MLP Coursework 3: Project Interim Report

G45: s1449692, s1410016, s1443569

1. Motivation

The question of whether human and machine intelligence are the same is one of the two fundamental questions of the philosophy of artificial intelligence as posed by (Russell & Norvig, 2018). Intelligence Quotient (IQ) tests have been used in standardized testing for decades and, despite scientific debate, results are often considered to be a proxy for general human intelligence and are used in practice to make critical decisions. We think that if one is to try and compare artificial and human intelligence, a framework for quantifying the intelligence of humans, such as IQ tests is a reasonable place to look first.

Raven's Progressive Matrices (Raven et al., 1998) (henceforth referred to as RPM) are a type of non-verbal intelligence test that has 60 separate items. Each item consists of a question matrix and an answer array. The question matrix is a single image or a set of images with a missing part. The test-taker must choose one of the images in the answer array that makes for the most appropriate completion of the question matrix. Refer to Figure 1 for an example RPM-style item.

RPM-type questions form a large part of Intelligence Quotient (IQ) tests and are designed to estimate fluid intelligence (Jaeggi et al., 2008). Fluid intelligence refers to the ability to perform well on novel tasks independently of previous acquired knowledge and is considered to be one of the most important factors in human learning (Bilker et al., 2012). Modern approaches to supervised machine learning tasks usually rely on their ability to learn patterns from large amounts of labelled data ((Russakovsky et al., 2015), (Chelba et al., 2013)) and solve qualitatively similar problems - e.g. recognize an object in an image after seeing millions of labelled images of this and other objects (Krizhevsky et al., 2012a). Intuitively, if neural networks simply learn a distribution over a domain of input-output pairs and RPM test the capability of an agent to perform well independently of acquired knowledge, we would not expect to yield great benefits of using a neural network for this particular task. Nonetheless, we have found two publications in which deep learning techniques are applied to the problem of solving RPM.

In their work (Mekik et al.) use neural networks in combination with a symbolic agent to solve the most popular type of RPM where question matrices are in the form of a 3-by-3 grid with the image in the bottom-right missing. They first use a convolutional neural network (CNN) to estimate difference vectors between pair images. These vectors encode information about whether the images in a given pair differ in size, orientation, color and count. Based on the difference vectors between pairs of images in the question matrix and images in the answer vector, a rule-based agent then chooses the best answer for the item. This system achieves 78.7% accuracy on a subset of 108 images taken from a dataset of RPM-like items generated by a software developed by (Matzen et al., 2010). This is also the dataset we plan to use in our work and it is described in Section 3.

The second piece of work that uses a neural approach to solve RPM is by (Hoshen & Werman, 2017). They train a CNN to solve the simpler task of predicting a third image from a sequence of two images, the shapes in which vary by size, shape, color, count or orientation. They train their algorithm on a synthetic dataset of such examples and achieve a test-set accuracy of over 90%. They also report a 5% error rate on a set of IQ questions. However, this set is not described or referenced in their work.

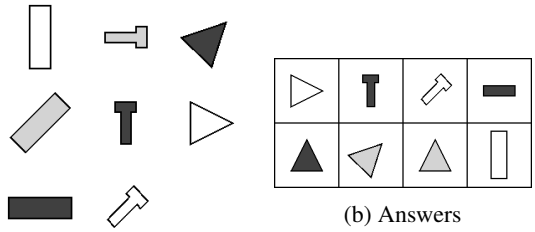
2. Research questions

The approach taken by (Mekik et al.) is not end-to-end differentiable and uses neural networks to solve a small sub-problem of the task of solving RPM. The architecture proposed by (Hoshen & Werman, 2017) uses only differentiable operations but it essentially solves a different problem and it is not immediately clear how to put it to use for solving RPM. We think that the question of how well an agent that uses only differentiable operations can perform in solving RPM is an interesting one and one that has not been previously explored in literature. We want to see if such end-to-end differentiable model can perform as well as the rule-based model used by (Mekik et al.)

Since Raven's Progressive Matrices require finding the next element in a sequence, we think that RNNs will be particularly well suited to solve the problem. We will test this hypothesis by experimenting with different architectures and in particular comparing LSTM networks with a fully-connected network. The exact details can be found in Section 5.

One challenge we expect to face is that when encoding an image, one of the first things a network learns is to distinguish edges, lines and shadows and it may not necessarily pick up spatial information and relationships between elements - both of those are important for solving RPM. How different autoencoders deal with this and what information is lost is something we will explore in our project.

Another challenge with our dataset is its size - we have a relatively small number of images (around 12000), which



(a) Question

(b) Answers

Figure 1. A sample question - answer pair. The correct answer label for this example is (6) - the gray triangle standing upward.

may make it hard to train very complicated networks. This is why we chose to use autoencoders to reduce the task complexity. But we still need to train the autoencoders and this poses an interesting question - whether we can combine other datasets such as MNIST and omniglot with ours to obtain better encodings. If we succeed, we would improve our prediction accuracy, but more interestingly it would mean that we have achieved a rudimentary form of knowledge transfer - a skill which is fundamental in human problem solving and which plays a big part in how humans deal with RPM (Jaeggi et al., 2008).

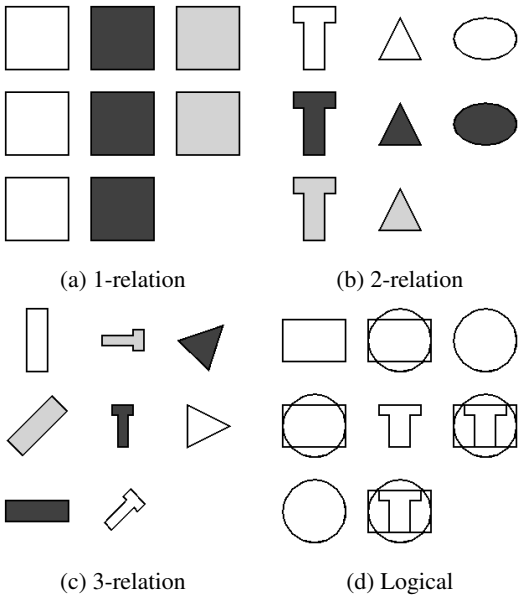
3. Data set and task

The original set of matrices developed by Prof. John Raven is not freely accessible so we do not plan to train or report performance on it. There have been several efforts to create RPM-like problems (Arendasy & Sommer, 2005; Christoff et al.; Matzen et al., 2010). We have chosen the dataset created by (Matzen et al., 2010) for two reasons. Firstly, it is used by (Mekik et al.), which gives us a point for comparison for the performance of our system. Secondly, the output of (Matzen et al., 2010)'s work is a free software, which we can use to generate more training data, if required.

The software in (Matzen et al., 2010) was developed in the Sandia National Laboratories and for that reason the matrices in the dataset produced by that software is referred to as Sandia matrices in (Mekik et al.). We adopt the same naming convention.

The dataset consist of 840 triples of question image, answer image and correct label for the answer. The question images are of size 308 by 308 pixels and represent a 3 by 3 grid of 100 by 100 pixel shapes with the bottom-right one missing. The answer images are of size 410 by 206 pixels and represent 8 possible 100 by 100 pixel shapes for the answer arranged in a 4 by 2 grid. The answer label is an integer between 0 and 7, and gives the index of the correct answer image, left to right and top to bottom, starting from the top left corner. An example question answer pair can be seen in Figure 1.

The questions in the dataset are split in two categories - object relation problems and logical problems. In object relation problems, the entries along the rows, columns or diagonals of the question matrix are related by size, shape, color, orientation. In logical problems, there are relations such as AND, OR and XOR.



(a) 1-relation

(b) 2-relation

(c) 3-relation

(d) Logical

Figure 2. Examples from each question type in our dataset

The object relation problems are subdivided further based on how many relations govern the change of entries along rows, columns and diagonals. Examples of 1, 2 and 3-relation problems as well as a logic problem can be seen in Figure 2. The Sandia matrices dataset we use consists of 1, 2 and 3-relation problems as well as logical problems. We use 90% of the data for training and hold the rest out for testing. The number of examples for each problem type in the data set is given in Table 1

As a pre-processing step, we downsample each 100×100 image to 28×28 pixels. We do this because as part of our future plans, we would like to perform joint training on our dataset and the MNIST and omniglot datasets, which consist of images of size 28×28 pixels.

Table 1. Number of examples for each question count

Type	1-rel	2-rel	3-rel	Logical	**Total**
Total	96	184	500	60	**840**
Train	86	166	450	54	**756**
Test	10	18	50	6	**84**

4. Objectives

Our first objective is creating an end-to-end differentiable network to solve the whole RPM problem instead of applying machine learning to only a part of it.

The second objective, which we already achieved is to build a model that can perform better than a random guess, as a way of proving neural networks are capable of picking up some rudimentary signal. After that is achieved our next goal is to improve on the performance of the planning

agent described in (Mekik et al.), which reported an overall accuracy of 78.7%.

If our models prove to be insufficient for the chosen task, as a back-up plan, we will try to solve the problem of type classification - each problem in the Sandia dataset can be put into one of four types as described in Section 3 and we would build an agent to achieve that goal.

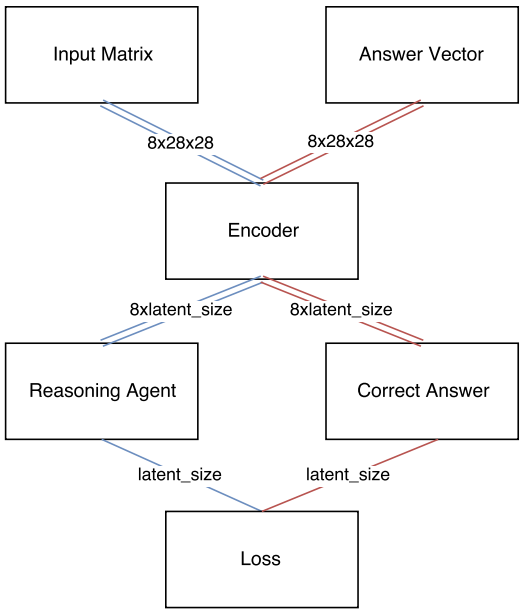
We plan on exploring the benefits of different autoencoders, reasoning agents and distance metrics. In Section 5, we explain the details of each possible choice.

5. Methodology

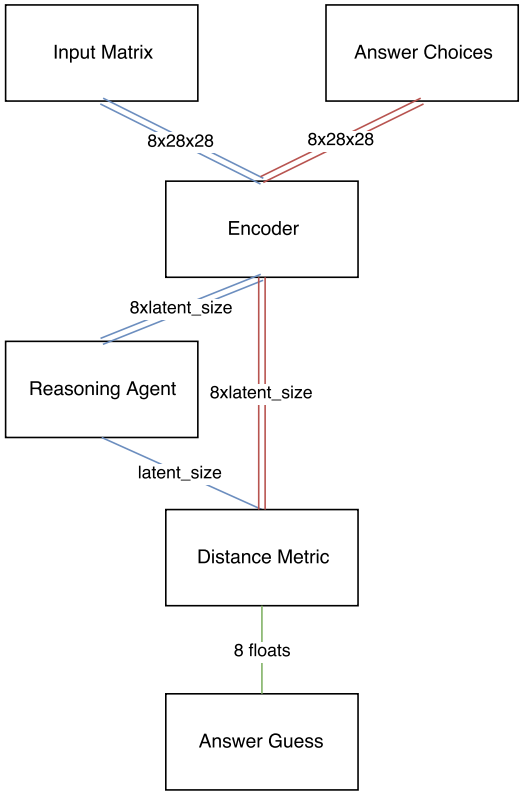
Our general architecture consists of an encoder, reasoning agent and a distance metric. The encoder is used for several reasons - firstly, our dataset size is small, so constricting the information flow in the architecture is a requirement for learning sensible features. Secondly, we aim to construct a general architecture, as independent of image size or type as possible so an encoding component provides an abstraction. The reasoning agent is the "computational" part of the architecture - it takes the encodings of the question images and produces a prediction for the encoding of the missing image. Finally, the distance metric is applied to determine which of the possible answers is closest to our prediction. An overview of the entire architecture is provided in Figure 3.

Principal Component Analysis (PCA) is a widely used method of dimensionality reduction and thus can serve as an encoder. We describe it briefly and defer a more in-depth explanations to Chapter 2 of (Goodfellow et al., 2016). During training, PCA takes a matrix of size N×D and finds its eigenvectors and eigenvalues. It then sorts the eigenvectors such that the corresponding eigenvalues are in decreasing order. For a given encoding size K < D, it takes the first K eigenvectors, thus forming a D×K matrix we will call V. Encoding of a batch of examples then is right multiplying the batch matrix (of size N×D) by V, producing a N×K matrix. Decoding is done by right multiplying the encoded matrix (of size N×K) by the transpose of V. PCA provides an optimal reconstruction procedure when the measure is mean squared error (MSE).

Autoencoders are another widely used method of dimensionality reduction. They consist of three or more layers - input, hidden and output. The objective is for the output to be as close to the input as possible, determined by some loss function. The hidden layers are restricted in some way, often by enforcing sparsity or by having a size much smaller than the input size. Linear autoencoders are the simplest of these, with the hidden representation being a result of multiplying the input by a matrix and the output is the result of multiplying the hidden representation by a matrix. It is worth noting that when mean squared error is used as a loss function, autoencoders' solutions converge to those of PCA, if we ignore the order of the hidden units. Convolutional autoencoders take advantage of the benefits of a convolu-



(a) Train architecture



(b) Test architecture

Figure 3. An outline of the architecture during a) training and b) testing.

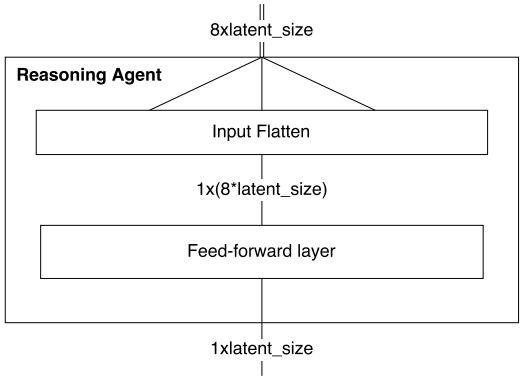
tional layer (Krizhevsky et al., 2012b) when dealing with images and also introduce a "convolutional transpose" as a match to the convolutions for the decoder. Convolutional transposes (also known as deconvolutions) (Zeiler et al., 2011) are a way of up-sampling an input. The best way to understand them is to think of normal convolutions, but with a fractional stride - we introduce empty cells between the cells of the input map and apply the convolution on this "sparse" input image. For building a more in-depth intuiton, please refer to the animations at this GitHub project.

Once we have encoded our inputs, we provide them to the reasoning agent. It is responsible for producing an encoding for the prediction of what the final answer should look like. We have three different architectures we plan on exploring for this component. Firstly, we built a feed-forward agent - when given 8 vectors of size *latent_size*, it concatenates them to form a single vector of size $8 \times latent\_size$. It then feeds this vector through a series of fully-connected layers and produces a single vector of size *latent_size*, which is our prediction. We use the encoding of the correct answer as the ground truth for this model and use the mean squared error between the prediction and this encoding as a loss function when training the model.

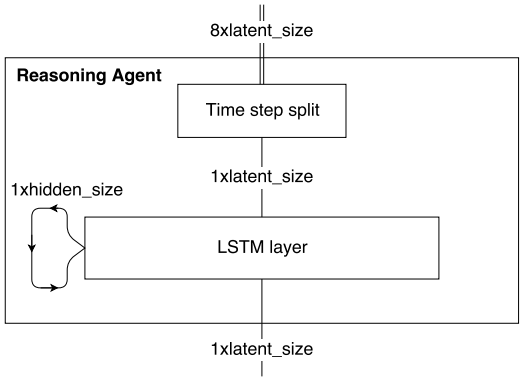
An alternative to the feed-forward agent is an agent that makes use of RNNs and in particular an LSTM. The LSTM takes in each of the vectors at a timestep and outputs the prediction latent vector at the last timestep. Additionally, we plan on introducing skip connections to bake-in the structure of the problem as shown in 4c.

Finally, we compute the distance between the output of the reasoning agent and the encodings of the answer images. We output the index of the image, the encoding of which was the closest to the output of the reasoning agent. For our baseline, we use the Euclidian distance between the prediction and the encoded correct answer. This approach is not differentiable, so in our future work we plan to explore methods that feed the prediction and the 8 answer choices through an architecture, through which we can propagate gradients.

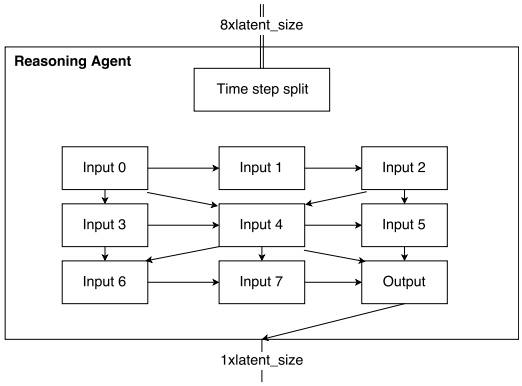
The architecture explored in this report consists of PCA as encoder, a feed-forward reasoning agent and an MSE loss as both distance metric and minimization objective. Although neither the encoding module (PCA) nor the method for choosing the best answer are differentiable, they can both be substituted with differentiable modules. The reason we chose this architecture is that it is simple to implement and gives us a solid baseline to which we can compare future work.



(a) Feedforward architecture



(b) LSTM architecture



(c) SkipLSTM architecture

Figure 4. A sketch of reasoning agent architectures.

6. Baseline experiments

We trained the architecture described in Section 5 on our training set and evaluated on the held-out test set. Table 2 shows the accuracy we obtained on the test set. This is a significant improvement over the performance of an agent making uniformly random guesses for the correct answer

Type	1-rel	2-rel	3-rel	Logic	**Overall**
Baseline	40%	56%	54%	50%	**52%**
UR agent	12.5%	12.5%	12.5%	12.5%	**12.5%**

Table 2. Comparison of the performance of our baseline system to that of an agent making random guesses for the four different types of questions in our test set

and can serve as a reliable baseline for our future work.

After obtaining a prediction for the latent vector of the correct answer, we can decode that using the second part of our autoencoder, to generate a prediction for an output image. Examples of this are tabulated in Table 3. It is interesting to note that our baseline agent seems to have grasped the concept of counting objects.

7. Interim conclusions

Our baseline experiments point that even a simple architecture produces reasonable results (>50% accuracy). We were able to encode the data, predict an encoding for the answer, then decode that prediction and get a reconstruction that is similar to the right answer. Our algorithm worked very well for some of the subtasks, such as counting objects or correctly predicting the form of a large object. However, it does not seem to grasp all types of problems, as exemplified by the third row of Table 3. Given these results we are confident we can obtain a lower classification error once we make use of a more complex and well suited for the task LSTM network. At this stage our results indicate that we are on the right track and our initial objectives seem feasible.

When we used PCA with bigger number of features, we were able to obtain clearer predictions for our images. This made us look at our initial belief that reducing the image size and dimensionality with autoencoding will help with predictions. Still because the size of our database is not big (840 sets of images) we think that training on the full size of the image will be a daunting task for a LSTM predictor and using an autoencoder that is able to pick up the most important features will help our task. Testing how much and if autoencoders help with prediction accuracy may be something worth exploring in the second part of our project, even though we did not consider it in the initial planning project.

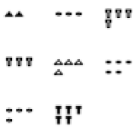
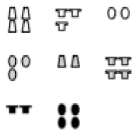
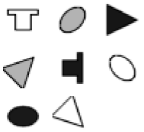
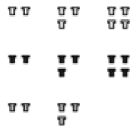
Question	Ground Truth	Reconstruction
		
		
		
		
		

Table 3. Results from our initial experiment with reconstructing question answers. The first column gives the question matrix, the second one is the correct answer and in the third one are the results of decoding the latent vector prediction of our feed-forward logical unit.

8. Plan for the rest of the project

Our initial work gives us a good baseline on which we can build up for the second part of the project. Since the results we obtained proved close to our expectations there are no major changes to our objectives.

We will proceed to create a fully differentiable network by:

- Changing the PCA encoding with a differentiable module - a linear autoencoder

- Changing our distance metric. We paln use one-hot encoding on the answer and then try to frame the problem as a classification challenge or we may explore other approaches if that proves insufficient

After that is complete we will implement:

- Different types of autoencoders - linear and convolutional and how they perform compared to PCA encoding and no encoding
- Different types of distance metrics - our initial ideas include least squares and NN distance
- An LSTM agent - and how it compares to the previously used fully connected agent
- A skip LSTM agent - an how it compares to the previous two

After we have experimented with different architectures, we think the fact that we use a dataset with relatively small size will give us time to perform extensive hyperparameter tuning.

Another interesting avenue we could take if we exhaust the possibilities listed above is to look towards transfer learning and domain adaptation. We plan to investigate if training our autoencoder to extract features from other datasets and transferring what it has learned to our task is feasible.

Our results so far indicate the aforementioned approaches will be able to handle the classification task, but we have a back up plan — in the form of a simpler task if this turns out to be too hard. If we are unable to correctly predict the next image in a RPM sequence, we will attempt to at least distinguish which of the four problem types described in Section 3 a RPM sequence has. We will then train our agent to solve this simpler problem.

References

Arendasy, Martin and Sommer, Markus. The effect of different types of perceptual manipulations on the dimensionality of automatically generated figural matrices. *Intelligence*, 33(3):307–324, may 2005. ISSN 01602896. doi: 10.1016/j.intell.2005.02.002. URL http://linkinghub.elsevier.com/retrieve/pii/S0160289605000206.

Bilker, Warren B, Hansen, John A, Brensinger, Colleen M, Richard, Jan, Gur, Raquel E, and Gur, Ruben C. Development of abbreviated nine-item forms of the Raven's standard progressive matrices test. *Assessment*, 19(3):354–69, sep 2012. ISSN 1552-3489. doi: 10.1177/1073191112446655. URL http://www.ncbi.nlm.nih.gov/pubmed/22605785http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC4410094.

Chelba, Ciprian, Mikolov, Tomas, Schuster, Mike, Ge, Qi, Brants, Thorsten, Koehn, Phillipp, and Robinson, Tony. One billion word benchmark for measuring progress in statistical language modeling. Technical report, Google, 2013. URL http://arxiv.org/abs/1312.3005.

Christoff, Kalina, Prabhakaran, Vivek, Dorfman, Jennifer, Zhao, Zuo, Kroger, James K, Holyoak, Keith J, and Gabrieli, John D E. Rostrolateral Prefrontal Cortex Involvement in Relational Integration during Reasoning. doi: 10.1006/nimg.2001.0922. URL http://reasoninglab.psych.ucla.edu/KHpdfs/Christoff.2001.pdf.

Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Hoshen, Dokhyam and Werman, Michael. IQ of Neural Networks. sep 2017. URL https://arxiv.org/abs/1710.01692.

Jaeggi, Susanne M, Buschkuehl, Martin, Jonides, John, and Perrig, Walter J. Improving fluid intelligence with training on working memory. *Proceedings of the National Academy of Sciences of the United States of America*, 105(19):6829–33, may 2008. ISSN 1091-6490. doi: 10.1073/pnas.0801268105. URL http://www.ncbi.nlm.nih.gov/pubmed/18443283http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC2383929.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012a. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012b. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Matzen, Laura E., Benz, Zachary O., Dixon, Kevin R., Posey, Jamie, Kroger, James K., and Speed, Ann E. Recreating raven's: Software for systematically generating large numbers of raven-like matrix problems with normed properties. *Behavior Research Methods*, 2010. ISSN 1554351X. doi: 10.3758/BRM.42.2.525.

Mekik, Can Serif, Sun, Ron, and Dai, David Yun. Advances in Cognitive Systems X (20XX) 1-6 Deep Learning of Raven's Matrices. URL http://www.cogsys.org/papers/ACS2017/ACS{_}2017{_}paper{_}23{_}Mekik.pdf.

Raven, John, Raven, J. C. (John C.), and Court, John H. (John Hugh). *Manual for Raven's progressive matrices and vocabulary scales.. Section 3, Standard progressive matrices : introducing the parallel and plus*

versions of the tests (the latter ...). Oxford, England : Oxford Psychologists Press, 1998 ed edition, 1998. ISBN 1856390276 (manual). Standard progressive matrices published 1989 by the Australian Council for Education Research.

Russakovsky, Olga, Deng, Jia, Su, Hao, Krause, Jonathan, Satheesh, Sanjeev, Ma, Sean, Huang, Zhiheng, Karpathy, Andrej, Khosla, Aditya, Bernstein, Michael, Berg, Alexander C., and Fei-Fei, Li. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Russell, Stuart and Norvig, Peter. Artificial Intelligence: A Modern Approach. http://aima.cs.berkeley.edu/, 2018. [Online; accessed 19-Feb-2018].

Zeiler, Matthew D., Taylor, Graham W., and Fergus, Rob. Adaptive deconvolutional networks for mid and high level feature learning, 2011.