Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Testing neural network intelligence using Raven's Progressive Matrices

Bachelor Thesis

2021
Nina Raganová

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Testing neural network intelligence using Raven's Progressive Matrices

Bachelor Thesis

Study Programme:	Computer Science
Field of Study:	Computer Science
Department:	Department of Applied Informatics
Supervisor:	prof. Ing. Igor Farkaš, Dr.

Bratislava, 2021
Nina Raganová

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Nina Raganová
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Testing neural network intelligence using Raven's Progressive Matrices
Testovanie inteligencie neurónovej siete použitím Ravensových progresívnych matíc

Anotácia: Ravensove progresívne matice (RPM) sa často používajú na testovanie IQ u ľudí, a to ako miera abstraktného uvažovania prostredníctvom neverbálnych otázok. V umelej inteligencii je jedným z najdôležitejších cieľov výroba strojov s rozumovou schopnosťou podobnou ľudskej, aby boli užitočné pre rôzne úlohy. Najsľubnejší prístup je založený na umelých neurónových sieťach.

Cieľ: 1. Implementujte vybranú neurónovú sieť a trénujte ju na riešenie problému RPM.
2. Otestujte úspešnosť modelu v rôznych scenároch týkajúcich sa variability a veľkosti trénovacej množiny.
3. Interpretujte získané výsledky.

Literatúra: Zhuo T., Kankanhalli M. (2020). Solving Raven's Progressive Matrices with Neural Networks. https://arxiv.org/pdf/2002.01646
Jahrens M., Martinetz T. (2020). Solving Raven's Progressive Matrices with Multi-Layer Relation Networks. https://arxiv.org/pdf/2003.11608
Zhang C. et al. (2019). Raven: A dataset for relational and analogical visual reasoning. In CVPR, https://arxiv.org/pdf/1903.02741

Vedúci: prof. Ing. Igor Farkaš, Dr.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 28.10.2020

Dátum schválenia: 31.10.2020

doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Nina Raganová
Study programme:	Computer Science (Single degree study, bachelor I. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Bachelor´s thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Testing neural network intelligence using Raven's Progressive Matrices

Annotation: Raven's Progressive Matrices (RPM) have been widely used for IQ testing in humans, namely as a measure of abstract reasoning through non-verbal questions. In AI, one of the most important goals is to make machines with reasoning ability similar to that of humans in order to make them useful for various tasks. The most promising approach is based on artificial neural networks.

Aim:
1. Implement a chosen neural network and train it for solving the RPM problem.
2. Test the model performance in various scenarios concerning the variability and the size of the training set.
3. Interpret obtained results.

Literature:
Zhuo T., Kankanhalli M. (2020). Solving Raven's Progressive Matrices with Neural Networks. https://arxiv.org/pdf/2002.01646
Jahrens M., Martinetz T. (2020). Solving Raven's Progressive Matrices with Multi-Layer Relation Networks. https://arxiv.org/pdf/2003.11608
Zhang C. et al. (2019). Raven: A dataset for relational and analogical visual reasoning. In CVPR, https://arxiv.org/pdf/1903.02741

Supervisor:	prof. Ing. Igor Farkaš, Dr.
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	prof. Ing. Igor Farkaš, Dr.
Assigned:	28.10.2020
Approved:	31.10.2020

doc. RNDr. Daniel Olejár, PhD.
Guarantor of Study Programme

Student

Supervisor

Acknowledgments:

I would like to thank my supervisor prof. Ing. Igor Farkaš, Dr. for his professional advice, ideas, patience and many valuable observations during my work on the thesis.

Abstrakt

Práca sa zaoberá schopnosťou neurónových sietí riešiť Ravensove progresívne matice, test neverbálnej inteligencie široko používaný na odhad ľudského intelektu. Za skúmaný model sme si zvolili ResNet-18, nadväzujúc na doterajšie dostupné výsledky uvedené v iných publikáciách. V rámci výskumu robíme experimenty s učiteľom aj bez učiteľa na datasete RAVEN a na podporu našich zistení sme použili vizualizáciu skrytých reprezentácií na predposlednej vrstve skúmaneho modelu. Analyzujeme správanie siete na rôznych typoch problému a ich kombináciách, taktiež v prípade, že niektoré typy z trénovacej množiny úplne vylúčime. Sieť v niektorých experimentoch vykazuje vysokú presnosť odpovedí, čo naznačuje, že môže byť schopná abstraktného myslenia.

Kľúčové slová: abstraktné usudzovanie, dataset RAVEN, ResNet

Abstract

The thesis deals with the neural network ability to solve Raven's Progressive Matrices, a nonverbal intelligence test widely used for estimation of human intellect. We chose ResNet-18 as the studied model, following up on the currently available results listed in other publications. Within the research, we perform both supervised and unsupervised experiments on the RAVEN dataset and support our findings with the visualization of the hidden representations on the penultimate layer of the studied model. We analyze the behavior of the network on various problem types and their combinations, also in case we eliminate certain types from the training set completely. The network demonstrates high accuracy of answers in certain experiments, which suggests that it might be capable of abstract reasoning.

Keywords: abstract reasoning, RAVEN dataset, ResNet

Contents

List of Figures

List of Tables

Introduction

The word "intelligence" is a frequently mentioned term concerning human mental abilities. Psychologists measure human intellect with various types of IQ tests, focusing on many aspects of reasoning. The purpose of this thesis is to attempt to measure the intelligence of artificial intelligence, specifically artificial neural networks. We study abstract reasoning, more precisely the ability to solve Raven's Progressive Matrices. With the rise of artificial intelligence, more and more people are concentrating on the understanding of the behavior of neural networks, bringing attention to this field. We consider the present link to human reasoning very fascinating, worthy of additional research.

However, the thesis should be understood from the point of view of computer science and follows up on closely related work, mostly the article by Zhuo et. al. [Zhuo and Kankanhalli, 2020]. Rather than invent a method that would produce better results, we focus on the already devised methods and further analyze the behavior of the network in various scenarios concerning the variability and the size of the training set. We believe that the results of our experiments could help to better understand the reasoning ability of neural networks. We also hope to inspire other researchers to study this promising field, because it is expected of intelligent systems to develop new skills and abstract reasoning is closely related to our ability to reason with information to solve unfamiliar problems.

Chapter 1 deals with the basic terms, explains the motivation and provides the reader with a brief description of the concept of neural networks. In addition, it summarizes the current progress related to the topic and compares the results of other authors. In chapter 2 are stated the aims of the thesis and outlined both supervised and unsupervised training methods. The chapter also contains a formal problem definition and an illustration of the implementation of the used model. Chapter 3 includes the descriptions of the performed experiments and points out the interesting results, supported by the visualization of hidden representations. The conclusion of the thesis contains the summary of the work that has been done and the propositions of possible future advancements in the problematics.

Chapter 1

Testing Intelligence

The concept of intelligence has been studied since the end of the 19-th century. Although the first mentions of intelligence and the aspirations to define it date back to the Ancient Greece, the deeper analysis of human intelligence has been on the rise since the study of Sir Francis Galton. He proposed the idea that intelligence was quantifiable, normally distributed and heritable [Galton, 1891]. However, the meaning of the term intelligence and its concept have evolved through the years and the complete definition of intelligence still remains unclear.

1.1 Intelligence Quotient

To be able to test intelligence, it is crucial to understand what we mean by the term. One of the earlier attempts to characterize intelligence comes from the French psychologist Alfred Binet who conceptualized intelligence as judgment, otherwise called good sense, practical sense, initiative, the faculty of adapting one's self to circumstances [Binet et al., 1916]. However, while Binet focused more on language abilities, Galton concentrated on testing human nonverbal skills.

Even though the opinions on defining intelligence differ, psychologists had to find common ground in order to be able to test and compare human intelligence. Evidence is increasing that traditional intelligence tests measure specific forms of cognitive ability that are predictive of school functioning, but do not measure the many forms of intelligence that are beyond these more specific skills, such as music, art, and interpersonal and intrapersonal abilities [Braaten and Norman, 2006]. Nevertheless, IQ tests have become a very popular method to measure human intellect. Intelligence quotient (IQ) was introduced as the scale for assessing intelligence on standardized tests. In the past, IQ was determined as a mental age score to chronological age ratio, where the mental age was obtained as the result of a standardized intelligence test.

1.2 Standardized Tests

Once we express the the possibility of assigning scores to every person based on their intelligence, we need a standardized technique to acquire universally comparable results. In 1890 American psychologist James McKeen Cattell published measures of several psychological functions, including, but not limited to, measures of reaction time for auditory stimuli, absolute pain threshold, and human memory span, in a paper entitled *Mental Tests and Measurements*. Thus, he was the first to use the term mental test. This research conducted by Cattell was affected by the work of Galton. Galton wrote a commentary to Cattell's 1890 paper endorsing the research and indicating that it would be useful to relate scores from the psychological measures to ratings of intellectual eminence [Sternberg, 2000].

However, Binet held a different opinion. He believed that intelligence cannot be measured by concentrating on elementary cognitive processes, but on complex mental processes instead. Consequently, he collaborated with his colleague Théodore Simon on the development of the first intelligence test, which became the precursor of IQ tests that are still in use today. The test, today known as the Binet-Simon Scale, did not focus on knowledge related to education, but on areas such as attention, memory, or problem-solving skills. The main purpose of the test was to determine the likelihood of school success. Binet soon realized that some children outperformed their peers and were able to answer more advanced questions that older participants were generally able to answer. Considering this remark, Binet came up with the notion of mental age. Moreover, he suggested to measure intelligence based on the average skills of a certain age group.

Nonetheless, Binet emphasized that the test had its limitations and that it did not entirely capture the broad spectrum of intellectual abilities. Since then, a lot of various intelligence tests have been proposed, focusing on diverse mental capabilities. Considering the fact that intelligence has been defined in many ways, it is also understandable that there are various tests trying to reflect the abilities corresponding to intelligence. Whereas verbal tests aim attention at the abilities associated with comprehension and reasoning using concepts related to or conveyed in words, measures of nonverbal intelligence contain questions concentrating on problem-solving and the analysis of information using visual reasoning, not demanding the understanding or the use of words. The latter ones include tasks linked with concrete and abstract ideas, performing visual analogies and recognizing relationships between visual concepts, and the abilities to recognize and remember visual sequences.

The current use of intelligence testing is quite extensive. For instance, it found application in predicting the potential of young children to detect giftedness at young age, so they can develop their talent. Nowadays, there are also frequently used tests,

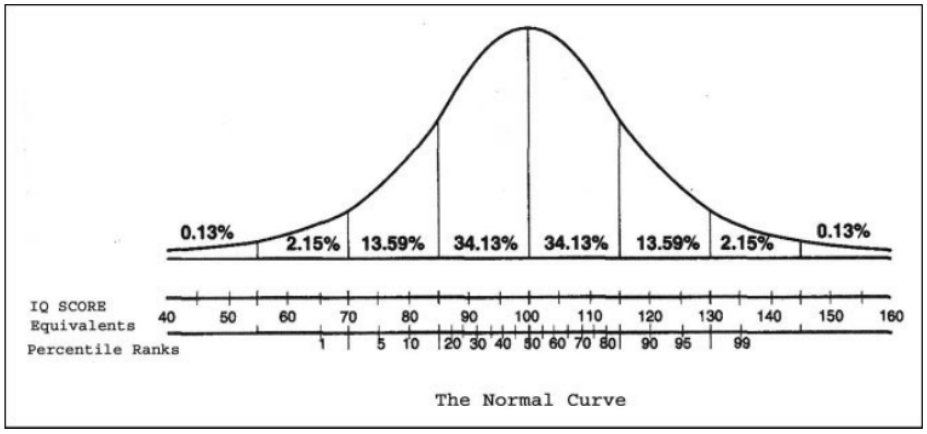

Figure 1.1: The distribution of IQ scores with the mean of 100 and the standard deviation of 15 [Braaten and Norman, 2006].

which are referred to as knowledge-based. These tests incorporate questions regarding acquired knowledge and experience. Many forms of these tests are widely used for college admissions.

For the purpose of this work and our testing, we focus on nonverbal tests and interpret intelligence as the ability to solve abstract reasoning tasks, particularly Raven's Progressive Matrices [Raven and Court, 1938].

1.2.1 Normal Distribution of IQ

For a given test, the distribution of scores depends upon the distribution of cognitive skills in the population. Many cognitive tests are constructed so that they yield a normal distribution of test scores in their intended population [Hunt, 2010]. Since IQ is a score to embrace the evaluation of every human being, the scores of the whole population are normally distributed on its scale.

The normal (Gaussian) distribution is a distribution of a continuous random variable X. The formula of its probability density function is given for all $x \in \mathbb{R}$ as

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{(x-\mu)^2}{2\sigma^2}) \tag{1.1}$$

where $\mu \in \mathbb{R}$ and $\sigma > 0$. The expected value of the random variable is $E(X) = \mu$ and the variance is $Var(X) = \sigma^2$, which means, in the case of IQ, that the scores are normally distributed with the mean of 100 and the standard deviation of 15.

As a result, as it is apparent from figure 1.1 and known from the properties of normal distribution, the scores of approximately 2/3 of population range between 85 and 115. The very high and very low scores are exceptionally rare, covering the cases of profound giftedness and, on the other side of the scale, profound mental disability.

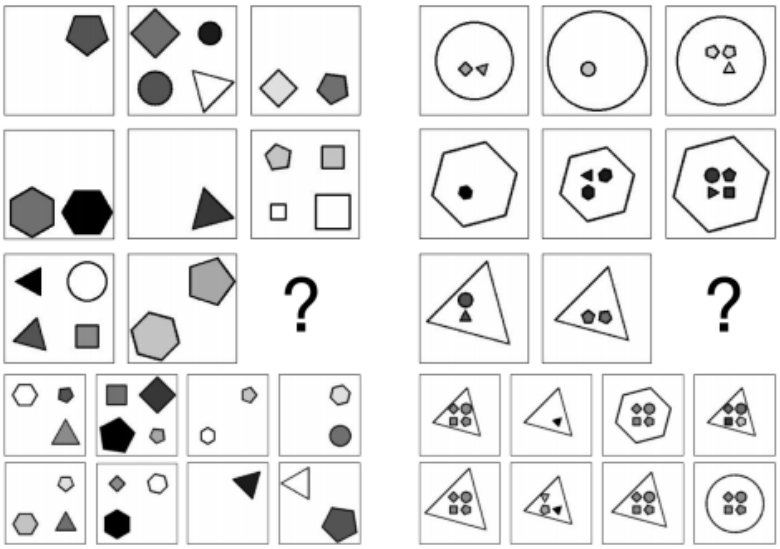

Figure 1.2: Two examples of RPM problems [Zhang et al., 2019].

1.2.2 Raven's Progressive Matrices

Raven's Progressive Matrices (RPM) are a type of test of nonverbal intelligence and abstract reasoning with multiple choice questions. It was developed in 1936 by English psychologist John Carlyle Raven. The test comprises several questions, usually placed in order by escalating difficulty. The test is designed to assess the reasoning ability of the participant.

Each problem is presented as a matrix with a missing piece. The task is to identify the relationships among the elements in each row and, based on the observations, choose the most suitable fulfillment among given options. An example of a test item is shown in figure 1.2. The formal definition of the problem and more detailed description can be found in chapter 2.

1.3 Artificial Intelligence

In the last decades, we can notice the observable increase of the mentions of Artificial Intelligence. But how can we interpret the term? According to American applied mathematician Richard Ernest Bellman, Artificial Intelligence (AI) can be understood as the automation of activities that we associate with human thinking, activities such as decision-making, problem solving and learning [Russell and Norvig, 2009]. Hence, AI represents intelligence demonstrated by machines, presenting virtual counterpart to human cognitive skills.

With the rise of computers and AI, the notion of describing the intelligence of AI started to be intriguing. One of the first attempts to provide the definition of the intelligence of machines was the Turing Test, proposed by British mathematician Alan

Mathison Turing in 1950 in his paper *Computing Machinery and Intelligence*. The objective of the test is to exhibit intelligent behavior, indistinguishable from that of a human. The principle of the test resides in the communication between a program and a human interrogator via online typed messages. The interrogator's task is to determine, whether the responses come from a computer generating human-like responses or a real person. The program successfully passes the test if the interrogator guesses incorrectly 30% of the time. Nowadays, computer programs still have problems with fooling sophisticated judges. However, many people have been misled when they had no idea they might be in a conversation with a computer.

In recent years, a trend to create virtual personal assistants has emerged. These humanlike companions are able to perform a wide variety of tasks. Evidently, the development of Artificial Intelligence is on the rise, with wide possible applications, ranging from the prediction of the customers' behavior to humanoid robots.

Many tasks that AI could carry out do not require intelligence (as the ability to solve abstract reasoning problems), similarly to human behavior. Despite this fact, we consider it an interesting subject for research to study this ability of computers. First of all, we need to select the right model for testing as a representative to simulate human cognition.

1.3.1 Artificial Neural Networks

One of the possible options, that could resemble humanlike reasoning, is to use Artificial Neural Networks (ANNs). ANN is an abstracted model of the brain's activity, simulating brain cells called neurons. A neural network is just a collection of units connected together; the properties of the network are determined by its topology and the properties of the "neurons" [Russell and Norvig, 2009].

The network consists of units, i.e. nodes, which are connected by directed links and organized into layers. The links between nodes propagate activations a, simulating the performance of the biological brain cells. Every link between nodes i and j has assigned a numeric weight w_{ij}, representing the strength of the connection, having either excitatory or inhibitory effect. Every neuron in the network produces output based on its input values and its activation function. To obtain the output of the unit, we need to compute a weighted sum of unit's inputs as

$$net_j = \sum_{i=0}^{n} w_{ij} a_i \tag{1.2}$$

where a_i denotes the activation from unit i connected to j and n is the number of connected neurons from the previous layer or, in the case of input layer, the input dimension. Then we apply the activation function as follows

$$a_j = g(input_j) = g(\sum_{i=0}^{n} w_{ij} a_i) \tag{1.3}$$

where g is the neuron activation function. Some of the typically used functions are hyperbolic tangent, logistic sigmoid or softmax.

Once we have defined the units, we need to put them together and form a network. Then we propagate the signal from the input through the layers to the output. NNs can be built in various ways, basically forming either a feed-forward or a recurrent network. The former type has connections between nodes in only one direction, so each node collects input from the units of the preceding layer. The latter also takes in the outputs, making them additional inputs, hence creating a dynamic system. The response of the network depends also on the initial state, possibly depending on previous inputs of the network. Hence, recurrent networks support short-term memory [Russell and Norvig, 2009], which can be advantageous in solving certain tasks.

Neural networks can be trained in several ways. Supervised learning is a method, when we train the network on input-target pairs $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$. Every y_i was generated by a function $f(x) = y$, which is unknown to the network. This way, we provide the network with a feedback, giving it the correct answers for the example inputs. The task is to find a function that sufficiently approximates f. Another approach is referred to as unsupervised learning, where the network learns patterns in the input without any explicit feedback (e.g. clustering).

Training the network involves a few stages. Firstly, we need to train the network on the training set, which is usually the largest part of the dataset. It consists of the example inputs, which are presented to the network so it can learn the features of the problem. The next step is to evaluate the model on the validation set to tune its parameters. The validation set serves as an instrument to estimate the performance of the network. Finally, once we have trained the best model on the training set, we test it on the test set to measure the accuracy of its performance. The test set is a set of questions of the same problem type, but yet not seen by the network. This way we test the generalization ability of the network, checking whether it is capable of dealing with new instances of the problem and not just the ones it has already seen (which is achieved by their memorization).

1.4 Related Work

The first attempts of solving RPM led to the design of many AI agents and algorithms, which were based on the observation of the relationships shared between the cells in the rows of the matrix [Joyner et al., 2015]. These approaches make use of additional knowledge about the problem structure.

However, the idea of solving Raven's Progressive Matrices with neural networks has emerged in recent years. Neural networks do not need any additional information, performing automatic feature extraction without human intervention.

1.4.1 RAVEN Dataset

Solving RPM is not a simple task, not even for a human individual. Complex problems demand the use of deeper networks, which can extract the higher-level features from the input. However, deep models require large datasets, otherwise they are prone to overfitting the training data. This phenomenon is very undesirable, because it mitigates the generalization ability. Therefore, to be able to train the network properly, we need a sufficiently extensive dataset.

We decided to use *RAVEN: A Dataset for Relational and Analogical Visual rEasoNing*, which was built in the context of Raven's Progressive Matrices and aimed at lifting machine intelligence by associating vision with structural, relational, and analogical reasoning in a hierarchical representation [Zhang et al., 2019]. The dataset is also provided with human performance as a reference. The dataset consists of 7 subtypes, which are depicted in figure 1.3. Each subtype contains 10 000 distinct problems comprising 16 gray-scale images, as can be seen in figure 1.2.

The RPM matrices are generated based on a set of applicable rules. After the sampling of the rules for the matrix, they are used to create valid rows. The problem matrix is then obtained as the concatenation of three distinct rows sharing the same rules. The answer set is generated by modifying the attributes of the correct choice in a way that breaks the rules of the matrix in the last row.

The rules are selected for every attribute individually and are chosen from 4 types: *Constant*, *Progression*, *Arithmetic* and *Distribute Three*. As can be derived from the name of *Constant*, this rule ensures that all the three images in the row share the same value of a certain attribute. *Progression* represents a quantitative increase or decrease between adjacent elements in the row and *Arithmetic* constrains the rows in a way that the attribute of the third item in the row is obtained by addition or subtraction of the attributes of the first two items. *Distribute Three* indicates three different attribute values distributed through the row.

The human performance was evaluated by testing human subjects consisting of college students on a subset of samples from the dataset types. First of all, they were presented with a simple matrix with only one non-*Constant* rule to become familiar with the problem. Then, they were tested on complicated questions with complex rule combinations. The results, also with the comparison of various models they experimented with, can be found in table 1.1.

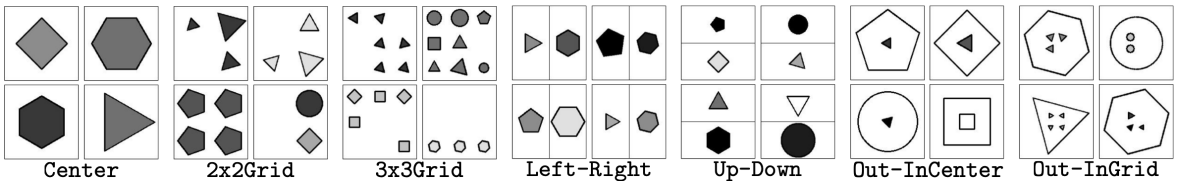

Figure 1.3: Different subtypes of RAVEN dataset [Zhang et al., 2019].

1.4.2 Solving RAVEN with Neural Networks

There have been proposed various perspectives and approaches for solving RPM with artificial neural networks so far.

The first attempts to experiment with the RAVEN dataset were performed right by its creators [Zhang et al., 2019]. They decided to try several well-known models to test their performance on the dataset. Considering the image representation of the problem, the authors decided to use models suitable for the task. In this case, as reasonable candidates were chosen computer vision models, specifically Long Short-Term Memory (LSTM), Convolutional Neural Network (CNN), Residual Neural Network (ResNet) and Wild Relation Network (WReN).

LSTM is a type of a recurrent neural network, designed to process sequential data with long-term dependencies. It is applicable to making the prediction of the next element in a series based on the dependencies in the input sequence. This seems to be convenient for the RPM problem, as it is of a partially sequential character. The authors decided to extract the image features by a CNN, then pass them into the LSTM sequentially and predict the final answer by feeding the last hidden feature into a multi-layer perceptron (MLP) with one hidden layer. Even though it may seem that this model suits the nature of the RPM problem, the results show that it was far from the best choice, yielding the average performance of only 13.07%.

In the second scenario, they set up a four-layer CNN followed by a two-layer MLP. The CNN extracts the image characteristics and the MLP applies the softmax activation function to the output layer to classify the answer. The convolutional layers of the CNN are alternating with batch normalization and ReLU [1] layers. The last hidden layer of the MLP uses dropout regularization to reduce over-fitting. This structure ended up more successful than the LSTM, the average result across the dataset types was 36.97%.

As the next model, they decided to try WReN, proposed specifically to solve RPM-like matrices [Barrett et al., 2018]. This model also uses a CNN for image feature extraction, but the features are then passed to a Relation Network module that forms representations between each context component and each candidate answer, and between the pairs of the context components themselves. It outputs scores for the can-

[1] activation function defined as f(x) = max(0, x)

Model	Avg	Center	2x2Grid	3x3Grid	L-R	U-D	O-IC	O-IG
LSTM	13.07%	13.19%	14.13%	13.69%	12.84%	12.35%	12.15%	12.99%
WReN	14.69%	13.09%	28.62%	28.27%	7.49%	6.34%	8.38%	10.56%
CNN	36.97%	33.58%	30.30%	33.53%	39.43%	41.26%	43.20%	37.54%
ResNet-18+MLP	53.43%	52.82%	41.86%	44.29%	58.77%	60.16%	63.19%	53.12%
Human	84.41%	95.45%	81.82%	79.55%	86.36%	81.81%	86.36%	81.81%

Table 1.1: Testing accuracy of the neural network models and human performance on the RAVEN dataset [Zhang et al., 2019].

didate choices and uses the softmax function at the output layer to predict the final answer. This model was well-designed for Procedurally Generated Matrices (PGM), which are similar to RAVEN, where it achieved the accuracy of 62.60%. However, PGM's average number of rules is 1.37, compared to RAVEN with 6.29 rules on average. PGM's gigantic size and limited diversity might disguise model fitting as a misleading reasoning ability [Zhang et al., 2019]. This caused the drop of the average accuracy on the RAVEN dataset to just 14.69%.

The most promising approach turned out to be the use of ResNet, which proved very effective in image feature extraction. The authors decided to replace the CNN foundation in the second network with ResNet-18 without pre-training. More information about the ResNet-18 model can be found in chapter 2. The average result was 53.43% with superior performance across all the subtypes. However, the gap between the human performance and the artificial models is still quite significant.

The RAVEN dataset is split into three subsets in 6:2:2 ratio for training, validation and testing, respectively. The models were trained with cross-entropy loss and Adam optimizer [Kingma and Ba, 2014], the problem being treated as a classification task. The complete results can be found in table 1.1.

The next attempt to study abstract reasoning was built on the belief that the aforementioned methods, which make use of deep neural networks, ignore some of the reasoning's fundamental characteristics. According the theory of the authors, one of the causes for the challenging nature of abstract reasoning is the presence of distracting features, which should not be taken into consideration while solving the problem. These models lack the ability to disentangle abstract reasoning and can't distinguish distracting features and reasoning features [Zheng et al., 2019]. Therefore, they present a different model, which is more fitted for abstract reasoning problems. Logic Embedding Network (LEN) is a neural network with a reasoning relational module. At first, the input images (8 context images from the problem matrix and 8 candidate images) are preprocessed by a shallow CNN and an MLP to extract the image features. The reasoning module then computes the scores for the consecutive row-wise and column-wise combinations of the context with the candidates. The answer is selected as the

Model	Avg	Center	2x2Grid	3x3Grid	L-R	U-D	O-IC	O-IG
LEN	72.9%	80.2%	57.5%	62.1%	73.5%	81.2%	84.4%	71.5%
ResNet-18	77.18%	72.75%	57.00%	62.65%	91.00%	89.60%	88.40%	78.85%
ResNet-50	86.26%	89.45%	66.60%	67.95%	97.85%	98.15%	96.60%	87.20%

Table 1.2: Testing accuracy of Logic Embedding Network (LEN) [Zheng et al., 2019] and ResNet models in supervised manner [Zhuo and Kankanhalli, 2020] on the RAVEN dataset.

highest value across the outputs of softmax activation function applied to the predicted scores.

As can be seen in table 1.2, this novel model with the reasoning module outperformed the best prior model by 19.5% on average. A remarkable observation is that the performance increase between the models on the *Center* dataset subtype is notably higher than the average, close to 30%. This might signify a closer similarity between this approach and human reasoning, considering that this dataset subtype shows to be the most elementary for human subjects, with the 95.45% accuracy of the answers.

Despite the opinion of the authors that the inferior performance of the deep networks was caused by not being well fitted for the problem, there have been major improvements in approaches without any reasoning modules. The most significant advancement, which inspired us to experiment with the provided solution more, was based on residual networks (ResNets). Different from the previous works that focus on designing an effective neural network, Zhuo et. al. mainly analyze how to reduce over-fitting in RPM, which is an important commonly occurring problem in supervised learning. Moreover, they wanted to reflect the human behavior and came up with an alternative based on unsupervised learning. They were interested whether NNs can solve RPM problems without any labeled data, thus they proposed a novel method called Multi-label Classification with Pseudo Target (MCPT). This approach is based on the observation that the third row in the problem matrix has to share the same rules with the first two rows.

The supervised method follows up on the initial work by Zhang et al. and further refines their foremost technique, which makes use of ResNet-18. Their results show that ImageNet pre-training [Paszke et al., 2019] and the use of deeper models is beneficial to solving RPM. Even though the type of images used in ImageNet pre-training is notably different from the RPM images, it can apparently help reduce over-fitting and speed up model convergence. Furthermore, they applied dropout on the penultimate layer and froze the parameters of all batch normalization layers in the model during training in all experiments. The models were trained using Adam optimizer and mini-batch of 32 samples, learning rate set to 0.0002 at the beginning and reduced every 10 epochs to its half. The input images were normalized to a range [0,1] and resized to 224x224,

Model	Avg	Center	2x2Grid	3x3Grid	L-R	U-D	O-IC	O-IG
Random	12.50%	12.50%	12.50%	12.50%	12.50%	12.50%	12.50%	12.50%
W/o MCPT	5.18%	7.60%	7.00%	7.00%	2.45%	3.20%	3.20%	5.30%
W/ MCPT	28.50%	35.90%	25.95%	27.15%	29.30%	27.40%	33.10%	20.70%

Table 1.3: Testing accuracy of ResNet-18 in unsupervised manner [Zhuo and Kankanhalli, 2020] on the RAVEN dataset.

as the standard input shape of ResNet.

Table 1.2 shows the results of the supervised experiments performed on the RAVEN dataset. The model with ResNet-50 backbone outperforms LEN by 13.36% on average, which seems to refute the theory that ResNet-based models do not exhibit the ability to distinguish between distracting and reasoning features. ResNet-50 surpasses the results of LEN on every problem configuration and even ResNet-18 performs better than LEN on every configuration except for *Center* and *2x2Grid*. But despite that, both of the ResNet models achieve the best results on the *Left-Right* and *Up-Down* subsets, whereas for human subjects was *Center* the least problematic. Furthermore, the *2x2Grid* and *Out-InGrid* are comparably complicated for humans, but the latter appears to be much more straightforward for the network. These differences might indicate that the reasoning ability on the human level works in a slightly different way.

The unsupervised technique, called Multi-label Classification with Pseudo Target, is based on the fact that the three rows of the problem matrix have to satisfy the same rules. The authors were inspired to explore the unsupervised solving techniques by the human approach, dealing with RPM without any supervision or labeled data. They decided to use the ResNet-18 model with ImageNet pre-training and the same implementation setup as in the supervised experiments. For the demonstration of the effectiveness of the method, they also provided the results of the model without a pseudo target and the accuracy of random guess. As the results show, unsupervised approach without MCPT seems to be counterproductive, with the average accuracy even below random guess. Given 8 candidate choices, the probability to select the right answer is 12.50%, which is 7.32% higher than the average correctness of the approach without MCPT. However, the approach with MCPT seems to be promising, more than doubling the accuracy of random guess. Another interesting observation is that the model trained with the MCPT approach performs the best on the *Center* subset, similarly to human reasoning.

The unsupervised learning is a fascinating area, because it does not require labels for training which makes it usable in vast areas where labelled data are not available, or only in small numbers. More detailed explanation of the MCPT approach is obtainable in chapter 2.

Chapter 2

Methodology and Implementation

In this chapter we state the aims and objectives of our work. The chapter also explains the methods used in the thesis and describes the implementation in detail.

2.1 Aims and Objectives

This thesis concerns the question, whether machines are capable of abstract reasoning, comparably to humans. In order to address this inquiry, we set the goals and specify the steps that need to be taken to achieve the desired outcome.

Our aim is to investigate the ability of neural networks to solve Raven's Progressive Matrices, a non-verbal IQ test widely used for testing human intelligence. We would also like to further analyze the reasoning ability of the chosen model by the visualization of its hidden representations. To achieve this, we implement a chosen neural network and train it for solving RPM. We test the model's generalization ability given various sizes of the training set and the types of problems it consists of.

We decided to follow up on the work [Zhuo and Kankanhalli, 2020] and use ResNet-18 as the studied model. Their results show superior performance of the models based on residual networks (see table 1.2). We use the publicly available implementation of ResNet-18 from the PyTorch library to make an unbiased comparison to the previous work. We implement the training according to the information provided in the article [Zhuo and Kankanhalli, 2020] and perform the experiments on the RAVEN dataset [Zhang et al., 2019].

2.2 Problem Definition

Let X be a problem matrix with k elements and Y the answer set with l potential candidates. To solve an RPM problem instance means to select the most fitting candidate y_z to complete the matrix X, where z is the index of the candidate y_z and $z \in \{1, ..., l\}$.

In the supervised approach, we are trying to learn a projection function f over (X, Y) and Z, given we have n training samples $\{(x_i, y_i, z_i) \mid 1 \leq i \leq n; x_i \in X, y_i \in Y, z_i \in Z\}$. The method can be formulated as

$$Z = f(\phi(X \cup Y), w) \tag{2.1}$$

where $\cup$ symbolizes the concatenation that stacks the images of the problem matrix with the answer set images, ϕ denotes the image features over $(X \cup Y)$ and w is the hyperparameter the network is supposed to learn. This way, the RPM problem is considered as a multi-class classification task. The network is trained using cross-entropy loss, which is calculated for every problem as

$$loss(out, z) = -\log(\frac{\exp(out_z)}{\sum_j \exp(out_j)}) \tag{2.2}$$

where z is the target, i.e. the index of the correct answer, and *out* the output of the network. As the final answer is selected the choice with the highest output probability. In the RAVEN dataset, we deal with matrices consisting of 8 images and the task is to select the 9-th element from 8 candidate choices to complete the matrix.

The aim of unsupervised approach is to find the correct answers for (X, Y) without giving the network the answers to the problem instances during the training. The rows of the correctly completed matrix, i.e. $(x_{i,1}, x_{i,2}, x_{i,3})$, $(x_{i,4}, x_{i,5}, x_{i,6})$ and $(x_{i,7}, x_{i,8}, y_{i,j}); j \in \{1, ..., 8\}$, have to share the same rules.

2.3 ResNet-18

ResNet-18 is a deep residual network having 17 convolutional layers and a fully-connected layer. Residual networks use skip connections, which makes them capable of jumping over some layers. The addition of the skip connections helps the model with the vanishing gradient problem and over-fitting, in case we use more than the suitable number of layers. The authors provide comprehensive empirical evidence showing that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth [He et al., 2016].

The PyTorch implementation of the standard ResNet-18 model can be seen in figure 2.1.

2.4 Supervised Learning

In this implementation, the images in the problem matrix X and in the corresponding answer set Y are stacked together as the input for the neural network. Therefore,

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

Figure 2.1: PyTorch implementation of standard ResNet-18.

we need to replace the first convolutional layer of the ResNet-18 model to expect 16-dimensional input. The output dimension of the fully-connected layer is set to 8, to coincide with the number of potential candidates.

2.5 Multi-label Classification with Pseudo Target

The MCPT approach uses a pseudo target in a form of two-hot vector, which does not contain the information about the correct answer. However, formally, it converts unsupervised learning to supervised. The vector is designed based on the observation that the rows are generated by applying the same rules. Therefore, the authors construct 10 rows as can be seen in figure 2.2. The first two are taken directly from the problem matrix and the rest 8 are produced as $(x_{i,7}, x_{i,8}, y_{i,j}); for\ all\ j \in \{1, ..., 8\}$. The first 2 values of the target vector are set to 1 and the other 8 to 0. This way, we force the outputs for the rows with the candidate choices to get close to zero. The expectation is that the row with the correct answer should keep producing higher output value than the rows with the incorrect ones, which should reflect the similarities of the correctly completed row and the two original rows from the matrix.

This unsupervised method can be represented by the formula

$$\tilde{Z} = g(\phi(X \vee Y), w) \tag{2.3}$$

where $\tilde{Z}$ represents the pseudo target, which is the same for every RPM instance, ϕ denotes the image features, g is the projection function, $(X \vee Y)$ are the rows, i.e. $r_1 = (x_{i,1}, x_{i,2}, x_{i,3}), r_2 = (x_{i,4}, x_{i,5}, x_{i,6}), r_j = (x_{i,7}, x_{i,8}, y_{i,j-2}); for\ all\ j \in \{3, ..., 10\}$. The rows constructed this way are then passed to the network to acquire their scores. The input layer of the standard ResNet-18 model is 3-dimensional, which is suitable for our input, although it is originally meant to take in RGB images. The images in the RAVEN dataset are grayscale, therefore they have only 1 color dimension, which means that we can fit in the whole row as the input in one step. The complete scores for each RPM problem are obtained in 10 consecutive steps. The output layer is replaced by a fully-connected layer with one neuron that predicts the score of the input row. The scores are then concatenated together and normalized by applying a sigmoid function to get scores in range [0,1]. The model is trained using Binary Cross Entropy as the loss function, which can be described as

$$loss(out, \tilde{z}) = -\sum_{i=1}^{n} [\tilde{z}_i \log out_i + (1 - \tilde{z}_i) \log(1 - out_i)] \tag{2.4}$$

where $\tilde{z}$ is the pseudo target vector, $\tilde{z}_i \in \{0, 1\}$, and out is the concatenated output of the network. During testing, the first two values of the prediction are removed and the candidate filled in the row with the highest score is considered the final answer.

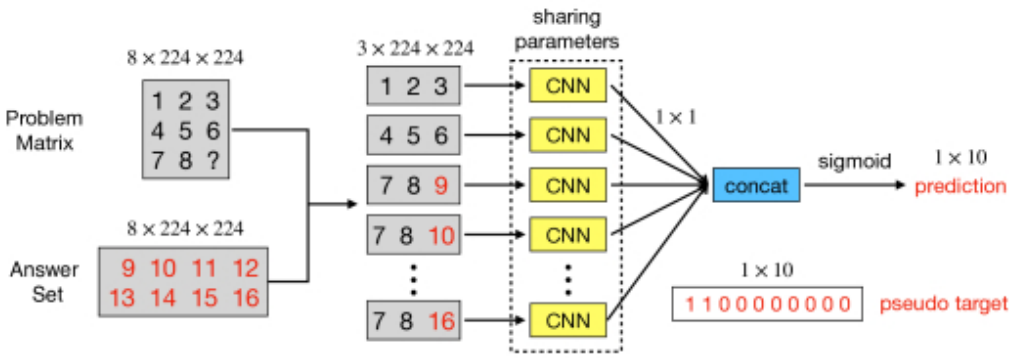

Figure 2.2: Schema of the MCPT approach [Zhuo and Kankanhalli, 2020].

2.6 Implementation Details

The differences in the model and training in the supervised and unsupervised approach are described in sections 2.4 and 2.5, respectively. This section presents the common features of the implementations.

From the wide variety of open source machine learning libraries, we decided to use PyTorch with the Python interface. ResNet-18 is one of the models that are implemented in the library, so we used the model pre-trained on the ImageNet dataset in all our experiments and made changes according to the article [Zhuo and Kankanhalli, 2020]. We added a dropout with value 0.5 before the last layer to reduce over-fitting, which is a significant problem in deep networks. Moreover, we froze all the batch normalization layers in the model. For the training of the network, we used Adam optimizer with the initial learning rate of 0.0002, which was reduced to its half every 10 epochs. The model was trained for 30 epochs, but it was validated after each epoch and as the final state for evaluation was selected the one with the highest validation accuracy. Then we performed the testing on the testing parts of the dataset subsets. We used 6 folds for training, 2 for validation and 2 for testing. The input images were normalized to the range [0,1] and resized using bicubic interpolation to 224x224, which is the standard input size for ResNet-18 in PyTorch. To speed up the training of the model, we ran the computations on the GPU using CUDA technology developed by NVIDIA. The CUDA Toolkit provides a comprehensive development environment for C and C++ developers building GPU-accelerated applications [Cook, 2012], which was utilized by the developers of PyTorch to accelerate the performance of the library.

We used UMAP for the visualization of the hidden representations on the penultimate (average pooling) layer to look closer at the decisions of the network. UMAP is a general purpose manifold learning and nonlinear dimension reduction algorithm [McInnes et al., 2018], that is very popular for the visualizations in machine learning.

For more detailed description of the attached source codes see Appendix A.

Chapter 3

Experiments

This chapter describes the experiments that were performed on the RAVEN dataset, lists and interprets the obtained results.

3.1 Supervised Learning Experiments

3.1.1 Joint Training

First of all, we decided to replicate the basic experiment, which is demonstrated in [Zhuo and Kankanhalli, 2020]. The authors report their results on ResNet-18 without pre-training, but we use the model with ImageNet pre-training in our experiments. Otherwise, the experimental setup is identical to the one in their work. As table 3.1 shows, the average accuracy of the models is very similar, which suggests that the pre-training does not play an important role in the final model generalization ability in this case. The results of this experiment are used as the baseline for the comparison of the latter experiments mentioned in the thesis.

Model	**Avg**	**Center**	**2x2Grid**	**3x3Grid**	**L-R**	**U-D**	**O-IC**	**O-IG**
w/o pre-train	77.18%	72.75%	57.00%	62.65%	91.00%	89.60%	88.40%	78.85%
w/ pre-train (ours)	77.06%	78.30%	56.10%	59.65%	92.50%	91.15%	86.25%	75.50%

Table 3.1: Testing accuracy of supervised ResNet-18 without pre-training [Zhuo and Kankanhalli, 2020] on the RAVEN dataset compared to our experiment with a pre-trained model.

The results show that the pre-training causes the network to perform slightly better on the subsets with less complicated image structure. The ImageNet dataset consists of real-world images, which are very different from the ones in the RAVEN dataset. However, according to the article, the pre-training helps significantly with the training of the deeper ResNet models. Here it seems to help with the categories *Center*, where

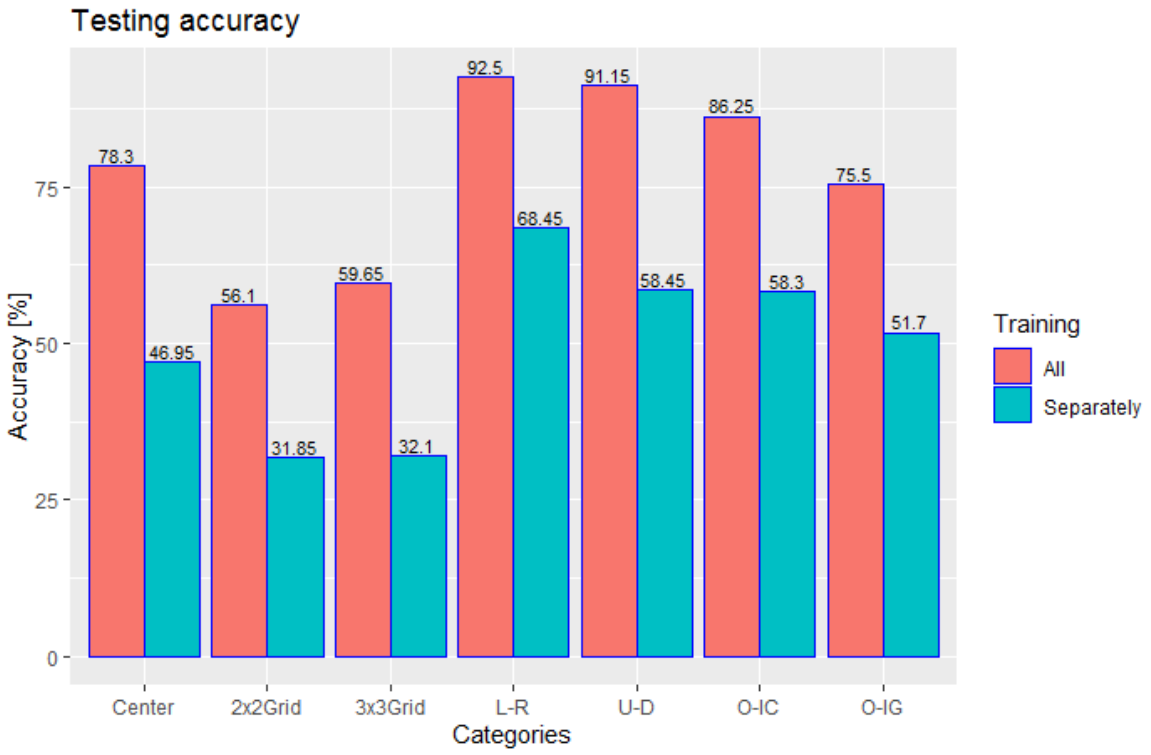

Figure 3.1: Comparison of the testing accuracy of the model trained on all subsets and trained separately for each category.

it led to quite notable increase in accuracy, *Left-Right* and *Up-Down*, where it caused just a minor improvement. The subsets *2x2Grid* and *Out-InCenter* show only small decrease in the testing accuracy and the results of *3x3Grid* and *Out-InGrid* fell by approximately 3%.

In the next experiment, we decided to study the outcomes of training the network on the subsets separately. Each subset consists of 6000 RPM problems dedicated for training, which is considerably less than the whole dataset combined. Our hypothesis was that the model would perform worse compared to the training on the whole set.

3.1.2 Separate Training

Figure 3.1 indicates that the experiment fulfilled the expectations. All the problem categories exhibit significant decrease in accuracy. The main problem with shrinking the dataset size seems to be over-fitting, because the model is too large for small datasets. However, the figure shows that even though the dataset size changed equally (from 42 000 for the whole dataset to 6000 for a single category), the testing accuracies did not decline by the same rate. When we train the model on the whole dataset, most of the categories perform well, which goes along with the average accuracy of 77.06%. Some of the subsets manifest worse results, which is intensified even more when trained

separately.

Left-Right is the most successful subset, with 92.50% accuracy, provided we train the model on all the subsets combined together. Its accuracy drops to 68.45% when trained separately. That represents decline by a quarter of the initial accuracy. A very similar result was obtained on *Up-Down*, where the training on the whole dataset ended with the accuracy of 91.15%. The decrease was a bit stronger, resulting in 58.45%. We consider this very interesting, given the difference between *Left-Right* and *Up-Down* was originally just 1.35% and one task is just a transposition of the other.

On the other hand, *2x2Grid* performs much worse already in the first experiment. The accuracy starts on 56.10%, which is less than the accuracies of 3 of the other subsets (*Left-Right*, *Up-Down*, *Out-InCenter*) trained separately. Moreover, if we train *2x2Grid* separately, the precision of the answers drops to 31.85%, which is the lowest of the results.

Another interesting point is that the ratio of the result obtained after the training on a single subset to the one after the training on the whole dataset varies with each category. The smallest decrease is observable on the *Left-Right* subset, where the result of the separate training presents 74.00% of the original result (decrease of 26.00%). *Up-Down* shows an absolute decrease of 32.70%, which is 35.87% relatively to the result of the training on the subsets combined. Also *Out-InGrid* and *Out-InCenter* demonstrate comparable fall of accuracy, of 31.52% and 32.41%, respectively. However, the results of the other categories diminished more significantly, *Center* by 40.04%, *2x2Grid* by 43.23% and *3x3Grid* by 46.19% relatively to the prior results.

3.1.3 Combinations of 2 Categories

Since there are obvious differences between the subset performances, we decided to study the influence of the training on more subsets at once. We expected that it would help the model to improve its generalization ability.

To begin, we tried several combinations with the *Center* category. When trained separately, *Center* results in the accuracy of 46.95%. We tried the combinations with *2x2Grid*, *Out-InGrid* and *Out-InCenter*. As can be seen in figure 3.2, the correctness of the answers on *Center* increased in every scenario, although by a bit different rate. The better the category scores separately, the more it helps to lift the accuracy of *Center*. However, the result of the other category in each experiment shows the opposite behavior, so the better the category scores separately, the less its accuracy is raised when trained together with *Center*. That means that the biggest increase is observable on *2x2Grid*, where it increased by 8.9%, while the smallest is on *Out-InCenter*, where the accuracy rose from 58.3% to 63.8%, which makes the growth of 5.5%.

In the next experiment, we tried combinations with *2x2Grid*. This is the subset

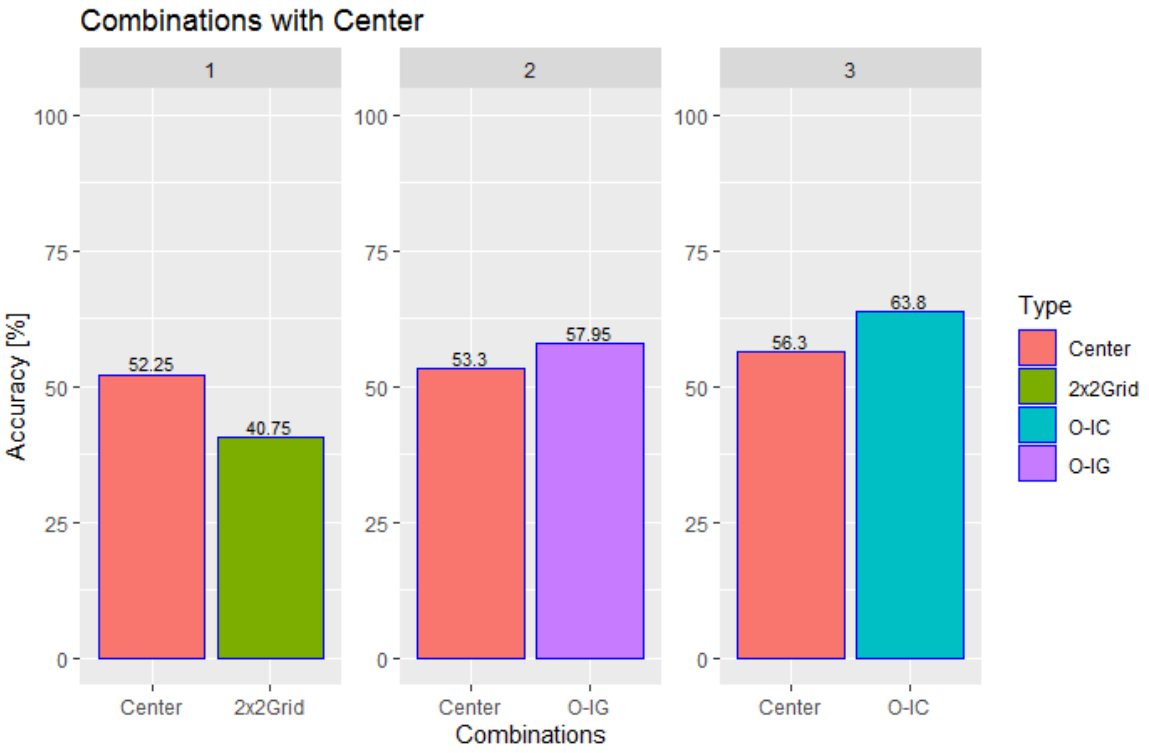

Figure 3.2: Results of the model trained on a combination of 2 subsets, where one of them is *Center*.

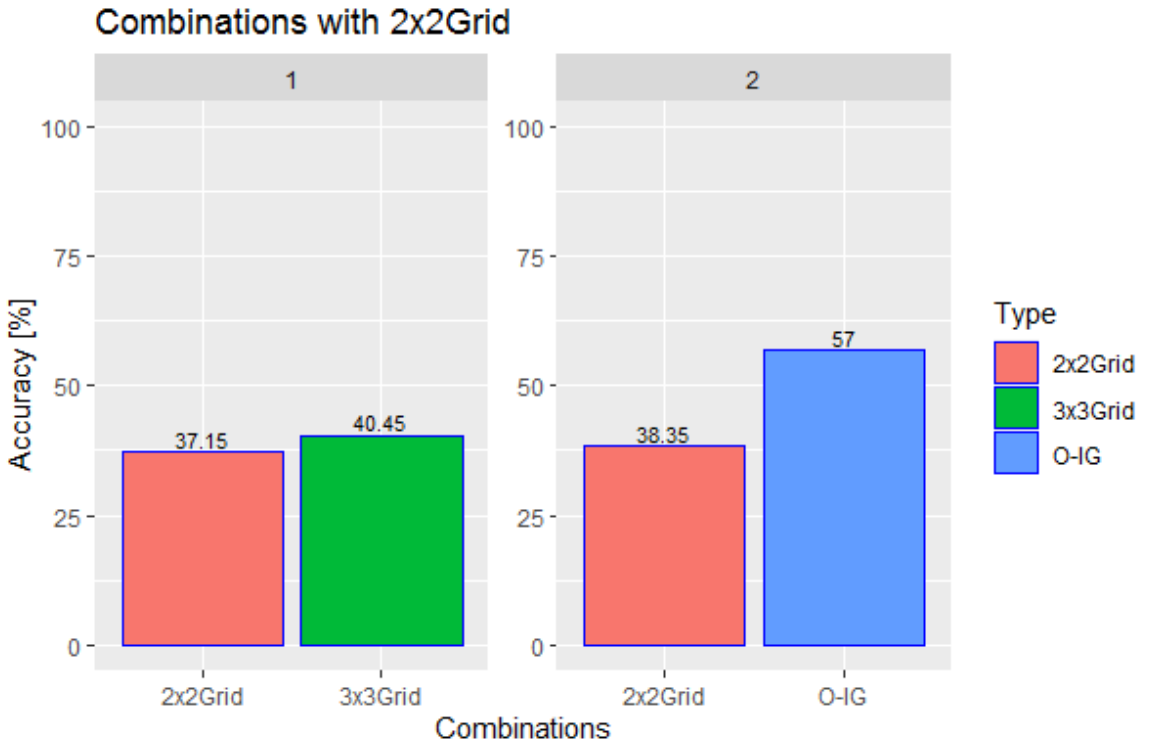

Figure 3.3: Results of the model trained on a combination of 2 subsets, where one of them is *2x2Grid*.

that performs the worst when trained separately. When trained with *Center* (figure 3.2), the accuracy rises to 40.75%. We tried to combine it with visually similar types - *3x3Grid* and *Out-InGrid*. However, *2x2Grid* did not perform better, compared to trained with *Center*, in any of these experiments. Also *Out-InGrid* does not demonstrate a greater increase in accuracy, showing the improvement of only 5.3%. The results are summarized in figure 3.3.

The next experiments deal with combinations of *3x3Grid* with *Left-Right* and *Up-Down*, as shown in figure 3.4. *Left-Right* and *Up-Down* are the types producing the best results when trained separately. They both seem to have a positive effect on the accuracy of *3x3Grid*, lifting it to 45.65% and 47.55%, respectively. On the other hand, the accuracies of both the other subsets decreased, *3x3Grid* having an inhibitory effect in the training of these sets.

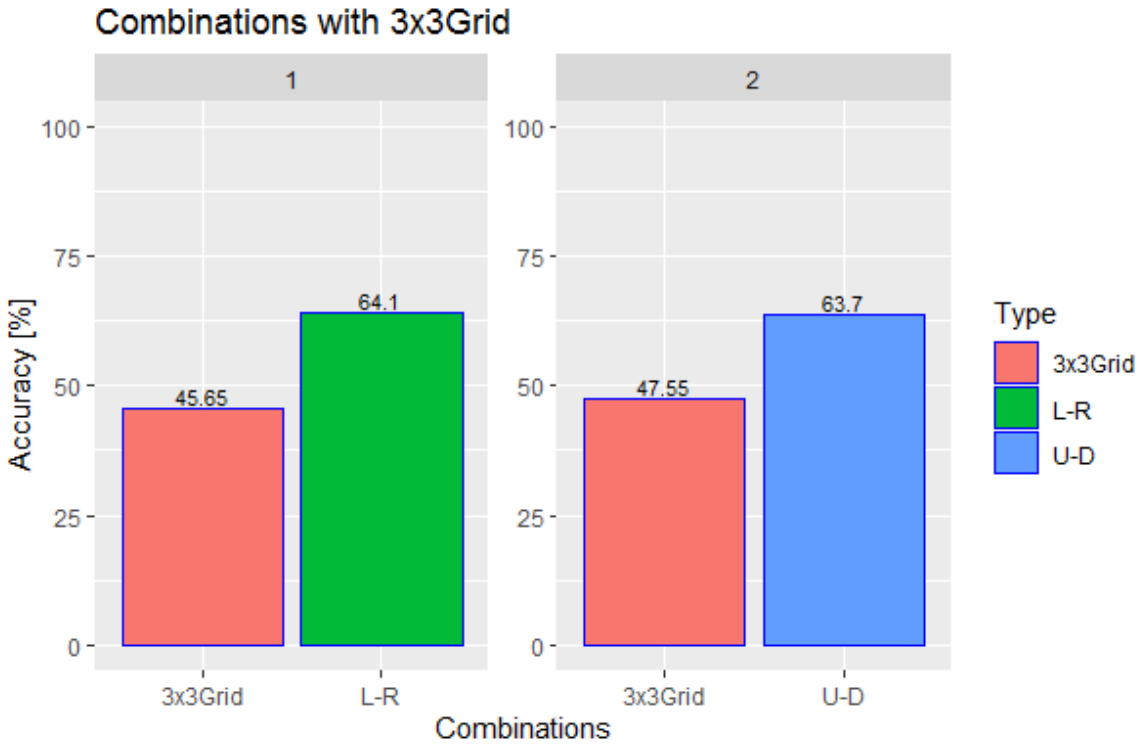

Figure 3.4: Results of the model trained on a combination of 2 subsets, where one of them is *3x3Grid*.

The last combinations of two subsets consist of categories that are similar to each other. As was mentioned above, *Left-Right* and *Up-Down* are a transposition of each other with similar testing accuracies after the separate training. Figure 3.5 implies that the comparable starting point resulted in increase in both of the accuracies. *Out-InCenter* and *Out-InGrid* both comprise of an outer shape with an inner structure. The separate training ended up with the results of 58.3% for *Out-InCenter* and 51.7% for *Out-InGrid*. The second graph of figure 3.5 illustrates that they also enhance results of

each other when trained together and the joint training reduces the difference between their accuracies.

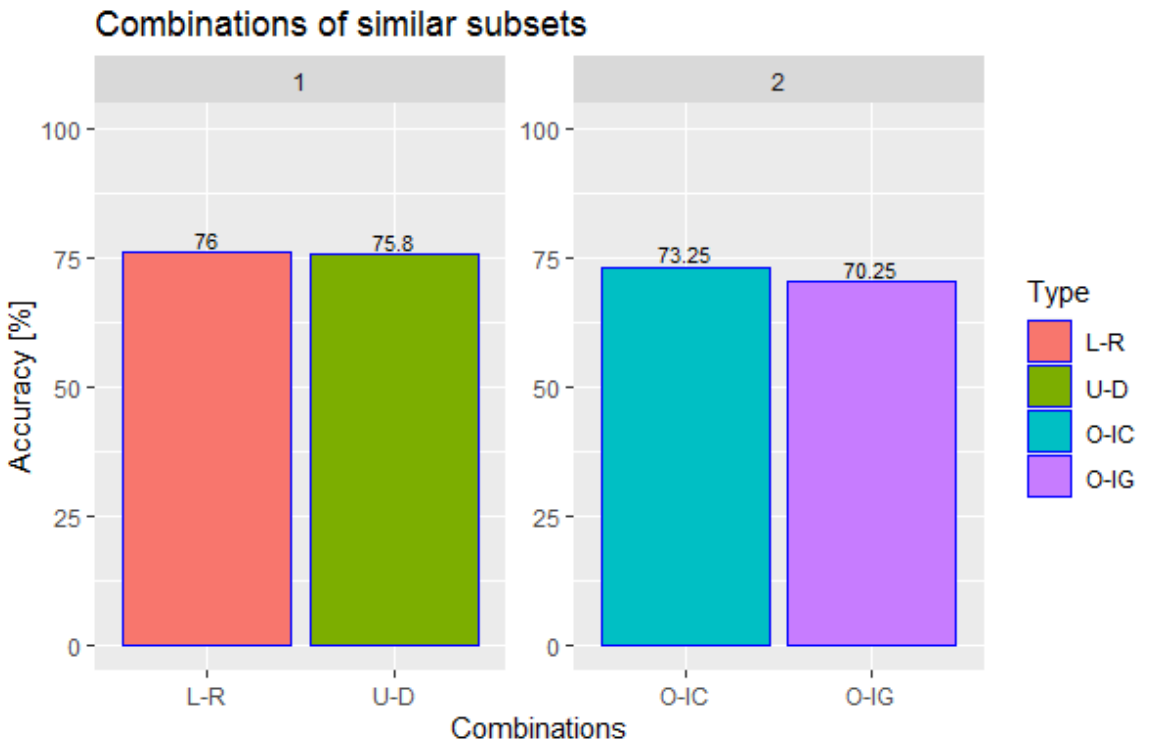

Figure 3.5: Results of the model trained on a combination of 2 visually similar subsets.

3.1.4 Combinations of 3 Categories

The results of the previous experiments suggest that the size of the training set plays a significant role in the model generalization ability. However, the outcome obviously depends also on which of the categories are joined together. Therefore, we decided to look into this further and perform experiments with triplets of categories.

First of all, we tried to combine three subsets that do not have much in common and then compare it to a triplet consisting of somehow connected subsets. The first group contains *Center*, *3x3Grid* and *Out-InGrid*. Figure 3.6 illustrates the contrast between the separate training and the joint training of these three problem categories.

The second triplet consists of *Center*, *2x2Grid* and *Out-InGrid*. These categories have a visually detectable connection. *Out-InGrid* comprises of an outer shape, which is similar to *Center*, and an inner structure of *2x2Grid*. Moreover, the results should be comparable, because *2x2Grid* and *3x3Grid* produce very similar results when trained separately. As figure 3.6 shows, there is a difference to some extent between the results of the experiment with *2x2Grid*, compared to the one with *3x3Grid* (figure 3.6). *3x3Grid* manifests a bigger increase in accuracy than *2x2Grid*, but *2x2Grid* enhances

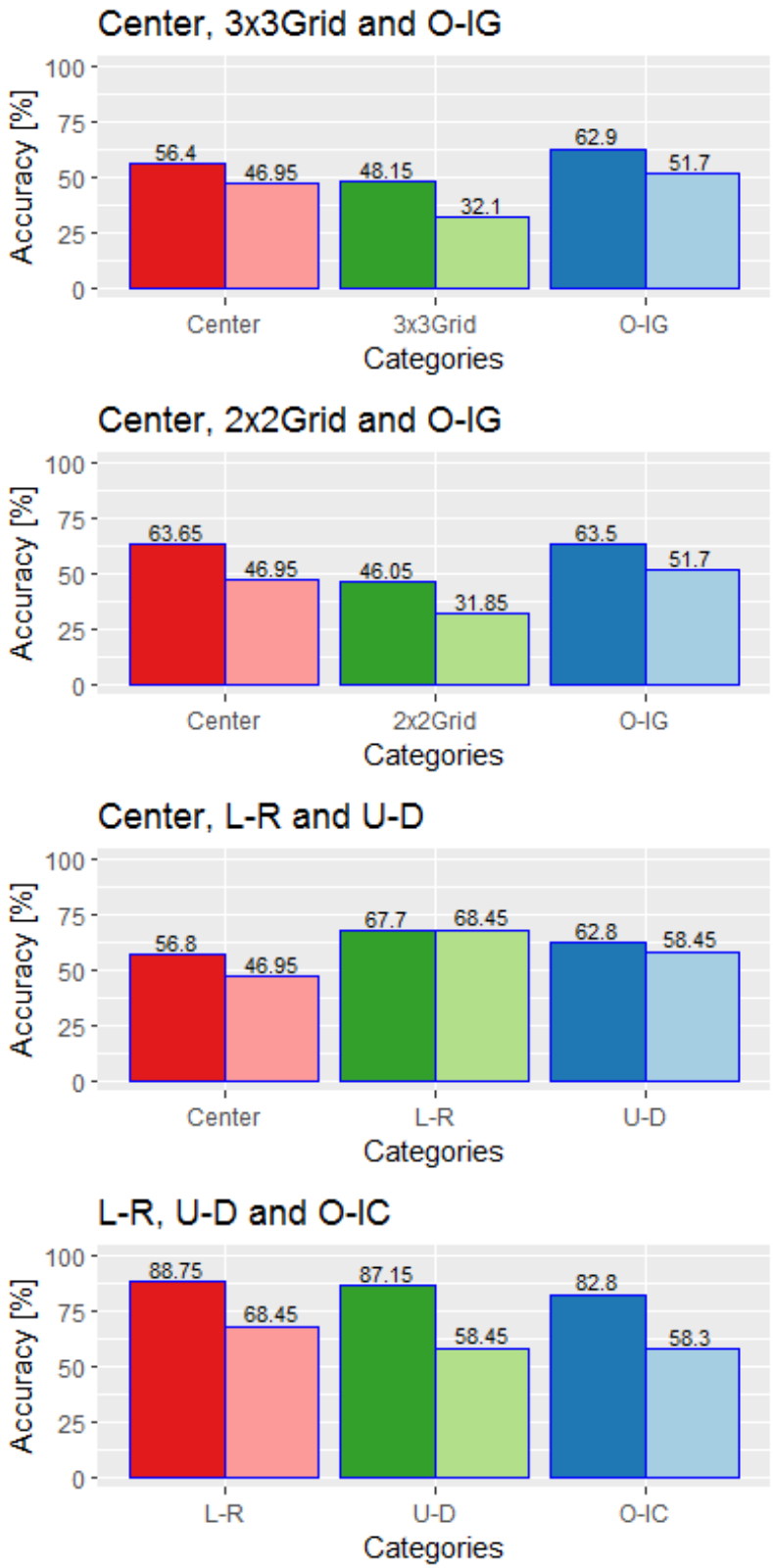

Figure 3.6: Results of the model trained on 3 subsets. Darker colors represent the accuracy of the combined training and lighter the results obtained after separate training.

the results of *Center* and *Out-InGrid* more.

The next experiment deals with subsets *Center*, *Left-Right* and *Up-Down*. The combination of *Left-Right* and *Up-Down* was very successful, therefore we were interested, what effect might have the addition of another category into training. It is a little complicated to decide which category has the most similar structure, so we chose *Center* as it has the most elementary structure. Figure 3.6 displays surprising results of the experiment. Not only did not the addition of *Center* help to increase the testing accuracy of *Left-Right* and *Up-Down*, it led to a significant decrease, compared to the joint training of these 2 categories.

As the last experiment with a triplet, we chose the 3 subsets producing the best results when trained separately, which are *Left-Right*, *Up-Down* and *Out-InCenter*. We assumed that they have a potential to improve each other's performance. This assumption was confirmed, supported by the results in figure 3.6. Each category exhibits a significant improvement, scoring over 80%. The average increase is 24.5%, which advances the accuracies almost to the level of the training of all the problem types together. In that scenario, *Left-Right* scores 92.5%, which is only 3.75% more, *Up-Down* 91.15%, which is a 4% difference and *Out-InCenter* 86.25%, which is 3.45% over the result of this combination.

3.1.5 Excluding Categories

After observing the differences between the separate training and the training of combinations, there arose a question, how would excluding certain categories from the whole training set affect the results. We decided to study 2 groups of types with visual connections - *Left-Right* with *Up-Down*, and *2x2Grid* with *3x3Grid* and *Out-InGrid*.

In the first set of experiments, we began with excluding only *Left-Right*. As can be seen in figure 3.7, the result of every category decreased a little. The most significant decline is observable on the *Left-Right* category, as expected. The average decrease among the trained categories is 2.38%, while the untrained subset got worse by 22.35%.

Then we excluded only *Up-Down* and compared the results. Figure 3.8 portays the comparison to the training of the whole training set. The results are similar to the first experiment, the average decrease is 2.25%, provided we take into consideration the trained subsets. The performance of *Up-Down* declined by 15.45%, which is notably less than the decrease of untrained *Left-Right*.

Lastly, we exclude both of the subsets, training on the remaining 5 categories. From figure 3.9 is observable that both of the excluded categories perform even worse than when excluded individually. Compared to the previous two experiments, all the problem types produce worse results. The average decrease of the trained subsets is 6.85% and among all the subsets 12.22%.

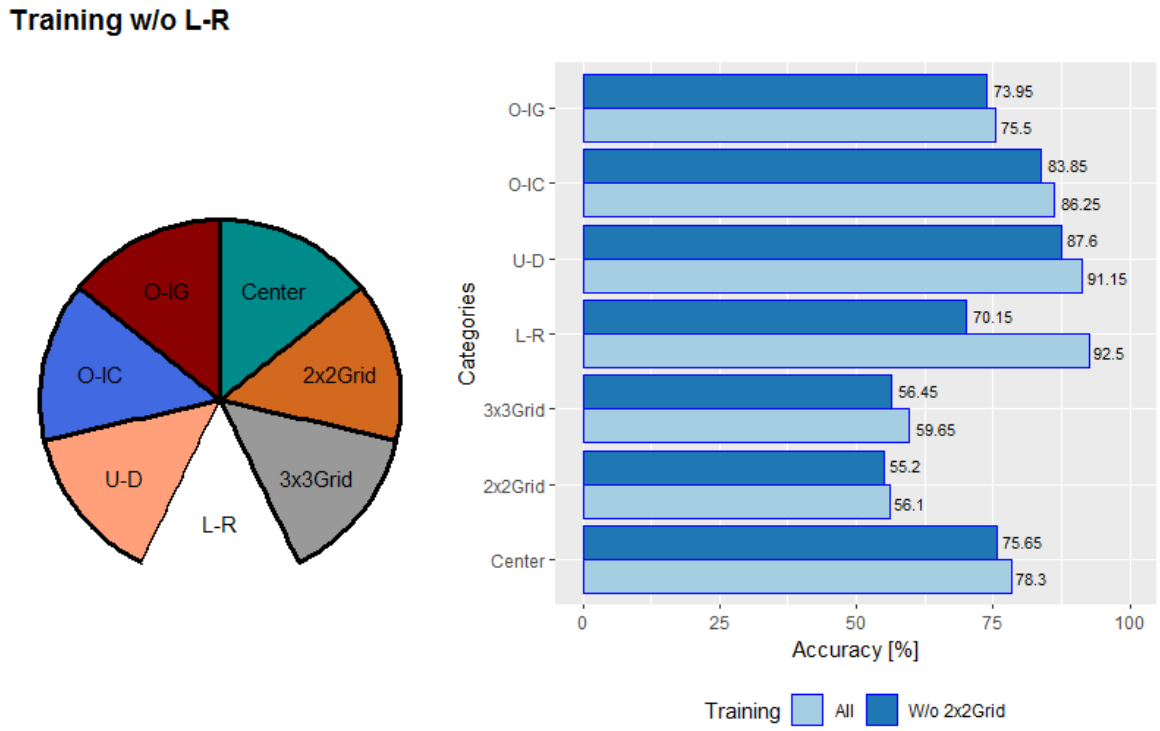

Figure 3.7: Results of the model trained on the whole training set, excluding *Left-Right*.

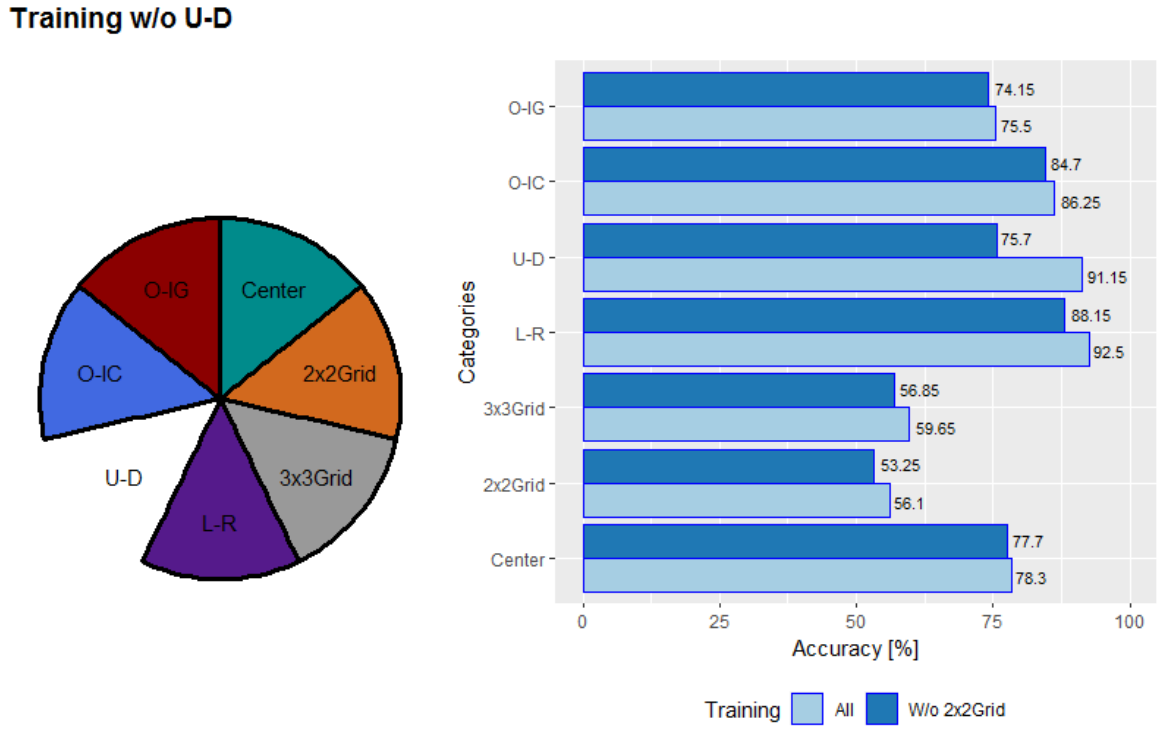

Figure 3.8: Results of the model trained on the whole training set, excluding *Up-Down*.

The second group was formed around the subset *2x2Grid*, because it produces the worst score. At first, we excluded only the one and observed the changes in the

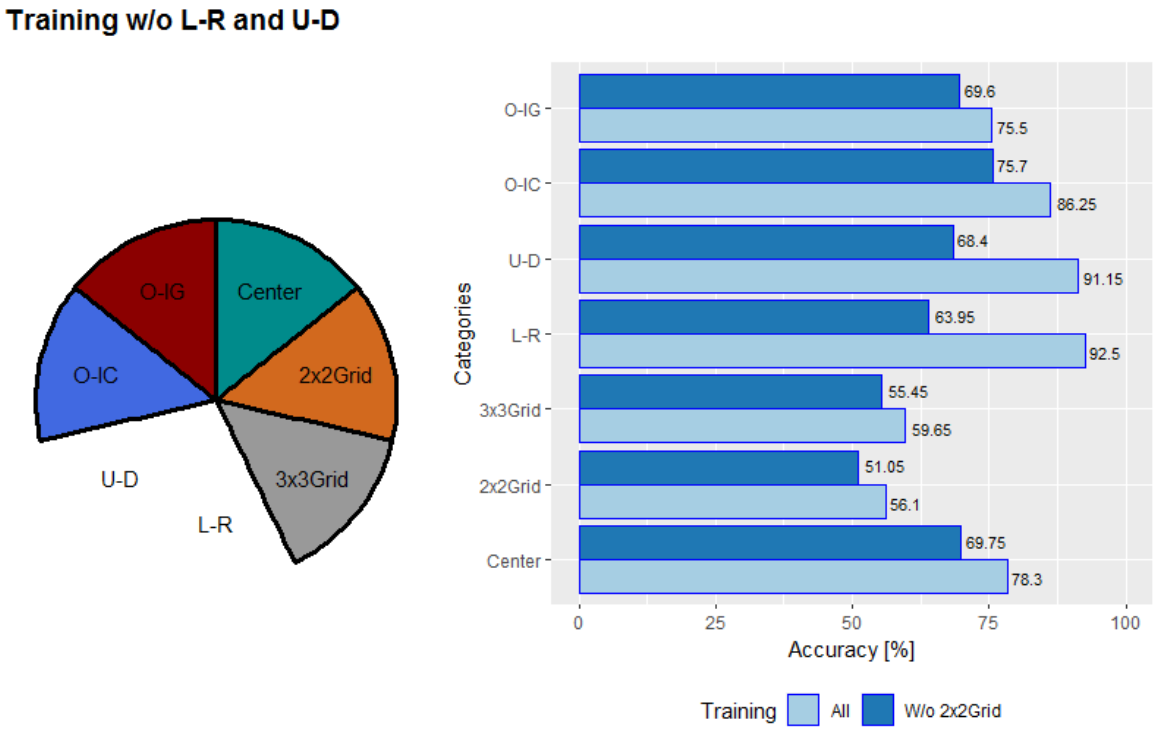

Figure 3.9: Results of the model trained on the whole training set, excluding *Left-Right* and *Up-Down*.

scores. Figure 3.10 shows the results of training without *2x2Grid*. As expected, all the categories display a decrease in accuracy. However, it is interesting that the excluded dataset did not drop very much, only by 10.6%, similarly to other categories.

When we exclude *2x2Grid* and also *3x3Grid*, all the categories, except for *3x3Grid*, manifest a smaller decrease in testing accuracy, as shown in figure 3.11. When excluding only *2x2Grid*, the average decrease is 9.07%, when excluding both *2x2Grid* and *3x3Grid*, the decrease changes to 7.08%. This is remarkable, given we reduce the size of the training set.

We further analyze the outcomes by excluding *2x2Grid* and *Out-InGrid*. Figure 3.12 implies a similar behavior as when we exclude *2x2Grid* with *3x3Grid*. This training set also results in a smaller decrease of accuracy of every problem type, except for *Out-InGrid*, compared to excluding only *2x2Grid*. The average decrease dropped to 7.35%, which is similar to case of excluding *2x2Grid* and *3x3Grid*.

The final experiment deals with the consequences of excluding the whole group (*2x2Grid*, *3x3Grid* and *Out-InGrid*), the results are summarized in figure 3.13. The experiment copies the trend of the previous ones, resulting in an even smaller decrease of 6.43%. All the included subsets show a better performance, compared to excluding less subsets, while the accuracy of the excluded ones slightly varies.

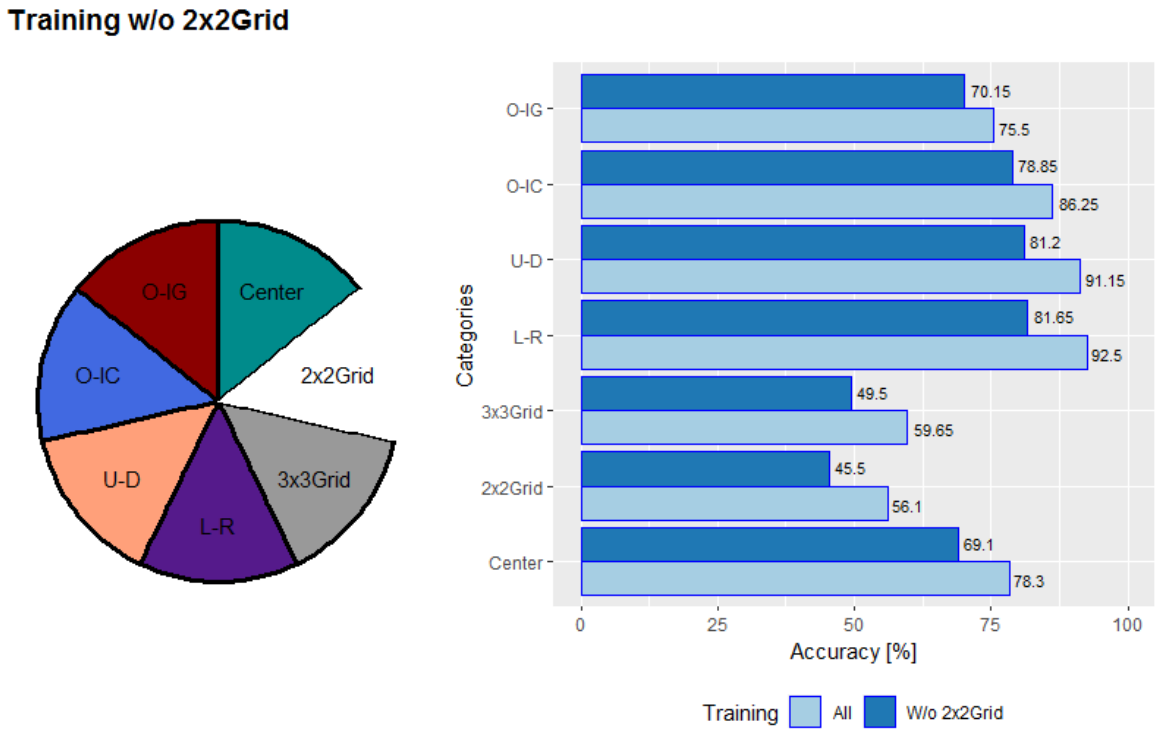

Figure 3.10: Results of the model trained on the whole training set, excluding *2x2Grid*.

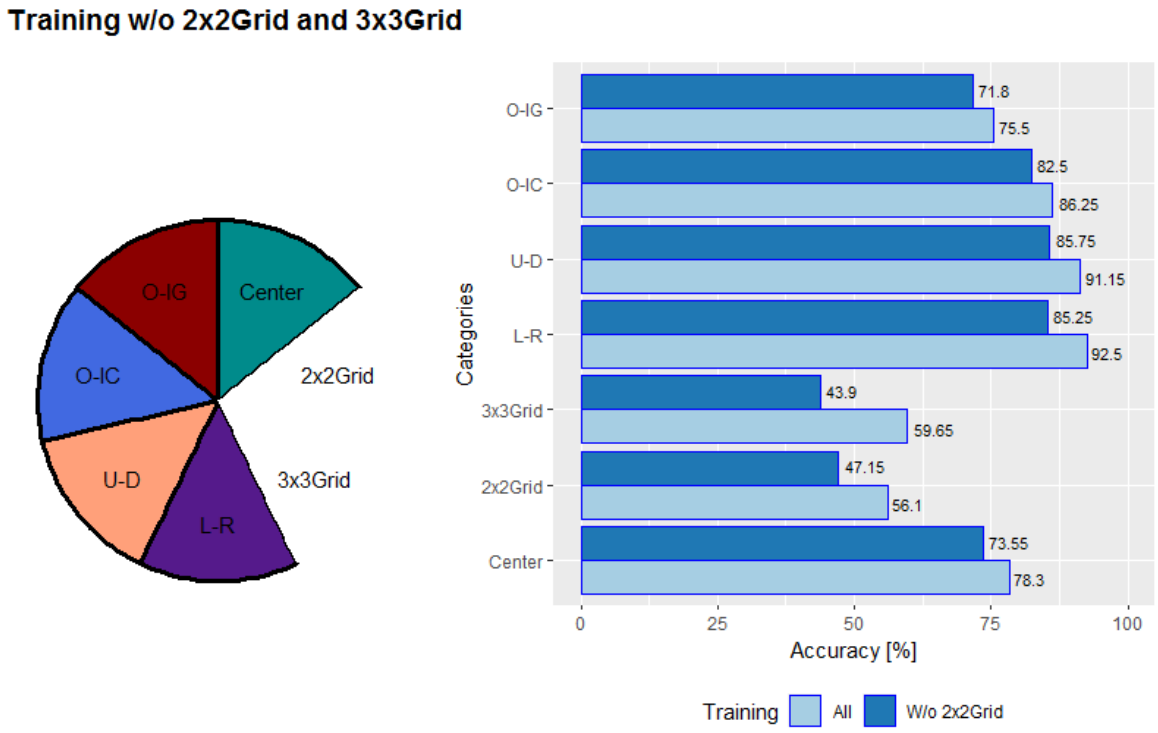

Figure 3.11: Results of the model trained on the whole training set, excluding *2x2Grid* and *3x3Grid*.

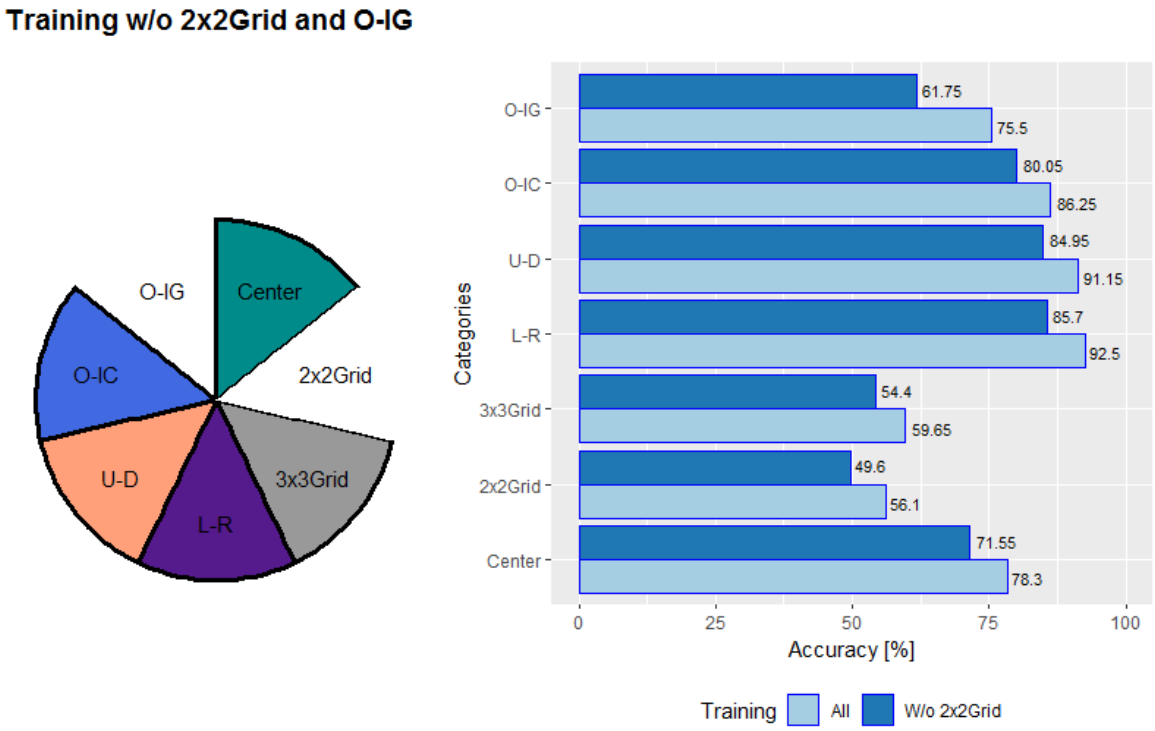

Figure 3.12: Results of the model trained on the whole training set, excluding *2x2Grid* and *Out-InGrid*.

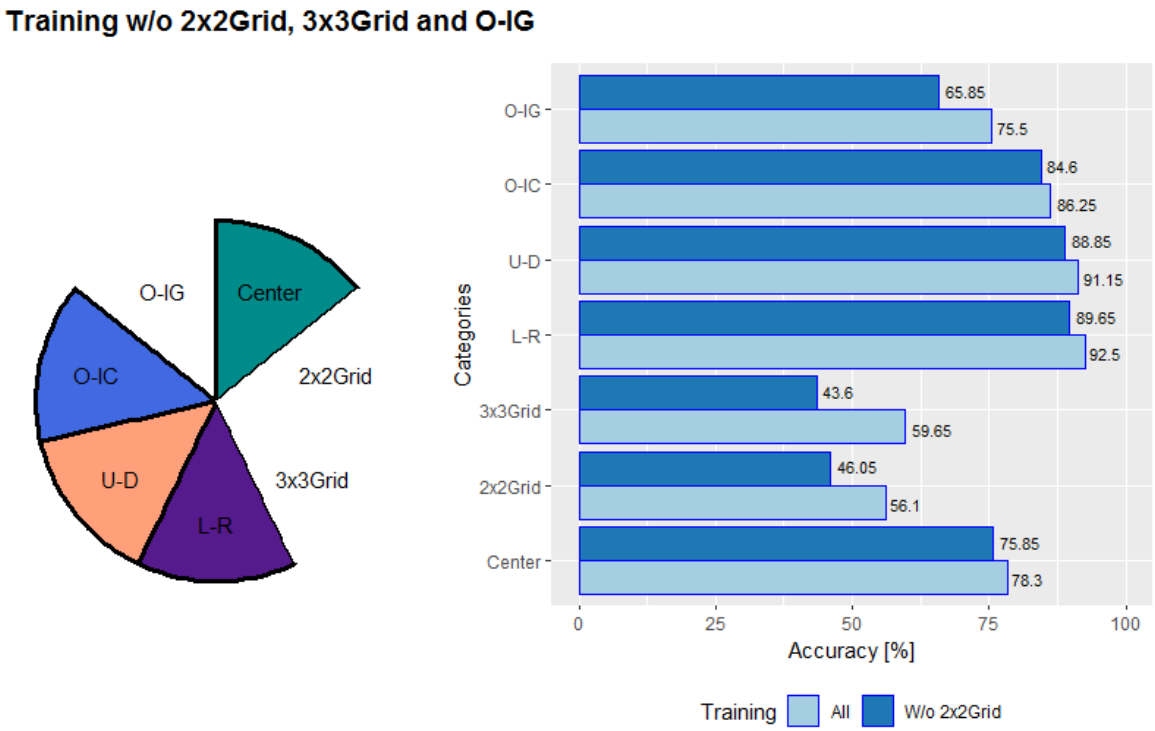

Figure 3.13: Results of the model trained on the whole training set, excluding *2x2Grid*, *3x3Grid* and *Out-InGrid*.

3.2 Unsupervised Learning Experiments

The unsupervised approach from the article is based on an interesting principle, so we also tried to perform a few experiments inspired by the MCPT approach. Firstly, we tried to replicate the experiment and compare the outcomes. As shown in table 3.2, the experiment ended up unsurprisingly, producing results very similar to the ones in the article.

Model	Avg	Center	2x2Grid	3x3Grid	L-R	U-D	O-IC	O-IG
Zhuo et. al.	28.50%	35.90%	25.95%	27.15%	29.30%	27.40%	33.10%	20.70%
Ours	27.79%	36.00%	23.95%	28.50%	27.20%	24.95%	32.15%	21.75%

Table 3.2: Testing accuracy of ResNet-18 in unsupervised manner [Zhuo and Kankanhalli, 2020] on the RAVEN dataset compared to our results.

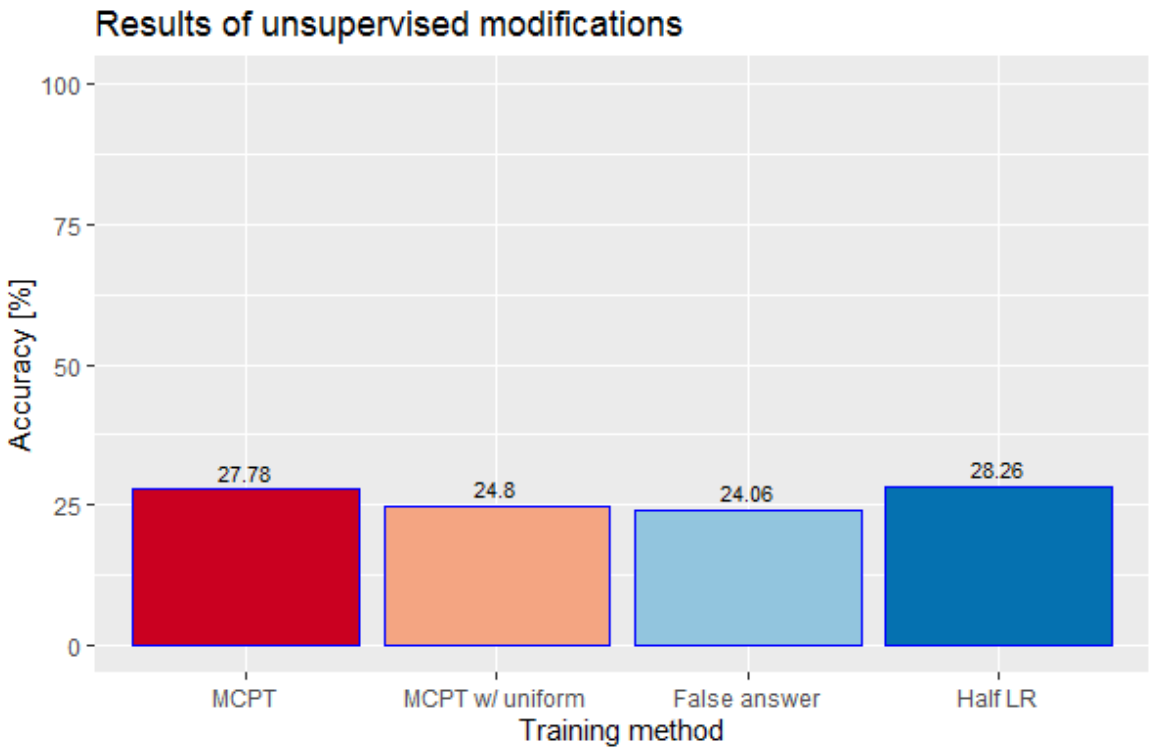

Figure 3.14: Results of the model trained on the whole training set in an unsupervised manner in various scenarios.

The next experiments preserve the nature of the MCPT approach. The average accuracies are depicted in figure 3.14 and compared to the basic MCPT approach. The first two experiments are based on the modification of the pseudo target vector. The target in MCPT is a vector of 2 ones and 8 zeros. In our first experiment, we keep the first 2 values and the 8 are chosen randomly from the uniform distribution in range [0,0.5]. The underlying hypothesis was that, rather than to force the outputs

to predict zero during training, the corresponding output neurons would be provided with arbitrary information, statistically balanced. In this aapoach, there could still be incentive to build internal representations that could help the correct output canidate to produce the highest output value drawing on shared similarities with the first two rows of inputs. This experiment did not end up better than the original MCPT approach, with the testing accuracy of 24.8%. In the second experiment, one of the 8 zeros is replaced with one, chosen randomly. This way, we do not tell the network the correct answer, but we let it know that there exists one. However, this did not help to improve the accuracy either, resulting in 24.06% accuracy. The last unsupervised experiment uses the original pseudo target, but the initial learning rate for the output layer is set to half the learning rate of other layers, being 0.0001. As the figure shows, the modification did not make a difference, compared to the MCPT. Unfortunately, we did not come up with a technique that would improve the results in an unsupervised manner.

3.3 Visualization and Interpretation of the Results

To further analyze the behavior of the network, we looked into it and collected the activations of the penultimate layer. To reduce the high dimensionality for visualization, we used UMAP and plotted it in 2D. Figure 3.15 displays the hidden representations of the model trained on the whole RAVEN dataset. The answers seem to form clusters based on the winner neuron. We noticed that the clusters of *Left-Right* are much more separated than the clusters of the worse performing subsets, e.g. *2x2Grid*. To explain this phenomenon, we measured the confidence of the network while answering the questions. This proved to be the answer, as captured in figure 3.16. The subsets with higher accuracy demonstrate higher confidence, which results in more distant clusters of answers.

It is difficult to interpret the results, more accurate interpretation would need to study and somehow precisely measure the similarities of the problem categories. We based the selections of the subsets in our experiments on their visual similarities. Moreover, we decided to always train the model either on the whole subset or on none of its items, never training just a certain portion. It is known that reducing the number of the training samples decreases the accuracy of neural network models and we focused on the dynamics of the model given various types of questions. However, some of the results look very interesting, suggesting that the network profits from the knowledge acquired from the other problem types.

Concerning the training on the combinations of 3 subsets, it is apparent that the smallest improvement was notable on the combination of 3 random ones, i.e. those that

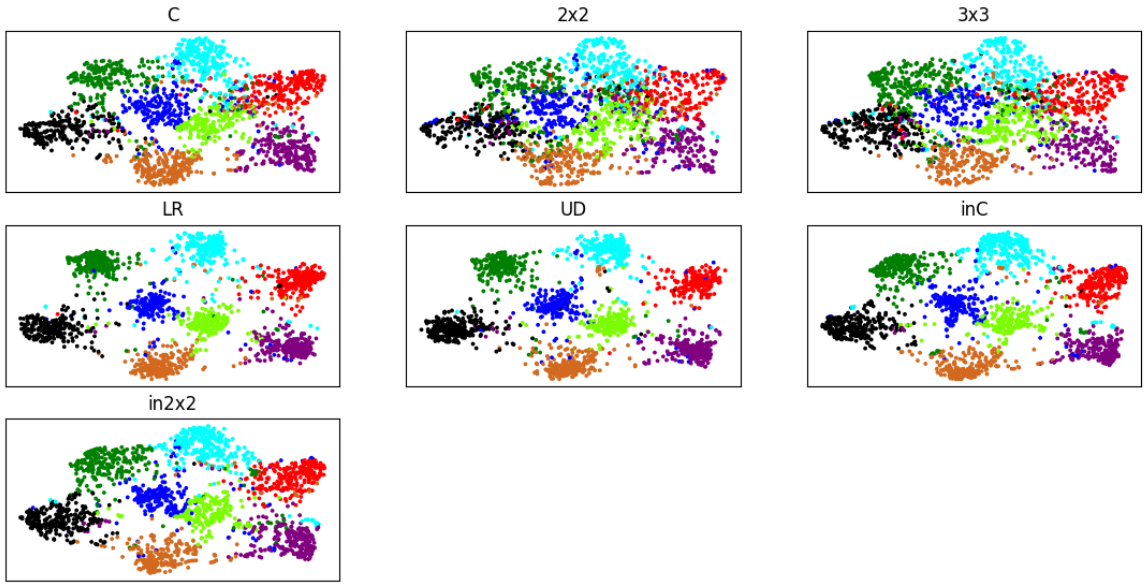

Figure 3.15: Visualization of the hidden representations on the penultimate layer of the model trained on the whole training set in a supervised manner. The different colors illustrate the order of the answer chosen by the network (1 to 8), i.e. the most activated output neuron.

are visually dissimilar. On the contrary, the highest advance showed the experiment with the subsets with visually observable connection - *Center*, *2x2Grid* and *Out-InGrid*. Although, we consider interesting that this combination resulted in significantly higher accuracy of *Center* and not *Out-InGrid* as we expected (compared to the random combination of *Center*, *3x3Grid* and *Out-InGrid*). This behavior remains unclear and would need further research.

Another interesting observation is that when we excluded the *2x2Grid*, the accuracy of the other subsets dropped more than when we excluded both *2x2Grid* and *3x3Grid*. The similar situation arose when we excluded *2x2Grid* and *Out-InGrid*. Furthermore, excluding all three of these subsets seemed to be the best option, suggesting these problem types are perceived in a similar manner by the network.

The second studied group of excluded subsets also produces intriguing results. When we trained all the subsets with the exception of *Left-Right*, the decrease of its accuracy was less significant than when we excluded both *Left-Right* and *Up-Down*. The same is notable when eliminating only *Up-Down* from the training, its accuracy is higher than when omitting both of the subsets. We assume there is a certain connection between these two problem types from the network's point of view.

The last observation we would like to point out is that the performance of *Up-Down*

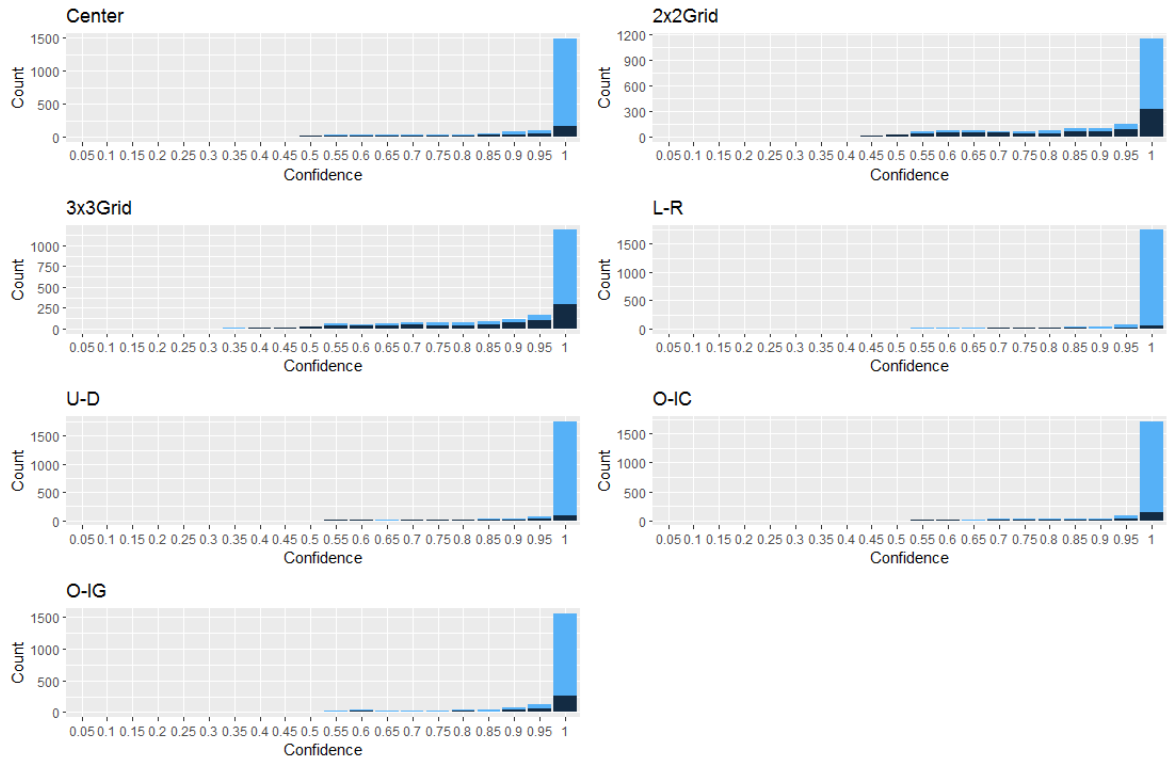

Figure 3.16: Histograms of the confidence of the model trained on the whole training set in supervised manner. The dark blue marks the incorrect answers and light blue the correct ones.

in the scenario of training the network on all the categories, except for the aforementioned one, is superior to its performance when trained only on the one, indicating that the network extracts knowledge from the other problem types. The correctness of the answers in the case without *Up-Down* represents 75.7% of the answered questions from this problem type, which could be considered a decent outcome even for a trained network. To emphasize the conditions of the experiment once again, our model did not see any sample from this problem type and still got right three quarters of the questions, which is a fascinating result.

Summary

Neural networks are often considered and used as "black boxes" by the unversed community. We chose the opposite approach and looked inside into the network to try to explain its decision-making process. We successfully tested the generalization ability of the network, resulting in various interesting observations. Some of our experiments ask for further investigation to clarify the behavior of the network.

Most importantly, the chosen model is very effective in image feature extraction, which makes a strong foundation for it to cope with more difficult visual problems, such as solving RPM. We consider it remarkable that the unsupervised MCPT approach achieves better results than certain well-known models or models dedicated to solve RPM-like matrices trained in supervised manner (LSTM, WReN). Although we did not succeed in improving the MCPT method, we realize the significance of unsupervised approaches and therefore encourage other researchers to attempt to refine the technique.

The experiments performed within the thesis point to the potential of neural networks in the field of abstract reasoning. The training on the whole RAVEN dataset produces superior results, suggesting that the networks profits from learning on several problem categories. However, the selection of the training subsets proved to be very significant as certain combinations produce better results than others, even though the size of the training set is the same. The worse results obtained during the separate training might be caused by the significant reduction of the size of the training set, cutting it to 1/7. ResNet-18 is a deep model, requiring extensive datasets, and 6000 training samples may not be sufficient to properly train the network. Nevertheless, *Left-Right* shows a decent performance, answering correctly 68.45% of the questions. Hence, *Left-Right* trained separately outperforms *2x2Grid* and *3x3Grid* trained together with the rest of the problem categories. The combinations of 2 similar subsets (*Left-Right* and *Up-Down*, *Out-InCenter* and *Out-InGrid*) also achieved good results, suggesting a connection between the combined types. The most successful triplet turned out to be the one consisting of the subsets with the highest accuracy demonstrated in the separate training. Also excluding certain problem types from the training set produced interesting results (e.g. excluding *2x2Grid*, *3x3Grid* and *Out-InGrid* produced better results than excluding only *2x2Grid*), worthy of additional research.

To conclude, the average testing accuracy of the supervised approach does not reach the level of humans, but some of the categories achieve even better results. To continue in our research, we propose to design a metrics to quantify the extent of similarity among the problem types based on their features and not only on our visual perception. It could be also interesting to quantify the similarities among the options in the answer set and study the wrong answers of the network. We tried to do this by visually checking the incorrectly answered questions and the corresponding answer sets and noticed that the answers chosen by the network were very similar to the correct ones, likely fooling also human subjects. Hence, the results of the performed experiments suggest that the tested model is showing signs of abstract reasoning.

Bibliography

[Barrett et al., 2018] Barrett, D., Hill, F., Santoro, A., Morcos, A., and Lillicrap, T. (2018). Measuring abstract reasoning in neural networks. In *International Conference on Machine Learning*, pages 511–520. PMLR.

[Binet et al., 1916] Binet, A., Simon, T., and Kite, E. (1916). *The development of intelligence in children (The Binet-Simon Scale)*. Williams & Wilkins Co.

[Braaten and Norman, 2006] Braaten, E. B. and Norman, D. (2006). Intelligence (IQ) Testing. *Pediatrics in Review*, 27(11):403–408.

[Cook, 2012] Cook, S. (2012). *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition.

[Galton, 1891] Galton, F. (1891). *Hereditary Genius*. D. Appleton.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Hunt, 2010] Hunt, E. (2010). *Human intelligence*. Cambridge University Press.

[Joyner et al., 2015] Joyner, D. A., Bedwell, D., Graham, C., Lemmon, W., Martinez, O., and Goel, A. K. (2015). Using Human Computation to Acquire Novel Methods for Addressing Visual Analogy Problems on Intelligence Tests. In *International Conference on Computational Creativity (ICCC)*, pages 23–30.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv–1412*.

[McInnes et al., 2018] McInnes, L., Healy, J., and Melville, J. (2018). UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv–1802*.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang,

E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

[Raven and Court, 1938] Raven, J. C. and Court, J. (1938). *Raven's progressive matrices*. Western Psychological Services Los Angeles, CA.

[Russell and Norvig, 2009] Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition.

[Sternberg, 2000] Sternberg, R. J. (2000). *Handbook of intelligence*. Cambridge University Press.

[Zhang et al., 2019] Zhang, C., Gao, F., Jia, B., Zhu, Y., and Zhu, S.-C. (2019). Raven: A dataset for relational and analogical visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Zheng et al., 2019] Zheng, K., Zha, Z.-J., and Wei, W. (2019). Abstract reasoning with distracting features. *arXiv:1912.00569*.

[Zhuo and Kankanhalli, 2020] Zhuo, T. and Kankanhalli, M. (2020). Solving raven's progressive matrices with neural networks. *arXiv–2002*.

Appendix A

Contents of the Electronic Attachment

The electronic attachment contains the source codes used for the experiments and the results. The file *network.py* defines the model, *util.py* defines the training functions and *main.py* loads the data and runs the program. The files are divided into supervised and unsupervised part, the source codes for the models slightly differ.