Imperfect Experience; or, Effects of withholding training data on multi-task question answering in convolutional neural networks

Imperfect Experience,

or,

Effects of withholding training data on multi-task question answering in convolutional neural networks

Matthew Donald Lutze

A Thesis in the Field of Software Engineering

for the Degree of Master of Liberal Arts in Extension Studies

Harvard University

December 2019

Abstract

This thesis explores the effects of training convolutional neural networks to perform conditional multi-task problems with training data that systematically excludes information. Using the MNIST database of handwritten digits, we prepare two collections of three versions of the database, adding color and an embedded question. Two question-embedding method are used.

The base version of each data set has 100 combinations of 10 digits and 10 colors, with a roughly equal distribution of questions. From these base preparations we extract color/shape combinations from the inputs using two strategies, forcing each network to infer progressively more answers during testing.

We demonstrate six Convolutional Neural Networks (CNNs) varied by the architecture of their output layers and output activation function, tested with two different question embedding processes. Without otherwise implementing advanced tuning techniques, the networks achieve between $96.78\%(\pm 0.41, n = 90)$ and $99.64\%(\pm 0.41, n = 90)$ accuracy on the more difficult task when training on all category combinations. At 50% combination extraction, one network variation demonstrates 98.33% $(\pm 0.35, n = 90)$ and $99.87\%(\pm 0.09, n = 90)$ accuracy on shape and color classification tasks.

The results indicate best performance from Sigmoid output activation and task-shared final fully connected output layers. The thesis contributes to strategy for network design when data is scarce.

Dedication

For Mom and Dad, Stefan and Letti;

for James, Tricia, Tom, Hugh, Liv, Pippa and Harvest;

I offer this work with gratitude and love.

Acknowledgements

I would like to thank Dr. Xavier Boix, who agreed after a thorough search to be my thesis director. Dr. Boix's expertise and wisdom in both the field and the writing process have been invaluable in helping me keep focus, refine my work and ultimately achieve a result that I'm proud of.

My thanks also go to Dr. Sylvain Jaume for many months of support in finding the right fit for me and in structuring my thesis process. Along with Dr. Jaume, my thanks to Prof. Eric Gieseke and Dr. Michael Gussert for their support whilst I developed my proposal, focused its scope and learned how to structure my goals into something achievable.

To Dr. Wayne Homstad I extend my appreciation for the inspiration many years ago to set my standards high and, even now, for helping me remember my voice. To Dr. Gussert once more, my thanks for the many conversations as I worked to get things exactly right.

Endless gratitude goes to my parents and family, whose love and support have been and remain my bedrock.

Most of all, my deepest respect and adoration goes to my partner, Stefan Ursul. This would not have been possible without his perpetual support, encouragement and joy. Thank you—this has meant the world to me.

Contents

List of Figures

List of Tables

List of Code

Chapter 1: Introduction

> There are four simple ways for the observant to tell Mr. Croup and Mr. Vandemar apart: first, Mr. Vandemar is two and a half heads taller than Mr. Croup; second, Mr. Croup has eyes of a faded, china blue, while Mr. Vandemar's eyes are brown; third, while Mr. Vandemar fashioned the rings he wears on his right hand out of the skulls of four ravens, Mr. Croup has no obvious jewellery; fourth, Mr. Croup likes words, while Mr. Vandemar is always hungry. Also, they look nothing at all alike.
>
> Neil Gaiman, *Neverwhere*

Computer systems will not attain artificial "general intelligence" until they can learn to communicate and develop reasoning skills (Mikolov et al., 2015, pp. 1, 24). These problem spaces are vast, but their potential solutions share both the need to ask and answer questions and the need to infer new information from related-but-distinct inputs (Mikolov et al., 2015; Bottou, 2014). At the intersection of these two needs we may ask the question, "how would a system perform if asked questions about scenarios it has not yet experienced?"

This thesis contributes answers to that and related questions. We require neural networks to selectively classify images based on a question about the image's encodes shape or color. Controlled stress is applied by increasingly excluding answers from the training data, requiring increasing inference. By observing the performance of the networks in these conditions, we can make better fundamental design choices that improve a network's ability to perform from less data.

But first, a question: why are communication and reasoning so important to the emergence of generally intelligent machines?

1.1. Questions are necessary for reasoning

A person, who we'll call the transmitter, encodes abstract information from their internal systems of perception and knowledge into a transferable format, hoping that their audience, another person, will be able successfully decode the transferred encoding within their own systems. The process of communication is imprecise and difficult to do with high accuracy and precision. The transmitter must adapt and account for the filters and differences in his or her audience's ways of perceiving and understanding. The audience similarly tries to guess the ways in which the transmitter sees and interprets the world, because they must as well compensate for uncorrected imprecision in the transmitter's encoding. Observed by William H. Whyte in an article for *Fortune* in 1950:

> LET US RECAPITULATE A BIT: The great enemy of communication, we find, is the illusion of it. We have talked enough; but we have not listened. And by not listening we have failed to concede the immense complexity of our society—and thus the great gaps between ourselves and those with whom we seek understanding.

Disciplines have grown around the pursuit of understanding and perfecting the process: the classics of grammar, logic and rhetoric, and with them the advances of critique and the exploration of semiotics. Communication, and unambiguously understanding it, are critical to our ability to interpret how we are expected to interact with the outside world.

Argued by Melvin Conway, "organizations which design systems [...] are constrained to produce designs which are copies of the communication structures of these organizations" (1968). He demonstrated that a system designed by more than a single

entity would reflect the way that those entities collaborated on the system's design and implementation. Expanding on this thesis, we might consider that non-solitary neural networks need both an ability to communicate, and the means to develop that communication around their own "organic" processes. Still, core to such a skill would be the capability to ask and answer questions. Only as these systems interrogate other emerging systems will they have the chance to develop truly new modes of understanding.

1.2. Reasoning is necessary for developing wisdom and understanding

Perhaps more fundamental to achieving an expression of intelligence is developing reasoning skills. Learning allows the network to collect impressions of the outside world, but reasoning would be the network's ability to interpret what those impressions mean. Observed by LeCun et al. in "Deep Learning" for *Nature*, "Human and animal learning is largely unsupervised: we discover the structure of the world by observing it, not by being told the name of every object" (LeCun et al., 2015, p. 442).

The task is so broad it can be difficult to define a success condition. Some approach logic tasks (Hohenecker & Lukasiewicz, 2018) and others relational problems (Santoro et al., 2017). Léon Bottou, in his essay on reasoning, describes the ultimate goal as *informal* reasoning:

> Human reasoning displays neither the limitations of logical inference nor those of probabilistic inference. The ability to reason is often confused with the ability to make logical inferences. When we observe a visual scene, when we hear a complex sentence, we are able to explain in formal terms the relation of the objects in the scene, or the precise meaning of the sentence components. However, there is no evidence that such a formal analysis necessarily takes place: we see a scene, we hear a sentence, and we just know what they mean. This suggests the existence of a middle layer, already a form of reasoning, but not yet formal or logical. (Bottou, 2014, p. 134)

He goes on to observe that learning how to model informal reasoning systems may allow us to short-cut the labor-intensive and hard-coded systems of traditional logic and instead use simpler, flexible designs. (Bottou, 2014, p. 136, 140).

We may find ways to model communication and informal reasoning in the further deconstruction of simpler fundamental processes. Multi-task Learning is one such motivation. Multi-task Learning (MTL) is simply the capacity of an entity to learn to perform more than one activity at the same time. It is a process natural to life: newborns learn to perform complex classification tasks by extending previously learned tasks. Humans regularly learn complex activities by first mastering fundamental tasks (Ruder, 2017, p. 2). A network that learns to perform multiple tasks is similarly better suited to learning more complex, general activities.

Various question answering and reasoning-based projects reviewed in Chapter 2 demonstrate a trend toward deeply complex and interconnected networks. This thesis distills some of the fundamental aspects of these projects' contributions. Using those distillations, we will report on how core architecture choices can improve a network's ability to make selective predictions with partial training information.

1.3. Thesis structure

The remainder of this thesis details the background, set up and execution of a project to investigate forced inferential reasoning in convolutional neural networks by excluding training information. Chapter 2 will review Prior Works related to question answering, reasoning, communication and cooperation. Next, Chapter 3 will present the Requirements for the project and detail the Design of the project's components. Then, Chapter 4 reviews the Methods taken to perform the project, whose Results are evaluated in Chapter 5. Finally, Chapter 6 will present Conclusions drawn from the evaluation and describe potential future works.

Chapter 2: Prior Work

> Therefore the problem of recognizing patterns and what to do under these circumstances is the thing that the computer engineers (they like to call themselves computer scientists) still find very difficult. It is certainly one of the important things for future computers, perhaps more important than the things I spoke about.
>
> Richard Feynman, *Computing Machines in the Future*

Two common deep learning neural network architectures are the Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). Networks built on either architecture, after undergoing a process of trial and correction, will adjust many internal weights and variables to signal when an input contains patterns they have previously seen. Where the RNN design works well for learning patterns between inputs (being helpful for tasks like sentence completion and audio decoding), CNNs are a traditional choice for learning the patterns within an input, such as image classification tasks.

Recurrent Neural Networks can be described as a software analogy for short term memory. They do a good job at discovering patterns experienced between inputs and being able to recall them when the pattern is discovered again. In their basic form, the process used to update the RNN with new information, backpropogation, tends to decay information when errors or poorer signals overwrite a good signal. This makes it very difficult to store information for a long time period if learning is an

active and ongoing process (Williams and Zipser, 1992; Werbos, 1988; Robinson and Fallside, 1987 as referenced in (Hochreiter & Schmidhuber, 1997)). To solve the need for long-term storage, Sepp Hochreiter and Jürgen Schmidhuber in 1997 introduced the Long Short-Term Memory model for the recurrent network. The LSTM added the capability to ignore an input or forget the state being effected, which helped reduce the issue of new inputs dramatically effecting stored information (or never getting to effect the stored model at all).

Still, the LSTM presents shortcomings with precision and accuracy. Weston in 2014 noted that contemporary research demonstrated memorization, like directly storing and returning the same information, are difficult for the RNN / LSTM alone (Zaremba & Sutskever, 2014). For Zaremba and Sutskever, in order for an LSTM to store and return a string of numbers consistently, it was necessary to reverse the input and double-train the model before getting its output.

2.1. Training neural networks to answer questions

Question answering is a fundamental requirement of interactive networks. Network architectures that can parse, process and reply to a question can be interrogated and can potentially participate within a system of actors.

The RNN and LSTM train a corpus of information into memory and refer to these storage patterns when classifying new inputs. This process predisposes the networks to losing learned facts over time (LeCun et al., 2015). Many groups made attempts to improve performance during the decades after Hochreiter and Schmidhuber introduced their improvement. In 2014, Jason Weston et. al. presented Memory Networks, which are designed to segment training input and store the learned patterns in external memory. Storing the model's "knowledge" allowed improved recall later, and successfully demonstrated Question Answering skills (Weston et al., 2014).

Published at approximately the same time as Memory Networks, a team of researcher at Google DeepMind, led by Alex Graves, published a similar but separate approach to the problem in Neural Turing Machines 2014.

The MemNN and NTM both approach the problem of long-term fact retention by adding an external memory component. Vectors generated by an initial parsing neural model are written to this memory by different schemes and the MemNN or NTM can later, when it parses a question, recall the information whose stored vector is closest or best matched to the question. Both models were significant improvements to the LSTM approach for neural-model-based discussion. Follow up work to improve both the Memory Network (Chandar et al., 2016) and Neural Turing Machine (Gulcehre et al., 2016, 2018; Woodward & Finn, 2017) have explored ways to improve memory addressing performance and reduced the supervision required for each one's training. In these approaches, however, question answering is handled as a parallel effort; a separate network for processing and understanding a question is introduced, working in concert with a classification network for the ultimate extraction and prediction of desired responses.

A Convolutional Neural Network (CNN), on the other hand, works by performing a series of operations on the input and the neural network layers, producing an output which describes how the input and the neural network layers interacted with each other (these operations are called convolutions, giving the network type its name).

A neural network layer in a CNN will have a set of "feature maps." A portion of the input's pixels, the size of a predefined small filter window, will be summarized and then stored in a corresponding pixel of one of the feature maps. If the network uses a 4x4 pixel filter, for example, the window would process 16 pixels from the input and store the result in the feature map. Different weights will apply to the

summarizing operation for each feature map, allowing each map to learn how to find different features (thus the name). An activation function normalizes the feature map values and, during a pooling stage, the parts of the activated convolution layer that do less often signal known inputs will be removed to keep the network from bloating. Backpropogation will, over time, adjust the weights and biases of the neurons in the feature maps. By working backwards through the network with adjustments for the correct answer, the network changes how input values are expressed layer-to-layer. This will eventually (one hopes) result in the network correctly classifying future inputs.

Convolutional feature extraction is helpful for visual classification but is not generally the first choice for learning semantic relationships. The RNN described earlier will learn the association between a set of letters that make up a word, or a set of words that make up facts. More robust recurrent networks will learn the association between parts of a set of words and parts of questions, allowing them to eventually predict answers to question configurations they may not have explicitly been trained on. RNNs underlie the Neural Turing Machine from Graves, Wayne and Danihelka at DeepMind 2014 and the Memory Network from Weston, Chopra and Bordes at Facebook 2014. The CNN is not as efficient at training word relationships. In Karpathy (2015), the author observes that

> A glaring limitation of vanilla Neural Networks (and also Convolutional Networks) is that their API is too constrained: they accept a fixed-sized vector as input (e.g. an image) and produce a fixed-sized vector as output (e.g. probabilities of different classes).

The CNN still presents an interesting challenge. A strategy that allows passing communication along with an image, to a CNN capable of interpreting the communication, would make a separate communication parsing network unnecessary and drastically simplify a Visual Question Answering (VQA) network's design.

2.2. Neural networks demonstrating reasoning characteristics

When we talk about teaching a CNN to learn new information, we begin to work with more complex problem spaces. The simplest problem space for classification is asking a neural network to identify whether an input contains a specific sort of representation: "Is there a cat?", or "Is this blue?" From this problem space, we can expand in two directions. If we add two or more representations to classify, then we've introduced a Multi-Classification problem. If we ask the network to classify two or more different kinds of representation (shape and color, for example) we've introduced a Multi-Task problem.

Both of these spaces are rich with interesting applications. The most complex contemporary example may be autonomous agents that operate vehicles. The computer vision networks for these machines need to simultaneously identify the road, vehicles, pedestrians, and other objects on the road, along with various environmental conditions (weather or signage) that might change the decisions the agent would make.

Breakthrough models have been both complex and limited in their applications. Like the Neural Turing Machine and the Memory Neural Network discussed in 2.1, these approaches have often looked at ways to add traditional computing structures into the Deep Neural Networks. Emerging models haven't required these external storage-and-recall mechanisms and are exploring natural language visual question answering.

Reviewed by Hudson & Manning (2018), the paper "FiLM: Visual Reasoning with a General Condition" introduces a computational method to influence a neural network's layers (Perez et al., 2017). In the case of CNNs, a network designer can insert, between two layers of the convolutional network, a FiLM (Feature-wIse Linear

Modulation) layer, which can use an RNN to influence the output of the first layer and therefore modify the perceived input at the second layer. This adaptability means that one can have the benefit of rich feature identification in the CNN's feature maps, as well as the pattern recognition or language interpretation inherit to RNNs. This combination can enable question-anwering about a visual input, and the authors indeed report a $97.4 \pm 0.4\%$ accuracy on one of their models when testing with the CLEVR dataset and evaluation model.

A downside to the FiLM model, observed by Hudson & Manning, is that the FiLM layer's influence is universally applied over the outputs of one CNN layer to the next. In Figure 3 of the FiLM paper, the authors illustrate the FiLM layer inserted between a convolutional layer and its activation function layer (Perez et al., 2017). Hudson and Manning observe that this full-layer effect reduces the ability of a CNN and FiLM network to perform attention-based tasks, which may limit the strategy's overall ability to answer detailed questions about regions of an input. The approach is illustrative, however, of a direction for achieving CNN-only VQA.

Drew A. Hudson and Christopher D. Manning of Stanford University, in April 2018, published the Memory, Attention, Composition (MAC) cell neural network component for using attention mechanisms to interrogating images during VQA (Hudson & Manning, 2018). They identify a common lack of inferential power in neural networks. The nature of Deep Neural Networks to develop vast amounts of hidden internal connections tends to mask the shallow connections that networks are often establishing:

> Most neural networks are essentially very large correlation engines that will hone in on any statistical, potentially spurious pattern that allows them to model the observed data more accurately. The depth, size and statistical nature that allows them to cope with noisy and diverse data often limits their ability to interpret and hinders their capacity to perform explicit and sound inference procedures that are vital for problem solving tasks.

They offer a network design of recurrent cells, with each cell containing: a control unit, which determines what the cell should be paying attention to; a read unit, which takes direction from the control unit to "extract information" from input or other information storage location; and a write unit, which performs an operation with the extracted information and the previous cell's memory output, creating a new output that gets passed to the next cell. In their description of the module, a reasoning prompt or question is delivered separately from the visual input, but both are processed simultaneously.

In the case of the CLEVR visual question answering problem, the cell may get a picture with some spheres in it as the memory output from a previous cell. The control unit may determine it is supposed to find one of the spheres, which it tells the read unit to extract. The read unit finds that sphere in the memory output from the previous cell, and tells the write unit where the sphere is. That location is then added to the currently held memory image, and is finally output from the cell as the new memory output. Notably, their model performed faster and achieved a higher overall accuracy on the CLEVR problem than FiLM and other then-state-of-the-art models.

Vincent Marois, T.S. Jayram, Vincent Albouy, Tomasz Kornuta, Younes Bouhadjar and Ahmet S. Ozcan with IBM Research AI responded to (Hudson & Manning, 2018) with "On transfer learning using a MAC model variant." In their paper, they introduce a simplified version of the MAC model that achieves comparable results. By removing some of the weights and biases from all three constituent units, the team reduce the number of parameters-per-cell by 50-67% and achieve a training and test accuracy less than 0.9% lower than the full MAC model (Marois et al., 2018).

The MAC model is interesting because it demonstrates how a network can take a question and an input together, understand the relationship between question and

input, and work its way toward an answer. Because the question decomposition and image analysis are happening concurrently (and not in separate layers of the network), it is potentially "understanding" where an object is in an image and analyzing that area to provide the correct answer. It also identifies that the simpler design performed nearly as well while requiring a significantly smaller processing resource footprint.

In Section 4.2 of (Marois et al., 2018), the authors note that MAC (and we assume their simplified MAC as well) networks fail to identify two shape and color combinations that were held out of training sets. The authors on page 5 of the report hypothesize that the MAC networks "did not correctly separate the concept of shape from the concept of color." we will explore the phenomenon of networks learning to separate tasks as a core question of this research project.

Santoro et al. present a composite function as a simple discussion model. This simple Relational Network (RN) has two multi-layer perceptrons and a set of objects. The input of the first MLP is the sum of outputs from the second MLP, one output for when each possible combination of two objects, from the total set of objects, is passed through that second MLP. The second MLP is learning the landscape of "distances" between each of the objects, while the first MLP is determining how to reflect those relationships or interactions.

To judge the CLEVR problem, the authors used a 4-layer CNN for image processing and 128 unit LSTM for question parsing, along with their RNs. CNN-parsed images features were given relative coordinates, which were fed into an RN to learn spatial relationships. LSTM-parsed question words were assigned unique integers, and the final output layer of the LSTM feeding the RN to learn the word relationships.

The study was influential for using a separate, simple module to learn how close or far apart things are, and it achieved state of the art performance on the

CLEVR tests. It seems to function like a look-up table for the other NNs, which appears to exhibit a problem confirmed by Palm et al.: the standard RN will learn single-order relationships between objects, but cannot infer second-order or deeper connections.

> The RN is a major step forward, but it has a limitation. The way it is constructed, each recognized object can only interact with the other recognized objects once, after which the network must give an answer. This limits the RN since it cannot reason about derived interactions, i.e. object A affecting object B, which in turn affects object C, and so on. In the RN, object A must directly affect object C, or not at all. Going through the interaction with object B is not an option.

Palm et al. (2017) extends the RN work by adding a messaging function, in this case a MLP that is passed by a node in problem space graph to all of its neighbors. Those messages are used to update a hidden state of the graph node (the authors used a Sudoku game to illustrate the process). This provides historical context about existing relationships that can be used by a neural network to make predictions based on greater that single-order relationships. They tested their model with a separate task also evaluated by Santoro et al. (2017), bAbI question answering, and the Recurrent Relational Network improved performance from 18 of 20 correct tasks to 19.17 ± 0.35 correct tasks while training days quicker.

Parts of a neural network sharing state messages amongst themselves in order to maintain historical context further decentralizes the influences network nodes have on each other, particularly within the same layer. This level of complexity allows for complex internal communication with its own form of memory to occur.

2.3. Neural networks cooperating and coordinating

One result of networks learning to communicate and perform nontrivial reasoning tasks would be cooperation between networks. Through the course of the project

we explored an initial collaborative-routing process for network question answering. In a fully realized form, networks that can ask and answer questions, make judgements and take actions would need a cooperation strategy for successfully interacting with their neighbors.

An approach by Foerster et al. (2017) in "Learning with Opponent-Learning Awareness" demonstrates multiple agents developing cooperation within different game environments, in scenarios where both agents have access to the exact parameters of their opponent and other scenarios where these parameters are hidden. The networks learn to infer the opponent's parameters from that opponent's action trajectories using an Argmax function. Previous papers by Foerster et al. used policy gradients to reinforce learning between agents 2016 and demonstrated Differentiable Inter-Agent Learning, a method for agents to share policies. Mordatch & Abbeel in 2017 demonstrated multiple agents in a learning environment developing a compositional language, without having previously been exposed to human language. The agents used that new language to provide instructions to each other in order to complete tasks.

A handful of researchers have recently demonstrated approaches of varying complexity for introducing inter-cooperation between specialized networks. Ruder et al. demonstrated in 2017 the Sluice Network, which uses a complicated strategy of cross-connecting neurons within the layers of peer networks, so that the patterns one has used can influence predictions of the other network. Andreas et al. (2015) describe a strategy of selecting from best-of-breed modules to build a custom solution for a problem. Rosenbaum et al. (2017) demonstrate an exciting design that encourages a network to routing inputs to different network layers for optimal processing.

2.4. Goals and Contributions

To summarize the previous work, there is evidence of Convolutional Neural Networks learning to communicate, perform reasoning tasks and even cooperate. These solutions separately are non-trivial and combined represent a dauntingly complex architecture. By taking fundamental components from them, whether that is embedding questions or training different layers to specialize in certain processing tasks, we will discover how different architectural choices effect a network's robustness.

A way to explore these capabilities is by using purposefully incomplete training environments for our networks. Architectures that perform well as less information is available will be better suited to real-world environments, where perfect knowledge is never achievable.

Interactive communication and reasoning are fundamental to generally useful and adaptable neural networks. We intend to explore fundamental components of recent advances in multi-task learning, network communication and cooperation in order to understand if we may be able to work toward simpler architectures. Simple but robust architectures that can transfer skills will be widely useful.

While deep learning neural networks excel at classifying previously trained information, building a model that is good at inferential reasoning is still difficult (Bottou, 2014). Transfer Learning—robustly pre-training a network before teaching it new tasks—is a powerful way to improve multi-task, multi-classification performance. Better understanding of how fundamental network architectures influence both multi-task performance and networks' inferential capacity may improve the quality of general use neural networks.

Chapter 3: Data and Experiments Design

> When solving a problem, try not to solve a more general problem as an intermediary step.
>
> Vladimir Vapnik, *The Nature of Statistical Learning Theory*

The design and implementation of this project is agile and iterative, with the success or failure of tests leading to refinements of both models and approach. For accessibility, overall design choices and rationale are presented here. Specific implementation details and methods are discussed in Chapter 4.

3.1. Overall project requirements

I hope to contribute to the Machine Learning field's collective ability to develop networks capable of learning many tasks with less-than-perfectly cultivated data. To accomplish that, this project focused on exploring features of different configurations of convolutional neural networks. More specifically, it explored how changing the configuration of a network would effect the network's ability to perform multiple tasks. We want to answer the following questions:

1. How simply can we implement a network that can perform multiple tasks and provide accurate multi-label outputs?
2. How can we extend this network to to use a shared vocabulary for answering different questions? How does that network change?

3. How does performance change when the data isn't exhaustive and doesn't represent the entire problem space?
4. How do these end-to-end networks compare to a network where all of the classifying components may not exist in a single feed-forward network?

This line of questioning provided the road map for exploration. The literature (see Section 2) describes many different design approaches that, for want of robustness, require complex and deep networks. Our first goal in this project is to look for a simple structure that would provide an easily reproducible foundation for the further questions, while still learning to perform more than a single task. From that baseline, the additional questions would direct exploration toward interesting aspects of these networks. How does withholding information change how well the network learns? Do other novel, simple approaches to developing multi-task networks provide better performance?

The project's solutions would need to provide a few commonsense capabilities. The overall goal is to develop a collection of different-configured networks, and use common testing data to compare their performance. To do this, the project will need to accomplish:

- Collect a series of input images and process the inputs in a predictable, describable and repeatable way
- Design the networks so that they can learn to perform more than one task based on the inputs.
- Design the networks, so that they can process a "question" or a switching command in relation to an input.
- Provide a process for collecting comparable results from training the networks.
- Provide visualization points into the networks to allow analysis of how they're

working.

3.2. Components design

This project hopes to understand fundamental performance characteristics of convolutional neural networks. To find actual signals, we need to carefully design and control the fixed parameters of the project data, neural networks and environments.

3.2.1 Environment

The industry standard for design and exploration of deep neural networks is done in Python, which we use here. It offers the most deep learning frameworks for designing and implementing the networks: Keras and Tensorflow are two of the most common, though others including Theano, mxnet, PyTorch, Caffe and many others have significant market share and benefits. While Keras makes model specification incredibly simple, abstracting some of the verbosity of a backend framework like Tensorflow or Theano, this project uses Tensorflow directly as it's deep learning framework. Tensorflow allowed me to provide greater flexibility in defining our training process. It further keeps the specification of network layer shapes verbose and easier to extract.

During network development, we used Tensorboard to review how networks were training in real-time. Being able to embed watchers on any part of the network (such as the final predictions, loss calculations, accuracy calculations) helped understand where the networks were training well and how to pick general parameters while we added and tested new functionality.

The underlying environment is generally summarized in Table 3.1.

Core tools and frameworks implemented as part of this project.

Type	Technology	Version
Software language	Python	3.6.4
Deep Learning framework	Tensorflow	1.12.0
Stats visualization	Tensorboard	1.12.1
Experiment data	MNIST	*
Nvidia GPU tools	CUDA	9.2.148
	cuDNN	7.2.1

**Note:* uses tensorflow.contrib.learn.python.learn.datasets.mnist distribution for MNIST data loading

Table 3.1: Experiment design technologies

3.2.2 Hardware and storage design

Hardware requirements are modest because these experiments will use the MNIST database of handwritten digits, based on a 28x28 pixel image (see Section 3.2.6). These are much easier to process than other more complex test sets. Images from the CLEVR database are much larger, while ImageNet images have variable sizes. The computing requirements are therefore moderate; an available desktop computer with a NVIDIA GeForce GTX 1070 graphics card provides sufficient performance to make the non-graph-manipulation operations the time-limiting part of training.

Beyond running tests, using git and GitHub will allow storing detailed version history of the work, and provide a platform to share both results and problems as they arise. To note: training and storing outputs from even simple neural networks can require significant size. As testing and development continued, the size of produced outputs increased significantly, until final outputs were exceeding 1 GB per experiment set run. In order to save from storing 100's of GB of partial experiments on GitHub, we store only summary run information for intermediate works.

3.2.3 Testing setup design

I began the project in Jupyter Notebook. The environment is easy to set up and makes testing parts scripts visual and trivial. It would be especially useful for inspecting and verifying the modifications to the MNIST inputs. Because we planned to both add color and noise, we needed to be able to review the process and determine that the modifications were sufficient and consistent. Jupyter Notebook is used initially to make visualizing the changes easier than running through the terminal.

A small testing platform will be necessary to efficiently run the volume of tests intended. While Jupyter Notebook is helpful for breaking scripts down or running tests, initial testing demonstrated that longer-running scripts stall or fail to complete. Running tests through the browser further reduces the amount of memory available for testing, and Jupyter Notebook presents issues with loading and running scripts from other files. We will therefore build an experiment "test bench" for training the networks directly in command shell-launched scripts to allow batch training and easily configuring training parameters.

3.2.4 Networks design

Each network explored should be able to learn features of our training inputs and respond to a "question" about those inputs. The composition of the network should be robust enough to efficiently learn its tasks, but not so excessively robust so as to waste resources or training time. The networks should also be identical, except for the specific component being alternated for the tests.

LeCun, Bottou, Bengio and Haffner in 1998 identified the following configuration as well-performing for MNIST training (LeNet-4, with 4 layers):

1. 4 features, with 8 sub-sampling maps
2. 16 features, with 16 sub-sampling maps
3. fully-connected layer having 120 units
4. output layer having 10 units

The best performing configuration from the paper used this network setup with 3 LeNet-4 networks(LeCun et al., 1998).

Simard, Steinkraus and Platt in 2003 discuss the effects of kernel size in the convolutional layers of a CNN processing MNIST images and determine that "padding the inputs did not improve performance significantly" They cite an optimal performing setup used 5 features in the first convolutional layer, 50 features in the second, 100 hidden units in the first fully connected layer and 10 output units in the second fully connected layer (Simard et al., 2003).

I reviewed the above recommendations and perform tests with different sizes for both the convolution window and the feature map depth. For the tests we will use 4x4 convolution window, and 25 feature maps for Convolution Layer 1. The review results are discussed in sections 4.2.4 and 4.2.

For this thesis, we test very similar versions of an end-to-end Convolutional Neural Network. Each network variant shares the first three layers, diverging at the final output layer. The shared layers are designed as follows:

1. *Input: a 28x28x4 image.
2. Convolution Layer 1 (Conv1): uses a 4x4 convolution filter to create 25 feature maps, for a resulting Tensor with shape (4, 28, 28, 25). Neurons are activated with ReLU, and a 2x2 max pooling filter results in an output Tensor with shape (4, 14, 14, 25).
3. Convolution Layer 2 (Conv2): uses a 4x4 convolution filter to create 50 feature

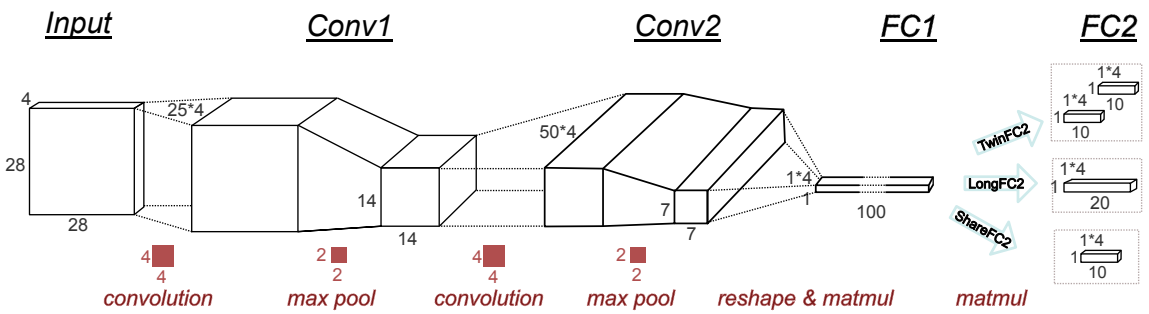


Figure 3.1: Shared and unique layers of the test network variants

maps, for a resulting Tensor with shape (4, 14, 14, 50). Neurons are activated with ReLU, and a 2x2 max pooling filter results in an output Tensor with shape (4, 7, 7, 50).

4. Fully Connected Layer 1 (FC1): The output Tensor from Conv2 is reshaped to (4, 2450) and then reduced through matrix multiplication to a (4, 100) Tensor. The neurons are finally activated with a ReLU function.

The dimensions of the final Fully Connected layer (FC2) are varied for each tested network type. Three network variations are used in this thesis:

1. **TwinFC2** is two peer Tensors with shape (4, 10). Each layer is independently fully connected to the FC1 layer neurons, and not connected to each other. Each layer is trained on either the shape or color ground truths.
2. **LongFC2** is one Tensor with shape (4, 20). Each output is assigned either a shape or a color label when training on the inputs' ground truths.
3. **SharedFC2** is one Tesnor with shape (4, 10). Each output is assigned both a shape and a color label when training on the inputs' ground truths, and must produce the correct prediction for the expected / embedded question.

The design of the networks are summarized in Figure 3.1. Each network variant will be tested with a Softmax output activation and with a Sigmoid out-

put activation. All networks use a Momentum optimizer. The Long and Share architecture variants optimize on the mean loss over a training batch calculated by `softmax_cross_entropy_with_logits_v2` or, for the Sigmoid-Channel trails series, `sigmoid_cross_entropy_with_logits`. The TwinFC2 architecture variants use an equally weighted average of the shape and color prediction mean loss.

3.2.5 Softmax & Sigmoid activation

Two sets of variants are used in these experiments, differed by output activation and loss function. LongFC2, ShareFC2 and TwinFC2 variants use a Softmax activation function, while the SigLongFC2, SigShareFC2 and SigTwinFC2 variants use a Sigmoid activation function. The output of a network variant is an array of probabilities. A Softmax function will transform this array so that the value of the array member with the highest probability is increased, while the other array members are decreased to 0. Softmax ensures that one member of the output array is always a strong prediction signal. A Sigmoid function will increase the value of array members that have higher probability, but does not concurrently decrease the value of other members. Sigmoid does not require any member of the output array to be a strong prediction signal.

When TwinFC2 is implemented with a Softmax activation, each of the peer output Tensors will always generate a prediction. To provide Shape and Color prediction task accuracy, we mask the outputs of each final layer to select the indices that correspond with each output's relevant question, and report the mean over those subsets. Because both outputs make a prediction, they both need a ground truth, and the network ends up not being penalized for answering the wrong question.

In order to compare the performance of the Twin configuration with the other two configurations, we need the network to output an array with a prediction for the

question task, and output an array with no prediction for the non-question task. We accomplish this by changing the output activation functions to Sigmoid and training the network with labels that provide a ground truth exclusively for the question asked. For the interest of thoroughness, we also collect and report on the performance of LongFC2 and ShareFC network variants using with a Sigmoid output activation.

```
## Get network outputs
model_output = my_conv_net(x_input)

## Loss
losses = tf.nn.softmax_cross_entropy_with_logits_v2(
        logits=model_output, labels=y_target)
loss = tf.reduce_mean(losses)

## Prediction
# Create prediction functions for digits and colors (_d and _c)
prediction = tf.nn.softmax(model_output)

## Accuracy
correct_prediction = tf.equal(tf.argmax(prediction, 1), tf.argmax(y_target,
    1))
accuracy = 100*tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

Listing 3.1: Softmax output activation for LongFC2 and ShareFC2

```
## Get network outputs
model_output = my_conv_net(x_input)

## Loss
losses = tf.nn.sigmoid_cross_entropy_with_logits(
  logits=model_output, labels=y_target)
loss = tf.reduce_mean(losses)

## Prediction
prediction = tf.nn.sigmoid(model_output)

## Accuracy
correct_prediction = tf.equal(
  tf.round(prediction), tf.round(y_target))
accuracy = 100*tf.reduce_mean(
  tf.reduce_min(tf.cast(correct_prediction, tf.float32), 1))
```

Listing 3.2: Sigmoid output activation for LongFC2 and ShareFC2

Listings 3.1 and 3.2 detail the specific method for calculating the loss and accuracy with each output activation method. Note that the Sigmoid accuracy

calculation uses `tf.reduce_mean(tf.reduce_min(correct_prediction,1))`. The `tf.equal(tf.round(),tf.round())` operation for evaluating the correctness of the prediction would produce the result "0.75" when comparing the prediction [1,0,1,0] with the ground truth [0,0,1,0], because 3 of the 4 positions would evaluate to "TRUE". Using the `tf.min()` operation and setting its minimum to 1 means that any `correct_prediction` that isn't completely correct will be considered a miss. This allows us to compare results with the Softmax network variants, whose `tf.argmax()` and Softmax functions are naturally comparing one-hot predictions and ground truths. It also ensures that strict accuracy is generally reported.

3.2.6 Data design

To test the questions posed in this design brief, we chose to augment the MNIST database images by adding to each a color and small amount of randomized noise. These will be fed into each of the network variations, and the resulting trained networks' performance compared to develop new understanding. This section discusses the design and rationale for data preparation.

The MNIST data set was published in conjunction with LeCun et al. (1998). It has become a gold standard for basic visual task analysis. The images are small and uniformly sized (28x28 pixels), which makes training networks simple for even a very modest home computer. Further, neural networks need not be exceedingly deep (see Section 3.2.4 above) in order to achieve very high accuracy. These properties make it a good choice for quickly testing design choices, as training a single feed-forward network on our hardware setup requires roughly 30-60 seconds.

An MNIST shape is a 28 by 28 pixel array with each pixel having a value from 0 to 1. In order to add color, we change this from a gray-scale to a color-scale image. We copy the gray-scale array twice, making each shape a 3x28x28 Numpy Array. To this

3-channel array we multiply a 1x3 list of simplified RGB color values; channel intensity values were 0.00, 0.25, or 0.75 (see the colors in Listing 3.3). The modified 3x28x28 is then clipped to 1, so that the pixel values remain in the 0 to 1 range. Finally, a small amount of noise is added via a `numpy.random.normal` Gaussian distribution, around 0 with a standard deviation of 0.2, after which the image is clipped one final time. This procedure gives us a 3-channel RGB color image, for each of which a color label is stored to enable supervised training of a feed-forward network on both shape and color tasks.

```
# 10 colors, with a 4th channel to match the processed MNIST image shape
colors10 = np.array([
    ["red", [.75,0.,0.,0.]],
    ["green", [0.,.75,0.,0.]],
    ["blue", [0.,0.,.75,0.]],
    ["pink", [.75,0.,.75,0.]],
    ["yellow", [.75,.75,0.,0.]],
    ["cyan", [0.,.75,.75,0.]],
    ["orange", [.75,.25,0.,0.]],
    ["periwinkle", [0.,.25,.75,0.]],
    ["purple", [.25, 0.,.75,0.]],
    ["chartreuse", [.25,.75,0.,0.]]
], dtype=object)
```

Listing 3.3: Array used for coloring the MNIST images

Once intermediary network designs are successfully learning shape and color identification tasks, we need to be able to ask the network a question (see Section 3.1). For this added complexity, the data needs to be further augmented. Question answering and reasoning networks benefit from attention-focusing: a target feature is highlighted with a mask that helps the classification network ignore other parts of the image. Using this general idea, we add a fourth channel to the images that stores a simple integer question. The process above is thus adjusted: the base shape array is copied 3 times, to make a 4x28x28 Numpy array, and a fourth 0 float is added to each color list. With this new 4x28x28 image, the fourth channel will be replaced

with a 1 for the "shape" question, and a 2 for "color." A list of questions is stored alongside the shape and color labels so that we can ultimately evaluate each network's performance answering either question.

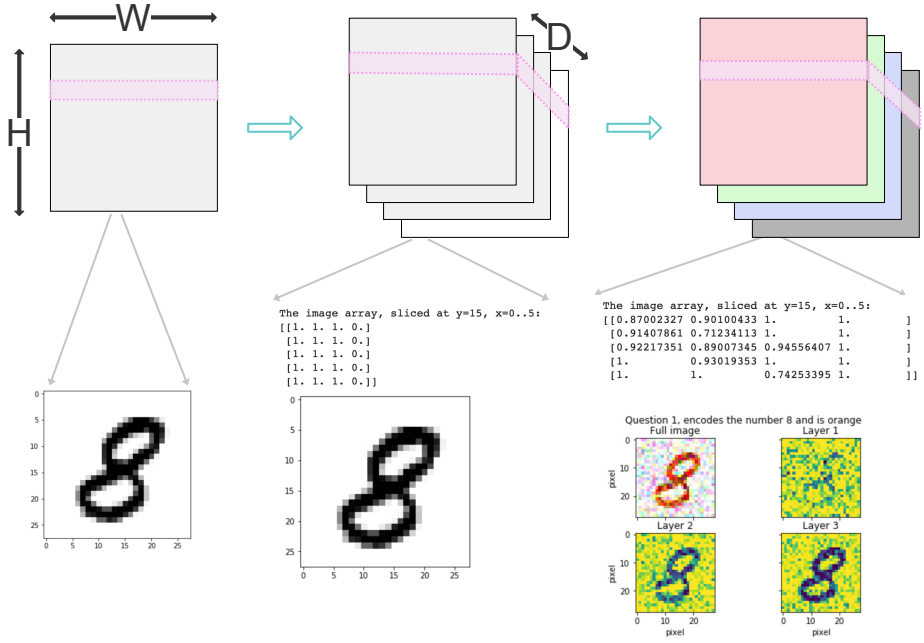


Figure 3.2: Process for expanding and preparing the MNIST inputs

An overview of the image preparation process is demonstrated in Figure 3.2, and the core function for coloring the images is demonstrated on page 35. In order to reduce potential inaccuracy in our final experiments, three different prepared sets of MNIST data were created. The runs in each prepared set for each specific network variant, learning rate and momentum, in each experiment can be averaged together, reducing potential unknown biases in any one particular set.

An unintended revision in the color strategy development process enables me to test two different methods of embedding the question. In the second preparation, later referred to as "Slice" trials, the question is embedded along the X-Z plane of the image at Y-index 3 instead of the X-Y plane at Z index 3. This leaves a 28x1 row of embedded Question in each of the 4 image channels (embedding it in

a slice of the image) instead of the 28x28 plane (using a full channel of the image). We present the results in comparison with the original question-embedding strategy during Evaluation in Chapter 5.

Chapter 4: Experiment Methods

> The first lesson on Roke, and the last is, Do what is needful! And no more. The lessons in between, then, must consist in learning what is needful.
>
> Ursula K. Le Guin, *The Farthest Shore*

We approach the thesis research in three main parts. First, we need to understand how the MNIST database looks and how it can be manipulated in order to effectively design ways to use that data. Next, we need to understand how the different parts of a CNN may change its robustness, in order to select which components or parameters we would vary. Finally, we need to set up a reproducible set of tests and collect sufficient data from them to make observations about our varied network architectures. I'll briefly describe these sections before expanding on important process points later in this chapter.

Part 1 involves understanding the MNIST database. We build and test scripts that manipulate and evaluate the inputs and labels. This includes exploring ways to add colors and noise to the database images. We test our understanding of the data by training a simple CNN on the different data configurations and end up with a network accurately predicting the encoded number and given color in example inputs.

For part 2, the exploration focused on implementing simple ways to ask a "question." Visual Question Answering is a still-developing field, while the state of the art in natural language processing for full-text question answering is a complex

process. To reduce the inherit complexity in these processes, we review simple ways to embed a question. In this part we also experiment with changing some of the parameter dimensions and network characteristics to understand what effect they had on the performance or robustness of these networks. A small script for standardizing the reporting of parameters and for running tests is also built.

Part 3 of the project results in the main discussion of Chapter 5. How do our best-performing network variants compare on fixed preparations of the MNIST database? What is their robustness—how well do they infer the correct classification—when we extract some of the training data in different ways? Significant efforts here include choosing how best to prepare the data, confirming that it was consistent, and collecting statistics about it before beginning the comparison tests.

The following sections will provide more detail on the specific process followed in setting up and running the experiments. Significant discoveries or observations will be discussed in more detail in Chapter 6, while results of experiments will be discussed in Chapter 5.

4.1. Understanding and Preparing the MNIST database

It is fundamentally important to understand as much as possible about what one is exploring, so that one knows what they can actually explore. The first phase of this research effort involves exploring the MNIST data in detail and selecting methods by which the images can be modified and made more difficult for the networks to learn.

Jupyter Notebook is used as the initial development platform. A Python script loads `tensorflow.contrib.learn.python.learn.datasets.mnist` and builds the training and test sets of inputs and labels. The images are 1x784 pixel arrays, which can be re-shaped to 28x28 pixels (this eases visualizing the actual digit). The labels can be stored as either an integer corresponding to the number in the image (0-9), or

a one-hot array, encoding the integer in the corresponding array position. The images themselves are gray-scale; each array position is a floating point number between 0 and 1, representing that pixel's intensity value.

Two initial modifications to the images are needed. First we add color. This is accomplished by making the 28x28x1 pixel image a 28x28x3 pixel image, adding two additional channels. With a 3-channel image, each pixel could then store a red, green, or blue (RGB) value. This is accomplished by adding defined channel values indicated for a given color (3.3) to the MNIST image along the axis grouping each pixel's three channel values. Clipping the values to 1 maintains the expected maximum value for each pixel, and results in a consistent color being applied to the image.

The second need is adding noise to make sure the colors and number shapes weren't too uniform. We create another array with `numpy.random.normal`, using the shape of the 3-channel colored MNIST image, centered on 0 with a spread of 0.2. The images were again added and then clipped to keep the value a maximum of 1.

Finally, to make visualizing the example images easier and to clip the minimum threshold of the images to 0, each array value is inverted and the absolute value is stored. This added slightly more noise for the background of the image and ensured that the encoded digit itself would be displayed with the colors, and the background around it displayed as white.

The initial testing network has 2 CNN layers and 2 fully connected layers, as described in Section 3.2.4. The initial network trains on only the shape labels, which described the number encoded in the MNIST image. Once the networks demonstrates better than 95% training accuracy on the shape labels colors are added and the network adjusted. A peer second fully connected layer split is added to the network to allow shape and color label training. The completion of this first task is the generation of the TwinFC2 network variant.

4.2. Fundamental characteristics of the networks

To support final testing, we need to identify network architectures that learn to differentiate between shapes and colors and to understand how the different hyperparameters change the performance of the networks.

4.2.1 Learning shape and color

Three network variants are explored as described in Section 3.2.4. TwinFC2 is used to explore selective color training. The training and test inputs use all 10 MNIST digits and either 2 or 3 colors. In the 2 color setup, digits 0-4 are colored with the first color and 5-9 with the second. In the 3 color setup, digits 0-4 are colored with either the first or third color and 5-9 with the second or third.

Initially the training is performed by manually setting all of the hyperparameters for the network, running a test and reviewing. This is improved by using an automation script to perform a grid search of different learning rates, momentum, Batch sizes and Epochs lengths. The grid search method eventually identified a high-performing parameter set, using the 10 shapes and 3 colors setup.

4.2.2 Selectively predicting shape or color

The next test demonstrates networks predicting either a color or shape based on an embedded "question." This is used to test whether the network is learning to differentiate between shape and color, rather than memorizing all combinations. Two major modifications are made to the setup. First, a 4th channel is added to the MNIST inputs, within which a question could be stored. Each value in the channel is made a 1 or a 2, representing the questions "which color is this?" or "which shape is this?" When the inputs are processed to add colors, the question would be randomly

selected and set in the 4th channel, after which the list of ground truths is modified to provide the correct one-hot encoded label for the color or the shape. Secondly, the LongFC2 network is created. This has a single second fully connected layer with 20 elements. Each element represents a class for either a shape or a color.

For testing, shapes 0-9 are output elements 0-9, and 10 colors are used occupying elements 10-19. The LongFC2 network is tested using the grid search script with both Softmax and Binary activation functions. A variation of the test is also run to demonstrate multi-hot learning. When the question is not specified—a 0. is left in the 4th channel of the input—the network learns to respond with the element for both the shape and the color. Hyperparameter configurations are discovered that demonstrate high training and test accuracy for the binary activation when testing single-hot classification (a question is always present) and in the multi-hot classification tests, when a question was sometimes absent.

4.2.3 Switching between tasks

Having demonstrated networks learning to classify shape and color independently, and networks selectively responding to a "question," the next level of refinement is training a network to use the same output elements for predicting different classes, depending on the context of the question asked. To test this capability the SharedFC2 network is added. SharedFC2 has a single final fully connected layer, with only 10 elements. In SharedFC2, instead of each class having an output, each output element represents both a shape and a color. The test continued to use the grid search script. Softmax and binary activation functions are tested with this setup, and for both the testing discovered hyperparameters that resulted in high training and test accuracy.

4.2.4 Hyperparameter exploration

In preparation for the final tests we review various hyperparameter configurations. FC1 size: The LongFC2 and ShareFC2 networks are given multiple batteries of the learning rate & momentum grid search, with the FC1 layer set at either 100, 25 or 10 entities. This test is used to determine how the networks' performance would change when there is less bandwidth between the convolution layers and the classifier.

Softmax vs Sigmoid: All three networks are configured with either a Softmax or Sigmoid activation function, and tested with mutual well-performing learning rate and momentum. These tests are used to determine which activation function results in the highest-performing network.

Sliding window & feature maps: During final testing, after 10 batches of learning rate/momentum grid searches of Set 1, we ran a 3-batch grid search to check alternative dimensions for the convolution window and the depth of the convolutional layers. The test is intended to reinforce that the parameters we have been using are providing the best accuracy, or to make an adjustment and use a new combination of hyperparameters for the final tests.

4.3. Preparing MNIST for the main tests

Initial training data is converted from single-channel black-and-white images to 4-channel black-and-white images, ready for a color and a question. Each training step, the batch of random input images would be selected from the test, then colored, given noise and modified with a question. This enabled the training tests to express a vast variety of combinations of data—left long enough, all 55,000 training images may have been experienced at least once with each color and each question. The networks are therefore able to express their end qualities, but it practically meant

that each run of a test is not internally consistent with the next run. Because the final tests would compare the averages of many runs, the data for each test needs to be consistent.

```
def make_color(img, label):
    """
    Return the adjusted image with question, position for the one-hot label,
    the question, and the shape and color label values
    """
    # pick color index
    c = random.randrange(0,9+1,1)
    #color the image, level it
    img = img + colors[c][1]
    img = np.clip(img, a_min=None, a_max=1)
    # add noise, level the image
    img += np.random.normal(0., 0.2, img.shape)
    img = np.clip(img, a_min=0., a_max=1.)

    # set the shape label on 0, color label on 1
    # set the question channel to 1 for shape, 2 for color
    if random.randrange(2) == 0:
        img[:,:,3] = 1. ## Used for Channel trials series embedding
        #img[:,3] = 2.  ## Used for Slice trials series embedding
        q = 1
        output_position = label
    else:
        img[:,:,3] = 2. ## Used for Channel trials series embedding
        #img[:,3] = 2.  ## Used for Slice trials series embedding
        q = 2
        output_position = c

    return img, output_position, q, label, c
```

Listing 4.1: Adding color to an MNIST image

For the final tests we generated three prepared sets of data, separated into training, validation and testing groups. Experiment 1, in which a network would train on all color-shape-question combinations, is the base data preparation for each of three Sets. For Experiment 1, the MNIST inputs and labels are loaded from the Tensorflow `tensorflow.contrib.learn.python.learn.datasets.mnist` distribution. After initial reshaping (as described in Section 3.2.6), 10 percent of the entries from the training set are masked via a random.choice() operation and extracted to

make a validation set.

All of the images are prepared with color and questions as described in Section 3.2.6. Once each image has a color and a question, the question is stored in an array, and a one-hot label for the shape and the color are stored. Additionally, a one-hot label that could be used by the LongFC2 and ShareFC2 networks is created and stored, along with the split shape and color labels in a labels dictionary. LongFC2 required a 20-entity array; "shape" questions used the first 10 positions in the array, and "color" questions used the second 10 positions in the array. ShareFC2 required a 10-entity array; the one-hot label for the asked question is used as the label for this input.

```
    ## Get a list of (shape,color,question) that should be excluded from the
    list
    drops = [(None,None,None)] * (pctdrop * 2)
    h = 1.
    j = 0
    while h < 3.:
        while j < pctdrop:

            # Make a combo, but make sure it doesn't already exist.
            isIn = True
            while isIn is True:
                s = random.randrange(0,9+1,1)
                c = random.randrange(0,9+1,1)
                ck_cmb1 = (s,c,1.)
                ck_cmb2 = (s,c,2.)
                isIn = ck_cmb1 in drops or ck_cmb2 in drops

            # Add to the drop list
            drops[j + (int(h)-1)*pctdrop] = (s,c,h)
            j += 1
        h += 1.
        j = 0
```

Listing 4.2: Experiment 2 process for extracting shape/color combinations

Order is maintained in Experiment 1 (E1) by not shuffling the various lists—the first position in the image array is also it's corresponding label in any of the label arrays, and its corresponding question in the question array. Experiments 2 and 3

(E2, E3) involved holding out 10% of the color-shape-question combinations using one of two strategies (show in Listings 4.2 and 4.3).

To prepare E2, the E1 data set is provided to a script, which selected 10 combinations from question 1 and 10 different combinations from question 2. Any input and corresponding label that has one of these combinations is removed from the main training, validation or test set. The values that are removed from the test set are moved instead to a set of excluded data, to enable testing the trained network on predicting these unseen combinations. The combinations that are excluded are stored in a dictionary of statistics, along with the number of inputs held out, and the distribution of inputs between all the combinations.

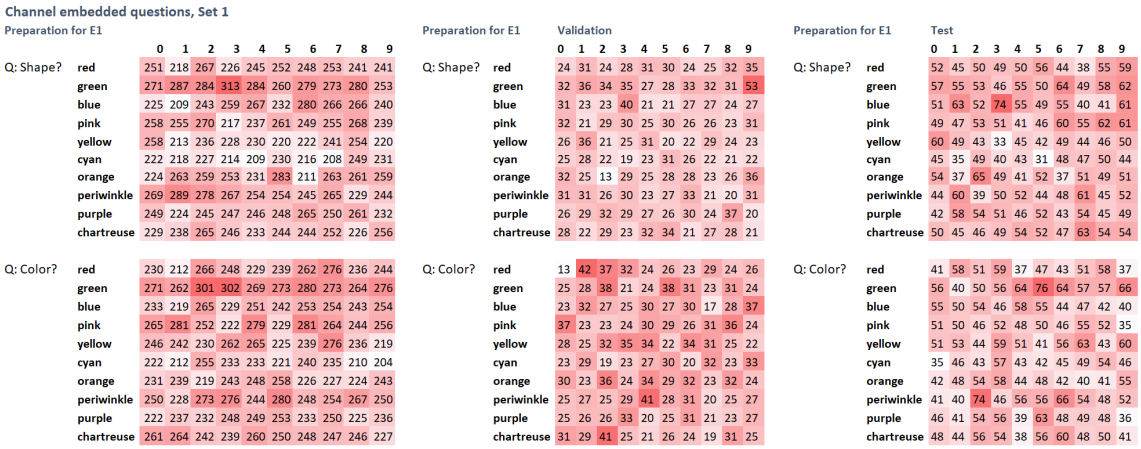
```
    #Get a list of (shape,color) that should be excluded from the list
    drops = [(None,None)] * pctdrop
    i = 0
    while i < pctdrop:

        # Make a combo, but make sure it doesn't aready exist.
        isI = True
        while isI is True:
            s = random.randrange(0,9+1,1)
            c = random.randrange(0,9+1,1)
            ck_cmb = (s,c)
            isI = ck_cmb in drops

        # Add to the drop list
        drops[i] = (s,c)
        i += 1
```

Listing 4.3: Experiment 3 process for extracting shape/color combinations

To prepare E3, the E1 data set is provided to a script, which selected 10 combinations across both question 1 and question 2. These are removed in the same way as E2, and the excluded test items are similarly kept for unseen combination testing. The combinations that are held out are stored in a dictionary of statistics, along with the number of inputs held out, and the distribution of inputs between all the combinations.



Channel embedded questions, Set 1

Preparation for E1

		0	1	2	3	4	5	6	7	8	9
Q: Shape?	red	251	218	267	226	245	252	248	253	241	241
	green	271	287	284	313	284	260	279	273	280	253
	blue	225	209	243	259	267	232	280	266	266	240
	pink	258	255	270	217	237	261	249	255	268	239
	yellow	258	213	236	228	230	220	222	241	254	220
	cyan	222	218	227	214	209	230	216	208	249	231
	orange	224	263	259	253	231	283	211	263	261	259
	periwinkle	269	289	278	267	254	254	245	265	229	244
	purple	249	224	245	247	246	248	265	250	261	232
	chartreuse	229	238	265	246	233	244	244	252	226	256
Q: Color?	red	230	212	266	248	229	239	262	276	236	244
	green	271	262	301	302	269	273	280	273	264	276
	blue	233	219	265	229	251	242	253	254	243	254
	pink	265	281	252	222	279	229	281	264	244	256
	yellow	246	242	230	262	265	225	239	276	236	219
	cyan	222	212	255	233	233	221	240	235	210	204
	orange	231	239	219	243	248	258	226	227	224	243
	periwinkle	250	228	273	276	244	280	248	254	267	250
	purple	222	237	232	248	249	253	233	250	225	236
	chartreuse	261	264	242	239	260	250	248	247	246	227

Preparation for E1 Validation

		0	1	2	3	4	5	6	7	8	9
Q: Shape?	red	24	31	24	28	31	30	24	25	32	35
	green	32	36	34	35	27	28	33	32	31	53
	blue	31	23	23	40	21	21	27	27	24	27
	pink	32	21	29	30	25	30	26	26	23	31
	yellow	26	36	21	25	31	20	22	29	24	23
	cyan	25	28	22	19	23	31	26	22	21	22
	orange	32	25	13	29	25	28	28	23	26	36
	periwinkle	31	31	26	30	23	27	33	21	20	31
	purple	26	29	32	29	27	26	30	24	37	20
	chartreuse	28	22	29	23	32	34	21	27	28	21
Q: Color?	red	13	42	37	32	24	26	23	29	24	26
	green	25	28	38	21	24	38	31	23	31	24
	blue	23	32	27	25	30	27	30	17	28	37
	pink	37	23	23	24	30	29	26	31	36	24
	yellow	28	25	32	35	34	22	34	31	25	22
	cyan	23	29	19	23	27	30	20	32	23	33
	orange	30	23	36	24	34	29	32	23	32	24
	periwinkle	25	27	25	29	41	28	31	20	25	27
	purple	25	26	26	33	20	25	31	21	23	27
	chartreuse	31	29	41	25	21	26	24	19	31	25

Preparation for E1 Test

		0	1	2	3	4	5	6	7	8	9
Q: Shape?	red	52	45	50	49	50	56	44	38	55	59
	green	57	55	53	46	55	50	64	49	58	62
	blue	51	63	52	74	55	49	55	40	41	61
	pink	49	47	53	51	41	46	60	55	62	61
	yellow	60	49	43	33	45	42	49	44	46	50
	cyan	45	35	49	40	43	31	48	47	50	44
	orange	54	37	65	49	41	52	37	51	49	51
	periwinkle	44	60	39	50	52	44	48	61	45	52
	purple	42	58	54	51	46	52	43	54	45	49
	chartreuse	50	45	46	49	54	52	47	63	54	54
Q: Color?	red	41	58	51	59	37	47	43	51	58	37
	green	56	40	50	56	64	76	64	57	57	66
	blue	55	50	54	46	58	55	44	47	42	40
	pink	51	50	46	52	48	50	46	55	52	35
	yellow	51	53	44	59	51	41	56	63	43	60
	cyan	35	46	43	57	43	42	45	49	54	46
	orange	42	48	54	58	44	48	42	40	41	55
	periwinkle	41	40	74	46	56	56	66	54	48	52
	purple	46	41	54	56	39	63	48	49	48	36
	chartreuse	48	44	56	54	38	56	60	48	50	41

Figure 4.1: Example prepared data distribution for Experiment 1 in Set 1, Channel embedded preparation

After the first three experiments are run, we extend each Set with a preparation of E2 and E3 that held out 30% of the combinations from the E1 data. The inputs from E1, and the stats dictionaries for Experiment 2 and 3 10% holdout tests, are fed to the E2 and E3 preparation script along with a new 30% holdout percentage argument. The script copied the 10% holdout information for each experiment and then extracted the additional combinations to meet the new holdout percentage. The same process is performed with 30% stats being provided for the 50% holdout versions of the experiments.

This process is specific to ensure that the combinations held out in the less-empty version of the experiment continued to not exist in the more-empty version, in order to make the results more comparable. The process is performed for each of the three Preparation Sets, resulting in seven experiments to run for three preparations of the MNIST data. The specific data preparations for the tests reported here is preserved for review.

As noted above, after preparing the data for an Experiment, the distribution of examples for each combination of color and shape for each question are added to

the Set. An example distribution is illustrated in Figure 4.1. The heat maps represent the relative distribution of the images in their combinations.

I update the data sets one final time in order to train TwinFC2 on only the ground truth exclusive to the question embedded in a given input. We add a set of labels to each experiment; for any input, the label for the classification category asked by the question was left intact, but the other label was replaced by a zeros array. The TwinFC2 network variant would have to learn to answer with an empty array for the ignored category, and with a prediction for the requested category. This label preparation is referred to as "exclusive split" labels. The findings in Section 5 refer to these runs as xTwinFC2.

4.4. Final tests

In order to report effects with some certainty, we needed to be able to report standard deviation as well as mean accuracy and loss. For each major set of trials—Softmax activation with X-Y axis slice question insertion ("Channel insertion" or "Channel"), Softmax activation with Y-Z axis slice question insertion ("Slice insertion" or "Slice"), or Sigmoid activation with Channel insertion—I first ran a grid search of Experiment 1. Each network variant is trained on 12-14 combinations of learning rate and momentum, with learning rates between 0.225 and 0.0002, and momenta either 0.9 or 0.09.

After 8-10 trials are collected for each prepared set, we selected the 2 or 3 learning rate (LR) and momentum (MM) combinations that provided the best mean validation accuracy and lowest mean standard deviation. We then collected 30-40 trials for Experiment 2 and 3, at 10, 30 and 50 percent input combination hold-out (and collected additional trials for E1 to maintain consistent/comparable results). In all, between 90 and 120 trials are averaged to report mean accuracy, loss and standard

deviation in our final reports. The results of the grid searches and the E2/E3 deep searches in Chapter 5.

When training the networks on either the Channel insertion or Slice insertion data preparations, identical methods are used to present the method to each network variant. When training the networks on Sigmoid activation versus Softmax activation, the presentation of data for the TwinFC2 network variant is modified. The Softmax version of TwinFC2, named just $TwinFC2$, received one array of inputs and a separate array of ground truths (called "Split" labels) that always contained the label for the shape and the color in the corresponding input image. Early tests running TwinFC2 with the Exclusive Split labels (see page 34) demonstrate that Softmax activation by default cannot return a 0's array. Therefore the network will surpass 50 percent accuracy. By converting TwinFC2 to use Sigmoid output activation the Twin network architecture variant is able to successfully train on the Exclusive Split. Twin architectures (either TwinFC2 or SigTwinFC2) that are trained on the Exclusive Split labels are prefixed with an "x" in the summary trial data, e.g. "xSigTwinFC2."

4.4.1 Automating training

Collecting consistent, comparable data is as important as the design of the experiments. Each network script includes a method for running validation of the network performance, which is then used to end training when the network stops improving. Each also includes a method for running tests after training completes and then reporting the test, validation and training results. These two observation methods produce consistent and comparable information from each neural network variant. Additionally, when a hold-out experiment is run, a test epoch is conducted as well on the test inputs that have been excluded; this allows us to compare end performance of the network on problems it explicitly learns the answer to, with problems

for which it has to infer the answer. When a network complete its training and test reporting, this information is returned to the network training management script. The training management script then stores the information in a designated CSV file. Training step, accuracy and loss is captured for the training/validation step at which peak training is determined to have occurred.

A high volume of trials would be difficult to collect manually. A batch management script automates the execution of many repeat trials. Each major set of trials, as noted above, standardized all but one significant architecture component and two hyperparameters. Therefore, the batch management script specifies a dictionary of common settings, the prepared sets to use, the experiments to run and the LR & MM combinations to test with. A batch management script would then be set to run on a computer for many hours, with that machine collecting varrying sets of tests.

4.4.2 Preparation of results

When a network fails to train, all accuracy, loss and generations results are set to -1. The problem of training an end-to-end CNN to predict MNIST digits is well-solved. Because the problem is well-defined, we heavily penalize configurations that are unstable enough to fail to train. Having said that, it is rare for a hyperparameter configuration to cause a network variant to fail, only ever showing up in the Softmax-Slice trials with an LR of 0.9 in both MM and LR of 0.225 and MM of 0.09.

Accuracy reported in Chapter 5 will be the mean taken over all three sets of test results.

Chapter 5: Results and Evaluation

> There isn't a way things *should* be. There's just what happens, and what we do.
>
> Terry Pratchett, *A Hat Full of Sky*

In this chapter we report the results of initial parameter testing and the main comparisons of network architecture variations over each data preparation. Additional charts and graphs are available in the Appendix on page 63.

5.1. Results of network characteristics tests

In Section 4.2.4 we described tests used to determine optimal internal parameters for the layers shared between the network variants. Their results were used to are briefly reviewed below.

5.1.1 FC1 size

I ran tests of setting the 1st fully connected layer's depth to 100, 25, and then from 10 down to 2. A single battery of tests was collected using the LongFC2 variant with Sigmoid activation. These trials demonstrated that the network lost no performance at least until setting to 10, and then slowly until the size was 5, after which validation accuracy dropped sharply. In order to remove the FC1 layer as a variable from the final analysis (focusing just on the effects of the output layer shape)

we decided to leave FC1 depth set at 100.

5.1.2 Softmax vs Sigmoid output activation

Multiple individual trials were run comparing the performance of the network variants using both Softmax and Sigmoid functions for output activation. The results in low volume were inconclusive, which encouraged a higher volume of testing. This additional exploration is reviewed in Section 5.2 and its subsections.

5.1.3 Convolution filters & feature maps

Mean accuracy of convolution window and feature sizes, %, average of 3 trials, select learning rates where accuracy > 90%

Momentum	Shape Test Accuracy				Color Test Accuracy			
Learning rate	4x4 5	5x5 25	4x4 25	5x5 5	4x4 5	5x5 25	4x4 25	5x5 5
$M = 0.09$	92.96	91.45	94.14	91.96	98.56	94.32	99.70	97.30
0.003515625	89.62	91.42	91.31	90.59	98.76	99.58	99.55	99.25
0.0140625	94.33	93.09	94.92	94.13	99.65	96.01	99.81	99.39
0.05625	94.93	89.84	96.19	91.16	97.27	87.36	99.75	93.26
$M = 0.9$	78.99	71.30	81.47	69.71	80.62	72.12	82.62	70.44
0.003515625	95.35	95.95	96.10	95.15	99.59	99.79	99.85	99.34
0.0140625	95.63	96.46	96.98	89.02	98.70	99.49	99.89	91.67
0.05625	45.98	21.49	51.32	24.97	43.58	17.09	48.13	20.32

Table 5.1: Convolution windows and feature maps, % mean test accuracy

Three batches of trials were run for each learning rate and momentum combination, with the Conv1 and Conv2 layer (see Section 3.2.4) convolution filter window dimensions set to either 4x4 or 5x5, and the first Conv1 layer feature map depth set to either 5 or 25. Table 5.1 shows the learning rate + momentum combinations where accuracy was greater than 90 percent. The 4x4 window with 25 feature map depth was roughly tied with the 5x5 and 25 settings for the very top rates, but the 5x5 and 25 settings combination fell off much quicker. Overall the 4x4 window with 25 fea-

ture maps provided a higher overall performing network across all the learning rates. Page 64 reports the full table of percentage difference that the different convolution windows and feature map depths had from the chosen 4x4 & 25 parameters.

5.2. Results of main experiments, multi-task training with increasingly scarce data

Experiment 1 best mean test results
Softmax and Sigmoid Channel n=90, Softmax Slice n=120
Momentum = 0.9 for all reported configurations

Network			Steps		Shape		Color	
Series	Variant	LR	Ave	SD	Acc%	SD	Acc%	SD
Soft C	Long	0.003515625	6347.02	1525.55	96.79	0.38	99.64	1.35
Soft S	Long	0.0140625	5442.13	1911.96	97.52	0.35	99.87	0.17
Sig C	Long	0.05625	6993.07	1958.54	99.64	0.04	99.98	0.02
Soft C	Share	0.003515625	6237.51	1741.27	96.78	0.41	99.77	0.30
Soft S	Share	0.0140625	5130.93	1762.99	97.33	0.44	99.85	0.21
Sig C	Share	0.05625	6058.67	1960.00	99.38	0.08	99.95	0.04
Soft C	Twin	0.0140625	5026.49	1545.22	97.49	0.33	99.79	0.25
Soft S	Twin	0.0140625	5535.47	1778.01	97.85	0.28	99.93	0.05
Sig C	xTwin	0.05625	6462.58	1813.68	96.82	0.37	99.75	0.46

Table 5.2: Network variant average Experiment 1 test accuracy, best learning rate & momentum

Table 5.2 presents the best performance from each network variant / data preparation. When the Softmax output logits activation function was used, the network variants training on the Slice question embedding demonstrates slightly higher overall accuracy. The accuracy for the LongFC2 network variant in particular is higher than the combined standard deviations of the two variants. The variants training on the Slice embedded questions also reach optimal validation accuracy quicker as well, requiring a mean 14.3 and 17.8% fewer training steps to reach completion for the Long and ShareFC2 variants, respectively. While the Slice versions of the Long and Share variants are both quicker and more accurate, the TwinFC2 variant was nearly

10% quicker to train in the Channel-embedded trials.

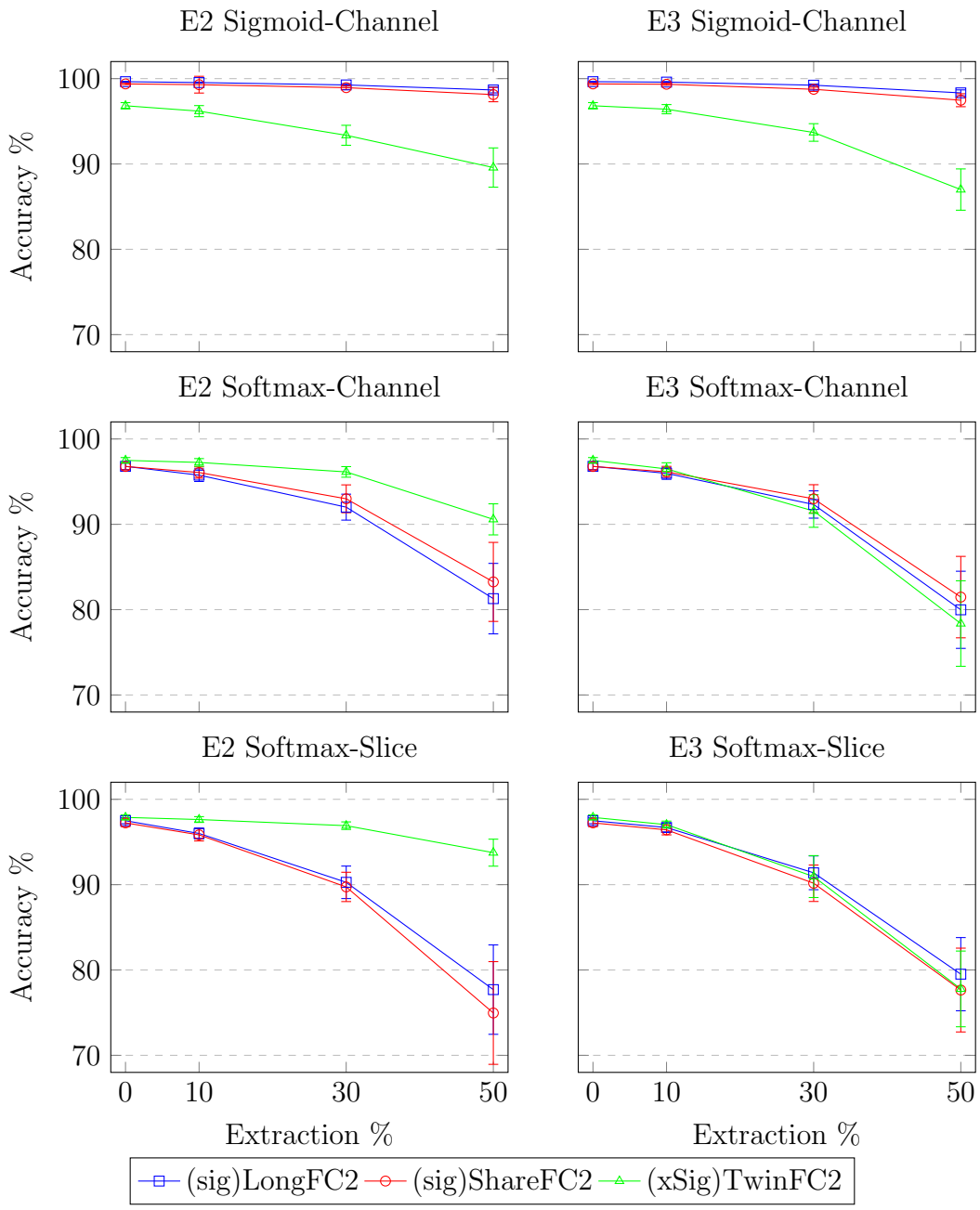


Figure 5.1: Comparison of shape prediction accuracy between trial series, best learning rate/momentum configuration. Mean accuracy with standard deviation. Softmax- and Sigmoid-Channel n=90, Softmax-Slice n=120

The learning rate & momentum configuration reported for each network vari-

ant was selected based on both highest mean shape and color prediction validation accuracy, and selecting for the configuration that presented lower standard deviation when results were too close to confidently separate them on accuracy alone. The results from Experiment 1 represented the network's maximum expected performance and were used to select for reporting.

Color accuracy is very stable through all of the experiments and configurations. These results are visualized in Figure 5.2 The lowest registered mean color accuracy was $96.92\%(\pm 1.41, n = 120)$ for Softmax-Slice ShareFC2. Shape accuracy demonstrates more performance loss as combinations were extracted from the training data. The shape accuracy performance of each network variant over the 0-50% extraction rate is illustrated in Figure 5.1, and the performance

5.2.1 Softmax output activation with Channel-embedded questions

Selecting one best configuration for the Softmax-Channel trials series is slightly complicated by the wide standard deviation for the 0.0140625 learning rate. While 0.0140625 for Experiment 1 produces marginally better accuracy than the trials at 0.003515625, the accuracy for the remaining experiments demonstrated significant deviation vs the slightly slower rate.

The mean shape accuracy for LongFC2 at $LR = 0.0140625$ in E2 and E3, with 50% combinations extraction, was $85.96 \pm 4.44\%$ and $85.15 \pm 4.52\%$, respectively. These results are within the standard deviation of the mean accuracy presented for 0.003515625, but would actually place LongFC2 as more accurate than the TwinFC2 network. For E3 with 30% extraction, however, LongFC2 presents, for this project, a massive standard deviation of $\pm 13.45\%$. We would prefer to report the more exciting results, but because of the high variability in 0.0140625 we report here instead the results for LR = 0.003515625. The test results for both learning rates are however

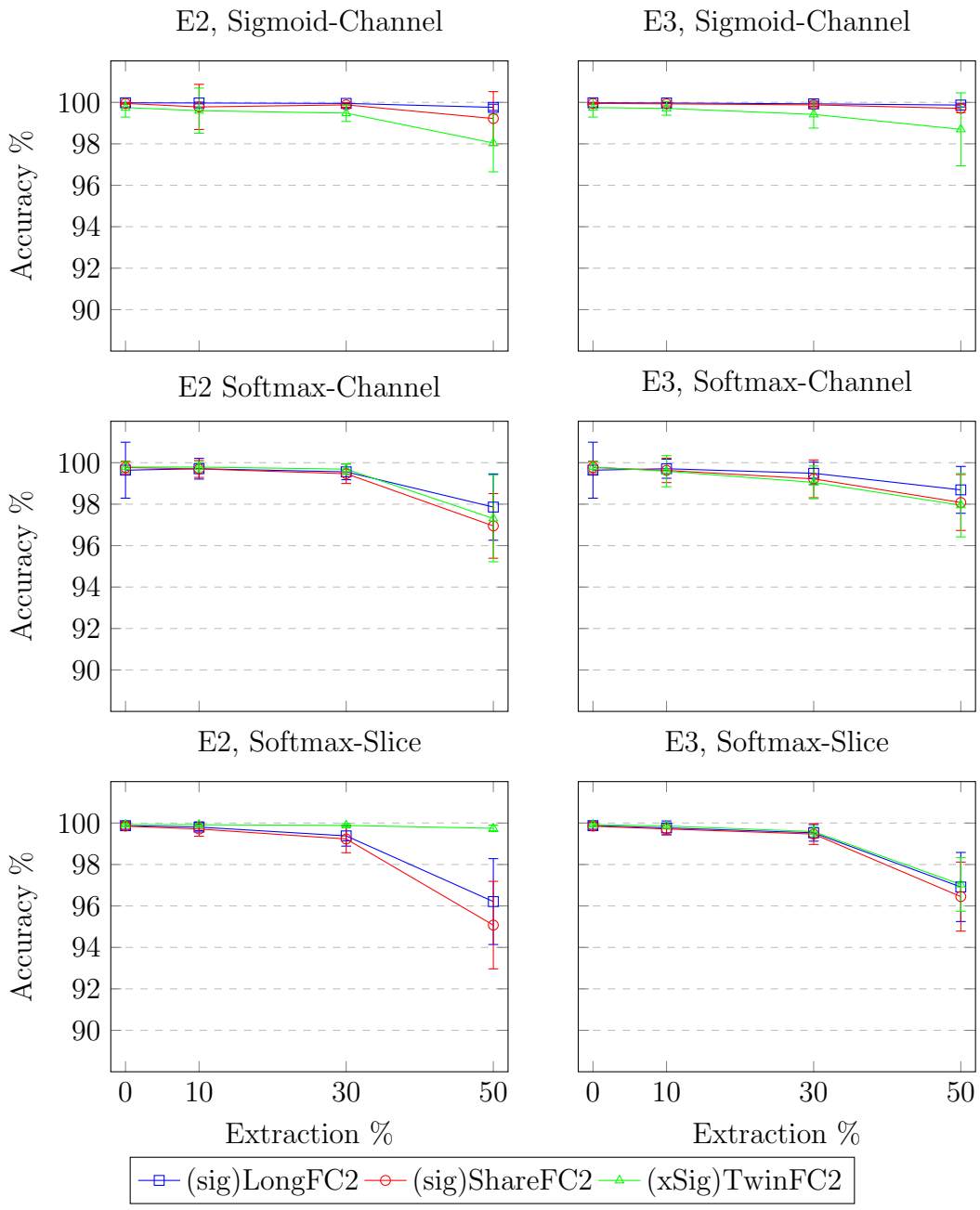


Figure 5.2: Comparison of color prediction accuracy between trial series, best learning rate/momentum configuration. Mean accuracy with standard deviation. Softmax- and Sigmoid-Channel n=90, Softmax-Slice n=120

included in the appendix for review (see page 65).

The Softmax-Channel trial series demonstrates little loss in color classification

Test results for the Softmax-Channel series.
Momentum = 0.9, n=90 per result
LR: LongFC2, ShareFC2 = 0.003515625, TwinFC2 = 0.0140625

	Long		Share		Twin	
Experiment	Mean	StdDev	Mean	StdDev	Mean	StdDev
Test results, shape prediction						
E1	96.79	0.38	96.78	0.41	97.49	0.33
E2 10	95.76	0.74	96.07	0.66	97.25	0.45
E2 30	92.01	1.52	92.99	1.63	96.14	0.62
E2 50	81.29	4.13	83.25	4.63	90.58	1.82
E3 10	95.97	0.70	96.15	0.64	96.49	0.72
E3 30	92.32	1.60	92.99	1.65	91.56	1.91
E3 50	79.98	4.52	81.47	4.77	78.37	5.01
Test results, color prediction						
E1	99.64	1.35	99.77	0.30	99.79	0.25
E2 10	99.71	0.50	99.70	0.41	99.79	0.23
E2 30	99.56	0.38	99.47	0.47	99.69	0.26
E2 50	97.86	1.60	96.95	1.56	97.31	2.09
E3 10	99.71	0.46	99.63	0.59	99.59	0.75
E3 30	99.49	0.54	99.22	0.91	99.05	0.79
E3 50	98.69	1.13	98.08	1.35	97.95	1.53

Table 5.3: Softmax-Channel Trial Series, prediction test accuracy

task accuracy and varying loss in shape classification task accuracy. The TwinFC2 network variant falls from shape prediction accuracy of $97.49\%(\pm 0.33, n = 90)$ to only $90.58\%(\pm 01.82, n = 90)$ for 50% combination exclusion in the Experiment 2 tests, versus $78.37\%(\pm 5.01, n = 90)$ accuracy in the 50% exclusion tests for Experiment 3. All of the networks performed roughly the same during the Experiment 3 tests and LongFC2 & ShareFC2 track each others performance for Experiment 2 as well.

During volume training with the Softmax variants on the Channel-embedded data, standard deviation remained significant for the 0.0140625 learning rate. Particularly for ShareFC2, across all sets there was a greater than 12% standard deviation in four of the seven tests, and higher than 6% standard deviation in all of them. We didn't see these effects with the Sigmoid output activation variants or with the

Slice-embedded data. Table A.3 in the appendix illustrates the significant variations experienced in the results for the two highest performing learning rates / momentum configurations.

5.2.2 Softmax output activation with Slice-embedded questions

As in the Softmax-Channel trials series, the LongFC2 and ShareFC2 networks performed roughly the same, and TwinFC2 retained significantly more accuracy in E2. For the E1 runs all three variants achieve a slightly higher mean accuracy of $97.52\%(\pm 0.35, n = 120)$, $97.33\%(\pm 0.44, n = 120)$, and $97.85\%(\pm 0.28, n = 120)$ for Long, Share and TwinFC2. 10% holdout remains slightly higher as well for E2 and E3, while drop-off is a slightly greater but less variable $80.99\%(\pm 3.74, n = 120)$, $79.50\%(\pm 4.14, n = 120)$, and $79.31\%(\pm 3.95, n = 120)$ at 50% exclusion for Long, Share and TwinFC2, respectively. Table A.4 in the Appendix illustrates the results for the two highest performing learning rates / momentum configurations.

5.2.3 Sigmoid output activation with Channel-embedded questions

The network variants trained on the Channel-embedded questions with a Sigmoid output activation vastly outperformed the network variants configured with Softmax output activation. Mean accuracy for E1 is $99.64\%(\pm 0.04, n = 90)$, 99.38% $(\pm 0.08, n = 90)$, and $96.82\%(\pm 0.37, n = 90)$ for the sigLongFC2, sigShareFC2 and xSigTwinFC2 variants, respectively. SigLongFC2 loses only 0.97% and 1.31% at 50% exclusion for E2 and E3. The Twin FC2 architecture in this setup is the poorer performer, tracking to $89.53\%(\pm 2.30, n = 90)$ and $87.00\%(\pm 2.42, n = 90)$ for E2 50% and E3 50%. Table A.5 in the Appendix illustrates the results for the two highest performing learning rates / momentum configurations.

Test results for the Softmax-Slice series.
Momentum = 0.9, n=120 per result
Learning Rate = 0.0140625

	Long		Share		Twin	
Experiment	Mean	StdDev	Mean	StdDev	Mean	StdDev
Test results, shape prediction						
E1	97.52	0.35	97.33	0.44	97.85	0.28
E2 10	96.14	0.67	95.94	0.74	97.58	0.37
E2 30	90.47	1.90	89.85	2.13	96.85	0.45
E2 50	76.32	4.56	72.04	4.72	92.56	1.47
E3 10	96.56	0.80	96.35	0.84	96.81	0.65
E3 30	91.62	2.17	90.34	2.47	91.47	2.42
E3 50	80.99	3.74	79.50	4.14	79.31	3.95
Test results, color prediction						
E1	99.87	0.17	99.85	0.21	99.93	0.05
E2 10	99.81	0.18	99.72	0.36	99.92	0.05
E2 30	99.32	0.57	99.18	0.71	99.88	0.09
E2 50	96.25	1.82	95.37	1.79	99.75	0.13
E3 10	99.78	0.34	99.73	0.27	99.87	0.09
E3 30	99.59	0.36	99.52	0.47	99.63	0.24
E3 50	97.40	1.50	96.92	1.41	97.51	1.14

Table 5.4: Softmax-Slice trial series, prediction test accuracy

5.2.4 Comparing Channel-embedded preparations and Softmax-activated variants

Between the two sets of results using the Channel-embedding strategy, the Sigmoid output activation is at a clear advantage. xSigTwinFC2 is an interesting comparison between the Sofmax and Sigmoid Channel sets. At 10 and 30% combination exclusion, it still roughly matches performance. At E2 50% holdout, xSigTwinFC2 underperforms TwinFC2 at $89.53 \pm 2.30\%$ vs $90.58 \pm 1.82\%$, but beats TwinFC2 by 8.53% ($87.00 \pm 2.42\%$ vs $78.37 \pm 5.01\%$).

The Sigmoid activation function provided overall more consistent results, but did so at the cost of training time. Sigmoid-Channel E1 average training steps to peak

Test results for the Sigmoid-Channel series.
Momentum = 0.9, n=90 per result
Learning Rate = 0.05625

	sigLong		sigShare		xSigTwin	
Experiment	Mean	StdDev	Mean	StdDev	Mean	StdDev
Test results, shape prediction						
E1	99.64	0.04	99.38	0.08	96.82	0.37
E2 10	99.54	0.06	99.29	0.98	96.19	0.65
E2 30	99.26	0.14	98.94	0.22	93.36	1.17
E2 50	98.67	0.34	98.13	0.84	89.58	2.30
E3 10	99.59	0.07	99.34	0.11	96.42	0.54
E3 30	99.23	0.13	98.75	0.25	93.69	1.03
E3 50	98.33	0.35	97.47	0.76	87.00	2.42
Test results, color prediction						
E1	99.98	0.02	99.95	0.04	99.75	0.46
E2 10	99.97	0.04	99.78	1.09	99.60	1.09
E2 30	99.95	0.03	99.88	0.11	99.48	0.40
E2 50	99.76	0.15	99.22	1.30	98.04	1.39
E3 10	99.97	0.02	99.93	0.10	99.71	0.33
E3 30	99.93	0.13	99.88	0.09	99.42	0.66
E3 50	99.87	0.09	99.71	0.20	98.70	1.76

Table 5.5: Sigmoid-Channel Trial Series, prediction test accuracy

performance was 6504.77 ± 1943.03, versus 5870.34 ± 1604.01 for the Softmax-Channel networks.

For the Sofmax-Slice trial series, the question was embedded as a 1x28x4 slice across all four channels of the input image. What is most interesting in this regard is the change in performance of the TwinFC2 network. For the Experiment 2 series of tests, TwinFC2 learning the Slice data outperformed the other two network variants by $16.24 to 20.52\%$. Color performance was closer overall; the variants lost less than 5% accuracy from 0 to 50% holdout. Except the TwinFC2 E2 accuracy, the overall performance of the Softmax-Slice networks was lower than the Softmax-Channel networks.

Chapter 6: Summary and Conclusions

In this project we explored the way fundamental components of an image-classifying neural network change the way that network responds to scarce training environments. Training a Convolutional Neural Network to classify the MNIST database is the "Hello World" of deep learning computer vision. By starting with this very well known problem and adding controlled complexity, we provided a set of results that can be compared against the corpus of analysis and understanding that extends from other MNIST work. Adding a second classification category and requiring the network to make a prediction provided a platform for testing how processing the final network output effects its overall accuracy. By removing increasingly more combinations of MNIST digit and color combinations from the training data, we forced each variant of that neural network to infer more of its predictions from related experience.

Some of the results from the testing were expected—it is well known that Sigmoid functions take longer to converge than Softmax functions. Some results were less expected. Performance for variants with single FC2 Tensors was relatively equal, whether the network had to use the same output position to make different predictions. Embedding the question across layers, instead of in a dedicated channel, resulted in slightly better full-combination training but overall worse performance for all the network variants. We are encouraged by the ease with with the networks learned to differentiate between the questions and am curious how detailed a question

could be, and by how much a question could be compressed, with the network still learning to understand and make correct predictions.

Surprising was the accuracy of the Sigmoid-activated networks. The default design for MNIST classification is to use a Softmax output activation, but the results here suggest that, excusing a slightly longer training period, the Sigmoid output activation is a superior choice. It is of particular note how little accuracy the Sigmoid series networks lost, even when half of the combinations were excluded. The effect of independently training each output channel from the network appears to make the network learn the shapes and colors and resist simply learning the specifically observed combinations.

The Sigmoid activation was also most effective when there was a single output Tensor instead of two peer output Tensors. This has the benefit of requiring half as many connections between the two fully connected layers, and could encourage a more generally usable architecture. When paired with continual learning strategies, for example, a ShareFC2 variant could potentially learn a third new task with its existing output layer.

6.1. Limitations and Known Issues

The TwinFC2 network variants used a very simple process for averaging the loss between the two output Tensors. It is possible that this variant would perform better if a more clever algorithm were developed to bias optimization toward the lagging task's loss.

The Validation set for the data was held statically separate from the training data – every trial run with a prepared set sees the same training and validation examples. This was done to introduce fewer variation in the training experience of the networks. However, it may be that part of the specific performance of the

networks is due to small irregularities in the validation set. Given additional time, we would reinforce the confidence of our findings by running the trials again with a strategy of selecting a random validation data set from the training data when each batch of tests start.

The Validation and Test data in each Set, while each building off the previous hold-out experiment's combination exclusions, do not explicitly use the same exclusion combinations that the Training data. Therefore, test accuracy is to some percentage for each Experiment representing both the network's performance on what it saw, as well as it's ability to infer answers. In the same way, the hold-out examples are not a true example of exclusively untrained predictions. The excluded combination sets for the training, validation and test data do intersect, and by weighting and combining the results for the test and excluded test results, we get an accurate test percentage. In repeat tests we would, however, use the hold-out combinations from training data for the validation and testing data as well. This would allow more direct interpretation of the network variant's ability to infer from not-experienced combinations.

6.2. Further work

The performance of the network variants with Sigmoid output activation under high combination exclusion was significant and warrants further investigation. Running similar training data classification exclusion setups, with more difficult problems like question answering in the CLEVR data set (Johnson et al., 2016) or real-world problems like identifying facial features (Jain & Learned-Miller, 2010), would be interesting. Such projects may develop further insight in how fundamental components influence a network's ability to learn when the data domain is scarce.

Late in the project's development we experimented with a network configu-

ration that would approximate a network routing protocol. Two ShareFC2 variants were trained with the split labels: one on the shape classification task and the other on the color task. A third ShareFC2 network was trained to select which of the experts to send an input to in order to get a correct result. The design was promising: the two "expert" networks trained their single tasks with fewer steps than the multi-task networks required, and the "router" network needed only dozens of steps to achieve near-ideal routing accuracy. This is similar to an architecture proposed in Rosenbaum et al. (2017), but used networks as discrete entities instead of an executive agent inside an end-to-end neural network selecting internal layers to pass an input through.

The collaborative-routing system was different enough from the rest of this thesis's work to set it aside for separate study. We would apply these lessons forward to study that collaborative-routing system more thoroughly. A network that can either answer a question, or, if it identifies that it does not know the answer, learns to "ask" the question to a different network, could be of particular use in broad-skill learning environments. A pre-trained network with this skill would hypothetically be able to "decide" to learn a new task with which its existing training is aligned, or still achieve success by declining to internalize the task, instead forwarding the problem to a different network that learns it quicker.

References

Andreas, J., Rohrbach, M., Darrell, T., & Klein, D. (2015). Neural Module Networks. *arXiv:1511.02799 [cs]*. arXiv: 1511.02799.

Bottou, L. (2014). From machine learning to machine reasoning. *Machine Learning*, 94(2), 133–149.

Chandar, S., Ahn, S., Larochelle, H., Vincent, P., Tesauro, G., & Bengio, Y. (2016). Hierarchical Memory Networks.

Conway, M. E. (1968). How Do Committees Invent? *Datamation*, 1968(April), 28–31.

Foerster, J. N., Assael, Y. M., de Freitas, N., & Whiteson, S. (2016). Learning to Communicate with Deep Multi-Agent Reinforcement Learning. *arXiv:1605.06676 [cs]*. arXiv: 1605.06676.

Foerster, J. N., Chen, R. Y., Al-Shedivat, M., Whiteson, S., Abbeel, P., & Mordatch, I. (2017). Learning with Opponent-Learning Awareness. *arXiv:1709.04326 [cs]*. arXiv: 1709.04326.

Graves, A., Wayne, G., & Danihelka, I. (2014). Neural Turing Machines. *arXiv:1410.5401 [cs]*. arXiv: 1410.5401.

Gulcehre, C., Chandar, S., Cho, K., & Bengio, Y. (2016). Dynamic Neural Turing Machine with Soft and Hard Addressing Schemes. *arXiv:1607.00036 [cs]*. arXiv: 1607.00036.

Gulcehre, C., Chandar, S., Cho, K., & Bengio, Y. (2018). Dynamic Neural Turing Machine with Continuous and Discrete Addressing Schemes. *Neural Computation*, 30(4), 857–884.

Hochreiter, S. & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780.

Hohenecker, P. & Lukasiewicz, T. (2018). Ontology Reasoning with Deep Neural Networks. *arXiv:1808.07980 [cs]*. arXiv: 1808.07980.

Hudson, D. A. & Manning, C. D. (2018). Compositional Attention Networks for Machine Reasoning. *arXiv:1803.03067 [cs]*. arXiv: 1803.03067.

Jain, V. & Learned-Miller, E. (2010). *FDDB: A Benchmark for Face Detection in Unconstrained Settings*. Technical Report UM-CS-2010-009, University of Massachusetts, Amherst.

Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Zitnick, C. L., & Girshick, R. (2016). CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning. *arXiv:1612.06890 [cs]*. arXiv: 1612.06890.

Karpathy, A. (2015). The Unreasonable Effectiveness of Recurrent Neural Networks.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2323.

Marois, V., Jayram, T. S., Albouy, V., Kornuta, T., Bouhadjar, Y., & Ozcan, A. S. (2018). On transfer learning using a MAC model variant. *arXiv:1811.06529 [cs]*. arXiv: 1811.06529.

Mikolov, T., Joulin, A., & Baroni, M. (2015). A Roadmap towards Machine Intelligence. *arXiv:1511.08130 [cs]*. arXiv: 1511.08130.

Mordatch, I. & Abbeel, P. (2017). Emergence of Grounded Compositional Language in Multi-Agent Populations. *arXiv:1703.04908 [cs]*. arXiv: 1703.04908.

Palm, R. B., Paquet, U., & Winther, O. (2017). Recurrent Relational Networks. *arXiv:1711.08028 [cs]*. arXiv: 1711.08028.

Perez, E., Strub, F., de Vries, H., Dumoulin, V., & Courville, A. (2017). FiLM: Visual Reasoning with a General Conditioning Layer. *arXiv:1709.07871 [cs, stat]*. arXiv: 1709.07871.

Rosenbaum, C., Klinger, T., & Riemer, M. (2017). Routing Networks: Adaptive Selection of Non-linear Functions for Multi-Task Learning. *arXiv:1711.01239 [cs]*. arXiv: 1711.01239.

Ruder, S. (2017). An Overview of Multi-Task Learning in Deep Neural Networks. *arXiv:1706.05098 [cs, stat]*. arXiv: 1706.05098.

Ruder, S., Bingel, J., Augenstein, I., & Sgaard, A. (2017). Sluice networks: Learning what to share between loosely related tasks. *CoRR*, abs/1705.08142.

Santoro, A., Raposo, D., Barrett, D. G. T., Malinowski, M., Pascanu, R., Battaglia, P., & Lillicrap, T. (2017). A simple neural network module for relational reasoning. *arXiv:1706.01427 [cs]*. arXiv: 1706.01427.

Simard, P., Steinkraus, D., & Platt, J. (2003). Best practices for convolutional neural networks applied to visual document analysis. In *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, volume 1 (pp. 958–963). Edinburgh, UK: IEEE Comput. Soc.

Weston, J., Chopra, S., & Bordes, A. (2014). Memory Networks. *arXiv:1410.3916 [cs, stat]*. arXiv: 1410.3916.

Woodward, M. & Finn, C. (2017). Active One-shot Learning. *arXiv:1702.06559 [cs]*. arXiv: 1702.06559.

Zaremba, W. & Sutskever, I. (2014). Learning to Execute. *arXiv:1410.4615 [cs]*. arXiv: 1410.4615.

Glossary

Argmax A function that determines the largest values in a given array.. 14

Batch size A predetermined number of inputs presented to a neural network for training, validation or testing during one Step. 32

Deep Neural Network A neural network containing more than 1 hidden layer. 9, 10

Epoch One training, validation or test pass through an entire data set. 1 Epoch is generally the number of steps determined by dividing the length of the data set by a batch size. 32

hyperparameter A shape or functional attribute of a neural network that can be externally modified, such as a learning rate or layer dimension.. 32, 34

multi-layer perceptron A form of artificial network containing at least an input, a hidden layer and an output, using non-linear activation.. 12

Multi-task Learning MTL, a form of Neural Network training in which a network learns to perform more than one classification, recall or response activity, either simultaneously or sequentially. 4

one-hot The name for an array where one array member is non-zero (commonly equals 1) and all the other array members are 0. The array [0,0,1,0] is a one-hot array.. 25

prepared set A collection of training, validation and test sets built around a single coloring and question-embedding preparation of the MNIST datasets. Three in total were created for the experiments in this thesis. 27, 35, 39, 41, 53

ReLU Rectified Linear Unit. An activation function with a minimum value 0 and maximum value x.. 21

train The process of passing information through a neural network and correcting the network's predictions, so that over time the network becomes more accurate at providing the correct predictions.. 6

Acronyms

CNN Convolutional Neural Network. iii, 5, 7, 8, 15, 21, 41, 52

LR learning rate. 27, 39, 41, 43

MM momentum. 27, 39, 41, 43

MTL Multi-task Learning. 4

RN Relational Network. 12, 13

RNN Recurrent Neural Network. 5

VQA Visual Question Answering. 8, 10

Appendix A: Supplemental Data

Additional explanation and analysis of the thesis experiment results are preserved here. These tables are referenced earlier in Chapter 5 or are included in this section for extra review and discussion.

Mean training steps to peak validation, % difference from 4x4 25 combination, select learning rates where accuracy > 90%

Momentum	Generation (Mean steps)				Generation (% diff)		
Learning rate	4x4 5	5x5 25	4x4 25	5x5 5	4x4 5	5x5 25	5x5 5
$M = 0.09$	7235.56	6276.35	7128.89	6816.40			
0.003515625	9057.19	8301.04	8888.89	8791.70	1.89	-6.61	-1.09
0.0140625	7560.30	6493.63	7413.33	6660.74	1.98	-12.41	-10.15
0.05625	5089.19	4034.37	5084.44	4996.74	0.09	-20.65	-1.72
$M = 0.9$	4632.10	4067.16	4628.15	3933.23			
0.003515625	6263.70	5494.52	5873.78	5390.22	6.64	-6.46	-8.23
0.0140625	4613.93	4604.44	4994.37	4419.56	-7.62	-7.81	-11.51
0.05625	3018.67	2102.52	3016.30	1989.93	0.08	-30.29	-34.03

Table A.1: Convolution windows and feature maps, mean training steps and % diff.

Table A.1 summarizes the mean steps to optimal training for each set of internal convolution layer parameters, and the % difference as compared to the 4x4 window and first convolution layer with 25 feature maps. The other combinations did achieve peak accuracy quicker with the other setups in the top performing ranges. As demonstrated in Table A.2, however, the 4x4 with 25 setup performed marginally better over more of the learning rates in the grid search. The goal with picking this combination was to ensure a more consistent training experience.

% Difference from 4x4 convolution filter and 25 features, 3 trials						
Momentum	Shape Test Accuracy			Color Test Accuracy		
Learning rate	4x4 5	5x5 25	5x5 5	4x4 5	5x5 25	5x5 5
$M = 0.09$	-3.02	-2.02	-7.06	-5.97	-4.14	-6.54
5.49316E-05	-31.55	28.65	-35.34	-45.66	23.53	-28.79
0.000219727	-54.23	55.98	-26.26	-58.58	17.16	-28.37
0.000878906	-18.41	5.15	-3.20	-9.62	5.57	-3.16
0.003515625	-1.85	0.12	-0.79	-0.79	0.04	-0.30
0.0140625	-0.62	-1.93	-0.84	-0.15	-3.80	-0.42
0.05625	-1.31	-6.60	-5.23	-2.49	-12.42	-6.51
0.225	55.75	-69.32	-27.93	58.06	-74.03	-15.00
0.9	38.59	26.44	30.14	39.62	43.18	58.70
$M = 0.9$	-6.13	-4.30	-7.29	-4.01	-4.87	-7.55
5.49316E-05	-27.96	11.07	-1.91	-17.49	6.83	-4.89
0.000219727	-3.65	0.70	-0.27	-3.73	0.25	-0.80
0.000878906	-0.73	0.28	-0.49	-0.25	-0.05	-0.45
0.003515625	-0.78	-0.17	-1.00	-0.26	-0.06	-0.51
0.0140625	-1.39	-0.53	-8.20	-1.19	-0.39	-8.23
0.05625	-10.41	-58.13	-51.34	-9.44	-64.48	-57.78
0.225	-18.32	1.53	2.27	21.03	3.68	-0.14
0.9	2.54	2.95	-1.93	4.13	-7.18	11.37

Table A.2: Convolution windows and feature maps, % difference from 4x4 and 25

Tables A.3, A.4 and A.5 present the top two performing learning rate and momentum combinations for each trials series, and the corresponding 30 or 40 batch-per-Set results that were collected. Note the performance of the Softmax-Channel results, and the high standard deviation in the quicker learning rate results.

The final appendix entry, Tables A.6 and A.7, show the % of test inputs that were extracted and placed in the extracted test data set. The mean results were used to add the test and extracted test epoch results back together and calculate the final test accuracy for the trials series. Also provided is the percentage of intersection between the training and test data for each experiment in each set.

Shape & color prediction validation & test for the Softmax-Channel series. Momentum = 0.9

Learning rate	LongFC2		ShareFC2		TwinFC2	
Experiment	Mean	StdDev	Mean	StdDev	Mean	StdDev
0.003515625	*Validation*					
Shape	97.07	1.72	97.20	1.81	96.90	1.15
Color	99.92	0.23	99.85	0.55	99.90	0.24
0.0140625						
Shape	97.91	1.52	97.36	1.72	97.52	1.03
Color	99.72	0.78	99.05	2.46	99.89	0.39
0.003515625	*Test results, shape prediction*					
E1	96.79	0.38	96.78	0.41	96.96	0.37
E2 10	95.76	0.74	96.07	0.66	96.57	0.47
E2 30	92.01	1.52	92.99	1.63	94.38	0.89
E2 50	81.29	4.13	83.25	4.63	87.48	2.12
E3 10	95.97	0.70	96.15	0.64	95.96	0.56
E3 30	92.32	1.60	92.99	1.65	90.80	1.92
E3 50	79.98	4.52	81.47	4.77	76.69	4.52
0.0140625						
E1	97.06	1.24	94.01	15.06	97.49	0.33
E2 10	94.38	13.02	92.99	15.04	97.25	0.45
E2 30	93.73	1.71	91.65	12.66	96.14	0.62
E2 50	85.96	4.44	84.39	9.63	90.58	1.82
E3 10	95.53	9.17	94.18	12.85	96.49	0.72
E3 30	92.36	13.45	92.86	9.00	91.56	1.91
E3 50	85.15	4.52	84.62	6.13	78.37	5.01
0.003515625	*Test results, color prediction*					
E1	99.64	1.35	99.77	0.30	99.85	0.14
E2 10	99.71	0.50	99.70	0.41	99.84	0.13
E2 30	99.56	0.38	99.47	0.47	99.69	0.32
E2 50	97.86	1.60	96.95	1.56	96.78	2.26
E3 10	99.71	0.46	99.63	0.59	99.76	0.24
E3 30	99.49	0.54	99.22	0.91	99.32	0.54
E3 50	98.69	1.13	98.08	1.35	98.65	0.77
0.0140625						
E1	99.43	1.63	95.97	16.15	99.79	0.25
E2 10	95.64	17.02	94.75	16.61	99.79	0.23
E2 30	99.16	1.72	94.96	16.18	99.69	0.26
E2 50	97.04	9.42	93.30	10.77	97.31	2.09
E3 10	97.43	12.23	96.18	13.80	99.59	0.75
E3 30	96.48	13.60	97.12	9.93	99.05	0.79
E3 50	98.73	1.56	95.56	6.55	97.95	1.53

Table A.3: Softmax-Channel Trial Series, validation and test for top 2 hyperparameter configurations

Shape & color prediction validation & test for the Softmax-Slice series. Momentum = 0.9
Single LR selected after E1 and E2 & E3 10% review.

Learning rate	LongFC2		ShareFC2		TwinFC2	
Experiment	Mean	StdDev	Mean	StdDev	Mean	StdDev
0.003515625	*Validation*					
Shape	97.05	1.62	97.17	1.69	97.24	1.24
Color	99.93	0.19	99.92	0.25	99.97	0.08
0.0140625						
Shape	97.71	1.46	97.61	1.63	98.06	0.97
Color	99.96	0.18	99.93	0.21	99.97	0.11
0.003515625	*Test results, shape prediction*					
E1	96.90	0.49	96.78	0.39	97.21	0.35
E2 10	95.42	0.74	95.22	0.87	96.80	0.48
E3 10	95.58	0.92	95.57	0.93	95.96	0.79
0.0140625						
E1	97.52	0.35	97.33	0.44	97.85	0.28
E2 10	96.14	0.67	95.94	0.74	97.58	0.37
E2 30	90.47	1.90	89.85	2.13	96.85	0.45
E2 50	76.32	4.56	72.04	4.72	92.56	1.47
E3 10	96.56	0.80	96.35	0.84	96.81	0.65
E3 30	91.62	2.17	90.34	2.47	91.47	2.42
E3 50	80.99	3.74	79.50	4.14	79.31	3.95
0.003515625	*Test results, color prediction*					
E1	99.87	0.09	99.84	0.09	99.90	0.09
E2 10	99.79	0.15	99.69	0.23	99.90	0.08
E3 10	99.80	0.13	99.72	0.27	99.86	0.13
0.0140625						
E1	99.87	0.17	99.85	0.21	99.93	0.05
E2 10	99.81	0.18	99.72	0.36	99.92	0.05
E2 30	99.32	0.57	99.18	0.71	99.88	0.09
E2 50	96.25	1.82	95.37	1.79	99.75	0.13
E3 10	99.78	0.34	99.73	0.27	99.87	0.09
E3 30	99.59	0.36	99.52	0.47	99.63	0.24
E3 50	97.40	1.50	96.92	1.41	97.51	1.14

Table A.4: Softmax-Slice Trial Series, validation and test for top 2 hyperparameter configurations

Shape & color prediction validation & test for the Sigmoid-Channel series. Momentum = 0.9

Learning rate	LongFC2		ShareFC2		TwinFC2	
Experiment	Mean	StdDev	Mean	StdDev	Mean	StdDev
0.0140625	*Validation*					
Shape	99.49	0.16	99.21	0.33	95.51	1.44
Color	99.99	0.04	99.96	0.16	99.91	0.17
0.05625						
Shape	99.67	0.17	99.55	0.23	97.28	1.18
Color	99.99	0.04	99.97	0.07	99.88	0.27
0.0140625	*Test results, shape prediction*					
E1	99.45	0.06	99.17	0.11	95.19	0.68
E2 10	99.36	0.11	99.00	0.16	94.02	0.84
E2 30	98.92	0.17	98.49	0.29	90.96	1.39
E2 50	98.40	0.34	97.50	0.79	87.05	2.50
E3 10	99.32	0.11	99.01	0.15	94.36	0.74
E3 30	98.93	0.15	98.28	0.29	91.01	1.20
E3 50	98.01	0.33	96.79	0.74	84.18	2.28
0.05625						
E1	99.64	0.04	99.38	0.08	96.82	0.37
E2 10	99.54	0.06	99.29	0.98	96.19	0.65
E2 30	99.26	0.14	98.94	0.22	93.36	1.17
E2 50	98.67	0.34	98.13	0.84	89.58	2.30
E3 10	99.59	0.07	99.34	0.11	96.42	0.54
E3 30	99.23	0.13	98.75	0.25	93.69	1.03
E3 50	98.33	0.35	97.47	0.76	87.00	2.42
0.0140625	*Test results, color prediction*					
E1	99.98	0.02	99.95	0.04	99.77	0.13
E2 10	99.96	0.05	99.95	0.05	99.71	0.23
E2 30	99.94	0.04	99.87	0.10	99.36	0.33
E2 50	99.64	0.17	99.30	0.33	96.80	1.37
E3 10	99.95	0.10	99.92	0.23	99.71	0.16
E3 30	99.92	0.06	99.86	0.09	99.20	0.43
E3 50	99.78	0.14	99.63	0.21	98.04	1.18
0.05625						
E1	99.98	0.02	99.95	0.04	99.75	0.46
E2 10	99.97	0.04	99.78	1.09	99.60	1.09
E2 30	99.95	0.03	99.88	0.11	99.48	0.40
E2 50	99.76	0.15	99.22	1.30	98.04	1.39
E3 10	99.97	0.02	99.93	0.10	99.71	0.33
E3 30	99.93	0.13	99.88	0.09	99.42	0.66
E3 50	99.87	0.09	99.71	0.20	98.70	1.76

Table A.5: Sigmoid-Channel Trial Series, validation and test for top 2 hyperparameter configurations

% of total test inputs in the excluded test set
% of inputs that require inference from network
For the Softmax-Channel and Sigmoid-Channel trials series.

Source		% of test inputs		% Test inference		% Excluded inference	
Set	Exp	Shape	Color	Shape	Color	Shape	Color
Set1	E2 10	9.41	9.53	11.29	10.51	0.00	9.64
Set1	E2 30	29.31	29.07	29.48	29.13	31.08	32.92
Set1	E2 50	49.69	49.35	51.69	52.62	49.03	47.77
Set1	E3 10	10.41	10.79	11.42	11.22	0.00	0.00
Set1	E3 30	29.65	29.81	30.88	31.48	27.95	25.87
Set1	E3 50	50.21	50.03	62.20	62.26	37.28	38.70
Set2	E2 10	9.66	10.32	7.35	11.24	27.95	0.00
Set2	E2 30	29.93	30.69	28.80	31.85	32.49	26.25
Set2	E2 50	49.37	50.31	54.21	53.40	47.41	47.97
Set2	E3 10	9.34	9.70	10.35	11.63	0.00	0.00
Set2	E3 30	29.93	30.03	28.63	28.75	33.49	33.89
Set2	E3 50	49.95	50.09	51.40	51.00	47.58	46.91
Set3	E2 10	10.34	10.38	11.42	11.19	10.53	0.00
Set3	E2 30	29.42	29.82	34.71	28.80	23.01	30.69
Set3	E2 50	49.83	49.61	54.90	53.03	46.95	45.78
Set3	E3 10	9.73	10.44	11.41	11.06	0.00	0.00
Set3	E3 30	29.03	30.45	33.28	33.57	26.09	25.36
Set3	E3 50	48.20	49.75	50.99	53.46	49.29	50.24
Mean	E2 10	9.80	10.08	10.02	10.98	12.83	3.21
Mean	E2 30	29.55	29.86	31.00	29.93	28.86	29.96
Mean	E2 50	49.63	49.76	53.60	53.02	47.80	47.17
Mean	E3 10	9.83	10.31	11.06	11.30	0.00	0.00
Mean	E3 30	29.54	30.10	30.93	31.27	29.18	28.37
Mean	E3 50	49.45	49.96	54.86	55.57	44.72	45.28

Table A.6: Percentage of test data that required prediction inference, Softmax- and Sigmoid-Channel

% of total test inputs in the excluded test set
% of inputs that require inference from network
For the Softmax Slice trials series.

Source		% of test inputs		% Test inference		% Excluded inference	
Set	Exp	Shape	Color	Shape	Color	Shape	Color
Set1	E2 10	0.10	0.10	9.86	10.72	9.98	6.15
Set1	E3 10	0.09	0.10	8.94	8.96	18.79	20.40
Set1	E2 30	0.31	0.30	27.81	28.11	39.07	39.04
Set1	E3 30	0.29	0.30	30.08	30.99	30.44	30.85
Set1	E2 50	0.51	0.51	45.89	48.02	54.36	55.24
Set1	E3 50	0.49	0.50	53.46	54.47	45.82	46.43
Set2	E2 10	0.10	0.10	10.29	10.20	10.93	12.35
Set2	E3 10	0.10	0.10	9.60	10.01	12.16	8.98
Set2	E2 30	0.30	0.29	30.95	31.33	25.86	27.96
Set2	E3 30	0.30	0.30	26.85	27.48	37.42	37.02
Set2	E2 50	0.50	0.49	54.43	53.23	46.23	46.19
Set2	E3 50	0.48	0.50	37.53	39.41	61.06	61.59
Set3	E2 10	0.09	0.10	9.64	9.44	20.42	9.64
Set3	E3 10	0.10	0.09	11.14	10.41	0.00	0.00
Set3	E2 30	0.29	0.29	36.59	27.65	16.30	34.38
Set3	E3 30	0.30	0.30	33.04	31.87	23.51	21.76
Set3	E2 50	0.49	0.49	47.31	47.60	53.84	51.95
Set3	E3 50	0.50	0.49	54.52	53.18	47.65	44.50
Mean	E2 10	0.10	0.10	9.93	10.12	13.78	9.38
Mean	E3 10	0.10	0.10	9.89	9.79	10.32	9.79
Mean	E2 30	0.30	0.29	31.78	29.03	27.08	33.79
Mean	E3 30	0.30	0.30	29.99	30.11	30.46	29.88
Mean	E2 50	0.50	0.50	49.21	49.62	51.48	51.13
Mean	E3 50	0.49	0.50	48.51	49.02	51.51	50.84

Table A.7: Percentage of test data that required prediction inference, Softmax-Slice