T.R.

GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

LEARNING BIJECTIONS

CAN DUYAR - 171044075

SUPERVISOR
ASST. PROF. ZAFEIRAKIS ZAFEIRAKOPOULOS

GEBZE
2022

T.R.
GEBZE TECHNICAL UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

LEARNING BIJECTIONS

CAN DUYAR - 171044075

SUPERVISOR
ASST. PROF. ZAFEIRAKIS ZAFEIRAKOPOULOS

2022
GEBZE

GEBZE TECHNICAL UNIVERSITY	GRADUATION PROJECT JURY APPROVAL FORM

This study has been accepted as an Undergraduate Graduation Project in the Department of Computer Engineering on 31/08/2021 by the following jury.

JURY

Member (Supervisor)	:	Asst. Prof. Zafeirakis Zafeirakopoulos
Member	:	Asst. Prof. Zafeirakis Zafeirakopoulos
Member	:	Assoc. Prof. Didem Gözüpek

ABSTRACT

My project is about training a neural network to act as a bijection. I picked two sets of combinatorial objects that are known to have the same cardinality and trained a invertible neural network to act as a bijection. It gives a different output for each input. The goal is to investigate best practices to develop a general framework in the direction of learning bijections based on Julia programming language.

Keywords: bijections, neural network.

ÖZET

Projem, birebir örten fonksiyonlar gibi davranmayı öğrenen bir sinir ağı mimarisi eğitmektir. Aynı eleman sayısına sahip olduğu bilinen iki kombinatoryal nesne kümesi seçtim ve ters çevrilebilir bir sinir ağını birebir örten olarak hareket etmesi için eğittim. Model, her girdi için farklı bir çıktı vermektedir. Proje amacı, Julia programlama dilinde birebir örten fonksiyonların öğrenme davranışını sergileyen genel bir framework oluşturmaktır.

Anahtar Kelimeler: birebir örten fonksiyonlar, sinir ağı.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my supervisor, Asst. Prof. Zafeirakis ZAFEIRAKOPOULOS, for his enthusiasm, patience, insightful comments, helpful information, practical advice and unceasing ideas that have helped me tremendously at all times in my research and writing of this thesis. His immense knowledge, profound experience and professional expertise has enabled me to complete this research successfully. Without his support and guidance, this project would not have been possible. I could not have imagined having a better supervisor in my study. Thanks for all your encouragement.

Finally, I must express my very deep gratitude to my parents for providing me with endless support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without them.

Thank you for everything.

Can Duyar

LIST OF SYMBOLS AND ABBREVIATIONS

Symbol or Abbreviation	:	**Explanation**
NN	:	Neural Network
INN	:	Invertible Neural Network
MMD	:	Maximum Mean Discrepancy
SL	:	Supervised Loss
USL	:	Unsupervised Loss

CONTENTS

LIST OF FIGURES

1. WHAT IS A BIJECTION?

A one-to-one correspondence function between the elements of two sets is known as bijection. Each element of one set pairs with precisely one element of the other set, and each element of the second set has exactly one paired element in the first set in such a function. 1.1.

Functions can be one-to-one functions (injections), onto functions (surjections), or both one-to-one and onto functions (bijections). Here is a brief overview of surjective, injective and bijective functions:

- **Surjective:** If f: P $\to$ Q is a surjective function, for every element in Q, there is at least one element in P, that is, f (p) = q.
- **Injective:** If f: P $\to$ Q is an injective function, then distinct elements of P will be mapped to distinct elements of Q, such that p = q whenever f (p) = f (q).
- **Bijective:** If f: P $\to$ Q is a bijective function, for every element in Q, there is exactly one element in P, that is, f (p) = q.

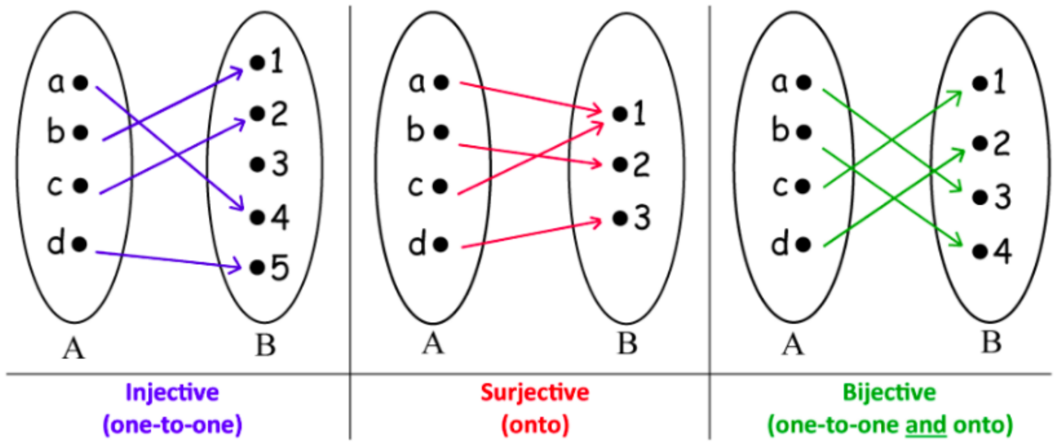


Figure 1.1: Bijective Functions

1.1. Bijective Function Properties

We know the function f: P → Q is bijective if every element q ∈ Q is the image of only one element p ∈ P, where element ‘q’ is the image of element ‘p,’ and element ‘p’ is the preimage of element ‘q’. Also,

- Each element of P should be paired with at least one element of Q.
- No element of P must be paired with more than one element of Q.
- Each element of Q must be paired with at least one element of P, and
- No element of Q must be paired with more than one element of P.

1.1.1. Example of Bijection

Let f: A → B. A, B and f are defined as:
A = {1, 2, 3, 4}
B = {5, 6, 7, 8}
f = {(1, 8), (2, 6), (3, 5), (4, 7)}

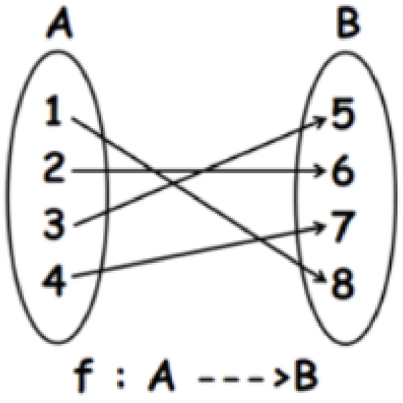


Figure 1.2: Example of Bijection

- Every element of B has a preimage in A. So f is onto function.
- Every element of A has a different image in B. That is, no two or more elements of A have the same image in B. So f is one to one function.
- Therefore, f is one to one and onto or bijective function.

1.2. Invertibility of Bijections

A bijection is a function that is one-to-one and onto at the same time. Naturally, we call a function bijective if it is a bijection. If a function f:A→B is a bijection, we can write another function g to invert the assignment rule associated with f. Then, by applying the function g to any element y from the codomain B, we can get an element x from domain A where f(x) = y.

More Concrete Definition: Let f:A→B be a bijective function. Its inverse function is the function f^{-1}:B→A with the property that f^{-1}(b) = a ↔ b = f(a) The notation f^{-1} is pronounced as "f inverse." See Figure 1.3 for a pictorial view of an inverse function.

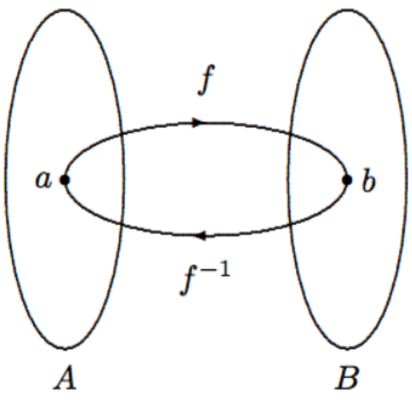


Figure 1.3: The pictorial view of an inverse function.

2. INVERTIBLE NEURAL NETWORK

I used an invertible neural network to learn bijections, because invertible networks have the feature of reconstructing the input from the output as well as learning the output from the input.

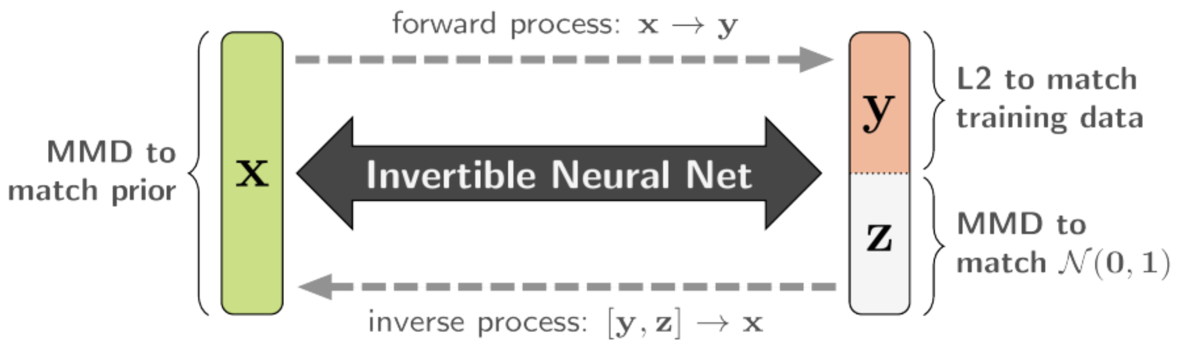

forward process: $\mathbf{x} \to \mathbf{y}$
MMD to match prior
$\mathbf{x}$
Invertible Neural Net
$\mathbf{y}$
L2 to match training data
$\mathbf{z}$
MMD to match $\mathcal{N}(0,1)$
inverse process: $[\mathbf{y}, \mathbf{z}] \to \mathbf{x}$

Figure 2.1: Invertible Neural Network Architecture.

Invertible networks allow for simultaneous loss optimization in both the input and output domains, allowing for more efficient training. By doing so, we can alternate between forward and backward iterations, accumulating gradients from both directions before performing a parameter update.

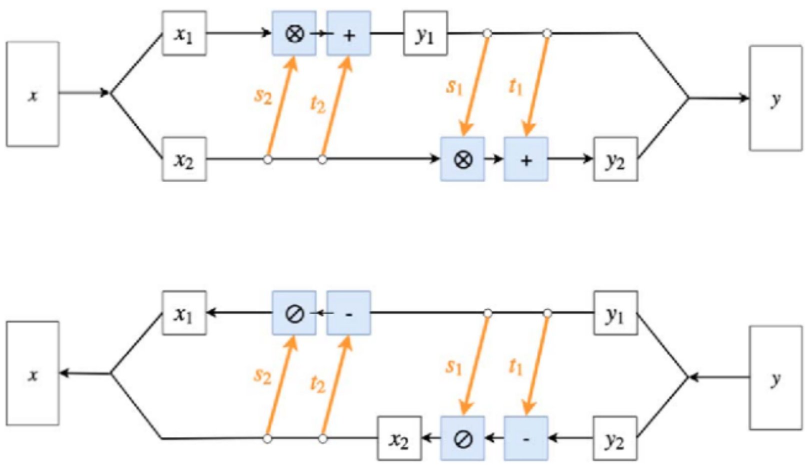
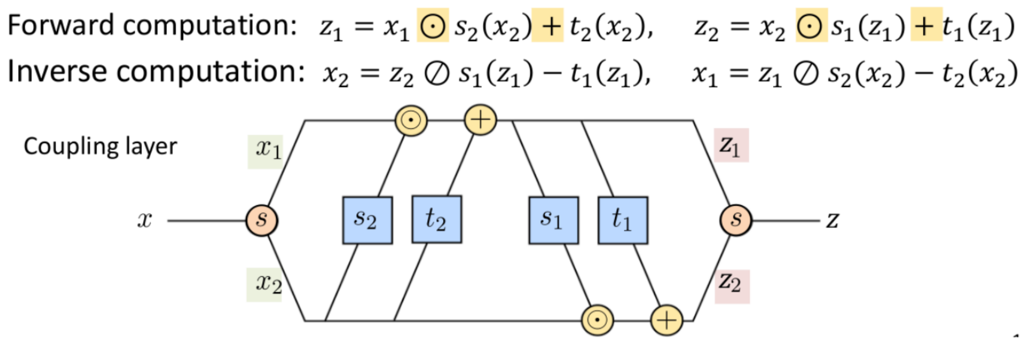


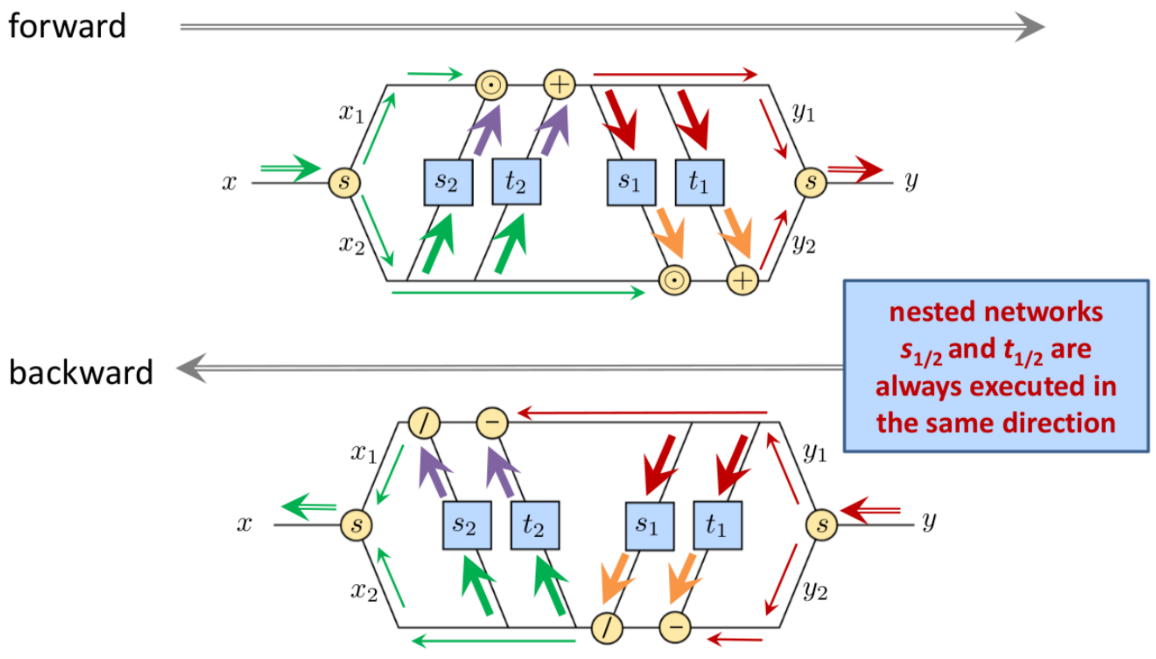
Figure 2.2: INN.

2.1. Forward and Inverse Computations in Invertible Neural Networks



Forward computation: $z_1 = x_1 \odot s_2(x_2) + t_2(x_2), \quad z_2 = x_2 \odot s_1(z_1) + t_1(z_1)$
Inverse computation: $x_2 = z_2 \oslash s_1(z_1) - t_1(z_1), \quad x_1 = z_1 \oslash s_2(x_2) - t_2(x_2)$
Coupling layer
x
x_1
x_2
s
s_2
t_2
s_1
t_1
z_1
z_2
z

Figure 2.3: Forward and Inverse Computations in INNs.



forward
backward
x
x_1
x_2
s
s_2
t_2
s_1
t_1
y_1
y_2
y
nested networks $s_{1/2}$ and $t_{1/2}$ are always executed in the same direction

Figure 2.4: Forward and Backward in INNs.

2.2. Abstract Comparison of Standard and Invertible Neural Networks

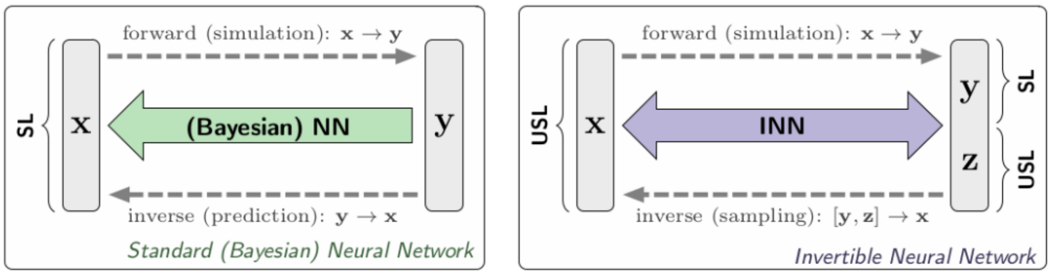

forward (simulation): $\mathbf{x} \to \mathbf{y}$
SL
x
(Bayesian) NN
y
inverse (prediction): $\mathbf{y} \to \mathbf{x}$
Standard (Bayesian) Neural Network
forward (simulation): $\mathbf{x} \to \mathbf{y}$
USL
x
INN
y
z
SL
USL
inverse (sampling): $[\mathbf{y}, \mathbf{z}] \to \mathbf{x}$
Invertible Neural Network

Figure 2.5: Standard vs. Invertible Neural Networks

The standard direct approach requires a discriminative, supervised loss(SL) term between predicted and true x, causing problems when y $\to$ x is ambiguous. Our network uses a supervised loss only for the well-defined forward process x $\to$ y. Generated x are required to follow the prior p(x) by an unsupervised loss(USL), while the latent variables z are made to follow a Gaussian distribution, also by an unsupervised loss.

INNs are characterized by three properties:

- The mapping from inputs to outputs is bijective, i.e. its inverse exists.
- Both forward and inverse mapping are efficiently computable.
- The mappings have tractable Jacobians, so that probabilities can be transformed explicitly via the change-of-variables formula.

3. CREATING DATA SET

It was not possible to use a ready dataset in this project because as a result of my research, I saw that no one had worked on such a project before. I needed to pick two sets of combinatorial objects that are known to have the same cardinality for dataset. In order to produce these objects, I implemented the algorithm of **Integer Partition with only odd numbers** in Python programming language.

3.1. Integer Partition with Only Odd Numbers

In number theory and combinatorics, a partition of a positive integer n, also called an integer partition, is a way of writing n as a sum of positive integers. I applied this method only for odd partitions to generate combinatorial objects for my project.

3.1.1. Examples of Integer Partition with Only Odd Numbers

Let N=5. Then,
1+1+1+1+1
1+1+3
5
are the partitions of 5 into odd parts hence, p(5|odd parts)=3

Let N=7. Then,
1+1+1+1+1+1+1
1+1+1+1+3
1+3+3
1+1+5
7
are the partitions of 7 into odd parts hence, p(7|odd parts)=5

3.2. Creating Bijection Data Set by Using Integer Partition with Only Odd Numbers

I needed to chose two sets that have same cardinality for my dataset. I used "Integer Partition with Only Odd Numbers" to create objects of these sets. For example, Let N = 5 then,
1+1+1+1+1
1+1+3
5

These are my partitions according to "Integer Partition with Only Odd Numbers" and p(5|odd parts)=3 in this case. I needed to elect one element from integer partitions to divide them into two equal halves so I elected single object(5 in this example), therefore my partitions are,
1+1+1+1+1
1+1+3

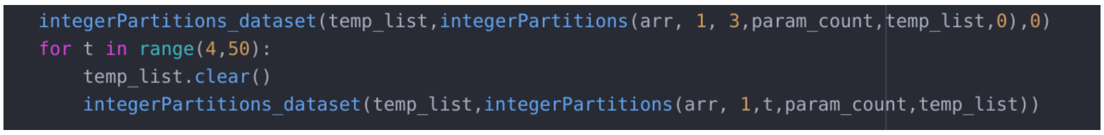
So I shared them with two sets as following,
Set1: [1,1,1,1,1]
Set2: [1,1,3]

Let N=6 then,
1+1+1+1+1+1
1+1+1+3
1+5
3+3
6

These are my partitions and p(6|odd parts)=5 in this case. I needed to elect one element from integer partitions to divide them into two equal halves so I elected single object(6 in this example), therefore my partitions are,
1+1+1+1+1+1
1+1+1+3
1+5
3+3

So, I shared these new partitions with actual two sets as following,
Set1: [1,1,1,1,1],**[1,1,1,1,1,1],[1,5]**
Set2: [1,1,3],**[1,1,1,3],[3,3]**

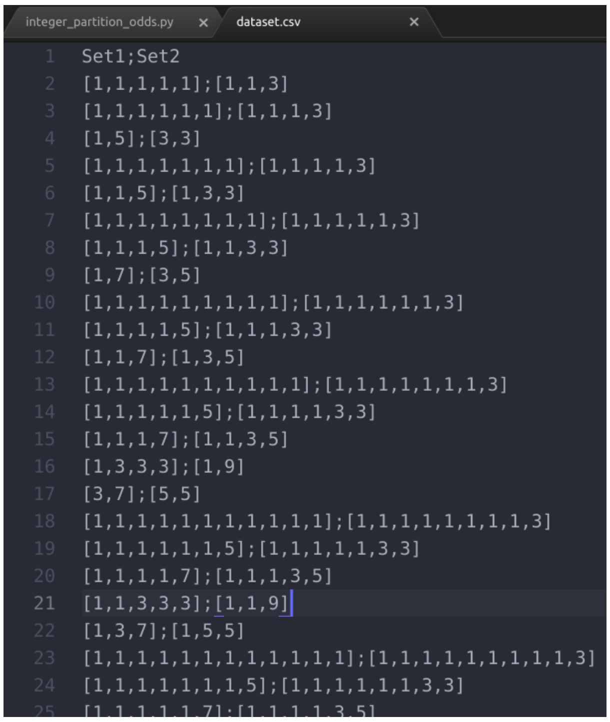
I wrote a script in Python programming language to generate a data set based on integer partition with only odd numbers. I selected N values in range(4,50) and I had 13937 rows in my data set.



```
integerPartitions_dataset(temp_list,integerPartitions(arr, 1, 3,param_count,temp_list,0),0)
for t in range(4,50):
    temp_list.clear()
    integerPartitions_dataset(temp_list,integerPartitions(arr, 1,t,param_count,temp_list))
```

Figure 3.1: Range of values of my script for integer partition with only odd numbers.

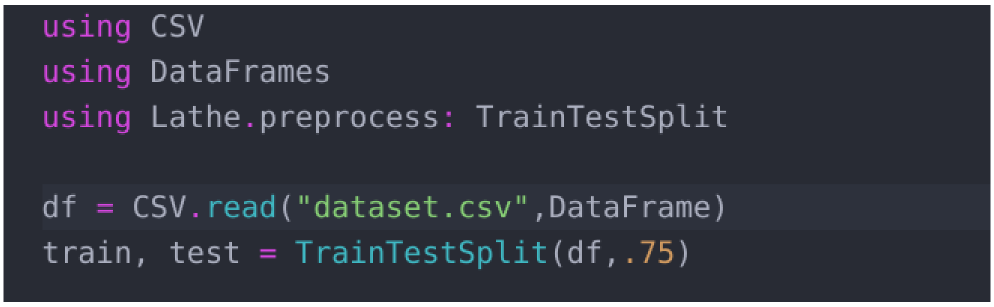
Cardinalities of Set1 and Set2 are equal to each other. The goal is to find bijections between Set1 and Set2 by using invertible Neural Network Architecture. Model gives an unique output(from Set2) for each input(from Set1) and also gives an unique output(from Set1) for each input(from S2). For example, if we give [1,1,1,1,1] as input then model gives us [1,1,3]. If we give [1,1,3] then it gives us [1,1,1,1,1] because model works as invertible.



```
Set1;Set2
[1,1,1,1,1];[1,1,3]
[1,1,1,1,1,1];[1,1,1,3]
[1,5];[3,3]
[1,1,1,1,1,1,1];[1,1,1,1,3]
[1,1,5];[1,3,3]
[1,1,1,1,1,1,1,1];[1,1,1,1,1,3]
[1,1,1,5];[1,1,3,3]
[1,7];[3,5]
[1,1,1,1,1,1,1,1,1];[1,1,1,1,1,1,3]
[1,1,1,1,5];[1,1,1,3,3]
[1,1,7];[1,3,5]
[1,1,1,1,1,1,1,1,1,1];[1,1,1,1,1,1,1,3]
[1,1,1,1,1,5];[1,1,1,1,3,3]
[1,1,1,7];[1,1,3,5]
[1,3,3,3];[1,9]
[3,7];[5,5]
[1,1,1,1,1,1,1,1,1,1,1];[1,1,1,1,1,1,1,1,3]
[1,1,1,1,1,1,5];[1,1,1,1,1,3,3]
[1,1,1,1,7];[1,1,1,3,5]
[1,1,3,3,3];[1,1,9]
[1,3,7];[1,5,5]
[1,1,1,1,1,1,1,1,1,1,1,1];[1,1,1,1,1,1,1,1,1,3]
[1,1,1,1,1,1,1,5];[1,1,1,1,1,1,3,3]
[1,1,1,1,1,7];[1,1,1,1,3,5]
```

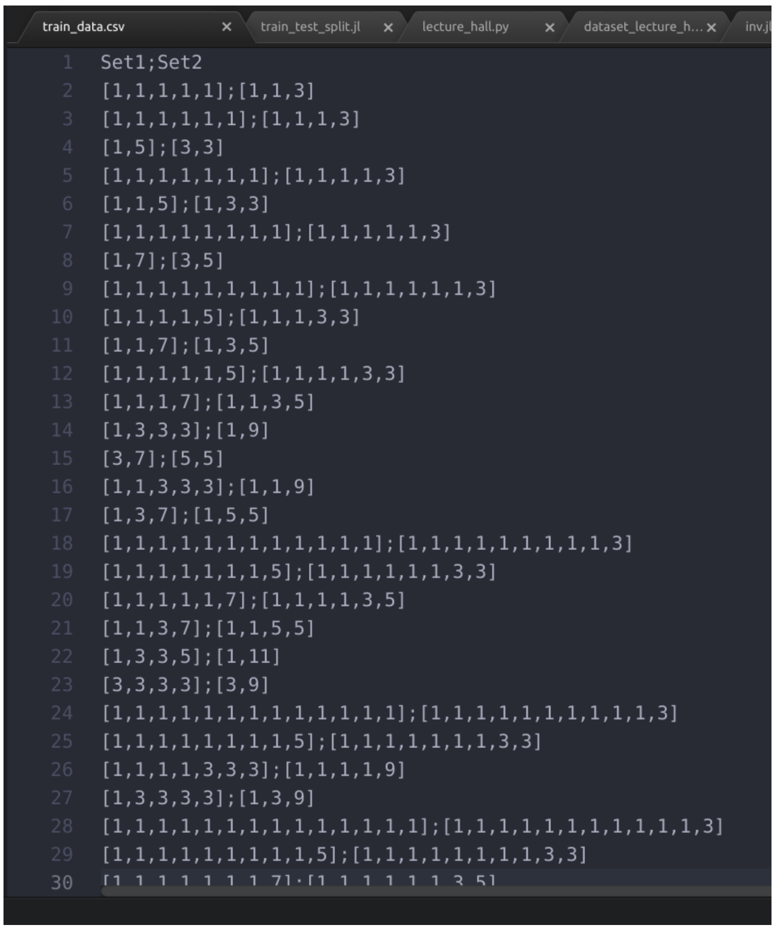
Figure 3.2: My dataset.

I created my train and test data by using "Lathe" library. I chose the train-test separation ratio as 0.75 and as a result of this process, I got 10517 train and 3457 test data.



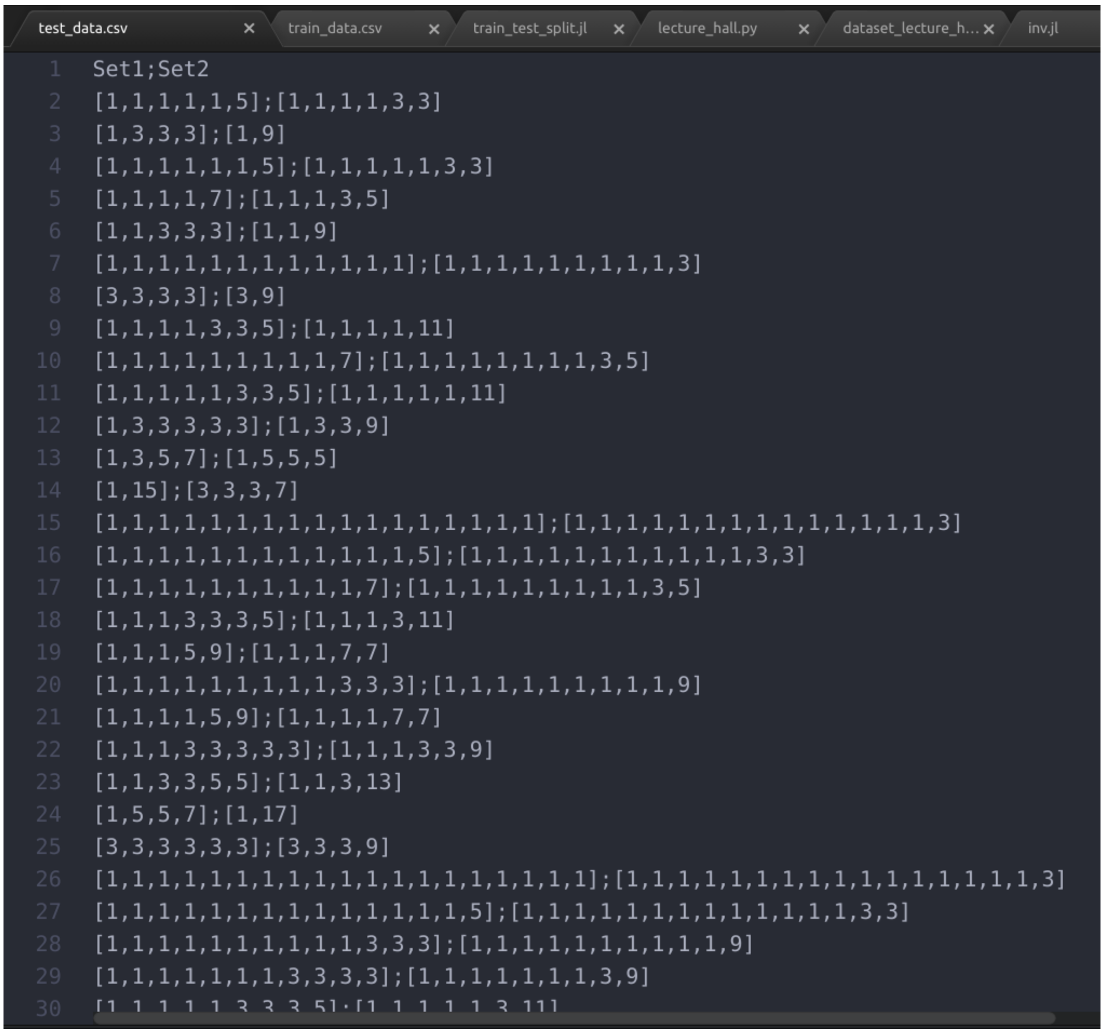
```
using CSV
using DataFrames
using Lathe.preprocess: TrainTestSplit

df = CSV.read("dataset.csv",DataFrame)
train, test = TrainTestSplit(df,.75)
```

Figure 3.3: Train-test split operation with "Lathe" library.



```
Set1;Set2
[1,1,1,1,1];[1,1,3]
[1,1,1,1,1,1];[1,1,1,3]
[1,5];[3,3]
[1,1,1,1,1,1,1];[1,1,1,1,3]
[1,1,5];[1,3,3]
[1,1,1,1,1,1,1,1];[1,1,1,1,1,3]
[1,7];[3,5]
[1,1,1,1,1,1,1,1,1];[1,1,1,1,1,1,3]
[1,1,1,1,5];[1,1,1,3,3]
[1,1,7];[1,3,5]
[1,1,1,1,1,5];[1,1,1,1,3,3]
[1,1,1,7];[1,1,3,5]
[1,3,3,3];[1,9]
[3,7];[5,5]
[1,1,3,3,3];[1,1,9]
[1,3,7];[1,5,5]
[1,1,1,1,1,1,1,1,1,1,1,1];[1,1,1,1,1,1,1,1,1,3]
[1,1,1,1,1,1,1,5];[1,1,1,1,1,1,3,3]
[1,1,1,1,1,7];[1,1,1,1,3,5]
[1,1,3,7];[1,1,5,5]
[1,3,3,5];[1,11]
[3,3,3,3];[3,9]
[1,1,1,1,1,1,1,1,1,1,1,1,1];[1,1,1,1,1,1,1,1,1,1,3]
[1,1,1,1,1,1,1,1,5];[1,1,1,1,1,1,1,3,3]
[1,1,1,1,3,3,3];[1,1,1,1,9]
[1,3,3,3,3];[1,3,9]
[1,1,1,1,1,1,1,1,1,1,1,1,1,1];[1,1,1,1,1,1,1,1,1,1,1,3]
[1,1,1,1,1,1,1,1,1,5];[1,1,1,1,1,1,1,1,3,3]
[1,1,1,1,1,1,1,7];[1,1,1,1,1,1,3,5]
```

Figure 3.4: Train Data.



```
Set1;Set2
[1,1,1,1,1,5];[1,1,1,1,3,3]
[1,3,3,3];[1,9]
[1,1,1,1,1,1,5];[1,1,1,1,1,3,3]
[1,1,1,1,7];[1,1,1,3,5]
[1,1,3,3,3];[1,1,9]
[1,1,1,1,1,1,1,1,1,1,1,1];[1,1,1,1,1,1,1,1,1,3]
[3,3,3,3];[3,9]
[1,1,1,1,3,3,5];[1,1,1,1,11]
[1,1,1,1,1,1,1,1,1,7];[1,1,1,1,1,1,1,1,3,5]
[1,1,1,1,1,3,3,5];[1,1,1,1,1,11]
[1,3,3,3,3,3];[1,3,3,9]
[1,3,5,7];[1,5,5,5]
[1,15];[3,3,3,7]
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1];[1,1,1,1,1,1,1,1,1,1,1,1,1,1,3]
[1,1,1,1,1,1,1,1,1,1,1,1,5];[1,1,1,1,1,1,1,1,1,1,1,3,3]
[1,1,1,1,1,1,1,1,1,1,7];[1,1,1,1,1,1,1,1,1,3,5]
[1,1,1,3,3,3,5];[1,1,1,3,11]
[1,1,1,5,9];[1,1,1,7,7]
[1,1,1,1,1,1,1,1,1,3,3,3];[1,1,1,1,1,1,1,1,1,9]
[1,1,1,1,5,9];[1,1,1,1,7,7]
[1,1,1,3,3,3,3,3];[1,1,1,3,3,9]
[1,1,3,3,5,5];[1,1,3,13]
[1,5,5,7];[1,17]
[3,3,3,3,3,3];[3,3,3,9]
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1];[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,3]
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,5];[1,1,1,1,1,1,1,1,1,1,1,1,1,3,3]
[1,1,1,1,1,1,1,1,1,1,3,3,3];[1,1,1,1,1,1,1,1,1,1,9]
[1,1,1,1,1,1,1,3,3,3,3];[1,1,1,1,1,1,1,3,9]
[1,1,1,1,1,3,3,3,5];[1,1,1,1,1,3,11]
```

Figure 3.5: Test Data.

4. MODEL IMPLEMENTATION USING INVERTIBLE NEURAL NETWORK

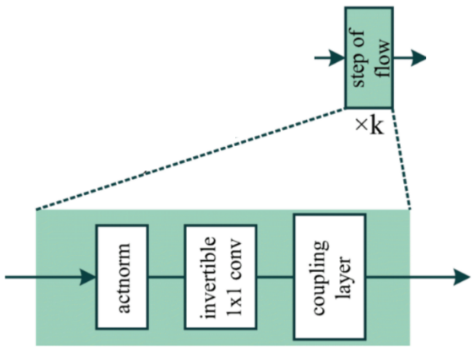


Figure 4.1: GLOW Model.

I used GLOW model for my project because I needed to implement invertible neural network structure and GLOW is a sort of flow-based generative model based on an invertible convolution. Each step of flow consists of an actnorm step, an invertible 1×1 convolution and an affine transformation.

Affine coupling is a technique for implementing a flow that is normalizing (where we stack a sequence of invertible bijective transformation functions). One of these bijective transformation functions is affine coupling. It's an example of a reversible transformation in which the forward, reverse, and log-determinant functions are all computationally efficient.

Theory of the Affine Transformation:

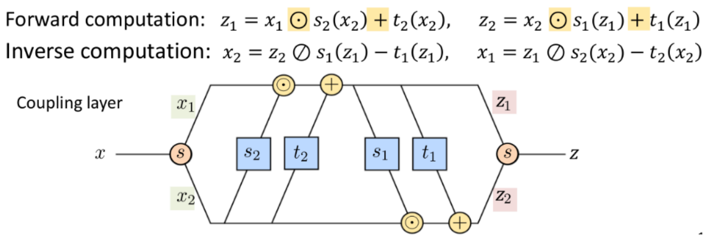

Forward computation: $z_1 = x_1 \odot s_2(x_2) + t_2(x_2)$, $z_2 = x_2 \odot s_1(z_1) + t_1(z_1)$
Inverse computation: $x_2 = z_2 \oslash s_1(z_1) - t_1(z_1)$, $x_1 = z_1 \oslash s_2(x_2) - t_2(x_2)$
Coupling layer
x
x_1
x_2
s
s_2
t_2
s_1
t_1
z_1
z_2
z

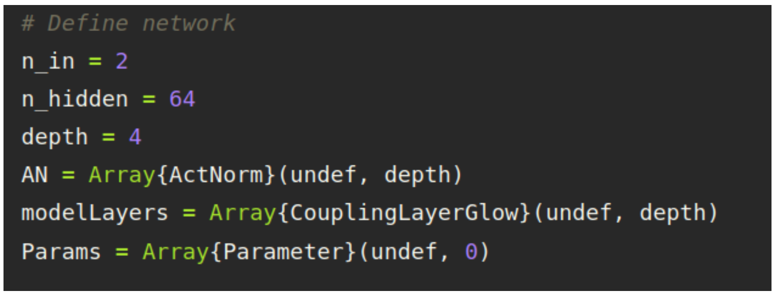
Figure 4.2: Affine Transformation.

An affine transformation is of the form y = A*x + b. This is the ”forward” transformation. The ”inverse” transformation is therefore x = A\(y-b)

I used "InvertibleNetworks.jl" and Flux libraries to implement my project.

InvertibleNetworks.jl: It is a machine learning package that includes invertible layers and networks. The invertibility allows back propagation through layers and networks without storing the forward state, which is recomputed on the fly when inverse propagating through it. This is the first package of its sort in Julia.

Flux: It is a machine learning library. It comes with many essential tools preinstalled, but we can also use the full power of the Julia language where we need it.



```
# Define network
n_in = 2
n_hidden = 64
depth = 4
AN = Array{ActNorm}(undef, depth)
modelLayers = Array{CouplingLayerGlow}(undef, depth)
Params = Array{Parameter}(undef, 0)
```

Figure 4.3: Defining Network for invertible structure.

CouplingLayerGlow creates a Real NVP-style invertible coupling layer based on 1x1 convolutions and a residual block.

Inputs of the CouplingLayerGlow:

n_in, n_hidden: Number of input and hidden channels.

k1, k2: Kernel size of convolutions in residual block. k1 is the kernel of the first and third operator, k2 is the kernel size of the second operator.

p1, p2: Padding for the first and third convolution (p1) and the second convolution (p2).

logdet: Bool to indicate whether to compute the logdet of the layer.

```
# Create layers
for j=1:depth
    AN[j] = ActNorm(n_in; logdet=true)
    modelLayers[j] = CouplingLayerGlow(n_in, n_hidden; k1=1, k2=1, p1=0, p2=0, logdet=true)

    # Collect parameters
    global Params = cat(Params, get_params(AN[j]); dims=1)
    global Params = cat(Params, get_params(modelLayers[j]); dims=1)
end
```

Figure 4.4: Creating layers with CouplingLayerGlow.

Data Flow in coupling layers:

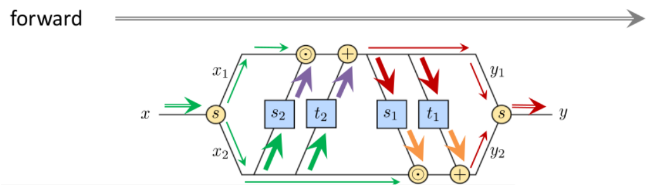


Figure 4.5: Data flow in forward direction.

```
# Forward pass of invertible structure
function forward(X)
    logdet = 0f0
    for j=1:depth
        X_, logdet1 = AN[j].forward(X)
        X, logdet2 = modelLayers[j].forward(X_)
        logdet += (logdet1 + logdet2)
    end
    return X, logdet
end
```

Figure 4.6: Forward pass of invertible structure.

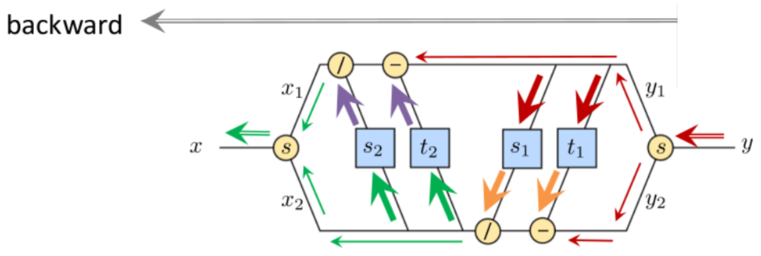
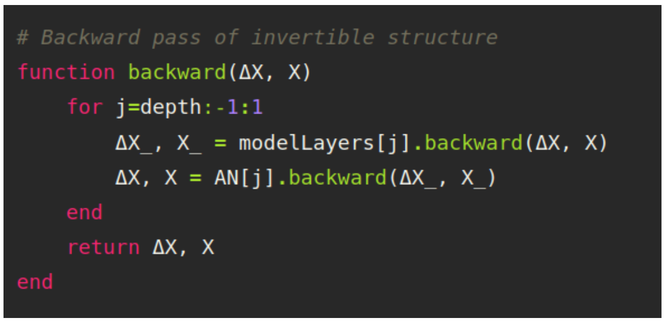


Figure 4.7: Data flow in backward direction.

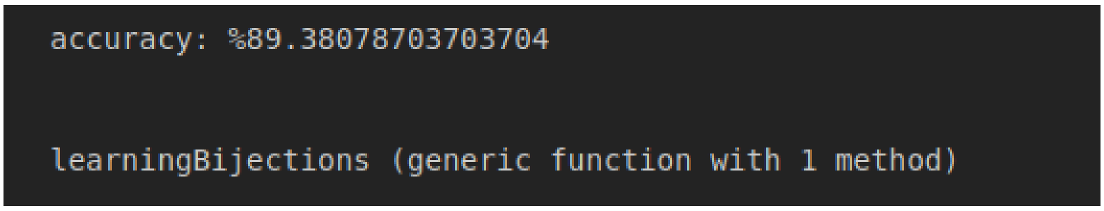


```
# Backward pass of invertible structure
function backward(ΔX, X)
    for j=depth:-1:1
        ΔX_, X_ = modelLayers[j].backward(ΔX, X)
        ΔX, X = AN[j].backward(ΔX_, X_)
    end
    return ΔX, X
end
```

Figure 4.8: Backward pass of invertible structure.

Accuracy Calculation:

I calculated the accuracy using 3457 different test data after training process. I used 10517 train data for my training operation. I used "onecold" method from Flux library to find the accuracy value. I calculated the accuracy value as %89.38



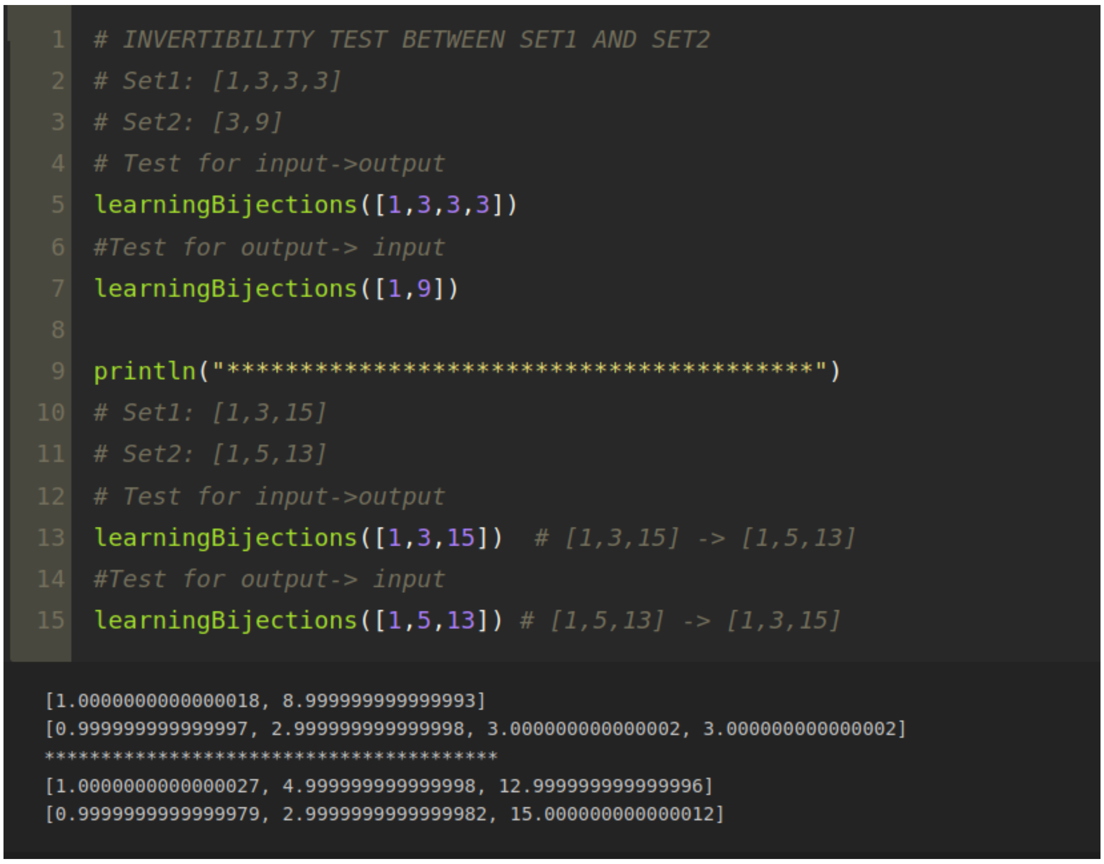
```
accuracy: %89.38078703703704


learningBijections (generic function with 1 method)
```

Figure 4.9: Accuracy with test data.

5. CONCLUSIONS

As a result of the implementation of my project, I saw that invertible networks can be used to understand the structure of bijection functions. Invertible Network architecture not only creates a certain output from the input, but also tries to recalculate the same input using this output. This is the main reason why I use the Glow model, which uses invertible networks in my project. I produced the dataset using the Integer Partition Theorem only with odd numbers, and it was very enjoyable to research and implement different mathematical theorems while creating the train and test data in the project. As a result of the project, I made an accuracy calculation using a test data other than the train data and reached the %89.38 accuracy value.

Test Results:



```
# INVERTIBILITY TEST BETWEEN SET1 AND SET2
# Set1: [1,3,3,3]
# Set2: [3,9]
# Test for input->output
learningBijections([1,3,3,3])
#Test for output-> input
learningBijections([1,9])

println("****************************************")
# Set1: [1,3,15]
# Set2: [1,5,13]
# Test for input->output
learningBijections([1,3,15])  # [1,3,15] -> [1,5,13]
#Test for output-> input
learningBijections([1,5,13]) # [1,5,13] -> [1,3,15]
```

```
[1.0000000000000018, 8.999999999999993]
[0.999999999999997, 2.999999999999998, 3.000000000000002, 3.000000000000002]
****************************************
[1.0000000000000027, 4.999999999999998, 12.999999999999996]
[0.9999999999999979, 2.9999999999999982, 15.000000000000012]
```

Figure 5.1: Test results.

BIBLIOGRAPHY

[1] H. Zhang, *Invariance and Invertibility in Deep Neural Networks*. 2020.

[2] P. A. Witte, M. Louboutin, A. Siahkoohi, and F. J. H. Gabrio Rizzuti, *InvertibleNetworks.jl – Memory efficient deep learning in Julia*. 2021.

[3] J. Behrmann, W. Grathwohl, R. T. Q. Chen, D. Duvenaud, and J.-H. Jacobsen, *Invertible Residual Networks*. 2019.

[4] L. Ardizzone, J. Kruse, S. Wirkert, *et al.*, *Analyzing Inverse Problems with Invertible Neural Networks*. 2019.

[5] N. Eriksen, *A Simple Bijection Between Lecture Hall Partitions and Partitions into Odd Integers*.

[6] B. Talay, *Yee's Bijective Proof of Bosquet-MÉLOU AND Eriksson's Refinement of the Lecture Hall Partition Theorem*. 2018.

[7] J. Behrmann, P. Vicol, K.-C. Wang, R. Grosse, and J.-H. Jacobsen, *Understanding and mitigating exploding inverses invertible neural networks*. 2020.

[8] U. Köthe, *Solving Inverse Problems with Invertible Neural Networks*. 2020.

[9] D. Guichard, *An Introduction to Combinatorics and Graph Theory*. 2021.

[10] H. S. Wilf, *Lectures on Integer Partitions*. 2000.