocode for the testing loop is as follows:

Algorithm 2 Pseudocode for Testing the Model

```
for each sample c do
    Y ← zeros                          ▷ reinitialise Y after each sample
    for times trained per sample do
        X ← f(YW)                      ▷ generating an output
        E ← (X − c)^2                  ▷ calculating error in the output
        Y ← Y − dE/dY                  ▷ gradient descent on Y
    end for
end for
```

After each sample, the prediction and output are shown in order to measure the accuracy of the predictive model on the testing set.

4.3.5 Additional consideration of the predictive model

Once an optimal configuration for training has been established that produces the best reconstruction of images during testing, the deep representations in Y can be saved as an entry in a new dataset, with the label for their corresponding image in MNIST.

The representations in both the new dataset of Y values, as well as the MNIST testing set, must be indexed in such a way that during classification each SVM can be trained on an equivalent set of data. This means that for a particular latent representation in the new dataset, the equivalent MNIST image from which the representation was obtained must be used in the training of the classifier.

These representations are also obtained from the MNIST testing set, as using values obtained from the training set may not be representative of values produced by a functional predictive

model. For example, the model could overtrain to the training set and have poor performance on the testing set, resulting in the Y values having unforeseen properties that skew the results of testing. This is why testing is performed on the predictive model before moving to classification, and values produced to be passed to the SVM are obtained from the set of data that the predictive model is not trained on.

4.4 SVM classifiers

So far, only the predictive model has been discussed. A process by which all the parameters in the predictive model can be updated and trained has been established. We must now establish the way in which the SVMs will interact with both the dataset, as well as the predictive model.

The structure, training, and testing of the SVMs are handled by the scikit learn [22] python library, and as such, we need only ensure that the datasets sent to each SVM are compatible with their input shape. This will be further discussed in the implementation.

As mentioned previously, each SVM is trained and tested on a different set of data.

- The first SVM is trained on the set of latent representations obtained from the predictive coding model and is used to show that these representations are categorically unique.
- The second SVM is trained on the MNIST testing set, and acts as a control to confirm that the SVM can perform categorisation of validated categorical data.

Each of the datasets used for training the SVMs will be split into a training and testing set. The exact size of these sets will be discussed in the implementation.

4.5 User Interface Considerations

Two main considerations for user interfaces were considered for this project. A graphical user interface, or a command line interface.

As this project is primarily research-based, a GUI adds unnecessary complexity without adding any benefits for the end user. There are not a large number of complex interactive modules within this project and, as such, a command line interface that is used to run each component of the project is more suitable. This interface will be responsible for running each discussed component of the project. This is important as breaking up the implementation into blocks of code that can be independently run allows functionality such as testing of a model without having to retrain it, or running testing on each individual dataset separately.

Using a command line interface also allows for input validation. For example, a user running the code through the command line interface will not be able to run the code in such a way that could cause catastrophic errors in the implementation. Each time an instruction is given to the command line, a number of checks will occur to validate whether it is safe within the current environment to execute that command. This will be discussed further in the implementation.

5 Implementation

In this section, the process of moving from a theoretical design to a working implementation is discussed. As previously stated, an anaconda [23] environment is used for the development of

this project. The specific additional libraries used will be introduced as they become relevant to the implementation. Multiple files are used in this project. The predictive coding model is contained in a python .py file, which saves the output produced for the svm to a numpy archive file, which is read by a different .py file containing the SVMs.

5.1 Dataset Preprocessing

In order to train any model, the dataset on which it is being trained must first be prepared in a form that is readable by a given training function. For the purpose of this project, the MNIST dataset needed to be flattened from the shape 28x28 to a numpy [29] of shape 784. Conveniently, the tensorflow[30] library contains a convenient function for loading the MNIST dataset, so this is used in order to retrieve the data.

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

Figure 7: Retrieving the MNIST data

Figure 7 shows the MNIST dataset being retrieved using the tensorflow library, and is split into training data and labels (x_train, y_train) and testing data and labels (x_test, y_test). The labels are only used for the SVMs. These arrays are then normalised to between the values 0 - 1 and reshaped to 784 as shown in Figure 8.

```
x_train = x_train/255
x_test = x_test/255
x_train = x_train.reshape(x_train.shape[0], 784)
x_test = x_test.reshape(x_test.shape[0], 784)
```

Figure 8: Preprocessing of the dataset

5.2 Implementation of the Predictive Model

The code described in this section is contained in the 'PredictiveModel.py' file. In order to represent the Predictive model, we create three numpy arrays to represent Y, W and X respectively.

```
X = np.random.rand(784) #output size
Y = np.zeros(10) #size of input layer
W = np.random.rand(Y.size, X.size)  # matrix for initial weights between neurons y and output
```

Figure 9: The decleration for the structure of the Predictive model

Figure 9 shows the shape of the predictive model. Here, the output layer of the model is of size 784, and the deep layer is of size 10, with a 2D array of weights representing the neural connections between the two layers. The initial number of nodes in Y was 10 and this proved to be the lower limit of what was possible in order to reproduce accurate images in the output layer. Testing was performed on these parameters in order to find this lower limit which will be explained later in the implementation. We initialise Y to an array of zeros, and W to an

array of random numbers between 0 and 1. W must be initialised randomly otherwise we would encounter a problem known as the symmetrical gradient problem [31] which would result in each set of weights connected to a single node in Y would train to exactly the same values, resulting in incredibly poor learning and no pattern extraction.

Functions were also created for the equations outlined in the Design section of this report. These functions are used for calculations that are common across both testing and training. Not all calculations were made into functions, as some were unique to the Y values and some to the W values. Those that were made into functions were those that were common across both the training of Y and W, shown in Figure 10.

```
def sigmoid(x):  #used to apply sigmoid to dot product results for forward prop
    return 1/(1+math.exp(-x))


def DsigmoidDyw(x):  #used to calculate the differential of the activation function
    return x*(1 - x)


def mse(predicted, target):  #mean square error
    return (target - predicted)**2


def DmseDactivation(predicted, target):  #derivative of the mean square error
    global X
    return -2*(target - predicted)
```

Figure 10: Functions used to calculate output, error, and derivatives.

5.2.1 Training Function and Loop

The function shown in Figure 11, is the final implementation of the training function.

Each time the training function is called, a prediction is made using the current values in Y and W. In the training loop, there is no calculation for the MSE error in the final implementation. While the MSE is a useful indicator for the performance of the network, only the derivative of the errors is needed in order to perform gradient descent. As such, the calculation of the MSE directly is omitted from the training function. Instead, calculation of the derivative of the MSE is performed across the entirety of the output using the 'currentData' arguement, representing the image being currently presented to the network, to calculate this value. It is then stored in the array dEdActivation. An array of the derivatives of the activation function is produced next and stored in dActivationdyw.

Using these values, as well as values we already have access to, we can now update the values in Y and W.

First, gradient descent is applied to the Y values, looping across all connected weights of each Y and performing the gradient descent algorithm on each connected branch and summing

```
def training(currentData):
    global W
    global X
    global Y
    global dEdActivation
    global dActivationdyw
    # forward prop
    X = np.dot(Y , W) #generates a prediction
    for n in range(0, X.size): #applies the sigmoid function to the output
        X[n] = sigmoid(X[n])
    # gradient descent
    for n in range(0, X.size): #calculates matrix of errors
        dEdActivation[n] = DmseDactivation(X[n], currentData[n])

    for n in range(0, X.size): #calculates matrix of derivitives of the activation
        dActivationdyw[n] = DsigmoidDyw(X[n])
    for n in range(0, Y.size): #performs gradient descent on inputs Y
        newY = 0
        for n1 in range(0, W[1].size):
            newY = newY + dEdActivation[n1]*dActivationdyw[n1]*W[n][n1] #DE/DY = DE/Df(y*w) * Df(y*w)/D(y*w) * D(y*w)/DY
            newY = newY
        Y[n] = Y[n] - LR * newY #applies learning rate
    for n in range(0, Y.size): #performs gradient descent on W
        W[n] = W[n] - LR*dEdActivation*dActivationdyw*Y[n] #DE/DW = DE/Df(y*w) * df(y*w)/D(y*w) * D(y*w)/DW
```

Figure 11: The training function of the Predictive Model

the results, before applying the learning rate and updating each neuron. Then, the gradient descent algorithm is applied to all W values.

It is wroth noting that in this implementation of the training function, gradient descent is performed only once on Y for before performing it on W. Usually, gradient descent needs to be performed a large number of times on Y before W is updated once. However, during training, obtaining a functional model would have taken weeks of training on the hardware used, due to the huge number of computations required to make a single change to W. As such, an alternate method of training was implemented with the hopes of greatly reducing this training time, in which for each call of the training function, W and Y are updated once. The repeated training of Y and W is achieved by calling the training function repeatedly from within a loop, and balancing out the number of training iterations for each image with the learning rate, as to hopefully avoid overfitting to each image. This method proved to be successful. However, on larger more complex datasets, this method is likely to be ineffective, as balancing out the learning rate and training loops for each sample would be incredibly difficult. The specific mechanics as to why this method of training produced a suitable set of weights for MNIST will require further research.

The main training loop, shown in Figure 12, denotes the parameters that control the training session. The loop shown depicts the final implementation for this loop. However, in initial implementations, the 'for i' loop was in the training function rather than the training loop. This is due to the previously discussed reasoning of updating Y multiple times for each W update, which was changed in this final implementation.

In this loop, 'i' denotes the number of times a prediction is made, and gradient descent is

```
def trainloop():
    global Y
    numpy.random.shuffle(x_train) #randomizes training array in order to improve reduction in MSE
    for x in range(5000):
        Y = np.zeros(Y.size)
        for i in range(500):
            training(x_train[x])
        print(str(x+1) + "/5000 complete")

    np.savez('TrainedWeights.npz', Y=Y, W=W)
```

Figure 12: The main training loop of the Predictive Model

performed for a single piece of data from the dataset. Through experimentation, it was found that by making this loop a relatively high value and setting the learning rate to a lower value, a more capable model was produced. x_train[x] is the current image from the training set of MNIST to be presented to the model. 'x' denotes the number of values from the training set that will be used in training. In the case of the final implementation, this was 5000. With this configuration, a single epoch across the training set took 8 hours to complete. The reason that only a single epoch was performed, is that excellent image reconstruction was achieved after only one epoch, likely due to the simplistic nature of the MNIST dataset. Any further training resulted in little improvement, and risked overfitting to the training data.

The values for the Y layer are also reinitialised to an array of 0s each time a new image from the training set of MNIST is presented to the network. This is because the set of Y values obtained will be unique for each image and, as such, the representations must begin training from the same value of 0.

After training, the weights are saved to a .npz numpy archive file. This is done so that if the weights proved to be trained well, then the saved set of weights can be used to obtain a new dataset of Y values, without the need to retrain the model. The file that these weights are saved to is called "TrainedWeights.npz".

5.2.2 Testing Function and Loop

Finally, the testing loop needed to be created for the predictive model. This largely borrows code from the training function, though it eliminates the part of the code that updates weights. In predictive coding, gradient descent is still performed on the values at each deep node, as described in the Design section, but the weights of a trained model are static. For this reason, we perform gradient descent on Y in the testing function, the same number of times as in the training loop.

As shown in Figure 13, the testing function is largely the same as the training function, excluding the removal of the weight updates. When called, this function performs gradient descent on each node in Y once. The number of times this occurs is determined by the testing loop, shown in Figure 14.

Similarly to to training loop, this loop calls the testing function to perform gradient descent

```
def testing(currentData):

    global W
    global X
    global Y
    global dEdActivation
    global dActivationdyw
    # forward prop
    X = np.dot(Y ,W) #generates a raw output
   # X = out
   # print(X[1]) # generates a raw output
    for n in range(0, X.size): #applies the sigmoid function to the output
        X[n] = sigmoid(X[n])

    # backward prop
    for n in range(0, X.size): #calculates matrix of errors
        dEdActivation[n] = DmseDactivation(X[n], currentData[n])

    for n in range(0, X.size): ##calculates matrix of derivitives of the activation
        dActivationdyw[n] = DsigmoidDyw(X[n])

    for n in range(0, Y.size): #performs gradient descent on inputs Y
        newY = 0
        for n1 in range(0, W[1].size):
            newY = newY + dEdActivation[n1]*dActivationdyw[n1]*W[n][n1] #DE/DY = DE/Df(y*w) * Df(y*w)/D(y*w) * D(y*w)/DY
            newY = newY #gives average of calculated Ys after gradient descent and applies learning rate
        Y[n] = Y[n] - LR*newY
```

Figure 13: The testing function of the Predictive Model

```
def testloop():
    global W
    global Y
    loaded = np.load('TrainedWeights.npz')
    W = loaded['W']
    for x in range(10): #tests to see if images can be generated using current set of weights
        Y = np.zeros(Y.size)
        for i in range(500):
            testing(x_test[x])
        image = X.reshape(28, 28)
        fig = plt.figure
        plt.imshow(image)
        plt.show()
        image = x_test[x].reshape(28, 28)
        fig = plt.figure
        plt.imshow(image)
        plt.show()
```

Figure 14: The testing loop of the Predictive Model

on the values at Y for a given image x_test[x]. For each image presented to the network, the Y values are set to 0, just as in the training loop. For testing, one option was to measure the mean square error. However for image reconstruction tasks, this does not always indicate that an accurate reconstruction of an image is being produced. Instead, we display a number of images generated using the model. The output layer is updated each time the testing function is called, so converting these values into a 28x28 format gives us the reconstructed image. This image is displayed using the MatPlotLib library[32].

5.2.3 Perfecting the Model Parameters

As mentioned previously, this iteration of the model outlined so far was not the first. Although the overall structure did not change, the number of nodes in Y and the learning rate did. Furthermore, due to this implementation updating W and Y simultaneously, the tweaking of these parameters in order to achieve successful training was paramount. In order to rigorously test hypothesis of this paper, we aim to minimise the cardinality of Y while maintaining a satisfactory reconstruction accuracy from the model. The initial value chosen to test for the number of Y values (10) coincidentally turned out to be the lower limit for accurate image reconstruction using this method.

With 10 as the starting number of Y values, 0.1 was used as an initial learning rate, with training occurring across 5000 images from the MNIST training set. With this configuration, the network showed little to no performance due to overtraining from the learning rate of 0.1.

The learning rate was then lowered to 0.0013, which is 1 divided by the cardinality of the output, in an effort to massively limit the rate of updates of W values. This adjustment proved successful and yielded satisfactory image reconstructions across the testing set.

Next was to test the lower limit of the possible cardinality of Y. This was done by training models with the same parameters as the functional model, but decreasing the number of nodes in Y each time. Image reconstruction immediately saw poor performance after reducing the number of nodes just once, and almost all of the reconstructive performance deteriorated beneath 5 neurons in Y. For this reason, the final model has a Y layer with 10 neurons and uses a learning rate of 0.0013.

Figure 15 shows a number of reconstructed images as well as the MNIST image from the testing set that they attempt to reconstruct. The reconstructions are not perfect, yet they retain a high enough degree of categorical uniqueness that the Y values used to generate them should also maintain the same degree of categorical uniqueness, allowing categorisation to successfully be performed.

5.3 Encoding the Dataset for the SVM

Now that we have a functional set of trained weights for the model that has been validated on the MNIST test set, we can convert a section of the MNIST dataset into a new dataset of low-level neurons in Y using the trained model.

In the implementation, this is achieved using the 'EncodeYvalues.py' python file. In this file, the same testing loop that is present in the 'PredictiveModel.py' file. However, in this

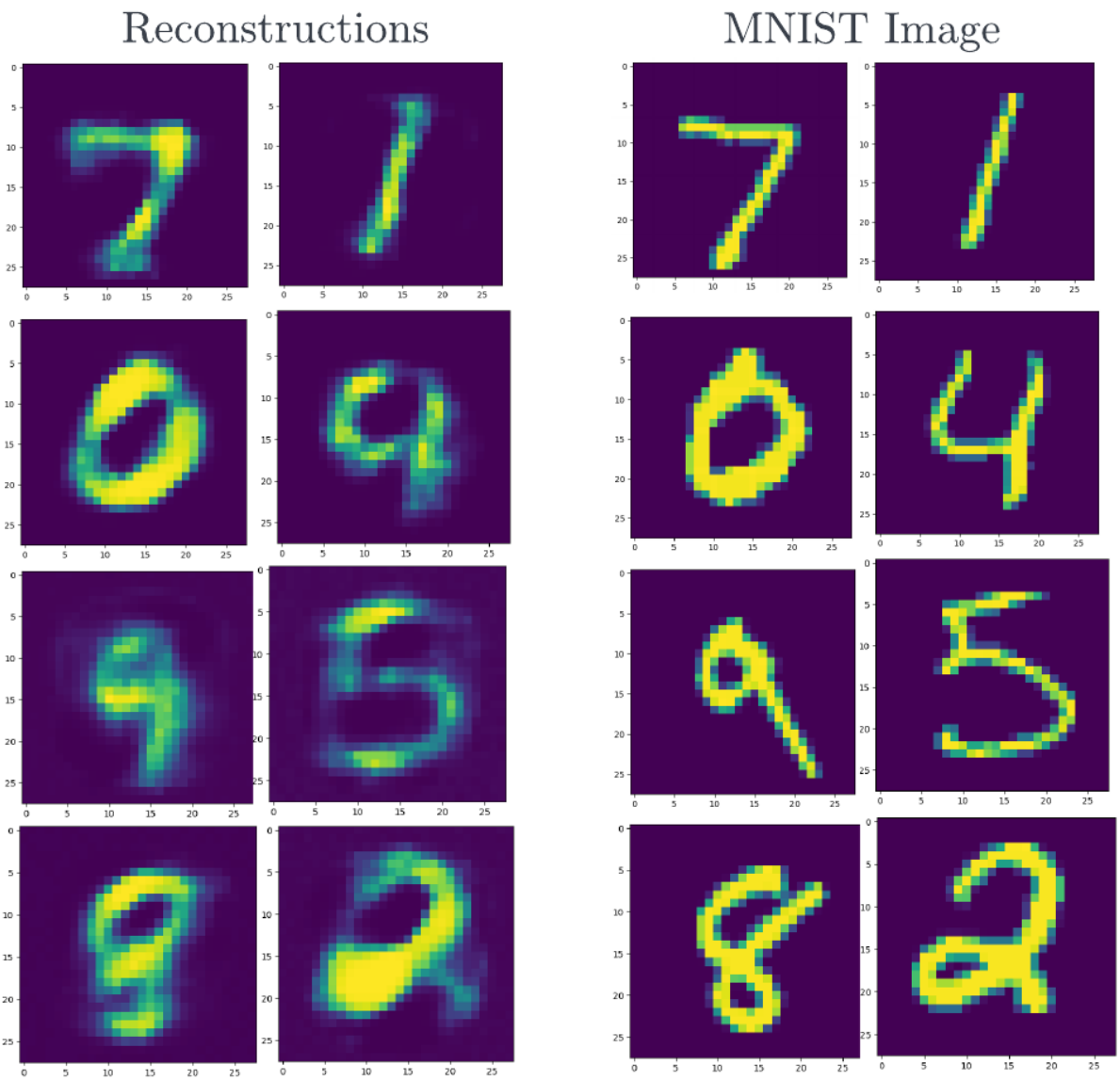

Figure 15: Demonstration of the trained predictive model reconstructing MNIST images from the MNIST test set

version, after gradient descent has been performed on Y 500 times for a given piece of data from the MNIST test set. The values present at Y are saved to the numpy array endodedData, and the corresponsing label for the category of image from MNIST is saved to the the array, encodedDataLabels. This is shown in Figure 16, where samples denotes the number of MNIST images to be encoded.

```
def encode():
    global W
    global Y

    loaded = np.load('TrainedWeights.npz')
    W = loaded['W']

    encodedData = np.zeros((samples, 10))
    encodedDataLabels = np.zeros(samples)


    for x in range(samples): #uses the trained predictive model weights to encode MNIST images
        Y = np.zeros(Y.size)
        for i in range(500):
            testing(x_test[x])
        encodedData[x] = Y
        encodedDataLabels[x] = y_test[x]
        print(str(x+1)+"/"+str(samples))
    np.savez('SVMdataset.npz', data=encodedData, labels=encodedDataLabels)
```

Figure 16: Loop for converting a selected number of images from the MNIST test set into their encoded representation using the trained predictive model

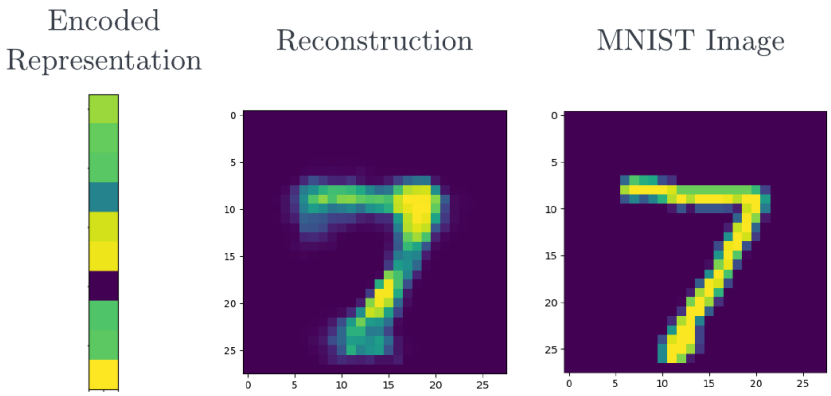

Figure 17: Example of an encoded representation used to generate a reconstruction of an MNIST image.

Both encodedData and encodedDataLabels are then saved to a numpy archive so that they can be accessed later by the SVM. Figure 17 shows an example of a set of Y activities that is added to the encodedData dataset, alongside its corresponding generated image, and original MNIST image. In the final implementation, a dataset of 1000 unique entries is produced with

the categories from 0-9 represented near evenly.

5.4 Training the Support Vector Machines

Finally, the SVMs need to be trained for categorisation of each dataset. In the python file 'SVM.py', the system by which each SVM is trained and tested is defined.

Before training the SVMs, the new dataset of latent representations and the MNIST test set must be prepared for training on the SVMs. As mentioned previously, the MNIST test set is used for training one of the SVMs here as this is the set that was used to obtain the latent representations.

First the MNIST dataset is reloaded, and the size of the testing set is reduced to the first 1000 pieces of data. These are the same 1000 images used to obtain the set of latent representations from the predictive model, shown in Figure 18.

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
#dataset preperation
x_train = x_train/255
x_test = x_test/255
x_train = x_train.reshape(x_train.shape[0], 784)
x_test = x_test.reshape(x_test.shape[0], 784)

x_test = numpy.delete(x_test, numpy.s_[1000:10000], axis=0)
y_test = numpy.delete(y_test, numpy.s_[1000:10000], axis=0)
```

Figure 18: Preprocessing of MNIST testing set for SVM classifier.

We then load the dataset produced by the predictive model, shown in Figure 19.

```
loaded = np.load('SVMdataset.npz')

data = loaded['data']
labels = loaded['labels']
```

Figure 19: Preprocessing of Latent Representation Dataset testing set for SVM classifier.

For each dataset, we must make a new training and testing split. A 4:1 ratio is used to split the data into a training and testing set, called 'trainData' and 'testData'. After training and testing of the SVM used to categorise the latent representations is finished, the 'trainData' and 'testData' variables are reassigned to the MNIST data.

We perform training with the dataset of latent representations first. Using the scikit learn python library [22], a support vector classifier is created and assigned to 'svc'. A support vector classifier is a support vector machine used for classification tasks in the scikit learn library.

Training of the classifier 'svc' is simple. The numpy arrays of the training data and labels are passed to 'svc' using '.fit', which performs training of the model using the training data. We can then test the performance of the model using a prediction function '.predict'.

The python package pandas[33] is used in order to produce a visualisation of the accuracy of the results of the classifications in the form of a confusion matrix. This will be further discussed in the evaluation.

The code for the training and testing process described can be seen in Figure 20.

```
svc.fit(trainData, trainLabels)
classifications = svc.predict(testData)

from sklearn.metrics import confusion_matrix
cm = np.array(confusion_matrix(testLabels, classifications))
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
confusion = pd.DataFrame(cm, index=['Actual 0', 'Actual 1', 'Actual 2',
                                    'Actual 3', 'Actual 4', 'Actual 5',
                                    'Actual 6', 'Actual 7', 'Actual 8', 'Actual 9']
                         , columns=['Predicted 0', 'Predicted 1', 'Predicted 2',
                                    'Predicted 3', 'Predicted 4', 'Predicted 5',
                                    'Predicted 6', 'Predicted 7', 'Predicted 8',
                                    'Predicted 9'])
```

Figure 20: Training and testing the SVM on one of the datasets, then saving then producing a confusion matrix of the results.

This process is then repeated for the MNIST testing set so that the categorisations of the two can be compared.

5.5 User Interface

In order to make the testing and training of Predictive Models, as well as running the classifiers less error-prone and more straightforward, a command line interface has been implemented for each python file. While a user of this project will still need to load each python file described so far independently in a python environment, the functionality within each file is executed using predefined commands in order to eliminate the risk of errors stemming from a user being inexperienced with this system.

Each of the user interfaces works by means of a while loop at the end of each python file, which takes an input as an argument. This input is compared to a number of possible inputs. When there is a match, the corresponding function is run after checking that the correct files required to run a given function exist. This ensures that no errors will occur as a result of the end-user's mistakes. In order to close the current python file in use, each of the CLIs accepts the command 'end', causing the loop to break and resulting in the file closing.

This CLI only allows for the running of the code in the final state that it is presented in. In order to change datasets or the architecture of the model, changes within the code are required to be made.

Figure 21 shows the code which is used to create the command line interface for generating a predictive model. the interface takes three possible arguments: 'train', 'test' and 'end'. 'train'

```
while True:
    print("Type 'train' to train a model, 'test' to test a trained model or 'end' to exit")
    condition = input()
    if condition == "train":
        print("beginning training of predictive model")
        trainloop()
        print("training complete")
    elif condition == "test":
        try:
            loaded = np.load('TrainedWeights.npz')
            print("beginning testing of predictive model")
            testloop()
            print("testing complete")
        except:
            print("no saved weights for predictive model found, please run training")
    elif condition == "end":
        break
    else:
        print("invalid input")
```

Figure 21: Code for the command line interface of the Predictive Model python file

is responsible for generating a new model. This has an incredibly long runtime, as it generates a new model from scratch without making use of the trained weights file. 'test' outputs a predetermined number of generated images using the trained model, as well as the original images from MNIST. If no model is found, it does not run predictions and requests the user to generate a model.

```
while True:
    print("Type 'begin' to generate a dataset using the current predictive model, to exit type 'end'")
    condition = input()
    if condition == "begin":

        try:
            loaded = np.load('TrainedWeights.npz')
            print("beginning generation of dataset")
            encode()
            print("dataset complete")
        except:
            print("no saved weights for predictive model found, please run training using PredictiveModel.py")
    elif condition == "end":
        break
    else:
        print("invalid input")
```

Figure 22: Code for the command line interface of the EncodeYvalues python file

Figure 22 shows the code for the CLI of the EncodeYvalues.py file. As this file is only responsible for generating a new dataset for categorisation, the only commands that it accepts are 'begin' and 'end'. 'begin' checks whether a set of trained weights exists, before compiling a dataset.

```
while True:
    print("Type 'trainEncoded' to generate classifications of the Latent "
          "Representation dataset, type 'trainMNIST' to generate classifications of MNIST, to exit type 'end'")
    condition = input()
    if condition == "trainEncoded":

        try:
            loaded = np.load('SVMdataset.npz')
            print("beginning training of SVM")
            trainingLat()
        except:
            print("no dataset found, please use EncodeYvalues.py")
    elif condition == "trainMNIST":
        print("beginning training of SVM")
        trainingMNIST()
    elif condition == "end":
        break
    else:
        print("invalid input")
```

Figure 23: Code for the command line interface of the SVM python file

The command line interface for the SVM python file, shown in Figure 23 uses the argument 'trainEncoded' to train and test an SVM on the dataset of encoded Y values and outputs a confusion matrix of the resulting categorisations. If the SVMdataset.npz file does not exist, or fails to load, then the function to train the SVM does not run. The argument 'trainMNIST' runs training for an SVM on the original MNIST test set and outputs the resulting confusion matrix. Once again, 'end' closes the python file.

6 Results

This section discusses the results obtained from the classifications using the SVMs and discusses ideas stemming from observations of both the predictive model and the final classifications.

6.1 Results of the Classifications

From both SVMs, high classification accuracy was obtained. Testing of each of these models was done with the remaining 200 images for each of the datasets. Firstly, we will review the results of the classifications of the MNIST images, so that we can compare these classifications with that of the latent representation dataset.

6.1.1 MNIST Classifications

Categorisations on the MNIST data were accurate. The SVM achieved an accuracy of 91% on the set of 200 test MNIST images. A confusion matrix for the categorisations of this dataset can be seen in Table 1.

6.1.2 Latent Representation Dataset Classifications

The accuracy of the SVM's predictions on the dataset of latent representations was also high. The SVM achieved an accuracy in categorisation of 84.5%.A confusion matrix for the

		Predicted									
		0	1	2	3	4	5	6	7	8	9
Actual	0	15									
	1		25		2						
	2	1		27	1						
	3				22		2			2	
	4					17					1
	5					1	11				
	6	1						18			
	7			1					19		1
	8			1	1		1			15	1
	9								1		13

Table 1: Confusion matrix showing the predictions produced by the SVM on MNIST

categorisations of this dataset can be seen in Table 2.

		Predicted									
		0	1	2	3	4	5	6	7	8	9
Actual	0	14						1			
	1		26		1						
	2	1		22	1	1			1	3	
	3				22		1			2	1
	4				1	14			1		2
	5						11				1
	6							19			
	7	1		3		1			15	1	
	8				2		2			15	
	9					3					11

Table 2: Confusion matrix showing the predictions produced by the SVM on the Latent Representation Dataset

6.2 Evaluation of Results

Across both the MNIST classifications and the Latent Representation classifications, we see a high classification accuracy. The majority of classifications across both datasets are correct, with a few outliers. These outliers have some interesting properties.

For the MNIST classifications, we see a higher overall accuracy, however, there are classification errors for most of the numbers at least once. While most of these are random, some stand out from the rest. Note Table 1, we see multiple instances of the number 3 being predicted as 5 and 7. There are a number of possibilities as to what could have caused this. It is possible that the number of pieces of data that the SVM was trained on (800) was not enough to accurately optimise a function that separated each category of data. If this were the case, the optimisation performed by the SVM would not be sufficient to categorise all instances of a number in a testing set. This is especially true for numbers that share similar features to other numbers, such as the round lower half of the number 3 which is also present in 5 and 8,

which are the numbers incorrectly categorised. The inverse of this can also be seen with the number 8 being incorrectly categorised as 5, 3 and 2. Overall, aside from these edge cases, the categorisation was a success and acts as a suitable control to compare to the other classification using the latent representations.

For the classifications of the latent representations, we also see a high accuracy. Although the accuracy is high, it is slightly lower than the categorisations of the MNIST data directly. This could be due to a broader variety of reasons. While the same arguments can be made about the performance of the SVM as on the MNIST data, there could also be artifacts of the predictive coding process that carry over and affect the categorisations. For example, as we test the lower bounds for the cardinality of layer Y in the predictive model, the overall reconstructive ability of the predictive model is not suboptimal. We aimed to find the lowest number of nodes in Y that would result in reasonable reconstruction, however, if this number of nodes in Y was increased, it is likely that the more detailed representations produced for the dataset of Latent Representations, would result in better categorisation.

Consider Table 2. We once again see an error in similarly shaped numbers being wrongly categorised, such as 3 and 8. However, we now also see an increase in the number 9 being confused with the number 4 compared to the SVM trained on MNIST. This is possibly due to the shapes of these numbers appearing similar in their reconstructions. In Figure 15, we see that the reconstructions of 9 and 4 share similar features, which is likely reflected in the latent representations, leading to this increase in categorical error.

Despite these errors, the categorisations were accurate more often than not. the fact that we see successful categorisation for both the MNIST set, as well as the Latent Representations.

7 Project Evaluation

7.1 Future Additions to the Project

In this project, we have seen excellent results for the most part. However, the accuracy in categorisations, as well as the predictive model, have ways in which they could be further improved. For the predictive model, the training loop can be further optimised as to produce a more powerful model, which would be capable of more accurate image reconstruction. This could reduce the gap in categorisation accuracy between the latent representations set and MNIST. For the task of categorisation, implementing a convolutional neural network could yield more accurate categorisations as they have shown higher accuracy than SVMs in some applications [19].

7.2 Directions for Future Research

The ability of predictive coding models to encode data while retaining categorical information about the original data has been proven for single-layer predictive models. Further could be undertaken to confirm the property of categorical uniqueness within predictive models of different architecture. It is possible that the different properties of layers in deep multi-layer predictive models, as described in the literature review, could lead to different results from categorisation of each of these layers. Different architectures that optimise the predictive coding

algorithm, such as efficient coding [10] could yield different deep representations of a given input from those obtained in this project, resulting in different categorical properties.

Further research into the potential applications of the results of this paper is also possible. Applications requiring lightweight hardware, embedded systems, or more lightweight datasets can use the method shown in this paper to obtain these representations, with the advantage of a predictive model being able to reconstruct an accurate representation of the original data.

This project also showed a means to train predictive models where weights and node values are updated simultaneously rather than performing training multiple times on node values for each weight update. The method used for this (outlined in the implementation section), while successful for this project, may not translate well to implementations outside of this project. Nevertheless, its success in the scope of this project warrants further research into the validity and scalability of this method of training.

7.3 Conclusion

The primary focus of this paper regards the categorisation of latent representations of a predictive model. The implementation and the results gathered are contributing to research on the properties of deep layers of predictive models. With further research, this paper can help to establish deeper understanding of the ways in which visual information is processed by predictive models.

Training an image classifier on a dataset of low-level activities of a predictive model showed that successful categorisation was possible using this data, proving the hypothesis that deep neural activity within predictive models retains the property of categorical uniqueness from a given stimuli.

A property of predictive model described in Rao's original paper [1] is also further validated by the results obtained in this project. By comparing the categorisation accuracy of the original data, as well as the lower performing categorisations of the latent representations, it is possible that details about an image are lost in low-level neurons to an extent. Such a loss can be seen in the slight reduction of SVM performance on the dataset of latent representations. However, further testing will need to be performed to confirm this, as the single-layer model in this project's implementation has many other variables that could have contributed to sub-optimal performance.

Before this research, the lower limit for the cardinality of neurons in a deep layer of a predictive model had not been fully tested. This project has shown the extent to which the size of a deep neural layer can be reduced while still producing accurate reconstructions of an MNIST image. Further research is required to test this property of predictive models outside the scope of this dataset.

To conclude, the research undertaken by this project opens the door to a deeper understanding of the properties of deep neural activity in predictive models. Now that the ability of deep neurons in a predictive model to encoding properties of categorical uniqueness within their response to a stimuli has been proven, it should be possible to produce an even more accurate model of the way in which information is encoded in predictive models.

References

[1] Rao, R.P. and Ballard, D.H., 1999. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. Nature neuroscience, 2(1), pp.79-87.

[2] Yann.lecun.com. n.d. MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. [online] Available at: http://yann.lecun.com/exdb/mnist/

[3] Chong, E.K. and Zak, S.H., 2004. An introduction to optimization. John Wiley & sons.

[4] Rumelhart, D.E., Hinton, G.E. and Williams, R.J., 1986. Learning representations by back-propagating errors. nature, 323(6088), pp.533-536.

[5] Amidi, A. and Amidi, S., 2022. The evolution of image classification explained. [online] Stanford.edu. Available at: https://stanford.edu/ shervine/blog/evolution-image-classification-explained

[6] Hubel, D.H. and Wiesel, T.N., 1965. Receptive fields and functional architecture in two nonstriate visual areas (18 and 19) of the cat. Journal of neurophysiology, 28(2), pp.229-289.

[7] Hubel, D.H. and Wiesel, T.N., 1968. Receptive fields and functional architecture of monkey striate cortex. The Journal of physiology, 195(1), pp.215-243.

[8] Grossberg, S., Mingolla, E. and Ross, W.D., 1997. Visual brain and visual perception: how does the cortex do perceptual grouping?. Trends in neurosciences, 20(3), pp.106-111.

[9] Mumford, D., 1992. On the computational architecture of the neocortex. Biological cybernetics, 66(3), pp.241-251.

[10] Karklin, Y. and Simoncelli, E., 2011. Efficient coding of natural images with a population of noisy linear-nonlinear neurons. Advances in neural information processing systems, 24.

[11] Laughlin, S.B., de Ruyter van Steveninck, R.R. and Anderson, J.C., 1998. The metabolic cost of neural information. Nature neuroscience, 1(1), pp.36-41.

[12] Rao, R.P., Olshausen, B.A. and Lewicki, M.S. eds., 2002. Probabilistic models of the brain: Perception and neural function. MIT press.

[13] Olshausen, B.A. and Field, D.J., 2004. Sparse coding of sensory inputs. Current opinion in neurobiology, 14(4), pp.481-487.

[14] Willshaw, D.J., Buneman, O.P. and Longuet-Higgins, H.C., 1969. Non-holographic associative memory. Nature, 222(5197), pp.960-962.

[15] Chalk, M., Marre, O. and Tkačik, G., 2018. Toward a unified theory of efficient, predictive, and sparse coding. Proceedings of the National Academy of Sciences, 115(1), pp.186-191.

[16] Whittington, J.C. and Bogacz, R., 2019. Theories of error back-propagation in the brain. Trends in cognitive sciences, 23(3), pp.235-250.

[17] Sacramento, J., Ponte Costa, R., Bengio, Y. and Senn, W., 2018. Dendritic cortical microcircuits approximate the backpropagation algorithm. Advances in neural information processing systems, 31.

[18] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25.

[19] Liu, Z., Mao, H., Wu, C.Y., Feichtenhofer, C., Darrell, T. and Xie, S., 2022. A ConvNet for the 2020s. arXiv preprint arXiv:2201.03545.

[20] Noble, W.S., 2006. What is a support vector machine?. Nature biotechnology, 24(12), pp.1565-1567.

[21] Hsu, C.W. and Lin, C.J., 2002. A comparison of methods for multiclass support vector machines. IEEE transactions on Neural Networks, 13(2), pp.415-425.

[22] scikit-learn. n.d. 1.4. Support Vector Machines. [online] Available at: https://scikit-learn.org/stable/modules/svm.html

[23] Anaconda. n.d. Anaconda — The World's Most Popular Data Science Platform. [online] Available at: https://www.anaconda.com/

[24] JetBrains. n.d. PyCharm: the Python IDE for Professional Developers by JetBrains. [online] Available at: https://www.jetbrains.com/pycharm/

[25] Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J. and Thomas, D., 2001. Manifesto for Agile Software Development. [online] Agilemanifesto.org. Available at: http://www.agilemanifesto.org/

[26] Lin, H., n.d. Data science: Neural networks: Deriving the sigmoid derivative via chain and quotient rules. [online] Data science. Available at: https://hausetutorials.netlify.app/posts/2019-12-01-neural-networks-deriving-the-sigmoid-derivative/

[27] Encyclopedia Britannica. n.d. chain rule mathematics. [online] Available at: https://www.britannica.com/science/chain-rule

[28] Probabilitycourse.com. n.d. Mean Squared Error (MSE). [online] Available at: https://www.probabilitycourse.com/chapter9/9_1_5_mean_squared_error_MSE.php

[29] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020)

[30] TensorFlow. n.d. TensorFlow. [online] Available at: https://www.tensorflow.org/

[31] Glorot, X. and Bengio, Y., 2010, March. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249-256). JMLR Workshop and Conference Proceedings.

[32] J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007.

[33] Pandas.pydata.org. n.d. pandas - Python Data Analysis Library. [online] Available at: https://pandas.pydata.org/