Machine Learning Methods on 2D Ising Model

Burak Civitcioglu
Project Supervisor: Prof. Rudolf A. Roemer
LPTM/CY Université
bcivitcioglu@cyu.etu.fr

October 1, 2020

Abstract

This report explores the study of Convolutional Neural Networks (CNN) prediction success on recognising the temperature by having the image of the configuration as an input for the Ising Model in 2D. Several studies have been done on transfer learning in various physical problems with promising results[14]. In the Ising Model case, we train a CNN on a square lattice and test it on a triangular lattice and vice versa; in order to conclude if the transfer learning methods work for the Ising model. The configurations are generated using Monte-Carlo Methods and Metropolis Algorithm and then Neural Networks are trained on square lattices of size 100×100 and the trained networks are tested on triangular lattices of the same size. Understanding the transfer learning for Ising model can help us discover if the transfer learning can be applied to any problem or not, moreover even if the transfer learning does not work, we can explore more on what the behaviour is when we try to do a transfer learning.

Contents

List of Figures

1 Introduction to Machine Learning

Machine learning can be considered to be a subset of artificial intelligence; consisting mainly of supervised learning, unsupervised learning and reinforcement learning [10]. In this project, our focus is the supervised learning. Supervised learning consists of mainly regression models, classification models and neural networks (NN). In this introduction, the goal is to present the basics of supervised learning starting from simple regression, explaining basic classification models and presenting the mathematical background of the structure of neural networks and deep neural networks (DNN).

We need to start with the basics in order to understand the basic concepts that is used in the more advanced topics such as DNNs and convolutional neural networks (CNN) which is the model that we use in this project.

1.1 Simple Linear Regression

Simple linear regression is suitable for data sets that are somewhat in a linear form. As we can see in Figure We can check the visuals in Figure 1. First let us start with getting familiar with the terminology and notation; along with some basic concepts that is important for all the machine learning models that will be covered.[5]

1.1.1 Model Representation

In a regression problem we are going to assume that we are presented with clean data that is somewhat resembling a straight-line like shape when plotted as a scatter plot. For simplicity and in order to develop intuition for the future models, we will be assuming a one dimensional model. Our data will be consisting of two variables. The number of data we have in this case will be called training examples m. Let us call these variables simply and conventionally, x and y. $(x^{(i)}, y^{(i)})$ will be used to represent ith target and feature.

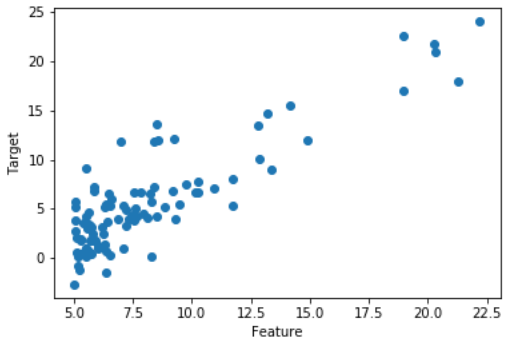

Figure 1: The scatter graph of our sample data

Figure 1 is the scatter graph of a sample data that is suitable for understanding a linear regression problem. We begin with a function of a straight line that will have the parameters to be found accordingly. The goal is to adjust the parameters so that the given data fits as good as possible. Let us define the hypothesis function as a straight line as follows:

$$h(x) = \theta_0 + \theta_1 x \tag{1}$$

where θ_i are the parameters that is to be determined. We define a function that will help us determine the parameters. This function is called the cost function. The cost function will be similar to the error when we try to fit the line to the data and therefore we will try to minimize the cost function.

1.1.2 Cost Function

The definition of the cost function will be an error expression that is independent of signs. The straightforward way of doing this is to take the square difference of the distance between our data points and the hypothesis that is our straight line. We sum over all the data points with the square difference expression; and if this function is minimum, it means that our hypothesis fits as good as possible given the conditions since the error is minimum.

First we subtract one from the other, $h(x^{(i)} - y^{(i)})$ and then we take the square so that we have a positive value, $h(x^{(i)} - y^{(i)})^2$. Now we sum over all the data points and we will get:

$$\sum_{i=1}^{m} \left(h(x^{(i)}) - y^{(i)}\right)^2$$

Conventionally for mathematical elegance, we will add a factor of $\frac{1}{2m}$. Thus the final version of the cost function is:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h(x^{(i)}) - y^{(i)}\right)^2 \tag{2}$$

Let us not forget the big picture. We want to minimize the cost function. Note that the value m is the number of total data points therefore it is constant. The values of $y^{(i)}$ are the data points therefore it is not a parameter. Therefore, what needs to be done is to find a pair of (θ_0, θ_1) that will minimize this function. Because these are the only values that are not known and the only values necessary to explicitly write down the hypothesis we started to look for in Equation 1.

1.2 Gradient Descent

In order to numerically minimize the cost function, we will use the method of gradient descent. Basic idea is to start at one random point, change it slightly to the decreasing direction; and check at each step if it is minimized. The formula needed is the following formula repeated until we converge to the minimum:

$$\theta_i := \theta_i - \lambda \frac{\partial}{\partial \theta_i} J(\theta_0, \theta_1) \tag{3}$$

where $i = 0, 1$, λ is the gradient descent parameter and the operator ":=" is not equality but the assignment operator; meaning that the right hand side of the operator is assigned

to the left hand side of the operator. In this expression, it basically means we update θ_i at each step with the given assignment operation. Equation 8 assures that when we update it, it will go into the direction of the extremum; which in this case always will be a minimum since the cost function is convex. We need simultaneous update of the parameters when we implement this method. The way to do that is the following considering t_0, t_1 to be temporary parameters for proper simultaneous update:

$$\begin{aligned} t_0 &:= \theta_0 - \lambda \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \\ t_1 &:= \theta_1 - \lambda \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \\ \theta_0 &:= t_0 \\ \theta_1 &:= t_1 \end{aligned} \tag{4}$$

The important points about this algorithm are:

1. If λ is too small, then it will take too much time to execute the algorithm.
2. If λ is too big then it will jump off the extremum value and it may not converge.
3. If the initial random point of the gradient descent is in fact the minimum, then we will not observe an update.
4. As the derivative of the cost function will approach to zero in time, we do not need to update λ. Nonetheless, we will take smaller gradient descent steps overtime.

1.2.1 The Implicit Calculation of the Gradient Descent

The important part that needs explicit calculation is the derivative part. Let us take the derivative of the cost function:

$$\frac{\partial}{\partial \theta_i} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_i} \left(\frac{1}{2m} \sum_{j=1}^{m} (h(x^{(j)}) - y^{(j)})^2 \right) \tag{5}$$

For $i = 0$ we have;

$$\frac{\partial J}{\partial \theta_0} = \frac{1}{m} \sum_{j=1}^{m} (\theta_0 + \theta_1 x^{(i)} - y^{(i)}) \tag{6}$$

For $i = 1$ we have;

$$\frac{\partial J}{\partial \theta_1} = \frac{1}{m} \sum_{j=1}^{m} (\theta_0 + \theta_1 x^{(i)} - y^{(i)}) x^{(i)} \tag{7}$$

More compactly:

$$\begin{aligned} \frac{\partial J}{\partial \theta_0} &= \frac{1}{m} \sum_{j=1}^{m} (h(x^{(i)}) - y^{(i)}) \\ \frac{\partial J}{\partial \theta_1} &= \frac{1}{m} \sum_{j=1}^{m} (h(x^{(i)}) - y^{(i)}) x^{(i)} \end{aligned} \tag{8}$$

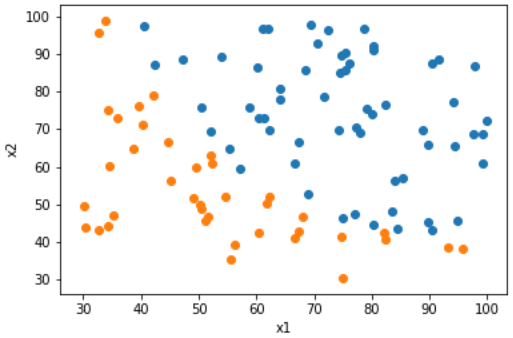

Figure 2: Scatter plot of a sample data set for logistic regression

Even though in the given example the cost function is indeed convex and thus will yield to a minimum extremum point when gradient descent is applied; in general, one needs to check whether the cost function is convex in order to conclude that the gradient descent will result in a minimum value.

1.3 Perceptron: Logistic Regression

The logistic regression is used to make classifications of two different types of data. Logistic regression works for the two-level classification; meaning that we will have only two possible outputs labeled as 0 and 1. We denote $y \in \{0, 1\}$

We want the hypothesis to be in the interval, $0 \leq h(x) \leq 1$. The following convention, although it is not used commonly as a first choice today, is the definition of a hypothesis: a sigmoid function.

$$\begin{aligned} h(x) &= \sigma(\theta^T x) \\ \sigma(x) &= \frac{1}{1 + e^{-x}} \end{aligned} \tag{9}$$

In this formalism, we will have the hypothesis function interpreted as the estimated probability that $y = 1$ on feature x and defined as:

$$h(x) = P(y = 1|x; \theta) \tag{10}$$

1.3.1 Decision Boundary

According to the above definition, the model will predict $y = 1$ whenever $h(x) \geq 0.5$. But substituting for sigmoid function in the hypothesis, we then get $y = 1 \rightarrow \theta^T x \geq 0$ Similarly, $y = 0 \rightarrow \theta^T x \leq 0$

Note that at zero we don't have a prediction. Then by finding these relations, we can plot the decision boundary. It can be a line, or a non-linear shape. For instance for a polynomial set of features, we will get a polynomial inequality relation therefore a non-linear decision boundary such as an ellipse or a circle. In order to find the decision boundary, we first need to define the cost function for the logistic regression.

1.3.2 Cost Function

Let us start by defining the sets and vectors that will be used in order to construct the logistic regression cost function. The training set will be:

$$T = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\} \tag{11}$$

The size of the training set is m. We similarly define:

$$x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix}, \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \cdots \\ \theta_n \end{bmatrix}, y = \{0, 1\} \tag{12}$$

and we also have $h(x) = \sigma(\theta^T x)$

In the logistic regression case, the cost function will be defined as an entropy expression which will be interpreted as an error expression. We will have:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} err(h(x^{(i)}), y^{(i)}) \tag{13}$$

with the $err(h(x), y)$ defined as:

$$err(h(x), y) = -y \ln(h(x)) - (1 - y) \ln(1 - h(x)) \tag{14}$$

Notice that this is in fact the entropy of a discrete two-level system found by using the micro-canonical partition function in statistical physics. The same procedure of gradient descent algorithm will be used for this cost function in logistic regression. The derivative that needed to be calculated in Equation 4, is:

$$\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} (h(x^{(i)}) - y^{(i)}) x_j^{(i)} \tag{15}$$

and we repeat this until convergence. Moreover, the vectorized version of this algorithm will be.

$$\theta := \theta - \frac{\alpha}{m} X^T (\sigma(X\theta) - y) \tag{16}$$

When this is applied on the data we are given in Figure 2, we will find the decision boundary as shown in Figure 3. Once the model allows us to find the decision boundary, then we check if the test data falls into which side; and we take the predictions of the model as that result.

1.4 Introduction to Neural Networks

Figure 4 is the very basic architecture of a single neuron. A neural network will receive inputs from the previous layer, and it will make the inputs go through an activation function.[12] The activation function can be chosen among many functions. For example we can go with

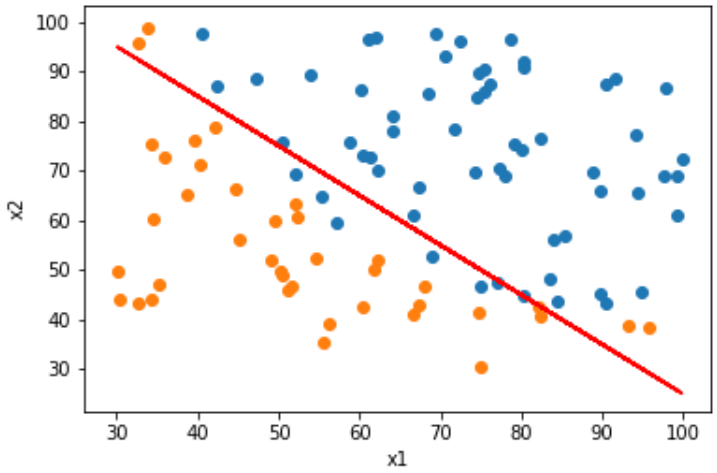

Figure 3: The decision boundary

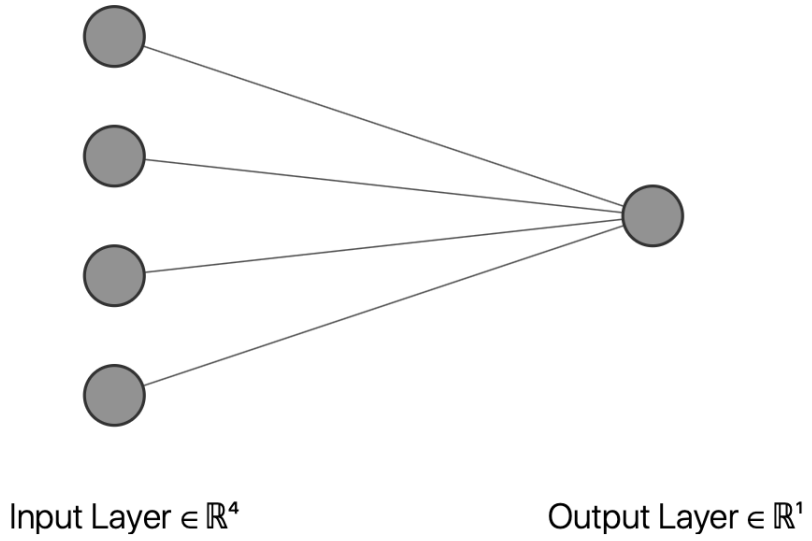

Figure 4: Perceptron

the sigmoid activation function, hyperbolic tangent function or more commonly used, ReLU function.[11]

$$h(x) = \text{ReLU}(x) = \max(0, x) \tag{17}$$

or

$$h(x) = \sigma(x) = \frac{1}{1 + e^{-\theta^T x}} \tag{18}$$

Figure 5 is a neural network with a more complicated architecture. The layers between

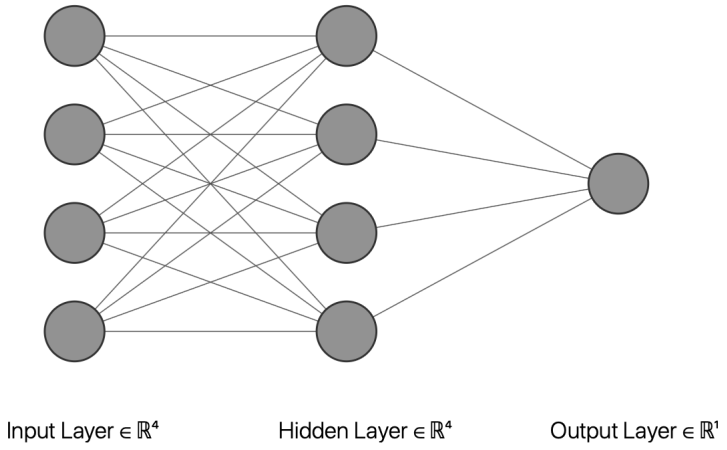

Figure 5: The visual representation of a deep neural network

the input layer and the output layer are called the hidden layers. We will define some terminology before we go deeper into the topic. $a_i^{(j)}$ is the activation of unit i in layer j. For example the second layer third unit's activation function would be denoted as $a_3^{(2)}$ being the weights of maps from layer j to layer $j+1$. We will denote the input layers with x's, and hidden layers with a's. Note that there is a bias unit, $x_0 = 1$.

Let us give an example calculation of the layers of this neural network. For instance if we want to calculate the second layer first unit, we will do the following operation:

$$a_1^{(2)} = \sigma[\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)}) x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3] \tag{19}$$

and similarly the output layer will be:

$$a_1^{(3)} = \sigma(\theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)}) \tag{20}$$

Therefore we can see that in order to calculate the output layer, we need all the values of the weights and all the values of the nodes of the hidden layers. Note that if the network's number of units in layer j is s_j then we have the matrix:

$$\theta^{(j)} \rightarrow s_j \times s_{j+1} \tag{21}$$

Therefore in order to calculate the hypothesis, we need to calculate all the values of all the nodes. They are defined by activation functions which depend on the weights. Therefore the thing we need to calculate here, is the weights. Once we have all the weights, then we can successfully calculate the hypothesis in the output layer. The way we calculate the weights is with forward and backward propagation.

1.5 Forward Propagation

We will again be considering the architecture in Figure 5. We have:

$$a_1^{(2)} = \sigma(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3) = \sigma(z_1^{(2)}) \tag{22}$$

and similarly we have

$$a_2^{(2)} = \sigma(z_2^{(2)}), a_3^{(2)} = \sigma(z_3^{(2)}) \tag{23}$$

then in order to calculate the output we have these functions. Then:

$$a_1^{(3)} = \sigma(z_1^{(3)}) \tag{24}$$

Now we will define the vectorized variables for simplification.

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}, z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} \tag{25}$$

We have then the following relations

$$\begin{aligned} a^{(1)} &= x \\ z^{(2)} &= \theta^{(1)} a^{(1)} \\ a^{(2)} &= \sigma(z^{(2)}) \end{aligned} \tag{26}$$

At this point we have a critical step. We add bias in every layer. Therefore we will have:

$$\begin{aligned} z^{(3)} &= \theta^{(2)} a^{(2)} \\ a^{(3)} &= \sigma(z^{(3)}) \end{aligned} \tag{27}$$

1.5.1 Example: Neural Networks of Logical Operators

First let us try to create a neural network architecture that will be equivalent to the logical AND operation. We will have two variables as inputs and one variable as an output.

$$\begin{aligned} x_1 & \\ x_2 \in \{0, 1\} & \\ y = x_1 \wedge x_2 & \end{aligned} \tag{28}$$

We will then have one output. We define the weights as $\theta = \begin{bmatrix} -30 \\ 20 \\ 20 \end{bmatrix}$ and therefore we will have the output layer as:

$$a_1^{(2)} = \sigma(-30 + 20x_1 + 20x_2) \tag{29}$$

We can now write down the logic table of this operation.

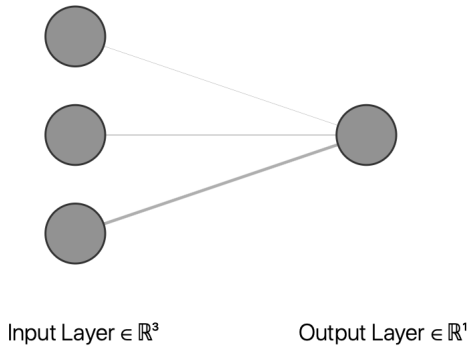

Figure 6: Logical AND operator as a neural network

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

Now let us try to understand the neural network for the logical OR operator. Similarly to the AND operator, we will have 2 input and 1 output nodes. Note that we will also have the bias added as another node. We change the weights as follows:

$$\theta = \begin{bmatrix} -10 \\ 20 \\ 20 \end{bmatrix} \tag{30}$$

This way we will end up with the following logic table.

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

As a final example let us do the NOT operator. This time we will have one input and one bias node; and one output node. With the following *theta* as our weights:

$$\theta = \begin{bmatrix} 10 \\ -20 \end{bmatrix} \tag{31}$$

And we end up with the expected logic table.

x_1	y
1	0
0	1

1.6 Multi-class Classification

When we have a problem of multi-class classification, we will have the possible outcomes denoted as,

$$h(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = y^{(3)} \tag{32}$$

when we have the third outcome. So we will have a set of fundamental basis; and the dimension will be the number of possible outcomes. We want to generalize the cost function of logistic regression. In this case we will have more than one possible outcome. Therefore the cost functions type will be the same, but we will add summations to cover all the possibilities.

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} err(h(x)) + R(\theta) \tag{33}$$

where error is defined as:

$$err(h(x)) = y_k^{(i)} \ln(h(x^{(i)}))_k + (1 - y_k^{(i)}) \ln(1 - h(x^{(i)})_k) \tag{34}$$

and $R(\theta)$ is defined as:

$$R(\theta) = \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_l+1} (\theta_{ji}^{(l)})^2 \tag{35}$$

Where L is the total number of layers. With s_l is the number of nodes in the layer l.

1.7 Back-propagation

At this point we have already formulated the cost function. Now we need to calculate the cost function and its derivatives. We need to compute $J(\theta)$, $\frac{\partial}{\partial \theta_{ij}^{(l)}}$. Assume that we only have one training example (x, y).

First we follow through the forward propagation.

1.7.1 Forward Propagation

Let us write down one by one each step of our nodes until we are in the output part which is going to be our hypothesis.

$$
\begin{aligned}
a^{(1)} &= x \\
z^{(2)} &= \theta^{(1)} a^{(1)} \\
a^{(2)} &= \sigma(z^{(2)}) + a_0^{(2)} \\
z^{(3)} &= \theta^{(2)} a^{(2)} \\
a^{(3)} &= \sigma(z^{(3)}) + a_0^{(3)} \\
z^{(4)} &= \theta^{(3)} a^{(3)} \\
a^{(4)} = \sigma(z^{(4)}) &= h(x) = \text{ the hypothesis}
\end{aligned}
\tag{36}
$$

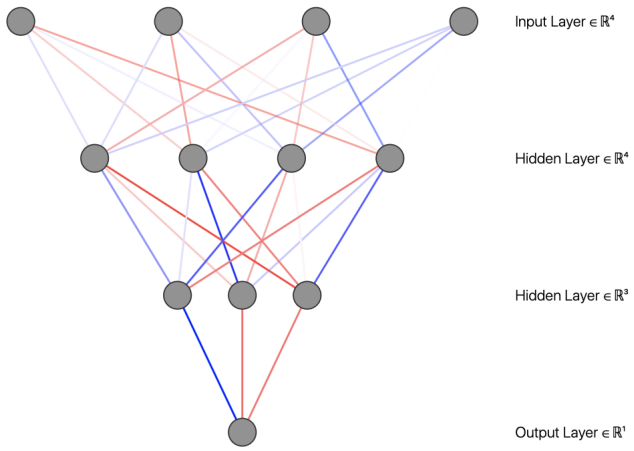

Figure 7: The neural network architecture used in the example for backpropagation

1.7.2 Backpropagation Step

At each time we do a forward propagation where we calculate $a^{(l)}$, we also calculate $\delta^{(l)}$ which is named as the error of nodes in layer l. But this time, we have to propagate backwards. We will start with the last one and then calculate the error of previous layers.[11]

$$
\begin{aligned}
\delta^{(4)} &= a^{(4)} - y \\
\delta^{(3)} &= (\theta^{(3)})^T \delta^{(4)} \odot g'(z^{(3)}) \\
\delta^{(2)} &= (\theta^{(2)})^T \delta^{(3)} \odot g'(z^{(2)})
\end{aligned}
\tag{37}
$$

The multiplication symbol is called the Hadamard Product and it basically means that we multiple the matrices element by element. We do not have any error in the first layer since it is the input layer. We have:

$$
\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = a_j^{(l)} \delta_i^{(l+1)} \tag{38}
$$

To sum up, for a training set $t = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$, we set initially $\Delta_{ij}^{(l)} = 0$ for all indices. Then we follow for all data points in the training set, the following set of operations, in the given order.

1. $a^{(1)} = x^{(i)}$
2. Forward Propagation for all $a^{(l)}$
3. Compute all $\delta^{(l)}$
4. $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

Now we have the necessary values to calculate finally the derivatives all at once. Finally we have for $j \neq 0$:

$$D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \theta_{ij}^{l} \tag{39}$$

and for $j = 0$

$$D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)} \tag{40}$$

These are the derivatives of the cost function. Now finally we have achieved to have both the cost function and all the necessary derivatives. After this point we will apply gradient descent and get the optimized hypothesis.

1.7.3 Dimensional Remarks

Let us have an architecture so that we have 10 nodes in the input layer, 1 hidden layer with 10 nodes and one output layer with 1 node. Then the dimensions will be:

$$\begin{aligned} d(\theta^{(1)}) = d(\theta^{(2)}) = d(D^{(1)}) = 10 \times 11 \\ d(\theta^{(3)}) = 1 \times 11 \end{aligned} \tag{41}$$

1.7.4 Training a Neural Network

Now let us sum up every step one needs to follow when training a neural network. Given that we have the problem given and we want to start doing machine learning on it for classification purposes.

1. Design a neural network architecture. Conventionally, all the hidden layers have the same number of nodes.
2. Randomly initialize the weights.
3. Implement forward propagation.
4. Compute the cost function.
5. Implement back propagation.
6. Get the derivatives of the cost function.
7. Apply gradient descent with respect to the weights.

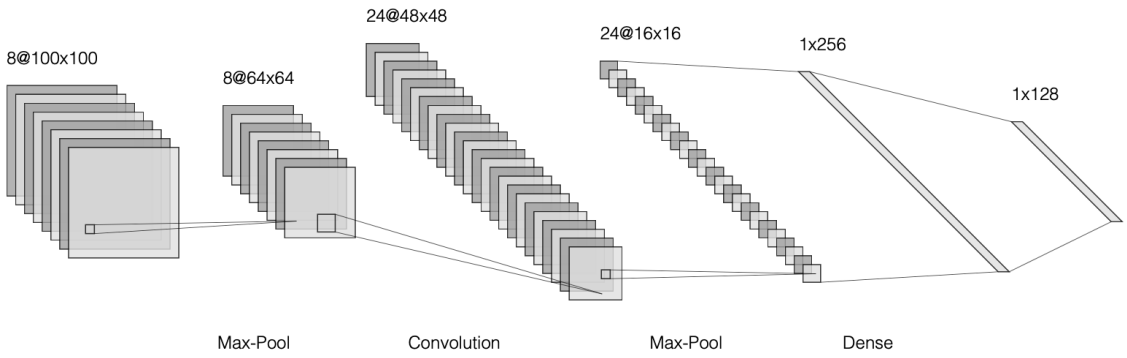

Figure 8: Sample layers of the CNN.

1.8 Convolutional Neural Networks (CNN)

In this project, we are only interested in a type of NN that is called CNN. CNNs are specifically used for the image type data. In our case we will use image snapshots of Ising configurations, and we will try to recognise the temperature from these images; therefore CNNs are the choice for the project [4].

CNNs, are similar to neural networks. They consist of neurons with weights and biases. Each neuron receives inputs, takes a weighted sum over the inputs, applies the activation function and produces an output. CNNs have a filter going through the image. As it goes through the whole image, the filter is altered at each step usually with a dot product operation [6]; resulting in a scalar value. Then we have the convolutional layer consisting of independent filters. For our comprehension, the convolutional layer basically learns different parts of images as seen in Figure 8 [9].

Another layer of CNNs is something called the Pooling layers. Pooling layer reduces the size in order to prevent ending up with too many parameters. Note that in machine learning applications we already have parameters on the scale of hundreds of thousands. For instance in Figure 10, we can see we have $47,460,807$ parameters, therefore it is important to have pooling layers to avoid having even more parameters.

1.9 Notes on Tensorflow/Keras Environment

The internship is done on the Keras environment that is built on tensorflow. Tensorflow is a library created specifically for machine learning and artificial intelligence models. Implementing these methods from scratch will be unnecessarily difficult and moreover, environments such as tensorflow made by the best computer scientists in the domain will have better optimizations and structure therefore it is more beneficial to learn these libraries and using them for our own purposes. [1]

2 The Ising Model in 2D

The Ising Model is one of the most studied models in physics, as it is really simple yet the critical phenomenon appears in two dimensions (2D). This project studies to apply machine learning methods on Ising Model in 2D to predict the temperatures from the images of the configurations. We start by briefly introducing the Ising Model.

2.1 Introduction

Ising Model we will study is a spin system that is on a lattice, where each site is either spin up or spin down. Therefore it is easy to represent binary. On a square lattice, we will attach up or down to each site. Moreover we will define an evolution by the following Hamiltonian[8]:

$$H(\sigma) = -\sum_{<i,j>} J_{ij}\sigma_i\sigma_j - h\sum_j \sigma_j \tag{42}$$

We also define the magnetization of an Ising configuration as:

$$m = \frac{1}{N}\sum_i \sigma_i$$

where σ_i's are the spins at site i, N is the total number of sites, h is the external magnetic fields parameter and J_{ij}'s are the interaction parameters.

The different values of parameters changes the behaviour of the Ising system's evolution. The configurations characteristics that we may expect is as follows:

$$J_{ij} > 0 : \text{ferromagnetic configuration}$$

$$J_{ij} = 0 : \text{non-interacting}$$

$$J_{ij} < 0 : \text{anti-ferromagnetic configuration}$$

and for the other parameter:

$$h > 0 : \text{tends to line up in the positive direction}$$

$$h < 0 : \text{tends to line up in the negative direction}$$

Note that Onsager's solution informs us that the critical temperature $T_c = 2.26$ is analytically calculated for 2D square Ising Model [7] [15].

2.2 Monte-Carlo Method for Collecting the Data

The generation of the Ising configurations are generated using Monte-Carlo Simulations, with Metropolis algorithm. The metropolis algorithm is an algorithm based on not only the Hamiltonian of the system but also on a random selection of spin flips. Metropolis

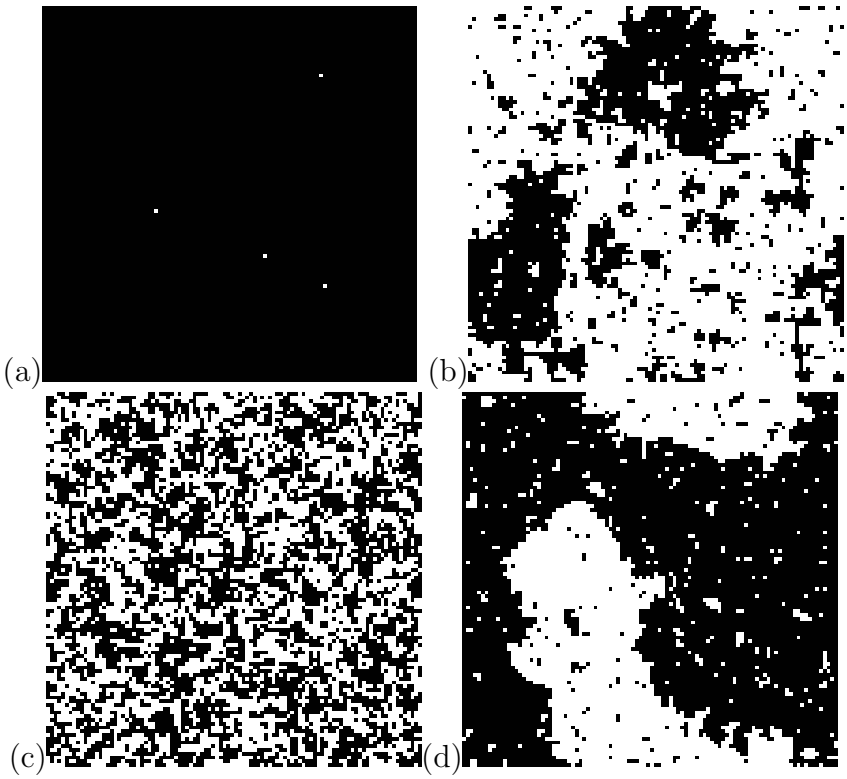

Figure 9: (a): A ferromagnetic ordered configuration. (b): A configuration at T_c. (c): A para-magnetic disordered configuration. (d): A ferromagnetic ordered configuration before it reached equilibrium.

algorithm generates valid Ising configurations and it can be explained simply as follows [16]:

Result: Metropolis Algorithm [22]
Pick a random site and calculate its energy;
Pick a random number $x \in (0, 1)$;
for all sites **do**
  if $\Delta E < 0$ OR $x < e^{-\frac{\Delta E}{k_B T}}$ **then**
    Flip spin;
  else
    Do not flip spin
  end
end

Algorithm 1: How to write algorithms

2.3 The Model Architecture

We are using a Convolutional Neural Network (CNN) that is a type of deep neural network and it is specifically used for image recognition. In our case, we will use image recognition techniques to recognise different phases and temperatures given configurations. The model we choose to work on is a rather complex model that is as given in Figure 10 [17]. Moreover, some studies have worked on identifying different phases using unsupervised learning instead

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 100, 100, 25)      700
_________________________________________________________________
batch_normalization_1 (Batch (None, 100, 100, 25)      100
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 100, 100, 25)      5650
_________________________________________________________________
batch_normalization_2 (Batch (None, 100, 100, 25)      100
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 50, 50, 25)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 50, 50, 25)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 50, 50, 75)        16950
_________________________________________________________________
batch_normalization_3 (Batch (None, 50, 50, 75)        300
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 50, 50, 75)        50700
_________________________________________________________________
batch_normalization_4 (Batch (None, 50, 50, 75)        300
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 25, 25, 75)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 25, 25, 75)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 46875)             0
_________________________________________________________________
dense_1 (Dense)              (None, 1000)              46876000
_________________________________________________________________
batch_normalization_5 (Batch (None, 1000)              4000
_________________________________________________________________
dropout_3 (Dropout)          (None, 1000)              0
_________________________________________________________________
dense_2 (Dense)              (None, 500)               500500
_________________________________________________________________
batch_normalization_6 (Batch (None, 500)               2000
_________________________________________________________________
dropout_4 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 3507
=================================================================
Total params: 47,460,807
Trainable params: 47,457,407
Non-trainable params: 3,400
_________________________________________________________________
```

Figure 10: The layers of the CNN that is used for all the trainings in the project.

of supervised learning that we are trying to do [19].

3 The Results of Training the Network

With the data that is generated, we create three data sets. First one is a data set with the greatest number of data: the training set. Second one is the cross validation data where we will check how good the training is before we actually test it on our test data. Third one is therefore the test data and it will consist of Ising configurations generated for a triangular lattice system. We explore if NNs in general are going to be sufficiently enough to make predictions [2].

3.1 Training the Network for Square Ising Lattice

First let us start by training the given model for a square Ising model in 2D, which will be tested in the next chapter for a triangular Ising data in order to see if it works on the triangular case. This is called the transfer learning and the goal is to understand if transfer learning works for the Ising model with square and triangular Ising configurations. [18]

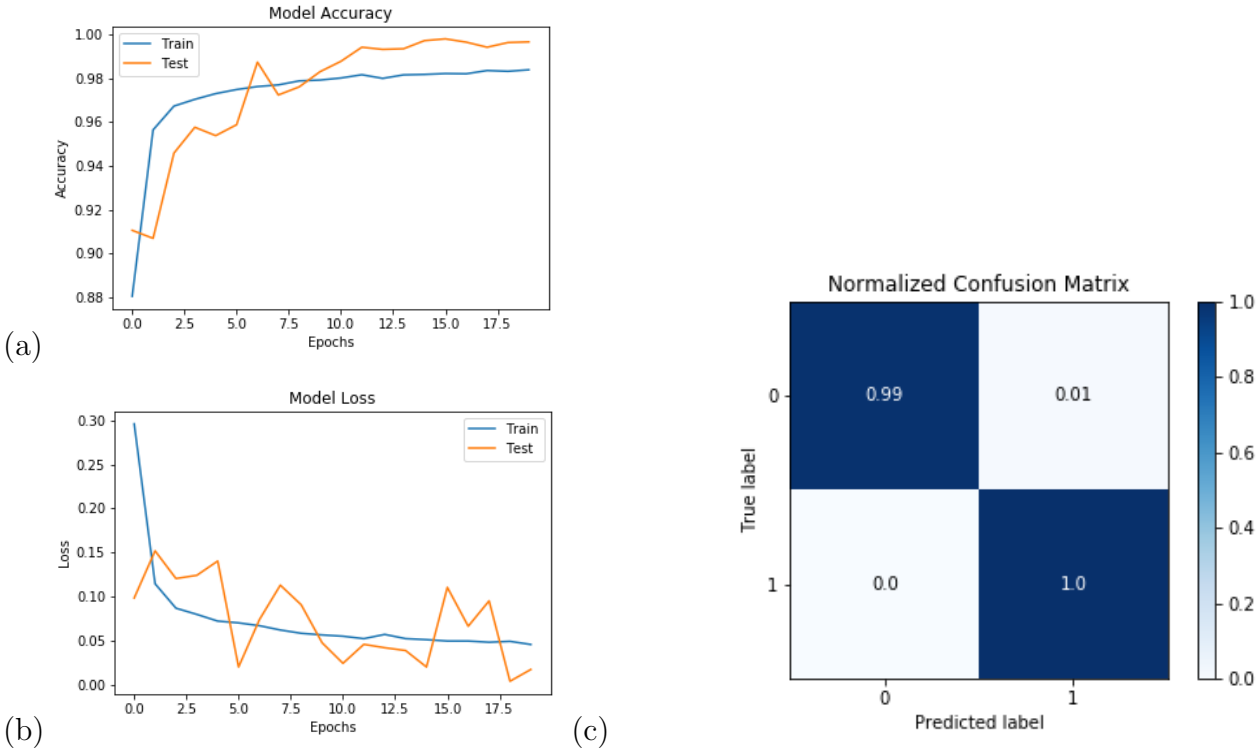

Figure 11: Results for the 2-class model with (a) training (blue) and validation (orange) loss (lines) and (b) accuracy as a function of the number of epochs as well as (c) the confusion matrix.

3.1.1 2-Class Classification

In the case of 2-Class, we achieve almost perfectly with the given convolutional neural network (CNN) architecture. Note that the CNN we are using is rather complex therefore we will see really high accuracy rates for smaller number of classes. [3]

Let us first present the loss function and accuracy with respect to the number of epochs. Note that one epoch is one training of all the data set once. In machine learning problems, we use many epochs, depending on the problem at hand and the number of data we have. We expect to see that the graph for training set and the cross validation set should be somewhat overlapping.

We can see in Figure 11 that we have somewhat correlated decrease for both loss functions. Note that in this case we have worked with 20 epochs; therefore the half epochs are only for visual convenience. Below we present the accuracy with respect to the number of epochs.

In Figure 11 it is visible that we do not need more than 20 epochs in this case since we are already in the 99% accuracy. Since the two are correlated, we do not need to change the hyper-parameters. Finally we need to see how well it actually does predictions by presenting the confusion matrix.

Finally in Figure 11 we have the confusion matrix. First let us clarify the confusion matrix and why it is one of the most important visual results one needs when working with machine learning models. The confusion matrix has the x-axis as the predicted labels and

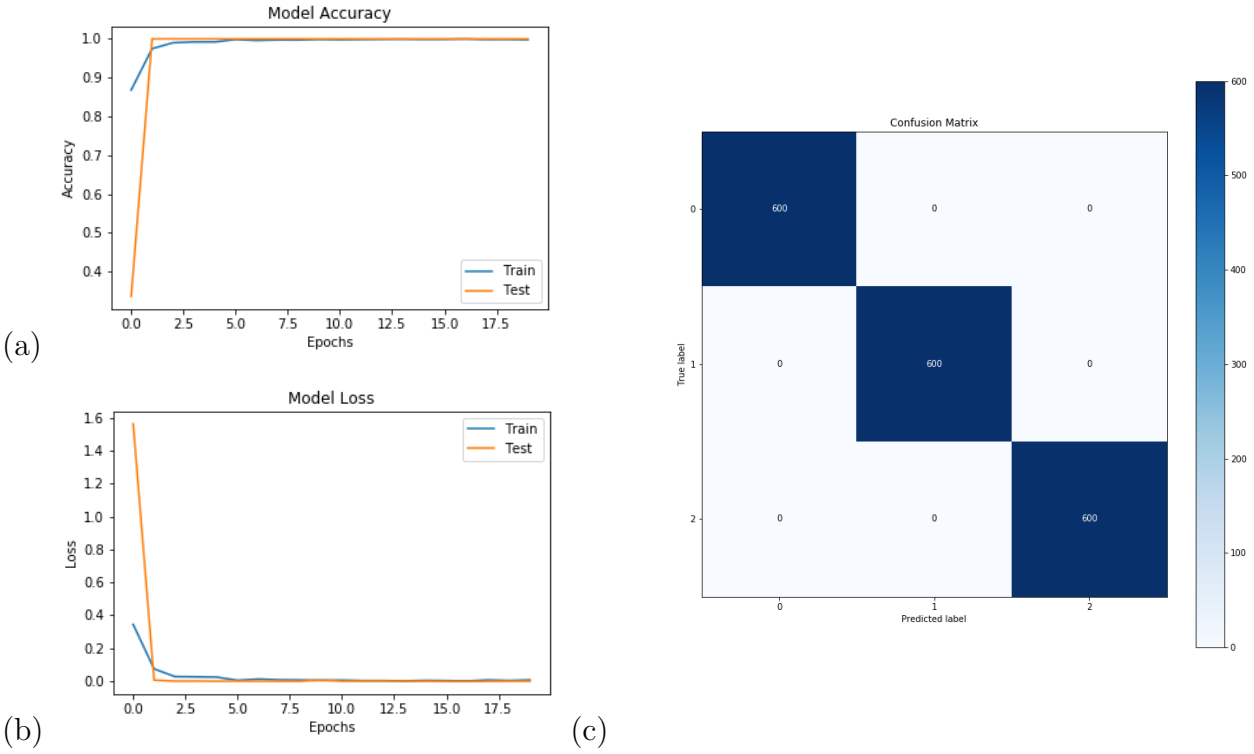

Figure 12: Results for the 3-class model with (a) training (blue) and validation (orange) loss (lines) and (b) accuracy as a function of the number of epochs as well as (c) the confusion matrix.

the y-axis as the true labels. Therefore we want the values to be dense in the diagonal. If we had the diagonal completely filled, this means that we have 100% accuracy.

In this specific visual , we can see that we have almost perfect score with just 0.01% mistake. This is an acceptable and actually a really good result for our model. Now we will increase the number of classes to see how well we can do with the most number of classes.

3.1.2 3-Class Classification

The 3-Class case is peculiar as we have achieved actually 100% accuracy. Our classes are chosen to be as far from one another as possible. Therefore we have the ordered class configuration as temperature $T = 1.0$, disordered class configuration as temperature $T = 4.0$ and the critical class configuration as $T_c = 2.26$. Below we start by presenting the loss function and accuracy as a function of number of epochs that is ran through the training.

One can see that the correlation between the training and the cross-validation sets are more than sufficient in Figure 12; therefore we do not need to adjust the hyper-parameters. Now we present the confusion matrix to see how well it actually worked and what mistakes it does.

In Figure 12 we have a perfect confusion matrix and we can observe that there is no mistakes done by our CNN. Therefore we can safely say that for 2-Class and 3-Class, our model works perfectly. We expect the accuracy to go down as we increase the number of

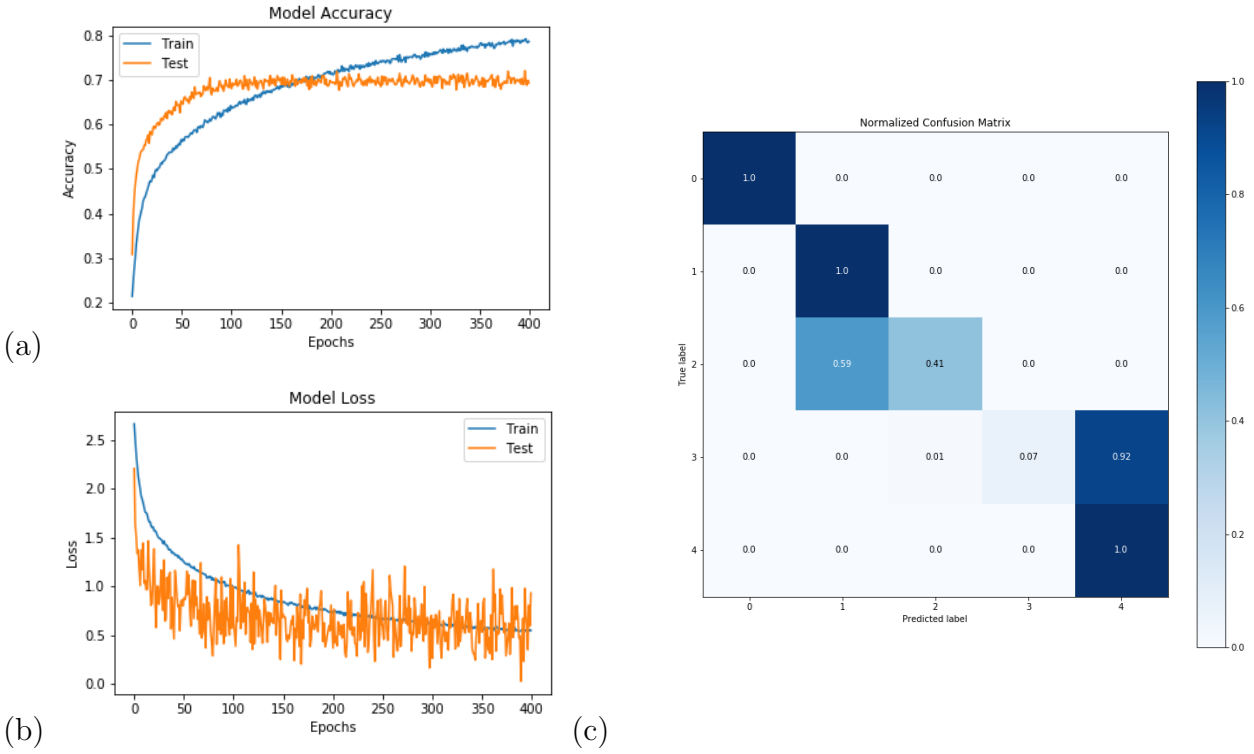

Figure 13: Results for the 5-class model with (a) training (blue) and validation (orange) loss (lines) and (b) accuracy as a function of the number of epochs as well as (c) The confusion matrix with the label 0 corresponding to $T = 1.0 - 1.4$, label 1 corresponding to $T = 2.70 - 2.11$, label 2 corresponding to $T = 2.24 - 2.28$, label 3 corresponding to $T = 2.45 - 2.49$ and finally label 4 corresponds to $T = 3.6 - 4.0$ with equal number of configurations in each class with a total of $25,000$ configurations divided as 80% training set and the remaining 20% cross validation set.

classes.

3.1.3 5-Class Classification

We present the 5-Class classification results. Passing from 3-Class to 5-Class we expect more changes and more confusion among the two classes above critical temperature, and among the two classes below the critical temperature.

As we can see below, the loss function is decreasing though fluctuates. In the end we can conclude that 400 epochs were enough since it is visible that the loss function stopped decreasing after some point. Then we can conclude that in order to increase the success of the model, we need to optimize the parameters or restructure the dataset in a better way; we cannot improve it by increasing the number of epochs anymore.

If we check the accuracy, it will be even more visible that the training has stopped getting better. The model is around 70% accuracy rate. If the confusion matrix is distributed around the diagonal matrix, then this accuracy rate is good enough for our purposes. If not, then the model needs revision.

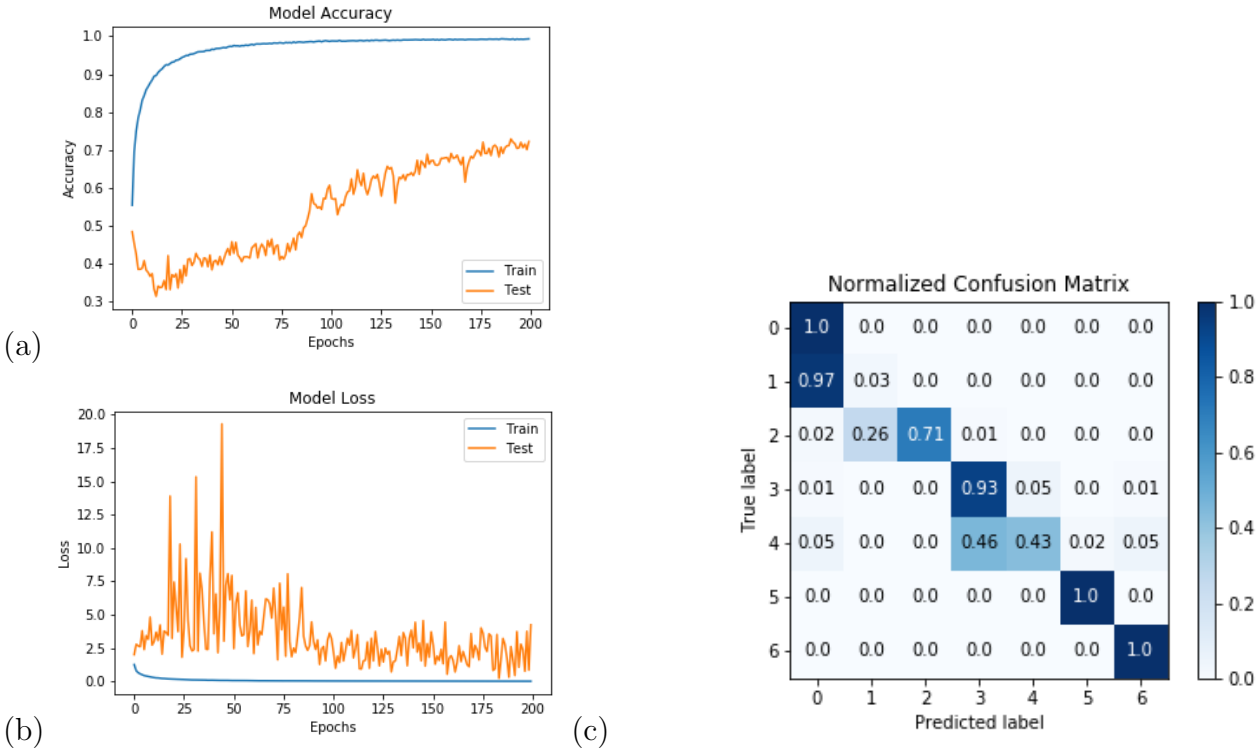

Figure 14: Results for the 7-class model with (a) training (blue) and validation (orange) loss (lines) and (b) accuracy as a function of the number of epochs as well as (c) The confusion matrix for 7-Class Case, 200 epochs and larger $\lambda = 0.00001$ training parameter, with $70,000$ data. Label 0 corresponding to $T = 1.0 - 1.4$, label 1 corresponding to $T = 2.09 - 2.5$, label 2 corresponding to $T = 2.12 - 2.16$, label 3 corresponding to $T = 2.24 - 2.28$, label 4 corresponding to $T = 2.38 - 2.42$,label 5 corresponding to $T = 2.53 - 2.60$, label 6 corresponding to $T = 3.6 - 4.0$ with equal number of data in each class.

In our case the confusion matrix in Figure 13 is distributed around the diagonal and it is visible that we have successfully classified the ferromagnetic cases. There is a significant confusion among the two para-magnetic cases, however that is expected because of the random nature of the para-magnetic configurations. The critical temperature's neighboring region is predicted correctly almost half the time.

3.1.4 7-Class Classification

In the 7-Class case, we achieve an optimum yet not perfect result. We will present two different training with different data structures and different number of data to see the significance of all the parameters plus we can see how the training parameter α changes the gradient descent drastically.

In the first case we have used $70,000$ data and we had the classification of the data as explained in the confusion matrix visual. The loss function and the accuracy is seem to be not very successful and this allows us to conclude that the training parameters should be optimized.

We can see in Figure 14 the model loss is fluctuating and not monotonically decreasing, therefore we can conclude that we need smaller training parameter since the gradient descent is probably jumping off the minimum and therefore never reaching it. The accuracy, on the other hand, is not correlated for the training and validation sets, therefore we can conclude that this model needs more hyper-parameter optimization.

Interestingly, as we have sufficiently enough data for this model, even though it is not optimized, we do have a rather acceptable confusion matrix in general. However we can observe that some mistakes done by the model are unacceptable since it confused the disordered para-magnetic high temperature configurations with low temperature ordered ferromagnetic configurations; and this is a mistake that one should not have; not even once.

The loss function in Figure 14 and as one can see, it fluctuates similarly to that of the 5-Class case. Yet it does not create a significant problem as long as it is overall decreasing in time. A smaller gradient descent parameter may be used to check if there is a jump happening, however since we observe an overall decrease, that may not be the case.

As the results could be improved, we change the training parameter and try to get a new training result; predicting it to be better than the one presented in Figure 14.

If we look at the accuracy of the model, we can see in Figure 15 the hyper-parameters are mostly fine as we observe a monotonic increase for both training and cross validation sets. After a while, the validation accuracy starts to stabilize, hinting that the number of epochs we have done, which is 400, is sufficiently enough. We achieve around 52% accuracy rate.

And finally, in the confusion matrix in Figure 15 we can see that we do not have any unacceptable mistake. It is dense in the diagonal, yet it is worth pointing out that this is not very good at predicting the critical temperature and its neighboring since it is only predicted correctly for the 5% success.

3.1.5 8-Class Classification

In the 8-Class case we achieve an acceptable yet not perfect results. In the first model have with a high training parameter $\lambda = 0.00001$ in Figure 16, we could not achieve any acceptable results. We conclude that the minimum of the cost function is being missed and jumped off by the gradient descent, therefore it never reaches to the minimum. As we can see in the figure, the validation accuracy stays the same and does not evolve and moreover the loss function keeps fluctuating for the validation set although it seems reasonably acceptable for the training set.

As a result, the confusion matrix in Figure 16 is not satisfactory. We can observe that the confusion in the higher temperatures are too many that there is almost no correct prediction in that region. We present non-normalized confusion matrix to see how many unacceptable configurations are predicted wrong. Though it seems like the highest and lowest temperatures are predicted correctly, we have absolutely unacceptable predictions such as temperature $T = 4.0$ is being predicted as $T = 1.0$ for 35 different configurations. There are 43 more false predictions in this region that are unacceptable. Therefore we try to remodel and make the results better.

Let us first start with the loss function again to observe how well we did. Note that in this case we have trained 400 epochs. As our classifications get more complicated, we need to decrease the training parameter λ; therefore it takes more time to reach the minimum of

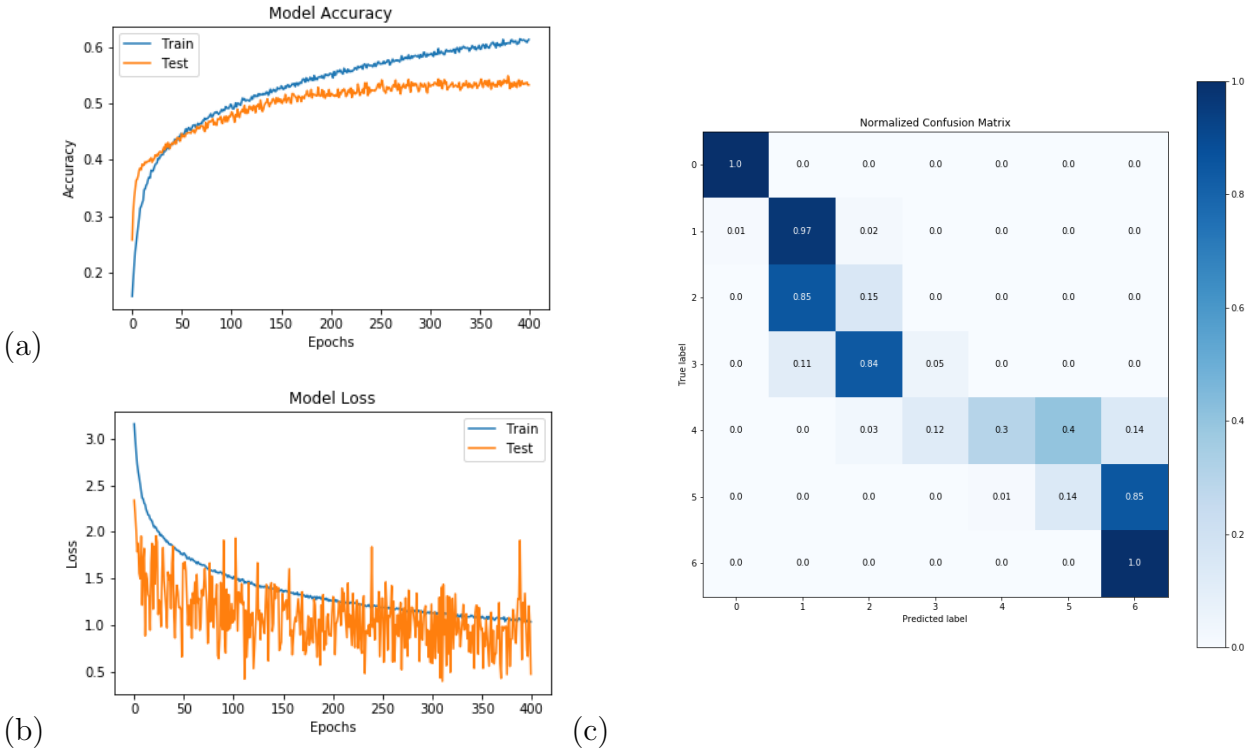

Figure 15: Results for the 7-class model with (a) training (blue) and validation (orange) loss (lines), The loss function for 7-Class Case, 400 epochs and smaller λ training parameter, with $35,000$ data. (b) accuracy as a function of the number of epochs as well as (c) The confusion matrix for 7-Class Case, 400 epochs and smaller $\lambda = 0.00000001$ training parameter, with $35,000$ data. Label 0 corresponding to $T = 1.0 - 1.4$, label 1 corresponding to $T = 1.6 - 2.0$, label 2 corresponding to $T = 2.06 - 2.1$, label 3 corresponding to $T = 2.21 - 2.25$, label 4 corresponding to $T = 2.26 - 2.3$,label 5 corresponding to $T = 2.46 - 2.50$, label 6 corresponding to $T = 3.6 - 4.0$ with equal number of data in each class.

the loss function.

The fluctuations in the loss function in Figure 17 of the validation set may be prevented with a more advanced hyper-parameter optimization techniques; however for our practical purposes as we could achieve enough computing power to have as many epochs as we did, it does not create a significant problem. Moreover, we could see below that the accuracy does not fluctuate as much as the loss function. As long as the fluctuations still follow along with the loss function of the training set, the model can be used.

The accuracy in Figure 17 increases with a smaller fluctuation that is visible, though the fluctuations are narrower than the fluctuations of the loss function. The reason is that the loss function does not by itself give us an idea about the success of our model, but it can only be interpreted by its monotonic decrease and ideally converging to the extremum point. By looking at the graph, we can comment that if we doubled the number of epochs we could indeed achieve a better result in the end. For our practical purposes and limited computation power caused by the time limit, we left it at 400 epochs with around 60% accuracy.

Observing the confusion matrix in Figure 17, we can say that even though we have

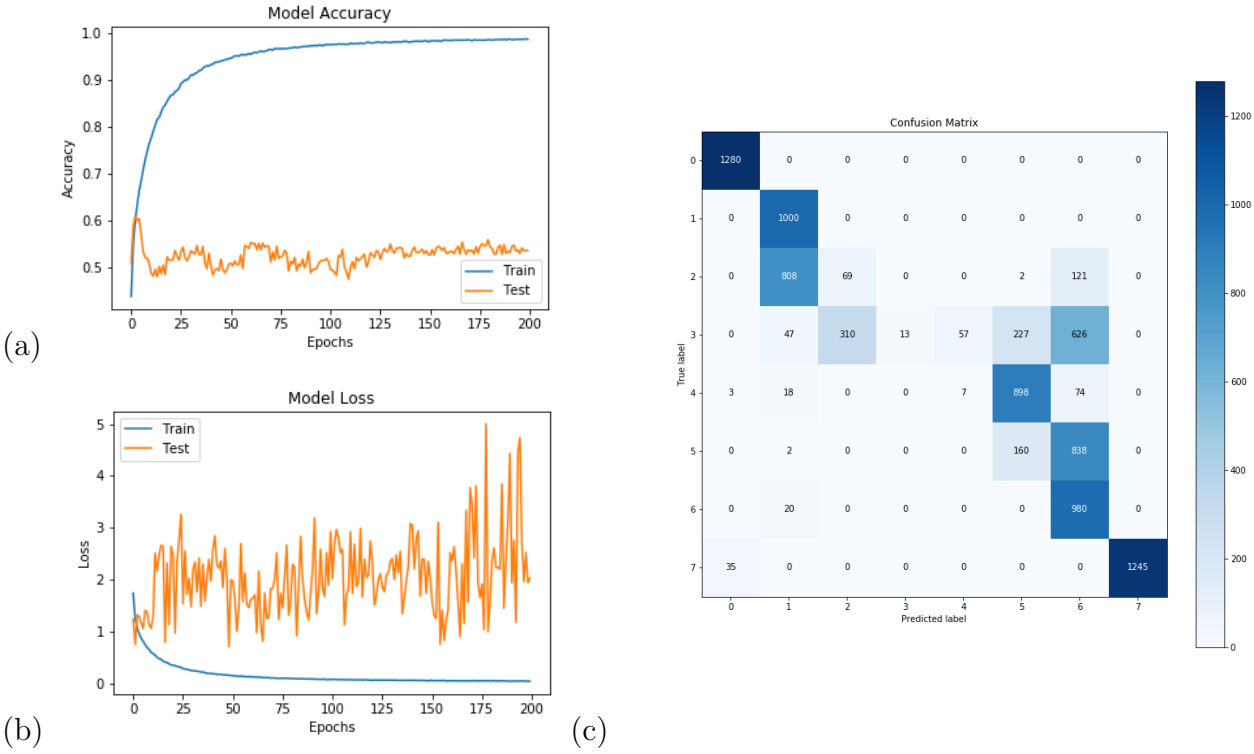

Figure 16: Results for the 8-class model with (a) Training (blue) and validation (orange) loss (lines) and (b) accuracy as a function of the number of epochs. Similar to the case of 8-Class, there is no correlation between validation and training results. (c) The confusion matrix of the 8-Class Model with the label 0 corresponding to $T = 1.0 - 1.4$, label 1 corresponding to $T = 2.09 - 2.5$, label 2 corresponding to $T = 2.12 - 2.16$, label 3 corresponding to $T = 2.24 - 2.28$, label 4 corresponding to $T = 2.35 - 2.39$,label 5 corresponding to $T = 2.45 - 2.49$, label 6 corresponding to $T = 2.59 - 2.9$ and finally label 7 corresponds to $T = 3.6 - 4.0$ with equal number of configurations in each class with a total of $70,000$ configurations divided as 80% training set and the remaining 20% cross validation set.

increased the number of classes and therefore narrowed the temperature range in each class, we can still get near-diagonal results where far ferromagnetic and far para-magnetic cases, meaning the Class 0 and the Class 7, are well predicted. However, the Class 3 in which the critical temperature lies, is not well predicted in this 8-Class model; with only 5% successful predictions.

3.2 Testing the Trained Network on a Triangular Ising Lattice

Transfer learning is when the machine learning model is trained in one system and it is tested on another system to see if the learning is indeed transferable or not. This is what we call the transfer learning. After we have trained the network on a square lattice, we will test it on triangular Ising configurations in 2D. What we try to understand is whether the transfer learning works for the two lattice systems, trained on a square. [21] [20]

In the Figure 18, we can observe a visual difference when compared to the square lattice

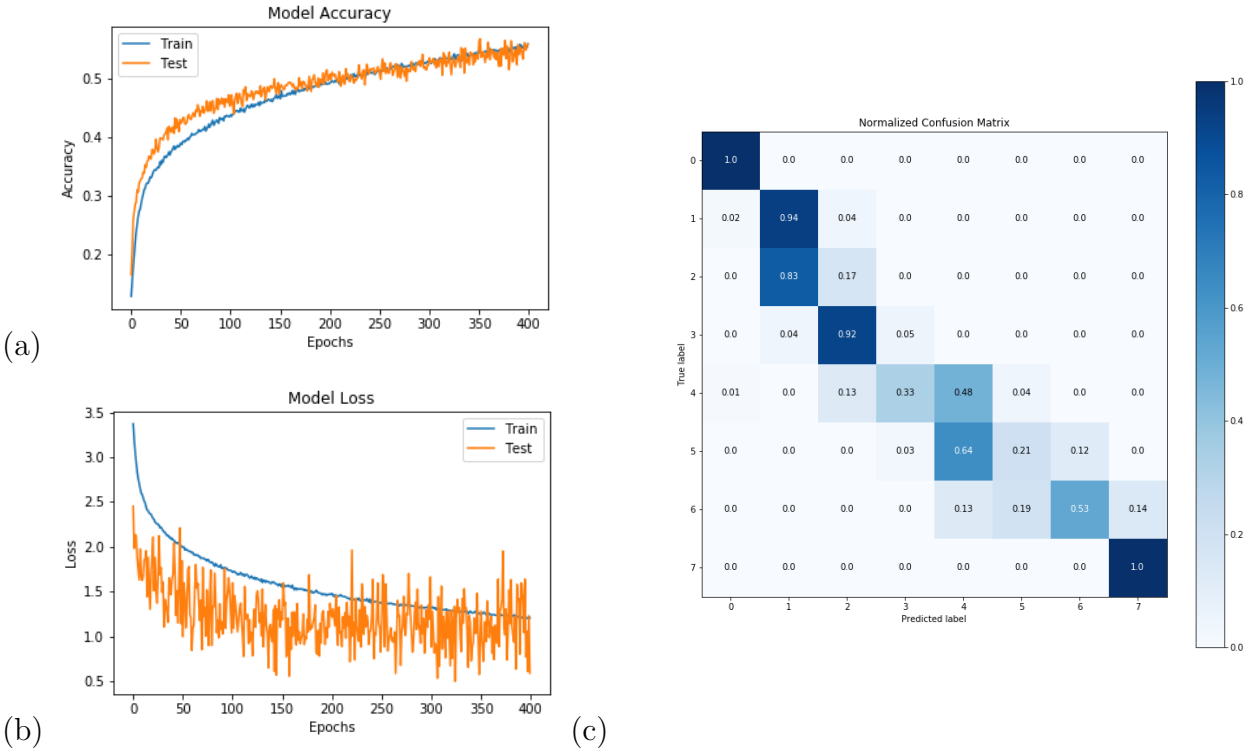

Figure 17: Results for the 8-class model with (a) Training (blue) and validation (orange) loss (lines) and (b) accuracy as a function of the number of epochs. Similar to the case of 8-Class, there is no correlation between validation and training results. (c) The confusion matrix of the 8-Class Model with the label 0 corresponding to $T = 1.0 - 1.4$, label 1 corresponding to $T = 2.09 - 2.5$, label 2 corresponding to $T = 2.12 - 2.16$, label 3 corresponding to $T = 2.24 - 2.28$, label 4 corresponding to $T = 2.35 - 2.39$,label 5 corresponding to $T = 2.45 - 2.49$, label 6 corresponding to $T = 2.59 - 2.9$ and finally label 7 corresponds to $T = 3.6 - 4.0$ with equal number of configurations in each class with a total of $40,000$ configurations divided as 80% training set and the remaining 20% cross validation set.

Ising configurations. Since we have in the triangular case 2 more interactions from upper left to bottom right of each lattice site. Therefore we can observe that the configurations are shaped as flows from upper right to bottom left.

3.2.1 2-Class Classification

In the 2-Class system, it seems like the transfer learning works perfectly and we can indeed train a network on a square lattice and observe that ferromagnetic and para-magnetic configurations are learnt. This is the basic case when we only have ordered and disordered classes as seen in Figure 19

In this configuration matrix it is visible that we have 100% accuracy and all the test set is classified correctly. Therefore we can conclude that for 2-Class systems, the learning is indeed transferable from a square lattice to a triangular lattice for Ising Model.

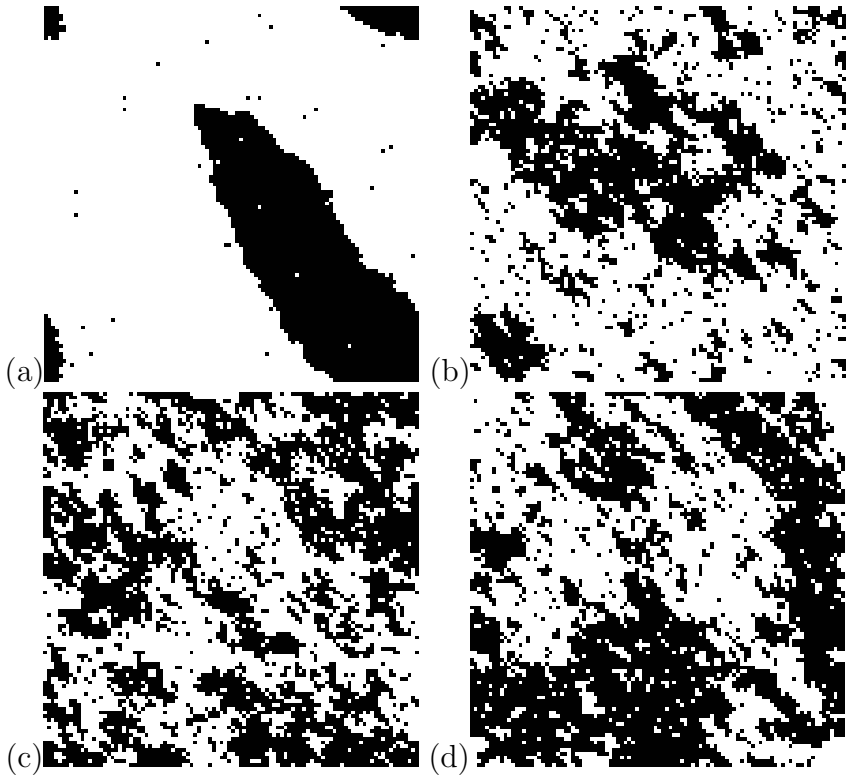

Figure 18: (a) A ferromagnetic triangular Ising configuration, (b) A critical temperature sample (c) A para-magnetic configuration (d) Another para-magnetic configuration

3.2.2 3-Class Classification

The 3-Class case is also almost perfectly predicted via transfer learning. This time we have the ordered case with temperature $T = 2.0 - 2.2$ and its neighboring, critical region where $T = 3.5 - 3.7$ and its neighboring and finally the para-magnetic region where $T = 4.3 - 4.5$ and its neighboring. The confusion matrix is presented in Figure 20

3.2.3 5-Class Classification

The 5-Class case shows us more conclusive results. We have an interesting property. The confusion matrix seems to be on one step down the diagonal yet keeping its diagonal shape, 21 on page 32. Let us call this structure "step-down diagonal structure" for future references. There are some possible reasons for this. In the square case where we train the network for 4 different interactions for each node; and when we test it for the triangular case, we have 6 interactions for each node. Note that these extra interactions have a direction; meaning that we only added cross interactions from upper left to bottom right. Moreover, as the critical temperature T_c increases for triangular case, going up to around $T_c \approx 3.6$, we will end up with a down-stepped diagonal since the critical label this time is label 3. And note that the critical temperature is not successfully predicted in this transferred learning model.

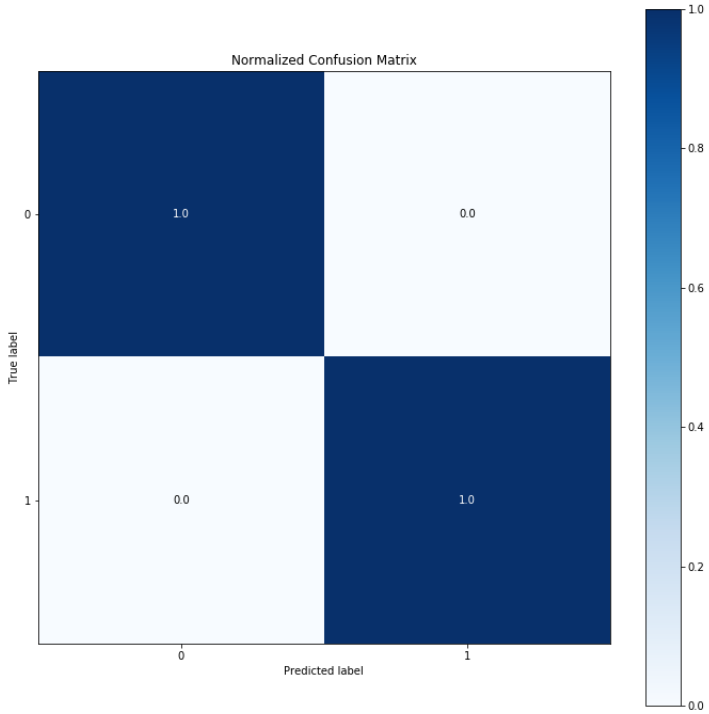

Figure 19: The confusion matrix of the 2-Class Model with the label 0 corresponding to $T < 3.0$, label 1 corresponding to $T > 4.0$, with equal number of configurations in each class with a total of $2,000$ configurations as the test set.

3.2.4 7-Class Classification

The 7-Class case of transfer learning supports the claims done in the previous section. We can see the confusion matrix in Figure 22 on page 33 We, again, do not have a well prediction of the critical temperature; however we do have correct predictions of ordered and disordered phases. And since the structure has still one step down diagonal structure, with further development, it may be possible to correct these; or it may be inherently nontransferable successfully for the Ising model.

3.2.5 8-Class Classification

The 8-Class case of transfer learning supports the claims done in the previous section too. We can see the confusion matrix in Figure 23 on page 34 The same property, step down diagonal structure, is visible and that we can see that the network is especially unsuccessful in the ordered phases. but this time we see that the disordered part is actually inclined to the diagonal structure, therefore the critical temperature is predicted better than the rest,

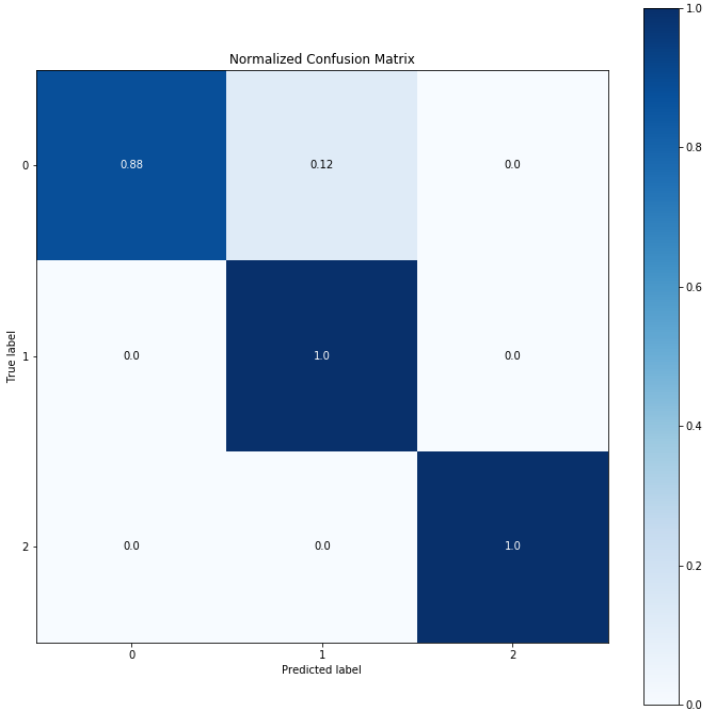

Figure 20: The confusion matrix of the 3-Class Model with the label 0 corresponding to $T = 2.0 - 2.2$, label 1 corresponding to $T = 3.5 - 2.7$, and label 3 corresponding to $T = 4.3 - 4.5$ with equal number of configurations in each class with a total of $9,000$ configurations as the test set.

yet it is not successful enough to claim that the transfer learning works.

3.3 Training the Network for Triangular Lattice

3.3.1 2-Class Classification

The 2-Class case is rather successful however since it is only ordered and disordered classes, even though one can achieve success, the result is limited. It is intuitively expected that distinguishing ordered and disordered configurations to be the easiest case. We can expect it to be successful in the second transfer learning we will do when we test it on a square lattice. We can check the visuals in Figure 24 on page 35.

3.3.2 3-Class Classification

The 3-Class case is surprisingly not as good as one would expect. It is not random but yet we can see the confusions between ordered and disordered, which is unacceptable. However

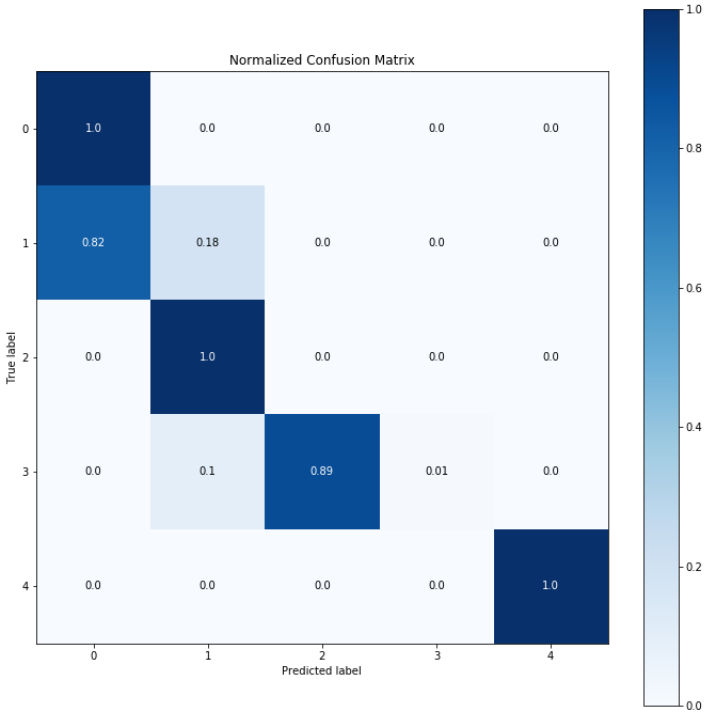

Figure 21: The confusion matrix of the 5-Class Model with the label 0 corresponding to $T = 2.0 - 2.2$, label 1 corresponding to $T = 2.4 - 2.6$, label 2 corresponding to $T = 2.9 - 3.1$, label 3 corresponding to $T = 3.6 - 3.8$ and label 4 corresponding to $T = 4.3 - 4.5$ with equal number of configurations in each class with a total of $15,000$ configurations as the test set.

we have a rather good predictions on critical temperature. The reason that it needs work is that we need way more data in order to have an actual conclusive result. We can check the visuals in Figure 25 on page 36.

3.3.3 5-Class Classification

In the 5-Class case we can observe that the critical and the disordered cases are predicted really well. However, the ordered case is not predicted correctly. Especially the first two ordered cases with $T = 2.0 - 2.6$, are predicted all as one class. This actually does not create a huge problem as long as we have well predictions on the critical temperature. If we have confusions amongst the ordered cases and amongst the disordered cases and not between the ordered and disordered cases, then we have an acceptable result. We can check the visuals in Figure 26 on page 37.

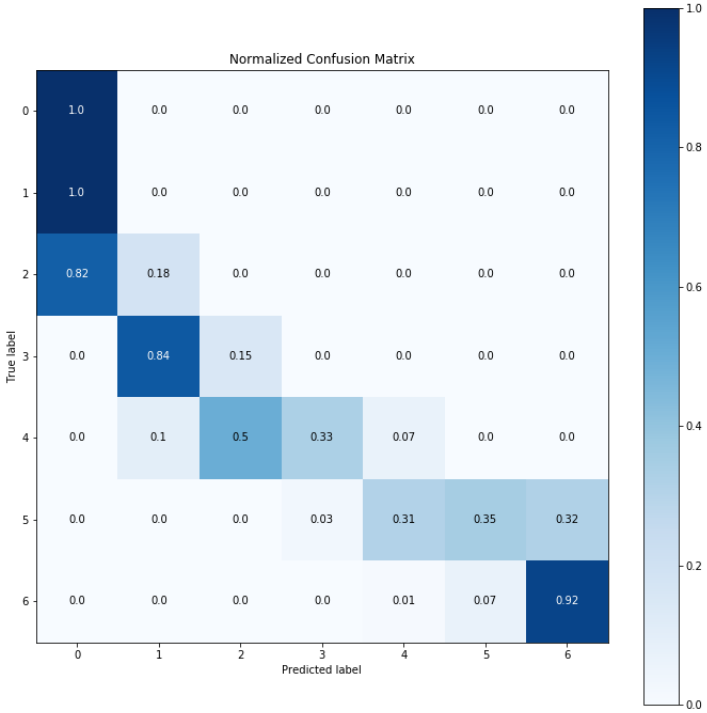

Figure 22: The confusion matrix of the 7-Class Model with the label 0 corresponding to $T = 2.0 - 2.2$, label 1 corresponding to $T = 2.4 - 2.5$, label 2 corresponding to $T = 2.8 - 2.9$, label 3 corresponding to $T = 3.2 - 3.3$, label 4 corresponding to $T = 3.6 - 3.7$, label 5 corresponding to $T = 4.0 - 4.1$ and label 6 corresponding to $T = 4.4 - 4.5$ with equal number of configurations in each class with a total of $14,000$ configurations as the test set.

3.3.4 7-Class Classification

7-Class case is one of the most promising ones in each training and test we did. The only issue is that on the triangular case the network cannot recognise the critical temperature explicitly. The rest of the results seem structured around the diagonal however the critical temperature is confused with the neighboring. One concludes that one needs more data. Note that the triangular training case has not as many data as the square training. Therefore we cannot be as conclusive in this case as we have been in the square case. We can check the visuals in Figure 27 on page 38.

The confusion matrix for 7-Class. We can check the visuals and confirm the claims done in the previous paragraph. The diagonal structure is one step down and even though some high and low temperatures are recognised well, the critical temperature is completely missed.

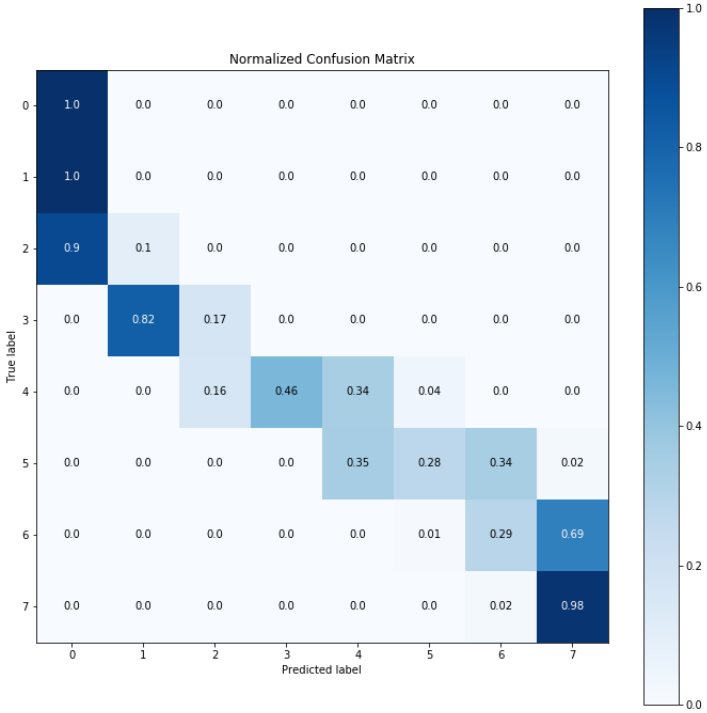

Figure 23: The confusion matrix of the 8-Class Model with the label 0 corresponding to $T = 2.0$, label 1 corresponding to $T = 2.4$, label 2 corresponding to $T = 2.8$, label 3 corresponding to $T = 3.2$, label 4 corresponding to $T = 3.7$, label 5 corresponding to $T = 3.9$, label 6 corresponding to $T = 4.2$ and label 7 corresponding to $T = 4.5$ with equal number of configurations in each class with a total of $8,000$ configurations as the test set.

3.3.5 8-Class Classification

Training on the 8-Class case is similar to the rest of the results and similar to the transfer learning tested on the triangular case. But interestingly, we have this time accuracy of the validation set higher than the accuracy of the training set. This is a bizarre phenomenon and usually unexpected. The loss function on the other hand seems correlated between the training and validation; therefore we can conclude that the training parameters are well optimized. We can check the visuals in Figure 28 on page 39.

The confusion matrix seems to be rather good around the ordered case this time. Interestingly, we have a specific issue with the critical temperature. The current model does not recognise the critical temperature at all and it is confused with the closest neighboring labels. This hints us that we definitely need more data.

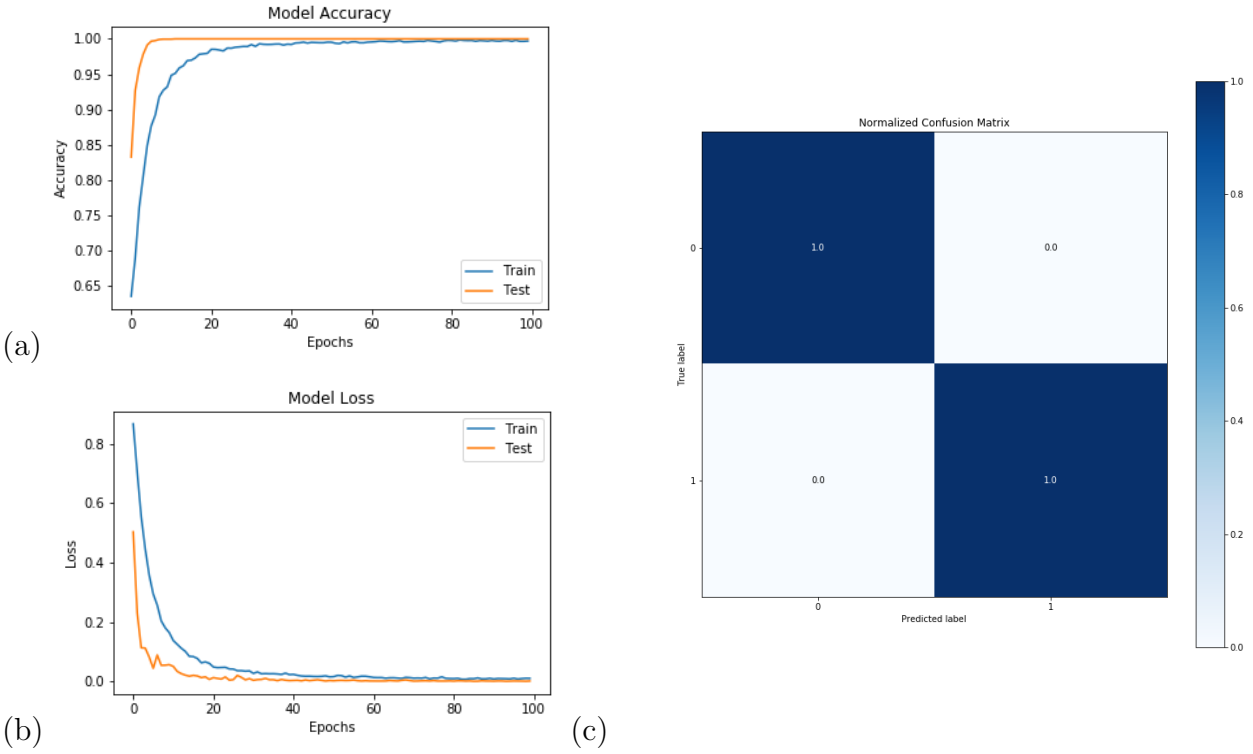

Figure 24: Triangular training on 2-Class. (a) Training (blue) and validation (orange) loss (lines) and (b) accuracy as a function of the number of epochs. The confusion matrix of the 2-Class Model with the label 0 corresponding to $T = 2.0 - 2.5$, label 1 corresponding to $T = 4.0 - 4.5$, with equal number of configurations in each class with a total of $12,000$ configurations as the training and validation set.

3.4 Testing the Trained Network on a Square Ising Lattice

In this section we will do the other case where we train the network on a triangular lattice and test the transfer learning on a square lattice. This time, different than the previous case, we expect to see a diagonal structure but this time we expect the diagonal to be one step up as opposed to the previous case being one step down.

3.4.1 2-Class Classification

In the 2-Class case we, as expected, achieve around 98% accuracy and for the case of ordered and disordered classes, the transfer learning works both ways. We can check the confusion matrix in Figure 29 on page 40.

3.4.2 3-Class Classification

The 3-Class case of the transfer learning has the accuracy of 91%. The critical temperature has predicted perfectly; however there is an unacceptable confusion between the ordered and disordered cases. Therefore we conclude we need more data in order to achieve an actual success. We can check the confusion matrix in Figure 30 on page 41.

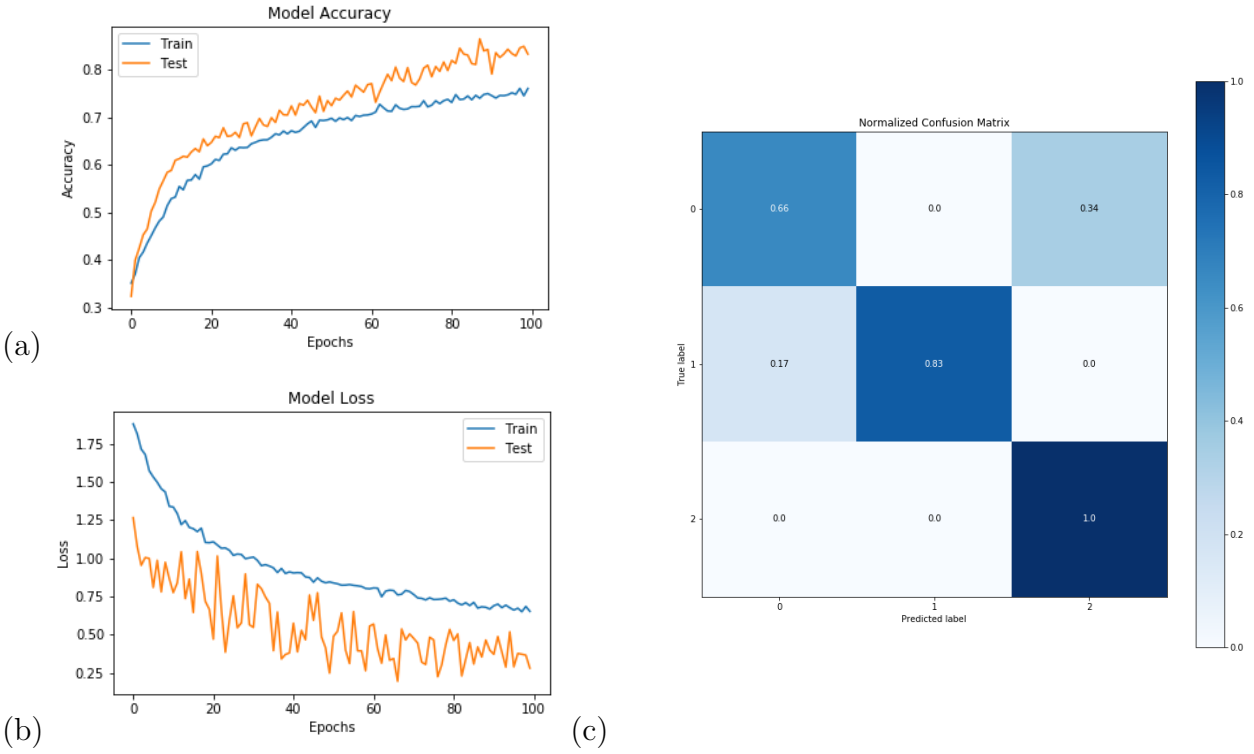

Figure 25: Triangular training on 3-Class. (a) Training (blue) and validation (orange) loss (lines) and (b) accuracy as a function of the number of epochs. The confusion matrix of the 3-Class Model with the label 0 corresponding to $T = 2.0 - 2.2$, label 1 corresponding to the critical temperature and label 2 corresponding to $T = 4.2 - 4.5$, with equal number of configurations in each class with a total of $9,000$ configurations as the training and validation set.

3.4.3 5-Class Classification

5-Class classification shows us the first example of our prediction about the one step up diagonal structure of the confusion matrix. The recognition of the ordered case is achieved with 100% with overall accuracy of 45%. Note that this time because of the difference between the critical temperatures of triangular and square lattices; and the data being structured differently, we have the critical region as the last label. Therefore we do have good predictions on critical temperature; yet the critical temperature is intertwined with the disordered cases. In order to have better conclusive results, we need not only more data for the existing data points, but also we need more data points; meaning that we need more configurations of different temperatures. We can check the confusion matrix in Figure 31 on page 42.

3.4.4 7-Class Classification

7-Class case we have more visible step-up diagonal structure. We do not see any unacceptable order-disorder confusion and we have better results in the ordered case. Considering the critical temperature, we have 23% accuracy but on considering the step-up diagonal

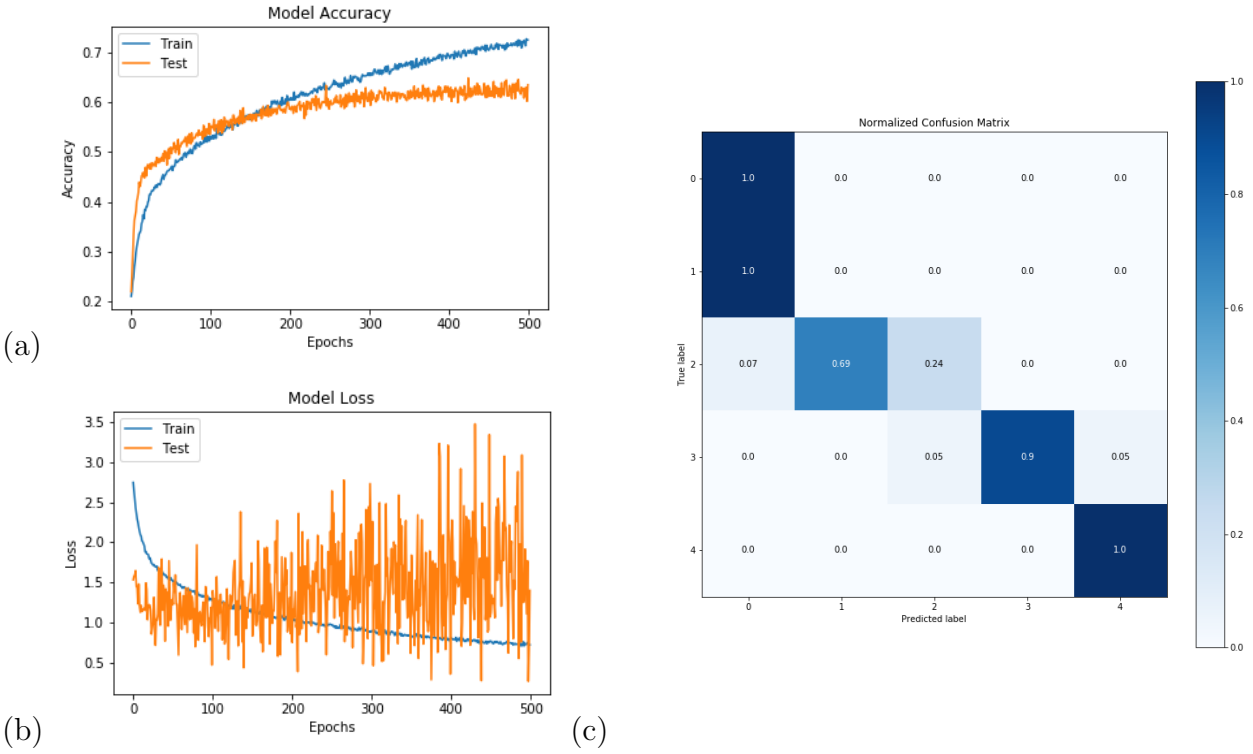

Figure 26: Triangular training on 5-Class. (a) Training (blue) and validation (orange) loss (lines) and (b) accuracy as a function of the number of epochs.The confusion matrix of the 5-Class Model with the label 0 corresponding to $T = 2.0 - 2.2$, label 1 corresponding to $T = 2.4 - 2.6$, label 2 corresponding to $T = 2.9 - 3.1$, label 3 corresponding to $T = 3.6 - 3.8$, and label 4 corresponding to $T = 4.3 - 4.5$, with equal number of configurations in each class with a total of $15,000$ configurations as the training and validation set.

structure, we may conclude more from this; though not in a straightforward way. In terms of number of data, the result investigated on the confusion matrix and feasibility, the 7-Class result is the optimum result we have in this section. We can check the confusion matrix in Figure 32 on page 43.

3.4.5 8-Class Classification

8-Class case is the case where we have not sufficient number of data in the training therefore the results are not as well as we had in the 7-Class case. The ordered case is well predicted however in the temperatures around $T \approx 2.0$ and $T \approx 2.5$ are not predicted correctly at all. With the sufficient training data, we expect to see the one step-up diagonal structure in this case too. We can check the confusion matrix in Figure 33 on page 44.

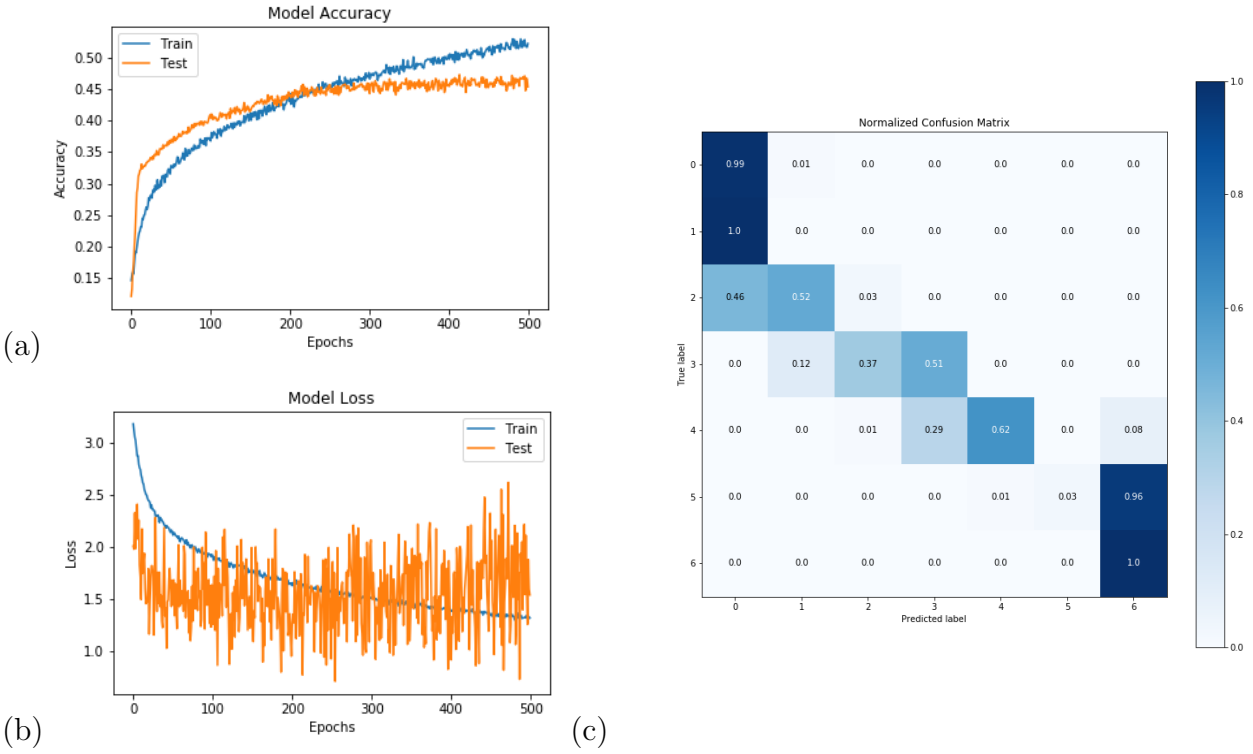

Figure 27: Triangular training on 7-Class. (a) Training (blue) and validation (orange) loss (lines) and (b) accuracy as a function of the number of epochs.The confusion matrix of the 7-Class Model with the label 0 corresponding to $T = 2.0 - 2.2$, label 1 corresponding to $T = 2.4 - 2.5$, label 2 corresponding to $T = 2.8 - 2.9$, label 3 corresponding to $T = 3.2 - 3.3$, label 4 corresponding to $T = 3.6 - 3.7$, label 5 corresponding to $T = 4.0 - 4.1$ and label 6 corresponding to $T = 4.4 - 4.5$ with equal number of configurations in each class with a total of $14,000$ configurations as the training and validation set.

4 Conclusions

During this project and the internship, I have learnt more deeply and thoroughly about the mathematical background and theoretical models of machine learning techniques. Moreover, applying these techniques for a real scientific problem by learning the necessary machine learning libraries such as Keras/TensorFlow, has helped me gain experience in the subject and helped me understand my domain of interests in physics.

The Ising configurations are generated by the code written as the first part of the internship and the code is executed in the cluster; once we checked the validity of the configurations. After we have started to work on the machine learning modeling part and hyper-parameter optimization, which requires high GPU power, there was some timing problems. Even though we have used the GPU's provided by the university computers, the time limitation of the internship has caused us to limit the amount of data to be trained and the amount of training epochs that we could do.

This internship was aimed to explore transfer learning for Ising Model. The network training for square and triangular models are done and tested on one another too see if

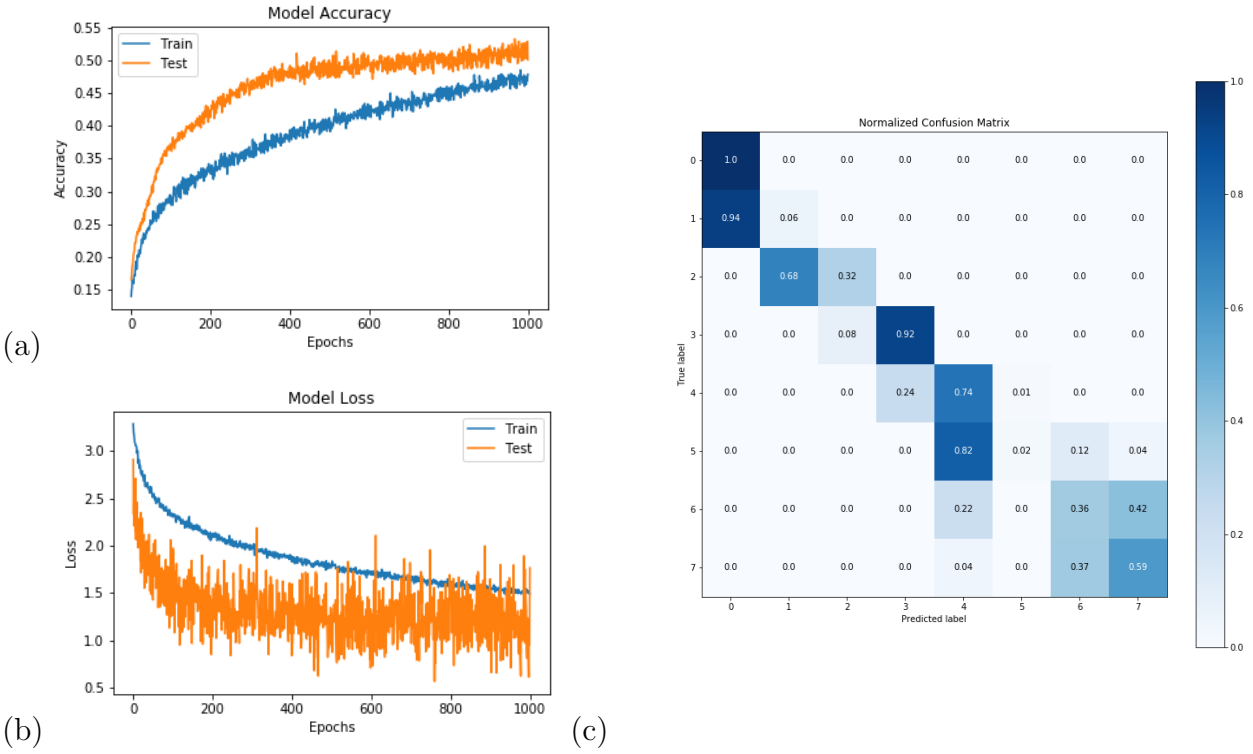

Figure 28: Triangular training on 8-Class. (a) Training (blue) and validation (orange) loss (lines) and (b) accuracy as a function of the number of epochs.The confusion matrix of the 8-Class Model with the label 0 corresponding to $T = 2.0$, label 1 corresponding to $T = 2.4$, label 2 corresponding to $T = 2.8$, label 3 corresponding to $T = 3.2$, label 4 corresponding to $T = 3.7$, label 5 corresponding to $T = 3.9$, label 6 corresponding to $T = 4.2$ and label 7 corresponding to $T = 4.5$ with equal number of configurations in each class with a total of $8,000$ configurations as the training and validation set; divided so that 80% is the training set.

the transfer learning works. We have concluded that for the small number of classification problems, namely 2-Class and 3-Class, it works sufficiently enough. However, for the higher number of classifications, namely 5-Class, 7-Class and 8-Class, it does not work. We observe step down and step up diagonal structure. We also observe in some cases the critical phenomenon predicted with a sufficiently enough percentage as suggested in [9].

Moreover it is possible to approach the problem with clustering methods of unsupervised learning methods; however since we are indeed able to generate as many configurations as we need, it fits to use supervized models [19]

4.1 Possible Future Work

In order to further understand the potential reasons for the one step up/down diagonal structure that is observed, one could go on to study more. The generated data should be more than we have used for both square lattice and triangular lattice with the same number of data points. For thorough analysis, the opposite case of interaction for another triangular

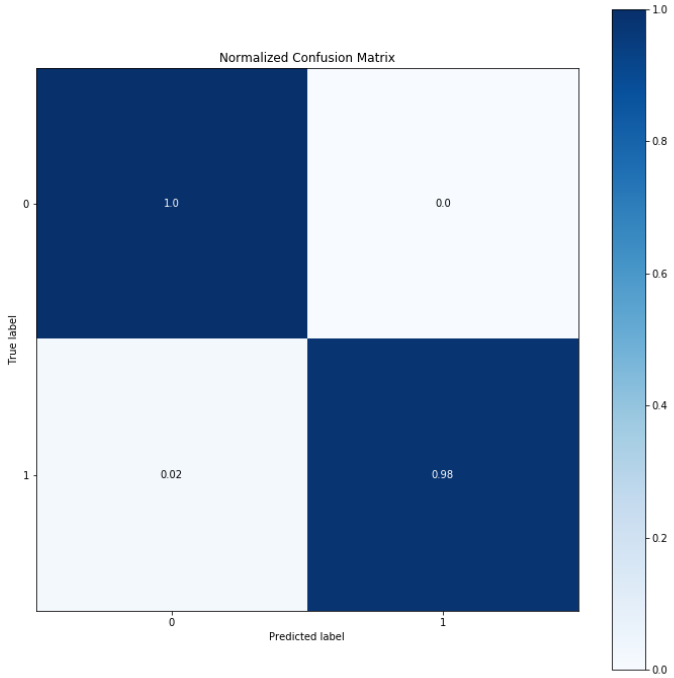

Figure 29: The confusion matrix of the 2-Class Model of the transfer learning of triangular network on square Ising model with the label 0 corresponding to ordered, label 1 corresponding to disordered cases with equal number of configurations in each class with a total of $60,000$ configurations as the test set.

case should also be introduced to see if there is the same result if we change the direction of the cross interaction for the triangular case. And moreover, it would be helpful to have both directions of interactions included in another triangular case. This time we may expect to have a transfer learning that is working; yet this would not be a triangular lattice; it would be the second closest site interaction. Therefore we may seek to observe that as long as the lattice structure is kept, we may have the transfer learning with different number of neighboring interactions [13].

Once we understand the structure and limitations of the transfer learning for the Ising model, then we can try to investigate the step up/down diagonal properties, the reasons behind these structures and whether it can be fixed or not. If we were to find that it cannot be fixed, then we would conclude that transfer learning for these specific case does not work. In this potential future work, it would strengthen the hypothesis to work the investigation for various lattice sizes, different Ising parameters and many different interaction connections for each lattice site.

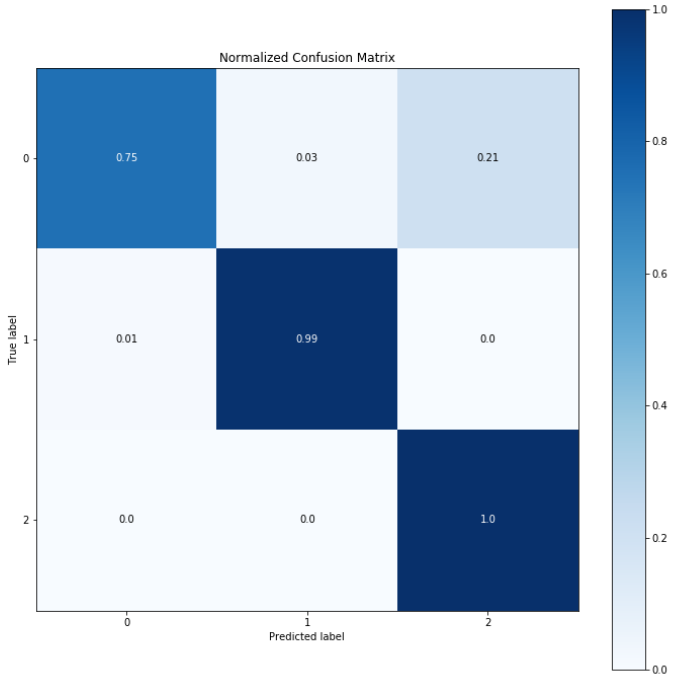

Figure 30: The confusion matrix of the 3-Class Model of the transfer learning of triangular network on square Ising model with the label 0 corresponding to ordered, label 1 corresponding to critical, label 2 corresponding to disordered cases with equal number of configurations in each class with a total of 15,000 configurations.

4.2 Final Remarks

Overall, during the internship we have reached the point where we can indeed see the potential possible work and exploration that can be done on this subject, and moreover I have learned and applied many notions on both physics and machine learning and experienced participating in research. Finally, I would like to thank my supervisor, Prof. Rudolf A. Roemer, for his guidance and advice throughout the internship and my colleague, Djenabou Bayo, for helpful discussions about the project.

References

[1] Martin Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems. Tech. rep. 2015. URL: https://www.tensorflow.org/.

[2] Zi Cai and Jinguo Liu. "Approximating quantum many-body wave functions using artificial neural networks". In: Physical Review B 97.3 (Jan. 2018), p. 035116. ISSN:

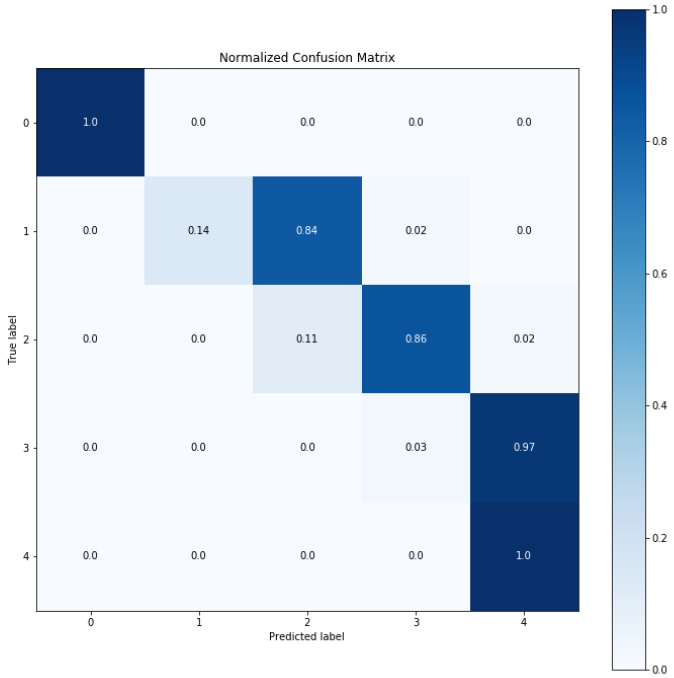

Figure 31: The confusion matrix of the 5-Class Model of the transfer learning of triangular network on square Ising model with the label 0 corresponding to $T = 1.0 - 1.4$, label 1 corresponding to $T = 2.07 - 2.11$, label 2 corresponding to $T = 2.24 - 2.28$, label 3 corresponding to $T = 2.45 - 2.49$, label 4 corresponding to $T = 3.6 - 4.0$ with equal number of configurations in each class with a total of $25,000$ configurations as the test set.

2469-9950. DOI: 10.1103/PhysRevB.97.035116. arXiv: 1704.05148. URL: https://link.aps.org/doi/10.1103/PhysRevB.97.035116.

[3] Juan Carrasquilla and Roger G. Melko. "Machine learning phases of matter". In: Nature Physics 13.5 (2017), pp. 431–434. ISSN: 17452481. DOI: 10.1038/nphys4035. arXiv: 1605.01735. URL: https://www.nature.com/articles/nphys4035.pdf.

[4] Kelvin Ch'Ng et al. "Machine learning phases of strongly correlated fermions". In: Physical Review X 7.3 (2017). ISSN: 21603308. DOI: 10.1103/PhysRevX.7.031038. arXiv: 1609.02552. URL: https://journals.aps.org/prx/pdf/10.1103/PhysRevX.7.031038.

[5] Francois Chollet. Deep Learning with Python. IEEE, July 2018, pp. 2437–2444. ISBN: 9781617294433.

[6] Stefanie Czischek, Martin Gärttner, and Thomas Gasenzer. "Quenches near Ising quantum criticality as a challenge for artificial neural networks". In: Physical Review B 98.2

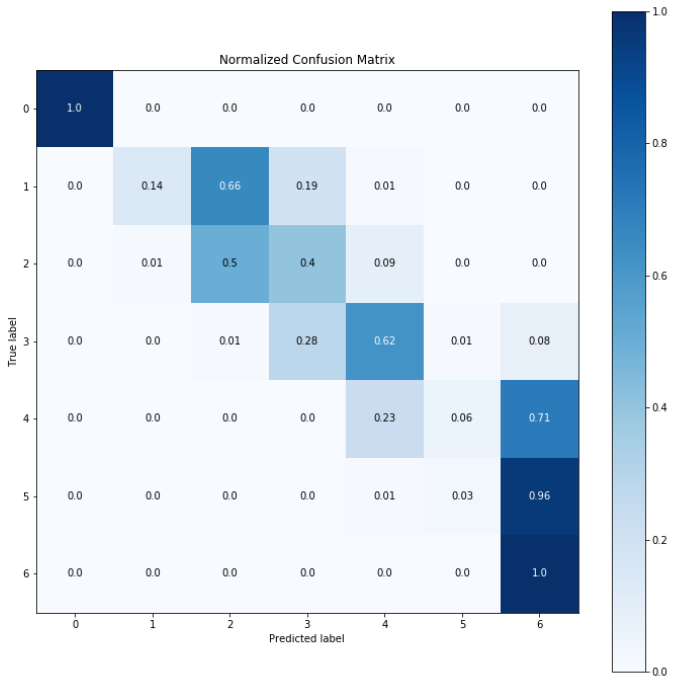

Figure 32: The confusion matrix of the 7-Class Model of the transfer learning of triangular network on square Ising model with the label 0 corresponding to $T = 2.0$, label 1 corresponding to $T = 2.4$, label 2 corresponding to $T = 2.8$, label 3 corresponding to $T = 3.2$, label 4 corresponding to $T = 3.7$, label 5 corresponding to $T = 3.9$, label 6 corresponding to $T = 4.2$ and label 7 corresponding to $T = 4.5$ with equal number of configurations in each class with a total of $14,000$ configurations as the test set.

(2018). ISSN: 24699969. DOI: 10.1103/PhysRevB.98.024311. arXiv: 1803.08321. URL: https://arxiv.org/pdf/1803.08321v1.pdf.

[7] Luo Di. "Machine Learning, Renormalization Group and Phase Transition". In: (2017), pp. 1–11. URL: http://guava.physics.uiuc.edu/%7B~%7Dnigel/courses/563/Essays%7B%5C_%7D2017/PDF/Luo.pdf.

[8] H T Diep. Frustrated Spin Systems. WORLD SCIENTIFIC, May 2013. ISBN: 978-981-4440-73-8. DOI: 10.1142/8676. URL: https://www.worldscientific.com/worldscibooks/10.1142/8676.

[9] Xiao-Yu Dong, Frank Pollmann, and Xue-Feng Zhang. "Machine learning of quantum phase transitions". In: Physical Review B 99.12 (Mar. 2019), p. 121104. ISSN: 2469-9950. DOI: 10.1103/PhysRevB.99.121104. arXiv: 1806.00829. URL: http://arxiv.org/abs/1806.00829%20http://dx.doi.org/10.1103/PhysRevB.99.121104%20https://link.aps.org/doi/10.1103/PhysRevB.99.121104.

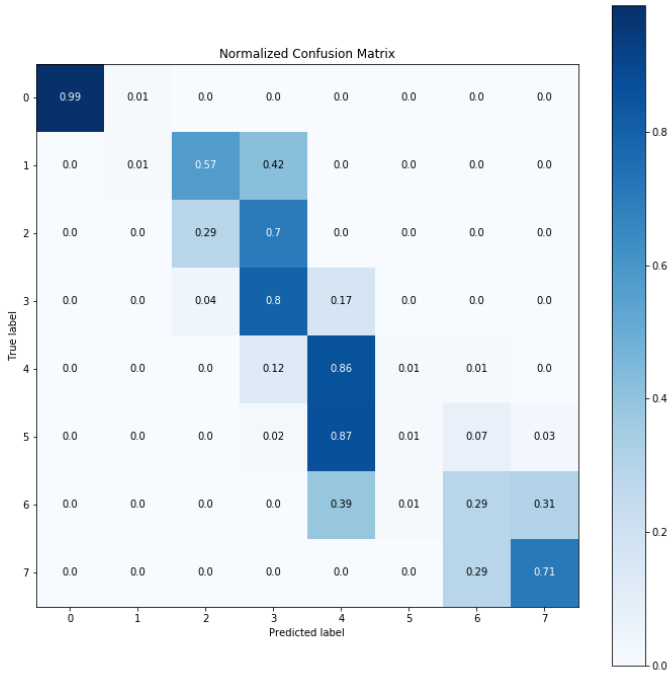

Figure 33: The confusion matrix of the 8-Class Model of the transfer learning of triangular network on square Ising model with the label 0 corresponding to $T = 1.0 - 1.4$, label 1 corresponding to $T = 2.09 - 2.5$, label 2 corresponding to $T = 2.12 - 2.16$, label 3 corresponding to $T = 2.24 - 2.28$, label 4 corresponding to $T = 2.35 - 2.39$,label 5 corresponding to $T = 2.45 - 2.49$, label 6 corresponding to $T = 2.59 - 2.9$ and finally label 7 corresponds to $T = 3.6 - 4.0$ with equal number of configurations in each class with a total of $40,000$ configurations as the test set.

[10] Wenjian Hu, Rajiv R.P. Singh, and Richard T. Scalettar. "Discovering phases, phase transitions, and crossovers through unsupervised machine learning: A critical examination". In: Physical Review E 95.6 (2017). ISSN: 24700053. DOI: 10.1103/PhysRevE.95.062122. arXiv: 1704.00080. URL: https://0-journals-aps-org.pugwash.lib.warwick.ac.uk/pre/pdf/10.1103/PhysRevE.95.062122.

[11] Pankaj Mehta et al. "A high-bias, low-variance introduction to Machine Learning for physicists". In: Physics Reports 810 (Mar. 2019), pp. 1–124. ISSN: 03701573. DOI: 10.1016/j.physrep.2019.03.001. arXiv: 1803.08823. URL: http://arxiv.org/abs/1803.08823%20http://dx.doi.org/10.1016/j.physrep.2019.03.001.

[12] Michael A. Nielsen. Neural Networks and Deep Learning. Chapman and Hall/CRC, 2015. URL: http://neuralnetworksanddeeplearning.com/.

[13] Tomi Ohtsuki and Tomohiro Mano. "Drawing Phase Diagrams for Random Quantum Systems by Deep Learning the Wave Functions". In: (2019). arXiv: 1909.09821. URL: https://arxiv.org/abs/1909.09821.

[14] Tomoki Ohtsuki and Tomi Ohtsuki. "Deep learning the quantum phase transitions in random two-dimensional electron systems". In: Journal of the Physical Society of Japan 85.12 (2016). ISSN: 13474073. DOI: 10.7566/JPSJ.85.123706. arXiv: 1610.00462. URL: http://journals.jps.jp/doi/pdf/10.7566/JPSJ.85.123706.

[15] Lars Onsager. "Crystal Statistics. I. A Two-Dimensional Model with an Order-Disorder Transition". In: Phys. Rev. 65.3-4 (Feb. 1944), pp. 117–149. DOI: 10.1103/PhysRev.65.117. URL: https://link.aps.org/doi/10.1103/PhysRev.65.117.

[16] Christian P. Robert. "The Metropolis-Hastings Algorithm". In: Wiley Mcmc (2015), pp. 1–15. DOI: 10.1002/9781118445112.stat07834. arXiv: 1504.01896.

[17] Akinori Tanaka and Akio Tomiya. "Detection of Phase Transition via Convolutional Neural Networks". In: Journal of the Physical Society of Japan 86.6 (June 2017), p. 063001. ISSN: 0031-9015. DOI: 10.7566/JPSJ.86.063001. URL: http://journals.jps.jp/doi/10.7566/JPSJ.86.063001.

[18] Evert P.L. Van Nieuwenburg, Ye Hua Liu, and Sebastian D. Huber. "Learning phase transitions by confusion". In: Nature Physics 13.5 (2017), pp. 435–439. ISSN: 17452481. DOI: 10.1038/nphys4037. arXiv: 1610.02048. URL: https://www.nature.com/articles/nphys4037.pdf.

[19] Lei Wang. "Discovering phase transitions with unsupervised learning". In: Physical Review B 94.19 (Nov. 2016), p. 195105. ISSN: 2469-9950. DOI: 10.1103/PhysRevB.94.195105. arXiv: 1606.00318. URL: https://link.aps.org/doi/10.1103/PhysRevB.94.195105.

[20] Sebastian J. Wetzel. "Unsupervised learning of phase transitions: From principal component analysis to variational autoencoders". In: Physical Review E 96.2 (2017). ISSN: 24700053. DOI: 10.1103/PhysRevE.96.022140. arXiv: 1703.02435. URL: https://0-journals-aps-org.pugwash.lib.warwick.ac.uk/pre/pdf/10.1103/PhysRevE.96.022140.

[21] Sebastian J. Wetzel and Manuel Scherzer. "Machine learning of explicit order parameters: From the Ising model to SU(2) lattice gauge theory". In: Physical Review B 96.18 (2017). ISSN: 24699969. DOI: 10.1103/PhysRevB.96.184410. arXiv: 1705.05582. URL: https://0-journals-aps-org.pugwash.lib.warwick.ac.uk/prb/pdf/10.1103/PhysRevB.96.184410.

[22] Kun Yang et al. "High performance Monte Carlo simulation of ising model on TPU clusters". In: International Conference for High Performance Computing (2019), pp. 1–26. ISSN: 21674337. DOI: 10.1145/3295500.3356149. arXiv: 1903.11714.