Master Thesis
Operations Research and Quantitative Logistics

ERASMUS UNIVERSITY ROTTERDAM
Erasmus School of Economics
December 30, 2021

Applying Convolutional Neural Networks for determining the chromatic number of a graph

Floris Koolen (569183)
Supervisor: dr. Olga Kuryatnikova
Second Assessor: dr. Wilco van den Heuvel

Abstract

This thesis investigates whether deep learning models such as Convolutional Neural Networks could be applied for determining the chromatic number of a graph. To do so, a data set of different types of random graphs is generated with their corresponding chromatic numbers. The graphs are transformed to binary grids using their adjacency matrices such that patterns can be recognised by Convolutional Neural Network models and used for determining their chromatic number. A model that predicts chromatic numbers of graphs of size 100 with an accuracy of 85% is obtained.

Contents

1 Introduction and Literature

1.1 Introduction

In this thesis we want to investigate whether deep learning can be applied for determining the chromatic number of a graph. Can deep learning models be trained to accurate prediction models for graph coloring problems and, if so, what are the complications or limitations of the resulting prediction models?
The problem considered in this thesis is a graph coloring problem. The mathematical formulation of a graph coloring problem is the following:

Let $G(V, E)$ be an undirected graph with a set of nodes V and a set of edges E. A *k-coloring* of graph $G(V, E)$ is an assignment of one of k different colors to each node in V such that no two adjacent nodes have the same color assignment. The chromatic number $\chi(G)$ of a graph G is the smallest number k such that the graph can be k-colored. Finding the number $\chi(G)$ is denoted as a graph coloring problem. Checking if a graph G can be colored with k colors is denoted as a k-colorability problem.

In [1] it is shown that finding the chromatic number $\chi(G)$ of a graph G is NP-hard. For $k \geq 3$ it holds that a k-colorability problem is NP-complete [2]. We provide the mathematical programming formulation of the problem and more details about our solution methods in Section 2.
Graph coloring problems are among the most studied topics within graph theory. The problems and solutions can be used in application of timetabling, computer registration allocation or electronic bandwith allocation [3]. Within modern computer science graph coloring problems can be applied to data mining, image segmentation, clustering, image capturing or networking for example [4]. Because of the many applications the search of efficient algorithms to solve graph coloring problems is important.
In this research we will try to apply deep learning models such as Convolutional Neural Networks on graph coloring problems for predicting the chromatic number of a graph. During this process we will generate a training set and compare multiple structures of the model to increase the performance.
The main research questions in this thesis are:

1. Can deep learning models be used for predicting chromatic numbers of graph coloring problems?

2. Is there a difference in performance of these prediction models for different types of graphs?

3. What are limitations of a Convolutional Neural Network as a prediction model of a chromatic number and possible ways out of them?

Brief descriptions of the corresponding answers that are also presented in the Conclusion are:

1. Yes, when graphs are presented as binary pixel maps to Convolutional Neural Networks the resulting model is able to predict chromatic numbers. We obtained results of around 85-90% accuracy on the final testing set containing different types of graphs and sizes of graphs, given that those types and sizes of graphs were used in the training data.

2. Yes there is a difference in performance for different types of graphs if the model is trained with a training set containing different types of graphs that are equally represented. Our final out-of-sample experiments show that for some types as random graphs as Barabasi-Albert or Watts-Strogatz it has accuracy's of 96% where for Erdos-Renyi random graphs it has only 74% accuracy.

3. The main limitation we found for a Convolutional Neural Network model as a prediction model for chromatic numbers of graphs is that the model did not perform well on the graphs types not used in training, such as graphs normally used in testing graph coloring algorithms where it only has 31.43% accuracy. To improve the performance on these type of graphs, it would be useful to include all online available graph coloring instances in the training sample. However, in our model the graph size was fixed, which was problematic as most of these graphs are too large. Hence we suggest that an approach to deal with varying graph sizes, e.g., pyramid pooling, is incorporated in future models to improve the performance.

   We have also observed that the data sample size could be increased using a data augmentation technique that generates as many as possible isomorphic graphs to those in the sample since the coloring numbers of such graphs are the same while adjacency matrices differ. However, the chromatic number of one of these isomorphic graphs needs still to be obtained before they can be added to the data and it may be computationally hard to find the chromatic number of large graphs of some types of graphs.

The setup of this thesis is as follows. First, we will introduce graph coloring problems and briefly discuss relevant literature. Then we will discuss mathematical formulations that we use for the graph coloring problem and we will introduce neural networks in general. This will be succeeded by the methodology section in which we will discuss how we will approach the application of deep learning on graph coloring problems. In the data generation section we will explain how we generated our data set and discuss some practical implementation issues. After this we will discuss our experiments and results and we will end with a conclusion.

1.2 Literature

In this section we will briefly discuss relevant literature to our research.
Due to its many applications, graph coloring problems are studied for several decades. There are only a few attempts to solve graph coloring problems using Neural Networks and none while using a Convolutional Neural Network.
A survey of recent advances in graph vertex coloring is given in [5]. Different coloring heuristic advances using local search heuristics, evolutionary algorithms or independent set extraction approaches are mentioned. This broad range of approaches illustrates the relevance of graph coloring solutions techniques. Despite that most of these approaches result in usable techniques, it is stated that there don't exist neural network coloring algorithms that are competitive.
In [6] a recurrent single layer Hopfield Neural Network model is used in an algorithm to color graphs. Its performance for some types of graphs is not far behind on other state of the art algorithms at that time and shows the potential of the applications of Neural Networks to graph coloring problems.

A hybrid algorithm that partly uses deep learning is designed for coloring graphs in [7]. They use a Recurrent Neural Network as their deep learning model and obtain competitive performance results. However, we aim at determining the chromatic number of a graph without finding the corresponding coloring while [7] is interested in the coloring instead of the minimum number of colors that is needed.
The approach in [8] uses Graph Neural Networks to predict the chromatic number of a graph. It retrieves promising results (82% prediction accuracy on training data) with which it outperforms other algorithms such as Neurosat, Tabucol and greedy baselines for some distributions of graphs. The research shows the possibilities of Graph Neural Networks and its application to graph coloring problems.
In [9] the artificial intelligence model AlphaGoZero is applied to a graph coloring problem. AlphaGoZero is a combination of a deep Neural Network and a reinforcement learning component. The training, test and validation set is partly randomly generated and derived from an existing graph coloring instance library. The results obtained by AlphaGoZero on graph coloring problems are very promising. The approach is different from this research in the way that they have an existing deep learning model and try to apply it for graph coloring where this research aims at finding a suitable deep learning model and understanding the issues or limitations that come with it.
Boolean binary constraint satisfaction problems are transformed to binary matrices in [10] such that they are similar to binary pixel grids. A Convolutional Neural Network is constructed for classifying the satisfiability of each boolean binary constraint satisfaction problem. The resulting deep learning model yields a $> 99.99\%$ prediction accuracy on random boolean binary constraint satisfaction problems.
We start our Convolutional Neural Networks experiments with model structures based on the structures used in [10]. We can transform our graphs to binary pixel grids and hence have similar input at hand from which the model needs to extract patterns and use them for classification or regression.

2 Problem description

In this section an Integer Linear Programming formulation is given. At the end, additional constraints for removing symmetrical solutions are presented as proposed in [11].

2.1 Integer Linear Programming formulation of a graph coloring problem

A graph coloring can be formulated using the following Integer Linear Programming formulation.
In this formulation, each node in the set of Nodes V is indexed: $V = \{1, 2, ..., n\}$ with a total of n nodes. If each node is given a different color, the resulting coloring of the graph is legal. Therefore, an upper bound on the number of colors that are needed is n, and hence, for the highest possible value m of $\chi(G)$, we have $m = n$. Each possible color in the set of Colors C is indexed: $C = \{1, 2, ..., m\}$.
Let the binary variables X_{ik} denote if node $i \in V$ is colored with color $k \in C$. Let X_{ik} be equal to 1 if color k is used in node i and 0 if otherwise. Let the binary variables W_k denote if color $k \in C$ is used in the solution. If color k is used, W_k is equal to 1 and 0 if

otherwise. We then have the following formulation:

$$\chi(G) = \min \sum_{k=1}^{n} W_k \tag{1}$$

$$\textit{Subject to:} \quad \sum_{k=1}^{n} X_{ik} = 1 \qquad \forall i \in V \tag{2}$$

$$X_{ik} + X_{jk} \leq W_k \qquad \forall \{i,j\} \in E, \forall k \in C \tag{3}$$

$$W_k \in \{0,1\}^n \tag{4}$$

$$X_{ik} \in \{0,1\}^{n \times n} \tag{5}$$

The objective of finding the lowest number of colors is stated by (1). The set (2) contains the constraints that ensure that only 1 color can be assigned to each node. The adjacency constraints leading to legal colorings of adjacent nodes are given in (3).

2.2 Extension of the formulation: eliminating symmetrical solutions

We are interested in solving graphs coloring problems as fast as possible with a solver because we need to generate a data set consisting of graphs and their colorability if we want to use deep learning. The formulation given in section 2.1 including (1), (2) and (3) embodies a complete graph coloring problem. Solutions using different colors but with an identical coloring structure could possibly slow down the cutting plane algorithm in a solver as feasible solutions of an integer programming formulation also suffer from that symmetry drawback [11]. Hence, as is done in [11], the implementation of this formulation in a solver could be improved by the extension of adding constraints that reduce the number of symmetrical solutions.

We consider three sets of additional constraints. The first set of additional constraints is constructed to remove symmetrical coloring solutions using colors with a label higher than χ. This means that for a coloring using m colors, only the colors with a label up to m may be used. The corresponding set of additional constraints is:

$$W_k \leq \sum_{i \in V} X_{ik} \qquad \forall k \in C \tag{6}$$

$$W_k \geq W_{k+1} \qquad \forall k \in \{1, 2..., n-1\}. \tag{7}$$

The second set of additional constraints is created to eliminate some permutations of symmetrical coloring solutions of the first m colorings:

$$\sum_{i=1}^{n} X_{ik} \geq \sum_{i=1}^{n} X_{ik+1} \qquad \forall k \in \{1, 2, ..., n-1\}. \tag{8}$$

The third and last set of additional constraints corresponds to the permutations of colors with the same cardinality in symmetrical coloring solutions. We remove these symmetrical coloring solutions with the following constraints:

$$X_{ik} = 0 \qquad \forall k \geq i+1 \tag{9}$$

$$X_{ik} \leq \sum_{j=k-1}^{i-1} X_{jk-1} \qquad \forall i \in V \setminus \{1\}, \forall k \in \{2, 3, ..., i-1\}. \tag{10}$$

After we include these three sets of additional constraints in the formulation, the performance of the branch and bound algorithm in the solver could be faster. However, including these additional sets leads to a large number of extra constraints. Therefore, a trade off between the speed gained by elimination of symmetrical solutions and the speed lost for including more constraints needs to be made. This trade-off is made when the size of the graphs that need to solved is known. If the graphs that need to be solved exceed a certain size, the additional constraints need to be dropped. However, in this thesis, this is not the case and, therefore, the additional constraints are used.

2.3 Introduction on Neural Networks

In this section, firstly a short introduction on deep learning and Neural Networks is given based on [12, Section 11]. Then we give a description of a Convolutional Neural Network and its structure that we used for our research based on [13]. Finally we discuss possible implementation issues with Neural Networks and how they can be solved.

2.3.1 Deep Learning

Deep learning is a family of machine learning models that uses Neural Networks consisting of multiple layers. A Neural Network is a circuit of artificial neurons, based on the working of the human brain. They are used in machine learning to construct non linear functions that can be used for regression or classification problems. An example of a Neural Network with a single hidden layer is given in figure 1. The vectors $\beta^{(1)}$ and $\beta^{(2)}$ correspond to the unknown weight parameters. Node z_j in the hidden layer has has value $h(\sum_{i=1}^{p} \beta_{ji}^{(1)} X_i + \beta_{j0}^{(1)})$. The function h is an activation function, for example a sigmoid $h(x) = \frac{1}{1+e^{-x}}$ or a rectified linear unit (ReLu) $h(x) = \max(0, x)$. The resulting values z_j determine similarly the output variables $\hat{y}$. When the Neural Network is used for classification, the softmax function

$$\sigma(x_k) = \frac{e^{x_k}}{\sum_{j=1}^{K} e^{x_j}}$$

is a popular choice for the output unit activation function σ.

The unknown weight parameters β in the Neural Network must be set to values such that the Neural Network fits the training data well. Therefore, a loss function $R(\beta)$ must be defined and minimized. A default loss function that is used when a Neural Network is used for classification is the cross-entropy function:

$$R(\beta) := -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log(\hat{y}_{ik}(X, \beta))$$

with $\hat{y}_{ik}$ as the estimated outcome k for the given observation i. The generic approach for minimizing the loss function is by gradient descent and is called back-propagation. Back-propagation starts at the last layer of the feedforward Neural Network and calculates the gradient of the loss function backward using the chain rule. In general a global minimum is not always preferred because if a global minimum is obtained, there could be overfitting of the training data [12].
The previously discussed components of a Neural Network can be extended to Neural Networks with multiple hidden layers. An example of a Neural Network with multiple hidden layers is illustrated in figure 2.

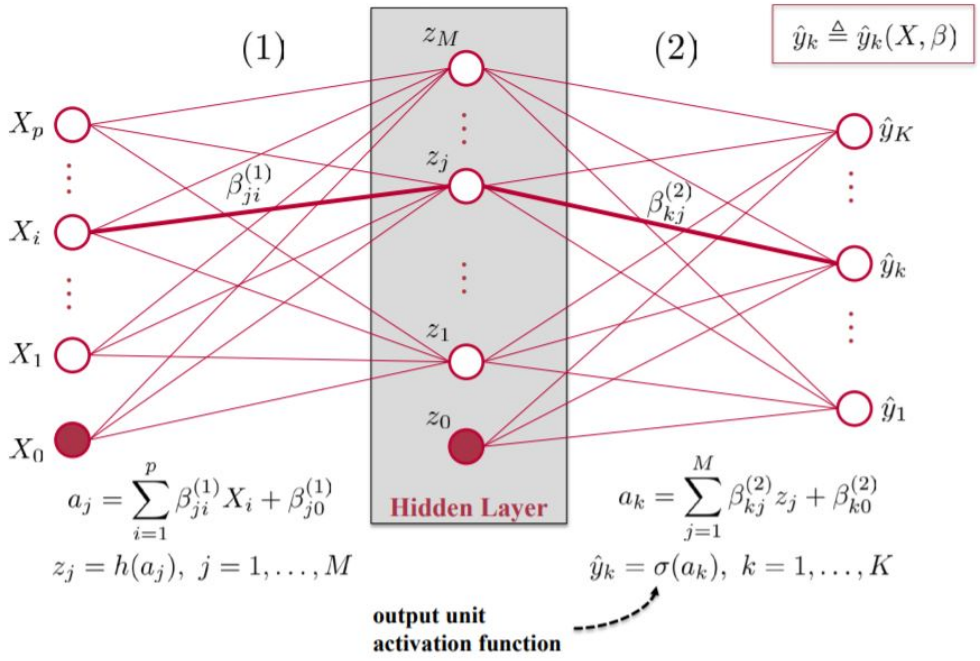


Figure 1: Example of a single hidden layer Neural Network with input variables X, output variables $\hat{y}(X, \beta)$ (depending on X and β), hidden layer variables z, weights $\beta^{(1)}$ and $\beta^{(2)}$. The activation functions are h and σ.

2.3.2 Convolutional Neural Networks

A specific type of Neural Networks with multiple hidden layers are Convolutional Neural Networks. Convolutional Neural Networks are designed to reduce the number of connections and parameters with respect to fully connected Neural Networks without losing their strength. Convolutional Neural Networks have three types of different layers: Convolutional layers, pooling layers and fully connected layers.

The convolutional layer connects each input node to hidden layer nodes that are only in their region of the input. So if a Neural Network is for example dealing with a picture, the input to the Neural Network will be pixels. The Convolutional layer connects each source pixel only to hidden layer nodes of pixels close to the pixel instead of a connection to all pixels. This is illustrated in figure 3. Another structural characteristic of the Convolutional layer is that the weight parameters of local connections are fixed. Fixing the weights of local connections results in an opportunity to detect and recognize features regardless of their positions in the image. The reduction of connection and fixing of weights reduces the amount of connections and parameters.

The pooling layers reduce the dimensional size by combining a group of connected nodes into a new one by taking for example the maximum or average of the original group of connected nodes. If a Neural Network has pixel data as input, it can be seen as that the pooling layers reduce the resolution of the data.

The last type of layer is the fully connected layer. These layers are general fully connected Neural Network layers as explained in section 2.3.1.

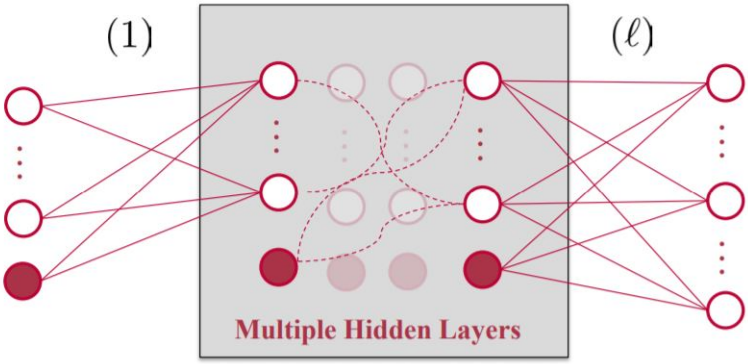


Figure 2: Example of a Neural Network with multiple hidden layers

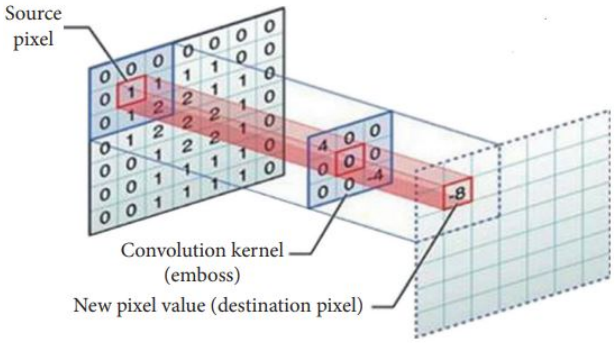


Figure 3: Illustration of a convolutional layer mapping pixels ([13])

3 Methodology

One of the main questions in our research is to investigate if it is possible to apply a Neural Network to the graph coloring problem. This means that we want to construct a Neural Network that is able to solve graph coloring problems as good as possible. We try to achieve this goal by experimenting with different Neural Network structures. This section consists of three subsections: graphs, input representation and Neural Networks respectively. In the first subsection we will discuss how we will obtain labels for all graphs in our data. In the second subsection we will explain how the data will be represented so it can be fed to our Neural Networks. In the third and last subsection we will discuss different structures of Neural Networks that we will experiment with and some implementation improvements.

3.1 Finding colorability bounds for graph data

For each random graph G that is created, a corresponding legal colorability number k must be found to complete the instance. The optimal colorability number to use would be $\chi(G)$. However, as graph coloring problems are NP-hard, it might not be practically possible to find $\chi(G)$, we therefore search for the tightest possible bounds on $\chi(G)$ that we can use for labeling a graph in our data.

3.1.1 Brooks' theorem

A general upper bound on the chromatic $\chi(G)$ of an undirected graph G was found by Brooks in [14]:

Theorem 3.1 (Brooks' theorem [14])**.** *Let G be an undirected graph. We than have that the chromatic $\chi(G)$ has an upper bound:*

1. *$\Delta(G)$ if G is not an odd cycle or a complete graph*
2. *$\Delta(G) + 1$ if G is an odd cycle or a complete graph.*

The notation $\Delta(G)$ denotes the maximum degree (maximum number of connected nodes to a node) of a graph G. The time complexity of finding the maximum degree of a graph is $\mathcal{O}(n^2)$ and hence, polynomial.
For our three types of randomly constructed graphs we can easily check if they are an odd cycle or a complete graph and apply Brooks' theorem. Sometimes, a check is even unnecessary depending on the model and parameters. For example, if a random graph G is constructed with the Watts-Strogatz model with $4 \leq K \leq n-2$ we know that it is not an odd cycle or a complete graph.

3.1.2 Solver bounds

Another set of practical bounds that could be useful for constructing an instance are the bounds found by an Integer Linear Programming solver before the branch and bound algorithm is fully terminated. If tight lower or upper bounds can be found in reasonable time, these bounds can also be used for creating an instance. The speed of the solver can potentially be increased by using the known upper bound (found by Brooks' Theorem for example) as a maximum for the number of decision variables that are created.
As we have that the Linear Programming solution is always as least as good as its corresponding Integer Linear Programming solution, we have that a lower bound can also be found by using the Linear programming relaxation of the solution. This means that we solve the formulation represented in section 2.1 by replacing the constraints 4 and 5 that ensures binary variables with:

$$W_k \in [0,1] \qquad \forall k \in C \tag{11}$$

$$X_{ik} \in [0,1] \qquad \forall i \in V, \forall k \in C. \tag{12}$$

The resulting new formulation is a Linear Programming formulation and is easier to solve then the first formulation. Assume that $\chi_{LP}(G)$ is the linear programming solution of a graph G, we then have that $\lceil \chi_{LP}(G) \rceil$ is a lower bound on the colorability of G.

3.1.3 Procedure for finding the best bound for a random graph

We need to combine the different types of colorability bounds in a procedure for finding the best lower and upper bounds. We therefore propose the following procedure:

Algorithm 1: Procedure for selecting bounds

Create a random graph $G(V, E)$ with a random graph construction model;
Let L be the resulting lower bound and initialize $L = 2$;
Let U be the resulting upper bound and initialize $U = n$;
Solve the relaxed graph coloring problem of G with an LP solver with resulting optimal value χ_{LP} ;
if $\lceil \chi_{LP}(G) \rceil \geq L$ **then**
  Set $L = \lceil \chi_{LP}(G) \rceil$*;*
end
if *G is an odd Cycle or a complete graph* **then**
  Set $U = \Delta(G) + 1$*;*
else
  Set $U = \Delta(G)$*;*
end
Try to solve the graph coloring problem of G with an ILP solver with a timelimit t_{max} and with set of colors C with $|C| = U$.
if *ILP solver finds $\chi(G)$ within timelimit* **then**
  Set $L = U = \chi(G)$ and terminate procedure;
else
  Let L_{ILP} be the best lower bound found for $\chi(G)$ by the ILP solver
  Let U_{ILP} be the best upper bound found for $\chi(G)$ by the ILP solver
  Set $L = \max(L_{ILP}, L)$ and $U = \min(U_{ILP}, U)$;
end

Algorithm 1 provides tight lower and upper bounds for all graphs in our data set so that each data point has labels and could potentially be used even when .

3.2 Input representation

If we want to feed graphs to a Neural Network we should transform them into data structures that are accepted by a Neural Network. We want to represent each graph by an image-like adjacency matrix because Neural Networks are known to perform well on image-like data. We therefore choose the adjacency matrix consisting of binary entries as our graph representation structure. A graph with n nodes can be represented by an $n \times n$ adjacency matrix in the following way.
Each node i in the graph is linked to row i and column i. If an edge between nodes i and j exists in the graph, the matrix entries with index (i, j) and (j, i) are equal to 1. If there exists no edge between nodes i and j, the matrix entries with index (i, j) and (j, i) are equal to 0. Note that the resulting matrix contains only binary values, is symmetrical and has a diagonal consisting of zeros.

For example, the adjacency matrix of the graph illustrated in figure 4 is: $\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$.

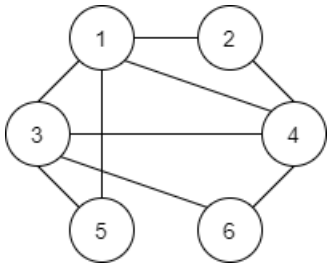


Figure 4: Example of an undirected graph

3.3 Neural Networks

In this subsection different Neural Network structures and implementation issues and improvements are discussed.

3.3.1 Fully Connected Neural Networks

The first, most basic, structures that can be considered are fully connected Neural Networks with one or multiple hidden layers. However, when the size of the fully connected layers or the number of fully connected layers increases, the number of parameters of the model increases drastically. The models complexity and its ability to recognise patterns and use them for predicting the chromatic number of a graph is a complex problem that could (if possible at all) require a practically impossible size of a fully connected Neural Network. For this reason, we want to extend our experiments to a different type of Neural Networks that are build with an architecture for pattern recognition.

3.3.2 Convolutional Neural Networks

In section 2.3.2 we explained the class of Convolutional Neural Networks with a structure that is highly compatible for pattern recognition. As we have our data represented in the form of adjacency matrices that can be seen as black and white pictures, Convolutional Neural Networks may be useful to extract significant information in the form of patterns from the data. These extracted information may then be used to determine the presence and/or absence of certain patterns to predict chromatic numbers.

3.3.3 Ordinal classification

When a graph coloring problem is approached as a classification problem, the classes are not entirely independent as the classes are clearly ordered by number of colorability. The problem is therefore called ordinal classification.

An efficient way to deal with ordinal classification is proposed in [15] and works as follows:
Suppose the set of labels to be predicted is $K = \{1, 2, 3, 4\}$ where we have the ordinal relations that class $i - 1 < i$ for all i. We define $k_{max} = \max_{k \in K} k$. Each label in K is converted to a k_{max} dimensional binary vector:

$$1 \longrightarrow (0,1,1,1)$$
$$2 \longrightarrow (0,0,1,1)$$
$$3 \longrightarrow (0,0,0,1)$$
$$4 \longrightarrow (0,0,0,0).$$

The output layer of the Neural Network must be a layer consisting of k_{max} output nodes. The activation function used at the output nodes should be a standard sigmoid-function. After a prediction $\hat{y}$ is made, the corresponding predicted class $\hat{k}$ can be obtained by defining $\hat{y}_i$ as the first entry $\hat{y}_j$ with $j \in [1, 2, ..., k_{max}]$ for which it holds that $\hat{y}_j > 0.5$. If there exists no $\hat{y}_i$ then the prediction is equal to the maximum label k_{max}. If $\hat{y}_i$ exists then we have that the prediction is:

$$\hat{k} = i - 1$$

A problem that may arise when the graph coloring problem is treated as a classification problem is that there may be 'missing' labels in the data set that can never be predicted by the model. A 'missing' label can be explained as a value for the chromatic number of a graph, for example 6, that is not included in the data while other surrounding values (5 and 7) are included.
This problem could be solved practically by transforming some labels next to missing labels to ranges that include the missing values. For example, if the label 5 is present in the data but the label 6 is not, then the label 5 will become the label 5-6. This approach, however, is not ideal as the corresponding models predictions are less meaningful for these range labels.

Another way to solve the issue of ordinal classification is to model the graph coloring problem not as a classification problem where each graph has one out of a finite set of labels corresponding to its chromatic number, but by modeling it as a regression problem instead. If the Neural Network is a regression model instead of a classification model, the issue of order between classes is directly taken care of by the structure of the regression model. But there is a new issue that arises in the context of graph coloring problems: the chromatic values that should be predicted by the regression model are all integer while a regression model predicts continuous values. This issue also leads to the fact that in a regression model the accuracy of rightly or wrongly predicted chromatic values can not be directly evaluated as the performance is given by the root mean squared error or an equivalent numerical error size statistic. This can in its turn be solved by rounding the prediction of the regression model and evaluate prediction accuracy again.

3.3.4 Optimizing architecture

We need to combine our knowledge about graph coloring problems and Convolutional Neural Networks with the results of our experiments to find the optimal structure of the Convolutional Neural Network for graph coloring problems. As a starting point for our experiments, we took a similar structure to [10] for the Convolutional Neural Network model as they consider Constraint Satisfaction Problems that are similar to Graph coloring problems. Our starting structure for the classification model is presented in Figure 5.

The first square in the Figure is the input (size 100x100), followed by three convolutional (with 8, 16 and 32 filters of size 3x3) and pooling layers. The last three layers are the two fully connected layers and the output layer. We then experiment with small deviations from this structure while we keeping the rest of the structure equal. In our experiments we will try variations in the number of convolutional layers and pooling layers, filter sizes of convolutional layers, number of filters, number of fully connected layers and sizes of fully connected layers.

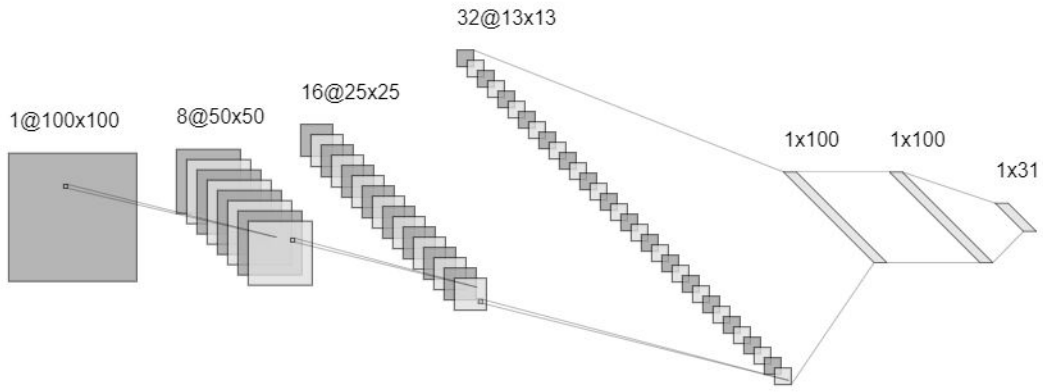


Figure 5: Starting structure of our Convolutional Neural Network

We expect to observe an optimal number of convolutional and pooling layers and an optimal filter size for our data set as this indicates that the patterns in adjacency matrices that are necessary for predicting the corresponding chromatic number have a level of complexity that can be recognised by the optimal number of convolutional layers. Increasing the number of convolutional layers could possibly lead to a model that is able to recognise patterns with a complexity level that is too high for graph coloring problems and can lead to worse out-of-sample performance.

3.4 Using unlabeled data

One of the issues that arises when we create our data set consisting of graphs is that we might not solve the colorability problem for each graph. However, for these unsolved graphs, we have found a lower and upper bound on their chromatic number with Algorithm 1. It would be nice to investigate whether the inclusion of these 'unlabeled' graphs in the data set results in an increasing performance of the prediction model. Not using the 'unlabeled' data points seems as a waste since they are already generated and could potentially be a large part of the data set as solving the graph coloring problem is NP-hard. The 'unlabeled' data points will therefore consist mainly of graphs that are relatively large and difficult to solve. Including information of difficult graphs into the training data set leads to enriching the information that is fed to the Neural Network and hence increases its performance.

To include the 'unlabeled' data points we can use the regression-based Neural Network structure and change the output structure of the regression Neural Network with one output node to a structure with two output nodes. Each data point (labeled or unlabeled)

will have two output values: a lower and an upper bound. If algorithm 1 however finds the exact chromatic number then in this case both output values will be set to the chromatic number.

3.5 Variation of input sizes

A technical issue for standard Convolutional Neural Networks as prediction models is that it requires fixed input sizes. This means in the context of graph coloring problems that all training and test graphs should have the same number of nodes. In this subsection we first discuss image cropping and warping. These are the most typical techniques for data augmentation for Convolutional Neural Networks, however, they turn out to be unsuitable for graph coloring problems.

3.5.1 Image cropping and warping

Basic methods to fit images of arbitrary sizes to Convolutional Neural Networks are image cropping and image warping. An illustration of these procedures and their place within a Convolutional Neural Network are shown in figure 6.

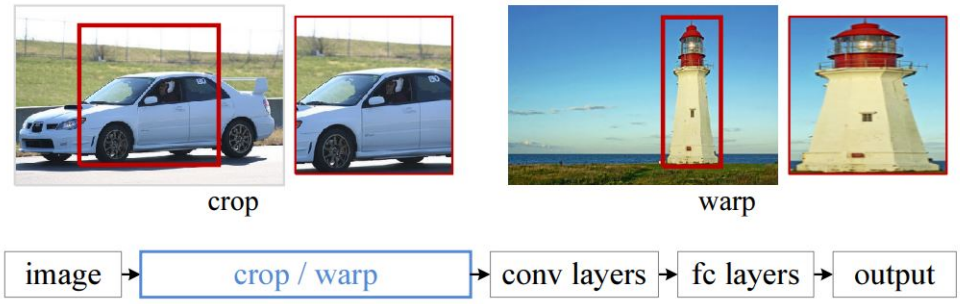


Figure 6: Cropping and warping ([16])

Image cropping is the removal of unwanted outer areas of the image. Cropping is not suitable for adjacency matrices of graphs as the entire adjacency matrix contains potentially import information on the chromatic number. Image warping is the resizing of an image to a new format but a disadvantage of image warping is that it could sometimes result in unwanted geometric distortions when the image is resized. Image warping is a technique that is constructed for resizing images consisting of pixels. If warping would be applied to the adjacency matrix of a graph, the resulting resized adjacency matrix could possibly not contain any of the old graphs structure. So it's original chromatic number could become totally undetectable from the resulted warped adjacency matrix. And, therefore, we do not use either of these two standard data augmentation techniques for Convolutional Neural Networks.

3.5.2 Zero padding

These standard image resizing techniques are not preferred for our adjacency matrices as described above. We therefore search for resizing methods of adjacency matrices that increase the size of an adjacency matrix (or graph) but don't affect the chromatic number

of a graph. The first method that probably comes to mind is to extend the adjacency matrix with rows and columns of zeros at the boundaries. We will call this method zero padding. The resizing with zero padding can then be interpreted as adding new unconnected nodes to the graph. It is easy to see that adding unconnected nodes to a graph never affects the chromatic number of the original graph as each unconnected node can be given the same color.
We apply the zero padding technique for increasing the size of adjacency matrices in the following way. Assume that we have a data set consisting of graphs of different sizes with their corresponding chromatic numbers. Let N be the size (number of nodes) of the largest graph in the data set. For each adjacency matrix of a graph that has a smaller size than N, we apply zero padding at the end of the adjacency matrix to resize the graph up to size N. Our Convolutional Neural Network is constructed for adjacency matrices of size $N \times N$.

4 Data generation

A set of instances is required for training and testing a Neural Network. Each instance consists of a graph and integer numbers L and U corresponding to the lower and upper bound respectively. If $L = U$ then we know the exact chromatic number of the graph: $\chi = L = U$. The larger the size of the set of test and training instances is, the more training can be performed on the Neural Network. By creating random instances for the training and test set, we are free to decide the size of the set. In this section we will discuss firstly the different types of random graphs that are used for the research, secondly data augmentation for increasing the size of the data set. After this we will discuss the different data sets that we use in our experiments and their specifics. Then, we will mention how we performed K-fold cross validation in our experiments to receive more robust results and finally we will discuss some performance measure issues that could occur during the experiments and how we deal with them.

4.1 Generating random graphs

We consider three types of random graphs for our data set: random graphs constructed by the Erdős–Rényi model, random graphs constructed by the Watts-Strogatz model and random graphs constructed by the Barabási–Albert model.

4.1.1 Erdős Rényi

The Erdős–Rényi model constructs random graphs as follows: a parameter for the desired number of nodes $n \in \mathbb{N}$ and a probability parameter $p \in [0, 1]$ are given as input for the model. Then n nodes are constructed and for each possible pair of nodes, an edge is constructed with probability p.

4.1.2 Watts Strogatz

In [17] Watts and Strogatz found that many real life graphs have a connection topology that lies between completely regular and completely random. These graphs are

highly clustered and have small characteristic path lengths and are therefore referred to as "Small-world" graphs. They therefore created a model for constructing random "Small-world" graphs that also have these properties and have a connection topology that lies between completely regular and completely random as is illustrated in figure 7.
The Watts-Strogatz model constructs random graphs in the following way: the desired number of nodes $n \in \mathbb{N}$, the mean even degree K and the probability parameter $p \in [0, 1]$ are given as input to the model. Construct a lattice with n nodes where each node is connected to its K neighbour nodes. Then, for every node all its connected edges are adjusted with probability p. If an edge from node i is getting adjusted, a random node is picked with a uniform probability until a node is picked that is not equal to i or does not have an edge connected to i.

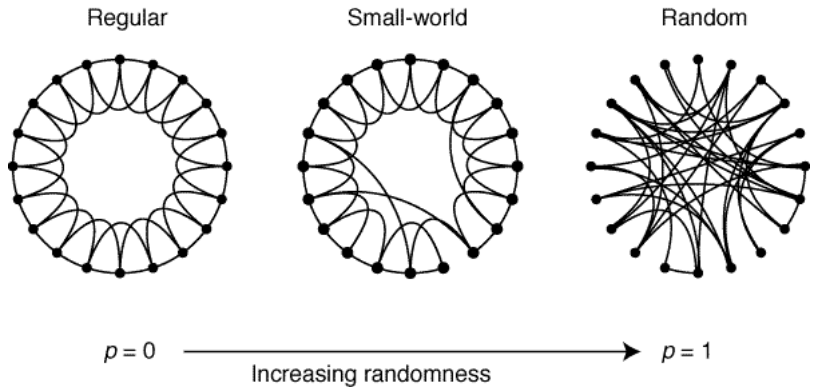


Figure 7: Watts-Strogatz model for interpolating between a regular ring lattice and a random network as is done in [17].

4.1.3 Barabási–Albert

In many real life networks there exists an inhomogeneous distribution of the degrees of nodes. The Erdős–Rényi model and the Watts-Strogatz model do not construct graphs with inhomogeneous degree distributions. We therefore consider a third type of model for creating random graphs.
A third type of random graphs we consider for creating instances are graphs constructed with the Barabási–Albert model ([18]). The idea behind this model is that in real life networks, the degrees of nodes are often inhomogeneous distributed. A shortcoming of the Barabási–Albert model with respect to real life networks is that it constructs graphs lacking the high level of clustering found in real life graphs and graphs constructed by the Watts-Strogatz model.
The Barabási–Albert model starts with a connected graph with m_0 nodes where each node has at least one edge. Then, at each time step, a node with m connecting edges to the graph is added, with $m \leq m_0$. The probability $p(d_i)$ that an edge of the new node connects to node i depends on the degree d_i of node i: $p(d_i) = \frac{d_i}{\sum_j d_j}$.

4.2 Data augmentation

To improve the performance and robustness of a Neural Network, we consider an increase in size of the data set. However, our method of generating data is time-expensive as it

could take up to a few minutes to solve the colorability problem of a single randomly generated graph. We therefore consider data augmentation techniques to generate new data with less time-expensive methods while we retain the same data-quality as with our previous data generating method.

4.2.1 Data Augmentation general

The main assumption for considering data augmentation techniques is that more information can be obtained from the existing data set through augmentation. As is explained in [19], when deep learning is applied for image classification, there are two main data augmentation techniques: data warping and oversampling. Data warping is a technique that focuses on changing a data point while retaining its label. For image classification this could mean that an image is rotated or a color filter is applied to the image. Oversampling augmentations are newly created data points by for instance combining two old data points or performing feature space augmentations. Data warping and oversampling can also be combined to augment a data set.

4.2.2 Data augmentation in the light of graph coloring

After a data augmentation technique is used to construct a new data point, the corresponding label should also be known. For graph coloring problems it is therefore not possible to apply oversampling data augmentation technique of combining two old data points to construct a new point. The exact chromatic number of the label can not be easily decided from the old two data points.
However, data warping is a technique for which graph coloring instances are highly suitable. To show this, we use the following definition for isomorphism between graphs:

Definition 4.1 (Graph Isomorphism ([20]))**.** *Two graphs G_1 and G_2 are isomorphic if there exists a matching between their vertices so that two vertices are connected by an edge in G_1 if and only if corresponding vertices are connected by an edge in G_2.*

We first note that two graphs that are isomorphic can be seen as two equal graphs with a different labeling of the nodes. It can then be expected that as the labeling of nodes in a graph does not influence properties such as the chromatic number of the graph. This is proposed in the following theorem:

Theorem 4.1 (Chromatic number of isomorphic graphs)**.** *If two graphs G_1 and G_2 are isomorphic, then their chromatic numbers $\chi(G_1)$ and $\chi(G_2)$ are equal: $\chi(G) = \chi(H)$.*

Proof. Suppose that the graphs $G_1 = (V_1, E_2)$ and $G_2 = (V_2, E_2)$ are isomorphic. From the definition of graph isomorphism we know that there exists a bijection $f : V_1 \longrightarrow V_2$ such that $(v, w) \in E_1$ if and only if $(f(v), f(w)) \in E_2$.
Let $c_1 : V_1 \longrightarrow \{1, 2, ..., n\}$ be a coloring of G_1 with n colors. We then have that the coloring $c_1 \circ f^{-1} : V_2 \longrightarrow n$ is a coloring of V_2. This indicates that all colorings of V_1 can be transformed to colorings of V_2.
Let $c_2 : V_2 \longrightarrow \{1, 2, ..., m\}$ be a coloring of G_2 with m colors. We then have that the coloring $c_2 \circ f : V_1 \longrightarrow m$ is a coloring of V_1. This indicates that all colorings of V_2 can be transformed to colorings of V_1.
As we have that the transformations between colorings of V_1 and V_2 are each other's inverse as f and f^{-1} are each other's inverse we have that there exists a bijection of colorings of G_1 and G_2 with a given number of colors. This means that $\chi(G_1) = \chi(G_2)$. □

Theorem **??** can be used to augment graph coloring data in the following way: if we can construct a new graph H that is isomorphic to an existing graph G in our data, the new graph H has the same label as graph G. A method for constructing an isomorphic graph H for a graph G is:
Assume we have a given random graph G of size n with chromatic number $\chi(G)$ and with node label set $V = \{1, 2, ..., n\}$. We can construct a new graph H with the same number of nodes and edges as G. We randomly re-assign each label in V to a new label in V. For each node in G we add a node to H. For each edge in G we add an edge to H such that the nodes connected by the edge in G correspond to the newly assigned labels of these nodes in H. The resulting graph H is isomorphic to the graph G and has, by using Theorem **??**, the same chromatic number as G.
If we apply this technique of data augmentation we can easily create a lot of new data with corresponding labels. All these new graphs could have completely different adjacency matrices compared to their original graphs. We could therefore potentially add new adjacency matrix patterns to the data that could hypothetically indicate the chromatic number of a corresponding graph.

4.3 Data sets

The number of nodes of instances of difficult graphs on which most coloring algorithms are tested in the literature are between 450 and 4000. However, for the largest of these graphs the chromatic numbers are not found but only upper bounds are found [5]. We tested for different types of random graphs considered in this thesis for which size they could 'easily' (in +- 10 seconds) be solved by a solver (Gurobi). We found that Erdos-Renyi graphs can be easily solved up to size 35, Watts-Strogatz graphs up to size 70 and Barabasi-Albert also up to size 70. These small instance sizes illustrate the difficulty of graph coloring problems and hence relevance of heuristics. We created a few different data sets for our experiments: Data set 1, 2, 2.1, 3, 3.1, 3.2 and some different final test sets 4.1, 4.2, 4.3 and 4.4.
Data set 1 has size 28636 and consists of graphs of size 50. For each graph in Data set 1 the exact chromatic number is found. The different types of random graphs are evenly represented in Data set 1.
Data set 2 includes Data set 1 but also contains 38328 random graphs of sizes 50 to 100 for which the exact label is known. However, in Data set 2 there are almost no Erdos-Renyi random graphs as their exact number could not be easily found. Data set 2.1 is Data set 2 minus Data set 1.
Data set 3.1 consists of the graphs of Data set 1 but with all its labels doubled such that each graph has an upper bound equal to its lower bound. Data set 3.2 consists of 10002 random graphs of sizes between 50 and 100 with bounds where the exact coloring number is not always found. Data set 3 is the union of Data set 3.1 and Data set 3.2.
Data set 4.1 to 4.4 are data sets to test the final models on. They therefore consists each of specific types of graphs. Data set 4.1 consists of 35 graphs up to size 100 that we found online. These graphs have different structures and are used for determining the effectiveness of graph coloring problem algorithms. Data set 4.2 and Data set 4.3 consist each of 1000 random 50-100 size graphs of Watts-Strogatz and Barabasi-Albert types, respectively. Data set 4.4 consists of 50 Erdos-renyi graphs of size 50.

4.4 K-fold cross validation

To evaluate the performance of a Neural Network for optimizing parameter settings and model structures, a test data set is required next to the training data set. This, however, may cause problems as the test data set could be imbalanced compared to the training data set. We also have limited data available as our data generation method is computationally heavy. We therefore use the method of k-fold cross validation to overcome these problems and obtain more general performances of our Neural Network models.
The method of k-fold cross validation works as follows. At the beginning the training data set S that is used is randomly shuffled. Then, it is split in k disjoint sets S_i of size $\frac{1}{k} * |S|$ such that the union of these sets is equal to the training data set at the beginning: $\cup_{i=1}^{k} S_i = S$. After these splits are done, the model is evaluated k times where in each iteration i with $i \in \{1, ..., k\}$ the training data set is equal to $S \setminus S_i$ and the test data set is equal to S_i. Each iteration i results in an accuracy δ_i and we compute the mean accuracy $\bar{\delta}$ as $\bar{\delta} = \frac{\sum_{i=1}^{k} \delta_i}{k}$.

4.5 Performance measures

For classification models the accuracy percentage of the model can be calculated relatively straightforward. The class corresponding to the output node with the highest value is chosen as the predicted class and the accuracy is calculated by dividing the number of all rightly predicted classes of data points by the number of all data points in the test set. This is however not possible for ordinal classification models and regression models.
Ordinal classification models return a vector with values in the interval $[0, 1]$ as predictions. We translate these resulting vectors to an integer value back in the following way: let $\hat{y_k}$ be the k_{th} entry of the resulting prediction vector $\hat{y}$ for which it holds that $\hat{y_k} > 0.5$, we then have that the chromatic number corresponding to prediction $\hat{y}$ is equal to $k - 1$.

Regression models predict a continuous number as the chromatic number of a graph while we have only integer values as true labels in our data test. In our regression models with one output node we solve this in the following way. After our regression model predicts the chromatic number we first round that number to the closest integer. Then we compute the accuracy of all rounded predictions against the true labels that are known. With this trick we are able to somehow compare classification and regression models with each other. This rounding method also provides regression models that can be used in practice to predict chromatic numbers as their outcome is then always integer.

Another performance measure that we take into account for evaluating a model is the interval of the accuracy's of the k-Fold cross validation. If this interval is relatively small, the model accuracy is more stable regardless of how test and training data are split. If the accuracy interval is relatively large, the model accuracy is more sensitive to different splits in test and training data.

In Experiment 6 we will test a regression model with two output nodes for predicting a lower and an upper bound of a graph. During this experiment we will test two different rounding methods: the 'standard' rounding method to the nearest integer and a rounding method that rounds to the outside integer values. The second rounding method to the outside integers rounds the value of the predicted lower bound by applying the floor function

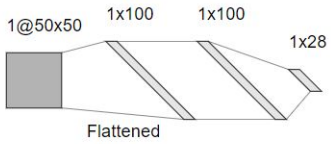
to it and it rounds the predicted value of the upper bound by applying the ceiling function to it. During this experiment the performance measurement is also different because the model predicts bounds instead of the chromatic number. A lower (or upper)bound is said to be predicted correctly if the bound is smaller (higher) or equal to the known bound.

5 Results

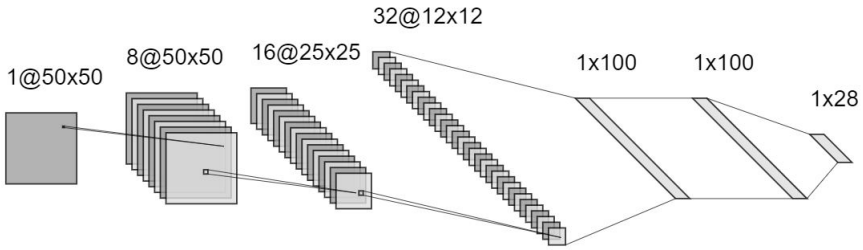
To compare different methods and Neural Network structures we performed several different comparative experiments. After each experiment, the results are evaluated and, if a model improvement is found, the basic model for later experiments is adjusted.
Experiment 1 is performed for comparing a Fully Connected Neural Network to a Convolutional Neural Network. In Experiments 2 and 3 the two methods of incorporating ordinal classification as explained in section 3.3.3 are tested and evaluated. Experiments 4 are for deciding on the optimal structure of the Convolutional Neural Network and are split into 3 sub-experiments: Experiment 4.1 for optimizing the structure of the convolutional layer part, Experiment 4.2 for optimizing the size and number of filters in the convolutional layers and Experiment 4.3 to decide on the optimal structure of the fully connected layer part of a Convolutional Neural Network. Experiment 5 aims at investigating if a model can be adjusted to fit different input sizes and in Experiment 6 we try to use unlabeled data. Finally, in Experiment 7 we test our best models to a range of final unseen test sets.
Experiments 1, 2, 3, 4.1, 4.2 and 4.3 are all performed with a data set consisting of graphs of size 50 for which the exact chromatic number is known. For each model that is tested a 5-fold cross validation is performed to evaluate the model. The number of training episodes for each split was 20.
The experiments were performed on a Intel(R) Core(TM) i7-9750H CPU using Python 3.8.8 in the environment Spyder 5.0.0. The solver that is used for generating the data sets is Gurobi Optimizer version 9.1.0.

5.1 Experiment 1: a Fully Connected Neural Network vs a Convolutional Neural network

In our first experiments we compared the performance of a Fully Connected Neural Network (1A) to the performance of a Convolutional Neural Network (1B). The architectures of both networks are presented in Figure 8.



(a) Structure 1A



(b) Structure 1B

Figure 8: Neural Network structures for Experiment 1

The results of Experiment 1 are presented in Table 1. We observe that 1B has a much higher mean accuracy and a lower accuracy variance. The higher mean accuracy means that 1B does a better job on prediction on average than 1A. The smaller accuracy interval is also preferable: it indicates that model 1B is more consistent over the data in its predictions than model 1A. We conclude that, as expected, a Convolutional Neural Network outperforms a Fully Connected Neural Network with our experiment settings.

	Number of parameters	Mean accuracy	Accuracy interval
Structure 1A	263 028	63.78	[56.22 , 74.90]
Structure 1B	134 116	73.20	[65.22 , 75.40]

Table 1: Results Experiment 1

5.2 Experiment 2: testing classification against ordinal classification

An attempt to improve our model of a Convolutional Neural Network found in section 5.1 can be made by implementing ordinal labels. These ordinal labels are implemented as is explained in the beginning of section 3.3.3: all data labels are transformed to a binary vector. The architecture of the model that makes use of ordinal labels is presented in Figure 9. The structure of the model is almost completely identical to that of model 1B, the model that it is compared to, and differs only in the size of its output layer.

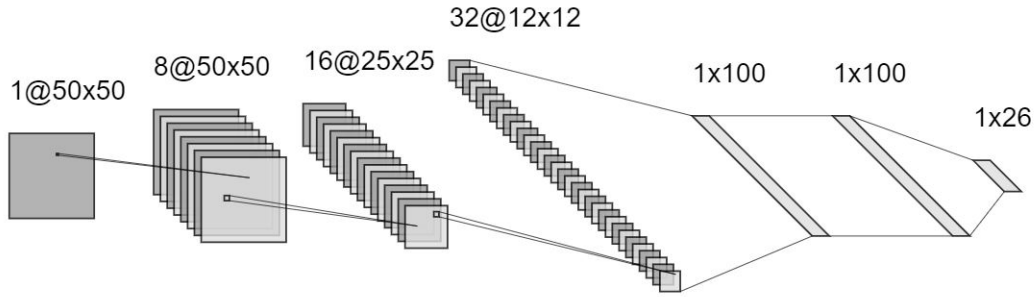


Figure 9: Neural Network structure 2B with ordinal classification for Experiment 2

The results of Experiment 2 are presented in table 2. We immediately observe that our new model with ordinal classification implemented performs significantly better then our original model. This is an expected outcome as the labels have a strict order in reality and this order is now exploited by the structure of our model. We can conclude that our new model with Structure 2B is better at predicting chromatic numbers than our previous model with Structure 2A.

	Number of parameters	Mean accuracy	Accuracy interval
Structure 2A	134 116	73.20	[65.22 , 75.40]
Structure 2B	134 015	81.75	[79.45 , 83.48]

Table 2: Results Experiment 2

5.3 Experiment 3: Classification vs Regression

Another method to incorporate the natural order between labels is to use a regression model instead of a classification model as is explained in section 3.3.3. This method is experimented with in Experiment 3. The structures that are used for Experiment 3 are structure 3A, identical to structure 2A and structure 3B that is also identical to structure 2A except for its single output node, making it a regression network. The regression model 3B has a different loss function that optimizes the 'mean absolute error' instead of the 'sparse categorical cross entropy' loss function that is used in structure 3A. To be able to still compare the two models we round each prediction of the regression model to the closest integer and take the rounded number as its prediction. In this way we can evaluate the accuracy of the regression model and compare it to the accuracy of the classification model (see Section 4.5).
The structure 3B is presented in Figure 10.

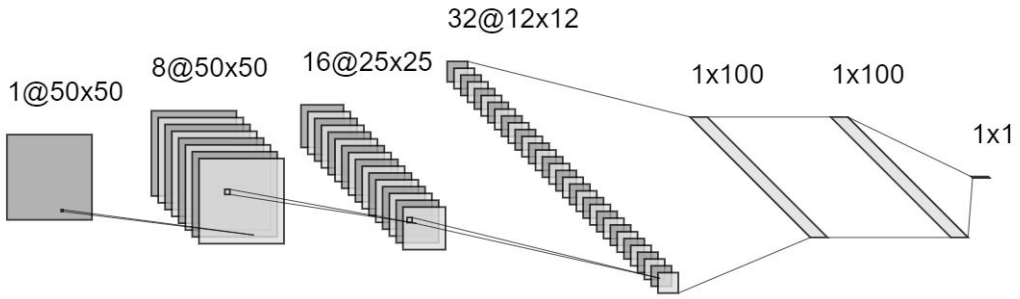


Figure 10: Neural Network structure 3B with regression for Experiment 3

The results of experiment 3 can be observed from Table 3. We see that incorporating the ordinal structure of the classes by creating a regression model leads to an even better model than when ordinal classification is used. We will therefore continue with experimenting with regression models to hopefully find an optimal model.

	Number of parameters	Mean accuracy	Accuracy interval
Structure 3A	134 116	73.20	[65.22 , 75.40]
Structure 3B	131 389	84.88	[84.39 , 86.07]

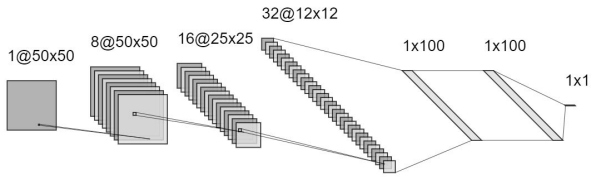
Table 3: Results Experiment 3

5.4 Experiment 4: Optimizing the Neural Network Structure

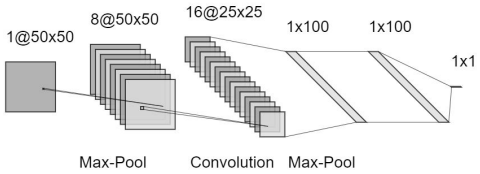
Experiment 4 consists of three sub experiments that are meant to obtain the optimal structure of the Convolutional Neural Network. However, due to this thesis' limited time we only test regression models as they showed the best accuracy results. In general, other structures should be tested for other models as well. In Experiment 4.1 models with different numbers of convolutional layers are tested to find the optimal number of convolutional layers. The effects of using different sizes and numbers of the filters that are used in the convolutional layers are investigated in Experiment 4.2. And finally, in Experiment 4.3 models with different structures in number and sizes of the fully connected layers are tested and evaluated.

5.4.1 Experiment 4.1: Convolutional layer structure

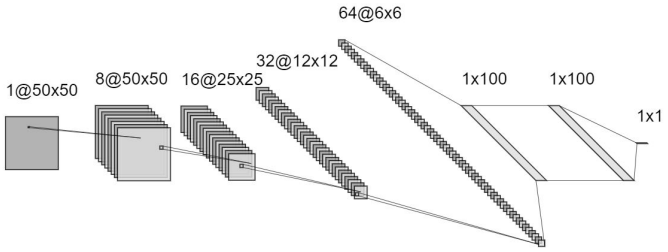
The first architecture component of a Convolutional Neural Network that we want to optimize is the number of convolutional layers. This is done in Experiment 4.1 in which we have our existing 'basic' Structure 4.1A with three convolutional layers and we experiment with small deviations from this structure ceteris paribus. The deviations are displayed in Figure 11: Structure 4.1B contains one less convolutional layer compared to Structure 4.1 whereas Structure 4.1C has one extra convolutional layer.



(a) Structure 4.1 A



(b) Structure 4.1 B



(c) Structure 4.1 C

Figure 11: Convolutional Neural Network structures for Experiment 4.1

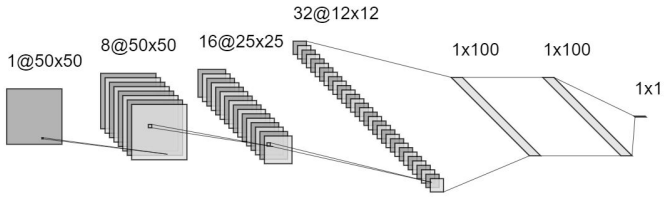
The results of all Structures of Experiment 4.1 are presented in Table 4. A decrease in the number of convolutional layers leads to more model parameters due to a larger first flattened layer in the fully connected structure component and an increase vice versa. We observe that an increase or decrease of the number of convolutional layers leads to a slightly worse mean accuracy. A significant increase in the size of the accuracy interval is also a result of both deviations of Structure 4.1A. Both results are undesired: for an optimal structure a highest possible mean accuracy with a smallest possible accuracy interval is best.
We know that a model with more convolutional layers leads to a model that can potentially recognise more complex structures than one with less convolutional layers. We can conclude that the optimal number of convolutional layers for our structure with the current data set is three.

	Number of parameters	Mean accuracy	Accuracy interval
Structure 4.1A	131 389	84.88	[84.39 , 86.07]
Structure 4.1B	241 949	82.44	[79.19 , 85.04]
Structure 4.1C	92 285	81.42	[75.19 , 84.60]

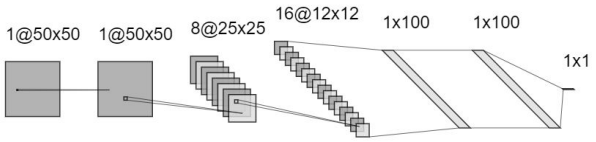
Table 4: Results Experiment 4.1

5.4.2 Experiment 4.2: Optimizing the filters

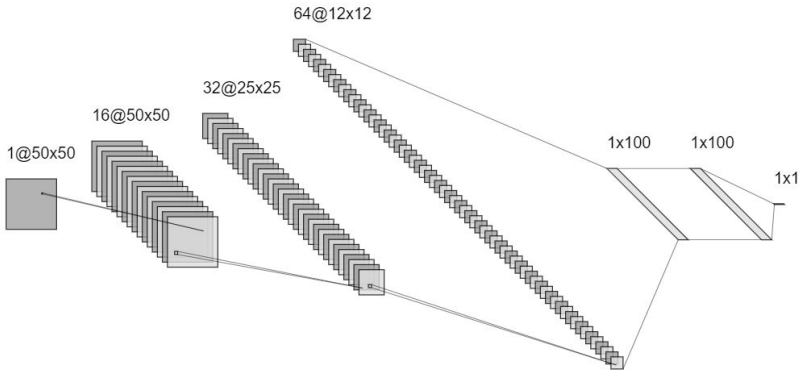
In Experiment 4.2 we aim for optimization of the filters in the convolutional layers. The two main structural components of the filters that can be adjusted are the filter size and the number of filters in a convolutional layer. Some of the different structures that are tested in Experiment 4.2 are presented in Figure 12.



(a) Structure 4.2 A



(b) Structure 4.2 C



(c) Structure 4.2 D

Figure 12: Convolutional Neural Network structures for Experiment 4.2

Structure 4.2A is the same structure as Structure 4.1A that was found to be optimal in Experiment 4.1. Structure 4.2B is equal to Structure 4.1A except for the fact that it has filters of size 5x5 instead of size 3x3. Structure 4.2C has less filters per layer than Structure 4.2A and Structure 4.2D has more filters per layer.

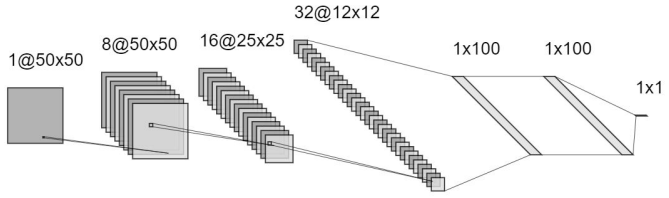
The results of Experiment 4.2 can be observed from Table 5. We first observe that Structure 4.2C has significantly less parameters than Structure 4.2A whereas Structure 4.2D has more than twice as many as Structure 4.2A. The mean accuracy of the 'basic' Structure 4.2A is slightly superior over the adjusted structures. The accuracy intervals of the measured accuracy's is better for Structure 4.2A. We therefore remain with Structure 4.2A as our 'basic' structure for later experiments.

	Number of parameters	Mean accuracy	Accuracy interval
Structure 4.2A	131 389	84.88	[84.39 , 86.07]
Structure 4.2B	141 757	83.34	[81.25 , 84.95]
Structure 4.2C	69 159	81.57	[77.86 , 83.27]
Structure 4.2D	263 997	81.67	[78.12 , 85.21]

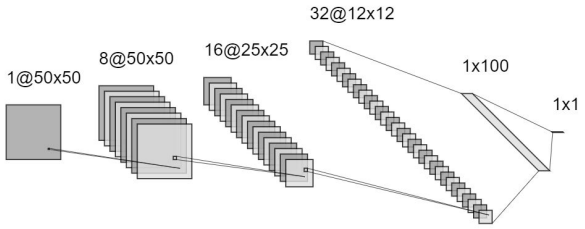
Table 5: Results Experiment 4.2

5.4.3 Experiment 4.3: Fully connected layers structure

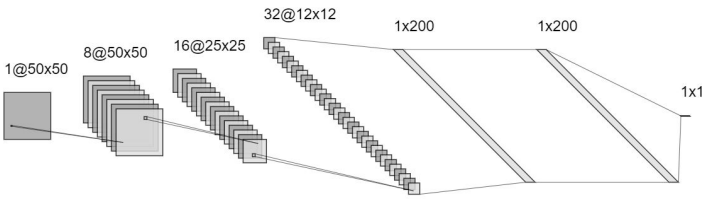
The last sub experiment that is performed to determine an optimal structure for our Convolutional Neural Network model looks at the structure of the fully connected layers in our model. There can be two types of variations: variations in the number of fully connected layers and variations in the size of fully connected layers. All structures that are used in Experiment 4.3 are presented in Figure 13.



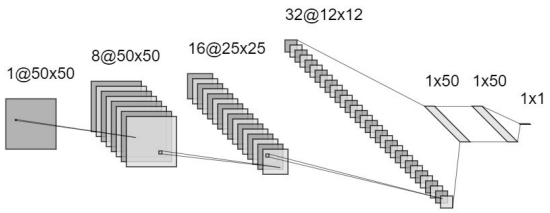
(a) Structure 4.3 A



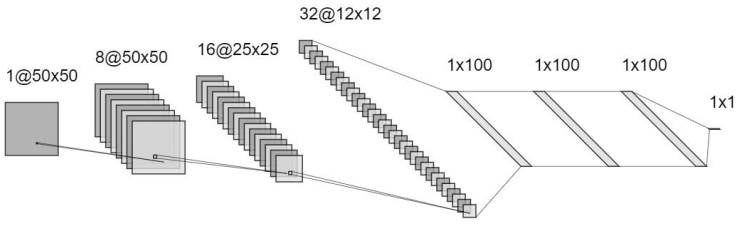
(b) Structure 4.3 B



(c) Structure 4.3 C



(d) Structure 4.3 D



(e) Structure 4.3 E

Figure 13: Convolutional Neural Network structures for Experiment 4.3

Structure 4.3A is our resulting 'basic' structure from Experiment 4.2. Structure 4.3B contains only one fully connected layer of size 100. Structure 4.3C has the same number of fully connected layers as Structure 4.3A but of size 200 instead of 100. Structure 4.3D has also the same number of layers but has smaller layers of size 50 and Structure 4.3E has 3 layers of size 100.
All results from Experiment 4.3 can be observed from Table 6. For the number of parameters we observe that the number of parameters structures 4.3A, 4.3B and 4.3E are relatively close while Structure 4.3C has significantly more parameters and Structure 4.3D has significantly less parameters. The mean accuracy's are almost the same for all structures with slightly higher accuracy's for 4.3A, 4.3C and 4.3D. We observe that the accuracy intervals for these three structures are also slightly smaller.
As the results for Structures 4.3A, 4.3C and 4.3D are relatively close, we will keep the three models as successful models for the final tests in Experiment 7. However, for further experiments we will continue with only one of them and that is 4.3A.

	Number of parameters	Mean accuracy	Accuracy interval
Structure 4.3A	131 389	84.88	[84.39 , 86.07]
Structure 4.3B	121 289	83.50	[79.59 , 85.32]
Structure 4.3C	276 889	85.02	[84.44 , 85.84]
Structure 4.3D	66 139	84.94	[84.01 , 85.65]
Structure 4.3E	141 489	84.00	[81.84 , 85.28]

Table 6: Convolutional Neural Network structures for Experiment 4.3

5.5 Experiment 5: Different Input sizes

In Experiment 5 we investigate whether it is possible to adjust our model such that it is able to receive graphs with different sizes as input. In Experiment 5.1 we performed 5-fold cross validation on Data set 2. In Experiment 5.2 we trained the same model with Data set 2 but now we evaluated the resulting trained model with different data sets 1 and 2.1. On all adjacency matrices in the data sets that are smaller then 100 by 100 we applied the zero padding technique as is explained in section 3.5. The structure of the Convolutional Neural Network model that is used in Experiment 6 is given in Figure 14. The main differences with the models from previous experiments are due to its 100x100 input size in the first convolutional layer.

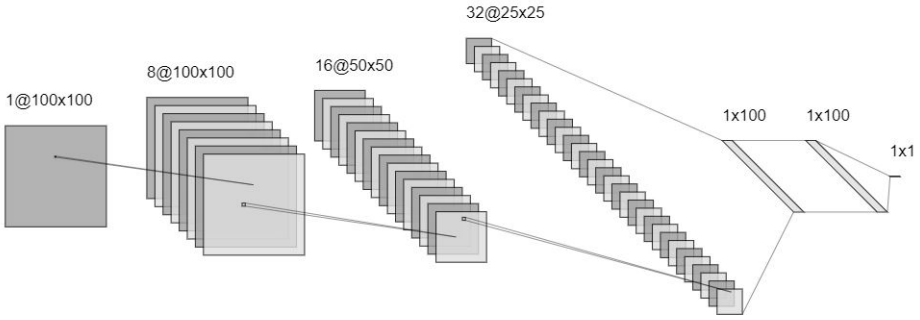


Figure 14: Convolutional Neural Network structure for Experiment 5

The results of Experiment 5.1 are presented in Table 7. Making the model suitable for graphs up to size 100 has increased the number of parameters significantly. The mean accuracy is slightly higher then the best models found in Experiment 4.3. However, Data set 2 contains relatively less Erdos-Renyi random graphs as their exact label is difficult to find for larger graphs we must be careful with drawing conclusions. We therefore perform Experiment 5.2 were we investigate the performance on two different data sets: Data set 1, containing only graphs of size 50 and Data set 2.1, containing graphs between size 50 and 100. The results of Experiment 5.2 are shown in Table 8. We observe that

	Number of parameters	Mean accuracy	Accuracy interval
Experiment 5.1	476 989	88.93	[86.27 , 91.22]

Table 7: Results Experiment 5.1

there is a significant difference in testing accuracy between the two test sets. The high performance on large graphs is caused by an 'easier' Data set 2.1 containing almost no Erdos-Renyi random graphs. Despite this difference between the data sets these results are still useful for our main question that we investigated in Experiment 5, namely, can a Convolutional Neural Network model for determining the chromatic number of a graph be adjusted such that it is able to process graphs of different input-sizes? The conclusion that we can draw of the results of Experiment 5.1 and 5.2 is that this adjusting is possible by using the zero padding technique that is discussed in Section 3.5. This result is in line with our theoretical expectations; pattern recognition by a Convolutional Neural Network is a process that is independent of the size of an image (or adjacency matrix in our case). However, practical problems that arise when our Convolutional Neural Network is adjusted is that the number of parameters within the model increases rapidly as does the time that is needed for the training phase. Another practical issue is that the memory size of the data set increases and that it may be difficult to include certain types of graphs in the data set because finding their exact label requires too much time.

	Size test set	Testing accuracy
Experiment 5.2A: testing on small graphs	28 636	81.83
Experiment 5.2B: testing on large graphs	38 328	95.63

Table 8: Results Experiment 5.2

5.6 Experiment 6: Using unlabeled data

Experiment 6 aims at investigating if unlabeled data , i.e. data for which the exact chromatic number is not known but an upper and lower bound are known, can also be used for improving a Convolutional Neural Network model for predicting chromatic numbers of graphs.
For Experiment 6 we use Data set 3. In section 4.3 it is explained that this data set consists of graphs of size 50 - 100 with two labels per graph: an upper bound and a lower bound on the chromatic number of the graph. If the chromatic number of the graph is known, these two labels are equal to each other. The data sets consists of graphs and two bounds and, hence, the model will also predict two bounds. We therefore have a different situation than all previous experiments because we have a model that is used for predicting bounds instead of chromatic numbers.

The structures that is used in Experiment 6 is presented in Figure 15. We consider two different models, Model 6A and Model 6B that are only different in their rounding method of a prediction to an integer. The rounding method that is applied to a prediction in Model 6A is 'standard' rounding to the nearest integer. The rounding method that is used in Model 6B is different for the prediction of the lower bound and upper bound: for the lower bound the prediction is rounded by the floor function and for the upper bound it is rounded by applying the ceiling function. The performance is measured as follows: if the rounded predicted lower bound is not higher than the known lower bound and the rounded predicted upper bound is not lower than the known upper bound, we say that the prediction is correct.

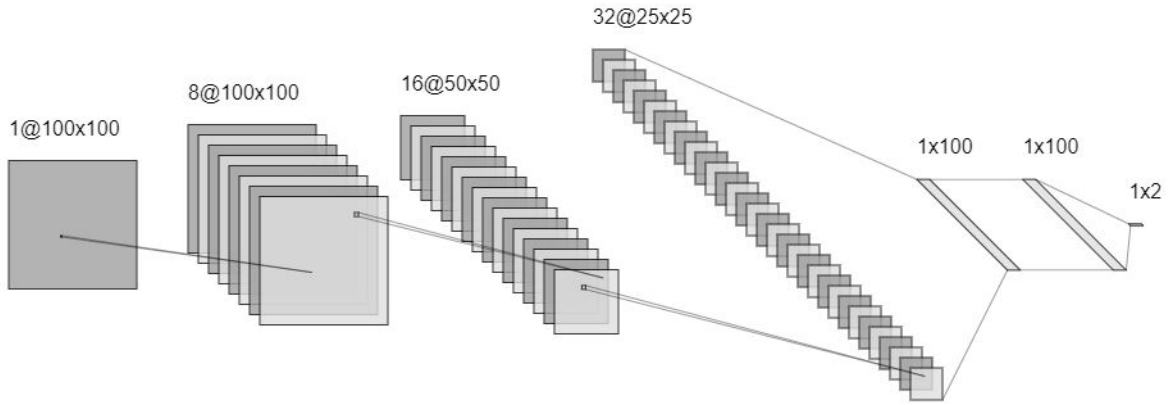


Figure 15: Convolutional Neural Network structure for Experiment 6

The results of Models 6A and 6B on Data set 3 are presented in Table 9. The accuracy of model 6B is higher than the accuracy of model 6A as could be expected by the difference in rounding methods as predictions are found to be correct as least as much by Model 6B as by Model 6A. Model 6A will despite that it is less accurate provide bounds that are closer to each other than the bounds predicted by Model 6B. A choice between both models could be made based on a trade-off between correctness of the bounds and absolute difference between the bounds. We also observe that for both models the upper bound is more difficult to predict than the lower bound. This could

be caused by several things. One possibility is that a predicted upper bound is too low because the upper bound is actually lower and the model is right about this prediction. Unfortunately, there is no way we can test these cases and we therefore say that the upper bound is wrongly predicted. In such a case, the wrongly predicted upper bound is of a graph for which the exact number is not known (there is a gap between the known upper bound and lower bound) and the model might be able to predict it smaller than the labels due to other exact data points that contain the same patterns as this unknown data point.

	Mean accuracy	accuracy interval	% wrong up-bound	% wrong low-bound
Model 6A	82.34	[76.11 , 86.79]	11.21	6.88
Model 6B	92.82	[89.74 , 94.53]	5.13	2.26

Table 9: Results Experiment 6.1

An interesting test to see whether this might be the case is testing the models 6A and 6B on a test set that contains only exact labels of size 50, Data set 3.1, and a test set that contains both exact and unknown labels, Data set 3.2 (see Section 4.3 for more information about these data sets). So the new experiment consists of the same training set as in Experiment 6.1 but now the test sets are different. The resulting experiment is Experiment 6.2.

The results of Experiment 6.2 are presented in Table 10. For both test sets it holds again that the second rounding method of Model 6B provides higher accuracy's. We now observe that for the test set that has exact labels with graphs of size 50, the percentage of wrongly predicted upper bounds and lower bounds are closer to each other than for the test set that also contains unknown labels. We expect that this is caused by the unlabeled data points that are only used in the experiment with graphs of size 51-100.

	test set	test accuracy	% wrong up-bound	% wrong low-bound
Model 6.2A	graphs size 50	88.51	5.08	6.40
Model 6.2B		97.57	1.88	0.56
Model 6.2A	graphs size 51-100	79.61	14.98	7.34
Model 6.2B		87.73	10.59	2.35

Table 10: Results Experiment 6.2

To check whether the predicted bounds of Model 6A and 6B are useful we also compare them with their trivial bounds. The trivial lower bound for each graph is 2 and the trivial upper bound for each graph is its maximum degree. We check if the predictions are higher (or lower) for these trivial lower (or upper) bounds. If the predicted interval includes the trivial bounds, the prediction model is not useful as these bounds can be found in less complex ways than our prediction model.
The results of this comparison are presented in Table 11. We see that there is never a prediction worse (lower) than the trivial lower bound 2. This does not hold for the upper bound as we see that for both models there are predictions that are above the trivial upper bound. If a prediction is equal or above the trivial upper bound the upper bound prediction is useless. We see that the predicted lower bounds for Model 6A are in 93.81 %

better than the predicted lower bound. For predicting lower bounds we have that Model 6A is prefered over Model 6B. This is not the case for the upper bounds as both models retrieve almost the same results when their upper bound predictions are compared to the know trivial upper bounds.

	lower bound		upper bound	
	% > than TB	% ≤ than TB	% < than TB	% ≥ than TB
Model 6A	93.81	6.19	70.94	0.86
Model 6B	78.53	21.47	70.79	0.48

Table 11: Comparison predictions Experiment 6.1 to trivial bounds (TB)

5.7 Experiment 7: Testing models on final data set

As a final experiment, we want to test a few of our previously successful models on some final test sets and evaluate their performance. The difference in test sets is described in Section 4.3. Model 7.1A is the same as the model used in Experiment 5.1 (see Figure 14). Model 7.1B and 7.1C differ only from 7.1A in their fully connected layer structure: Model 7.1B has instead of two layers of size 100 two layers of size 50 and Model 7.1C has two fully connected layers of size 200. We try these three structures as we found that they all performed well in Experiment 4.3. Model 7.2 is an ordinal classification model instead of a regression model and is similar to Structure 2B but is now adjusted to fit size 100. All models are trained with data set 2.

	Accuracy			
	Set 4.1	Set 4.2	Set 4.3	Set 4.4
Model 7.1A	31.43	96	96.5	74
Model 7.1B	17.14	96	96.5	56
Model 7.1C	31.43	96	96.5	72
Model 7.2	17.14	95.9	95.5	62

Table 12: Results Experiment 7

The performance of our final models on the test sets are presented in Table 12. We see that all models perform extremely well on Set 4.2 and Set 4.3. On the graphs found online captured in Test set 4.1 the performance is disappointing. However, this could be expected as they are not represented in the training data set. The two models that have the highest accuracy on the graphs of Test set 4.1 have also the highest accuracy on the Erdos-Renyi graphs from Test set 4.4. These models differ by being regression models and/or having larger fully connected layers. They could therefore be able to combine recognised patterns in more complex or more different ways which could be necessary to classify these types of graphs. We can therefore conclude that the structure of the graphs matters for the prediction accuracy of the models.
We also tested Models 6.2A and 6.2B on Test set 4.1 to check whether it was able to predict useful bounds for these difficult graphs. The results are presented in Tables 13 and 14. We see that both models have precisely the same performance on the Test set.

So all predictions of Model 6A that are wrong can not be fixed by the 'outside' rounding method of Model 6B. This indicates that it is not useful to use the Model 6B for these graphs as it never leads to better predictions. When we look at the comparison between predictions and trivial bounds we see that Model 6A is preferred over Model 6B as it predicts fewer bounds which are outside the trivial bounds interval. It seems that at least for some graphs the predicted bounds are useful as they are different than the trivial bounds.

	Accuracy	% wrong upper bound	% wrong lower bound
Model 6.2A	65.71	28.57	5.71
Model 6.2B	65.71	28.57	5.71

Table 13: Performance results bound prediction model on Test set 4.1

	lower bound		upper bound	
	% > than TB	% ≤ than TB	% < than TB	% ≥ than TB
Model 6A	93.81	6.19	70.94	29.06
Model 6B	78.53	21.47	70.79	29.21

Table 14: Comparison predictions on Test set 4.1 to trivial bounds (TB)

6 Conclusion

6.1 Main conclusions

In the introduction we mentioned the main research questions of this thesis:

1. Can deep learning models be used for predicting chromatic numbers of graph coloring problems?
2. Is there a difference in performance of these prediction models for different types of graphs?
3. What are limitations of a Convolutional Neural Network as a prediction model of a chromatic number?

To answer these questions, we started with testing two types of deep learning models: a fully connected Neural Network and a Convolutional Neural Network. We represented each graph in our data sets with its adjacency matrix, such that it can be seen as a binary picture consisting of zeros and ones. The Convolutional Neural Network outperformed the fully connected Neural Network with 73.20% against 63.78% accuracy. This could be expected as Convolutional Neural Networks are known to perform well in image classification tasks. Our graph coloring problem can also be seen as some type of image classification task as we presented our graphs in adjacency matrix and the model could recognise certain patterns in these binary pictures that indicate features that are important for the chromatic number of a class.
As the results of the Convolutional Neural Network model were promising we experimented with different structures of the Convolutional Neural Network. We first tried to

take advantage of the known order between classes. We tried to incorporate this by creating an ordinal classification model and a regression model, two model structures that can both exploit this feature. The results improved for both models with 81.75% accuracy for the ordinal classification model and 84.88% accuracy for the regression model.
We therefore continued with experiments aimed at finding the optimal Convolutional Neural Network structure of our regression model for our data set with an accuracy of 85%.
To receive a more practical model that could be used in practice we tested whether we could adjust our model such that it could fit and predict graphs of different sizes. We tested this by applying zero padding to all smaller adjacency matrices in our data set such that they all would have the same size (100x100). We then tested and evaluated the results and found that the model performed still reasonably well with an accuracy of 88.93%. However, the higher performance is due to relatively less Erdos-Renyi random graphs in the data set as it is hard to generate data points of large Erdos-Renyi graphs.
When we generated our graph data with corresponding chromatic numbers there were a lot of graphs (especially larger graphs) for which we could not find the exact chromatic number. However, for these graphs we did find a lower and upper bound on the chromatic number. In Experiment 6 we tested if we could use these unlabeled graphs by changing our regression model with one output node to a regression model with two output nodes. The new model is a model that does not predict the exact chromatic number as the other models, but it predicts an upper and a lower bound for each graph. The performance of the new models was fine with accuracy's of 82.34% and 92.82% of correct bounds, but it is less strong as it predicts bounds that could have a large gap between them instead of the prediction of one chromatic number. We compared the bound predictions with the trivial bounds of the graphs and we found that the predicted bounds are better than the trivial bounds for most of the predictions: up to 93.81% for the lower bounds and 70.94% for the upper bounds. It therefore seems that the bound predictions and corresponding models can be useful as they provide new information.
In Experiment 7 we tested three regression models and one ordinal classification model on different unseen test sets. The best performance on Test set 4.1, consisting of different graphs found online used for testing graph coloring algorithms, was not that good with 31.43% accuracy. The performance on the Erdos-Renyi Test set 4.4 could also be improved as it is now a 74% accuracy. We can conclude that the type of graph matters for the performance of our prediction models.
Our main conclusion is that it is definitely possible to use deep learning models for determining the solution of a graph coloring problem. A Convolutional Neural Network is a structure that performs well on graphs represented by their adjacency matrix as they are image-like. However, generating training data is difficult as for each graph that is randomly generated the chromatic number must be found as its label. Especially for large graphs or certain types of random graphs (Erdos-Renyi) it is not possible to solve their graph coloring problem exactly and add the graph and corresponding solution to the data set. For difficult graphs a model that predicts bounds on the chromatic number instead of the exact chromatic number may be considered.

6.2 Limitations

There are some limitations for this research and its resulting models that are listed below:

- As Neural Network models' performances benefit from large data sets, it would be

ideal to generate a large number of data points. Unfortunately, it is computationally difficult to retrieve a data set that contains a large number of large graphs with corresponding chromatic numbers. Techniques such as data augmentation can help for creating more data points but despite these techniques it is still hard to retrieve large graph data.

- The models considered in this thesis were only used for predicting chromatic numbers of bounds on chromatic numbers. However, in most practical situations a coloring of the nodes corresponding to the chromatic number is wanted. An estimation of the chromatic number may help other algorithms to obtain a coloring but only an estimation of the chromatic number is not always useful in practice.

- The retrieved models attempt to exploit patterns found in the training data to determine chromatic numbers. Different types of graphs contain different patterns in their adjacency matrices. The resulting models only perform well on graphs that are included in the training data.

6.3 Further research directions

In this subsection we will mention some possible further research directions.

1. In practice, a model would only be useful if it is able to receive large graphs as input. An interesting further research direction might therefore be to investigate whether it is possible to construct a model that can predict the chromatic number of large graphs of a large size n without having graphs of size n in the training data. Implementing techniques such as Spatial Pyramid Pooling [16] could be investigated for graph coloring problems. Spatial Pyramid Pooling adds a spatial pyramid pooling layer between the convolutional layers and the fully connected layers of a Convolutional Neural Network such that the input size for the fully connected layers is fixed.

2. An interesting research direction would be to investigate whether our resulting models can be combined with existing algorithms to create improved heuristics for graph coloring. They could, for example, be implemented in algorithms that decompose graphs into smaller subgraphs. Our models could then maybe be used to determine the chromatic number of subgraphs.

3. As a set of isomorphic (non trivial) graphs can be represented by multiple adjacency matrices (see Section 4.2, Theorem 4.1) but has one chromatic number, the patterns that can be recognised from one of these graphs adjacency matrix could be totally different from another adjacency matrix representation of a graph that is isomorphic. An idea for improvement may be to use as many reshufflings of adjacency matrices as possible such that all patterns of adjacency matrices are included in the data. A further research direction may therefore be to investigate how to extract as much information as possible from permutations of adjacency matrices without generating all permutations. This could lead to an advanced data augmentation technique that would be useful to create a more complete data set for pattern recognition. It could also be helpful by creating more data samples from a small number of existing difficult graphs with a known chromatic number.

Bibliography

[1] Harry R Lewis. Computers and intractability. a guide to the theory of np-completeness, 1983.

[2] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[3] Anuj Mehrotra and Michael A Trick. A column generation approach for graph coloring. *informs Journal on Computing*, 8(4):344–354, 1996.

[4] Shamim Ahmed. Applications of graph coloring in modern computer science. *International Journal of Computer and Information Technology*, 3(2):1–7, 2012.

[5] Ivan Zelinka, Vaclav Snasael, and Ajith Abraham. *Handbook of optimization: from classical to modern approach*, volume 38. Springer Science & Business Media, 2012.

[6] Arun Jagota. An adaptive, multiple restarts neural network algorithm for graph coloring. *European Journal of Operational Research*, 93(2):257–270, 1996.

[7] Dibyendu Das, Shahid Asghar Ahmad, and Kumar Venkataramanan. Deep learning-based hybrid graph-coloring algorithm for register allocation. *arXiv preprint arXiv:1912.03700*, 2019.

[8] Henrique Lemos, Marcelo Prates, Pedro Avelar, and Luis Lamb. Graph colouring meets deep learning: Effective graph neural network models for combinatorial problems. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 879–885. IEEE, 2019.

[9] Jiayi Huang, Mostofa Patwary, and Gregory Diamos. Coloring big graphs with alphagozero. *arXiv preprint arXiv:1902.10162*, 2019.

[10] Hong Xu, Sven Koenig, and TK Satish Kumar. Towards effective deep learning for constraint satisfaction problems. In *International Conference on Principles and Practice of Constraint Programming*, pages 588–597. Springer, 2018.

[11] Isabel Méndez-Díaz and Paula Zabala. A cutting plane algorithm for graph coloring. *Discrete Applied Mathematics*, 156(2):159–179, 2008.

[12] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[13] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. Ieee, 2017.

[14] Rowland Leonard Brooks. On colouring the nodes of a network. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 37, pages 194–197. Cambridge University Press, 1941.

[15] Jianlin Cheng, Zheng Wang, and Gianluca Pollastri. A neural network approach to ordinal regression. In *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*, pages 1279–1284. IEEE, 2008.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

[17] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440–442, 1998.

[18] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[19] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.

[20] Emanuel Lazar. Lecture notes in ideas in mathematics, Spring 2016.