Classification of liquid crystal phases via machine learning

James Harbon

10137627

Department of Physics and Astronomy

University of Manchester

MPhys Project Report

Project performed in collaboration with Joshua Heaton

January 2021

Abstract

The viability of using machine learning to classify liquid crystal phases, based on only their texture images, was tested. The architectures we used included six sequential networks, ResNet50 and a down-scaled version of Inception V3. Classifiers were built for four distinct data sets where the first contained isotropic, nematic, cholesteric and smectic phases. The model with the best performance for the first data set achieved a test set accuracy of $(89 \pm 4)\%$. The second data set was built from the fluid, hexatic and soft crystal smectic groups and our best model achieved a test accuracy of $(86 \pm 6)\%$. Following this, smectic A and C binary classifiers were created and ResNet50 achieved a test accuracy of $(92 \pm 4)\%$. Finally, we created a data set with six individual smectic phases and our best classifier was ResNet50, however this time with an accuracy of $(54 \pm 1)\%$.

Machine learning proved to be viable for three out of the four classification problems we approached. The poor performances of all models for the six phase data set could be improved by the addition of data for underrepresented phases. Other networks with greater complexity and capacity than ResNet50 could be used to distinguish between the six classes.

Contents

1 Introduction

The field of artificial intelligence is concerned with understanding and building intelligent agents, which can be a general object or system that can interact with an environment. Initial work, which dates as far back as Aristotle, was focused on trying to define formal rules for how a mind works. Practical applications did not exist until the field of mathematics was used to apply logic, probability and computational methods to solve problems. Specific problems could be solved if humans defined all of the parameters and rules before hand for how an agent should solve the given problem. For complex, general systems it is no longer possible to initially specify exactly how the agent should solve a problem or behave and even if it is, the time it would take to do this becomes impractical. The solution is to allow the agent to learn parameters and rules for how to behave intelligently in a given environment. Specifically, we often want it to learn an optimal function or mapping for a problem from a family of parametric or non-parametric functions. Machine learning is a sub field of artificial intelligence which provides frameworks and tools for doing exactly this [1].

Machine learning approaches have found great success in a multitude of fields and industries. One approach is concerned with computer vision algorithms, which can be used for image classification. This is where an image, drawn from some distribution with different classes of images, is fed into the algorithm and a label for the class type is outputted. Alongside computer vision, an approach called reinforcement learning (RL), which provides a framework to learn the optimal actions to take in the context of decision making, has been applied to robotics and self-driving cars [1, p. 1-29]. RL has also been used to exceed human performance in a game called Go, which is a very complex game due to the large search space of positions and moves. The company DeepMind developed a program in 2014 called AlphaGo which achieved a win rate of 99.8% against other Go programs and beat the human European champion in five out of five games. This was the first time ever that a program managed to beat a human at Go [2].

The application of machine learning in the sciences has gained popularity in recent years. One particular area with a lot of potential is the use of predictions for occurrence, recurrence and mortality likelihoods in diagnoses and prognoses of cancer. A study titled "Applications of Machine Learning in Cancer Prediction and Prognosis" analysed 79 papers focused on the diagnosis and prognosis of cancer. They found that machine learning methods can improve the accuracy of the predictions by as much as 15-25% [3]. A more recent development emerged in 2020 with the reveal of the AlphaFold algorithm. It was shown that the 3D structure of a protein could be predicted, using only its genetic sequence, with accuracy comparable to the conventionally used experimental instruments. The latter is expensive and time consuming, which means that AlphaFold could massively increase the speed of drug and disease research [4].

This project was focused on the use of machine learning to classify phases of liquid crystals (LC). The phases, if the LC is thermotropic, will vary with temperature. The phase can be represented by an image obtained via polarised microscopy (PM), and the current convention uses human observation of an image to infer the type of phase. We used a range of different models to test the viability of using ML to learn how to classify LC phases from PM images.

2 Machine Learning

2.1 Fundamental Concepts

In the field of artificial intelligence, problems are often approached through consideration of an agent in one or more environments. The agent can perceive and act upon the environment, within which metrics can be defined to assess its behaviour. A rational agent is simply defined as one which can maximise the performance metrics for the given environment. The field of machine learning provides methods for agents to learn this optimal behaviour. [1, p. 693-717]

The type of feedback the agent receives from the environment determines the type of learning which will take place. In order for the agent to learn the optimal behaviour, it must be exposed to some form of data, which could already exist or be created as a result of the actions of the agent. One of the most commonly used learning frameworks is supervised learning, within which a data set is comprised of pairs $(\boldsymbol{X}, \boldsymbol{y})$, where $\boldsymbol{X}$ is some M-dimensional vector $\boldsymbol{X} = (x_1, x_2, ..., x_M)$ used as an input and $\boldsymbol{y}$ is a vector output for that given input (it has one dimension when there is a single output). It is assumed that a function $\boldsymbol{y} = f^*(\boldsymbol{X})$ exists which specifies the true relationship between an input and an output. Methods in machine learning aim to approximate this function with a model $\hat{\boldsymbol{y}} = f(\boldsymbol{X}; \boldsymbol{\omega})$, where $\boldsymbol{\omega}$ is a vector of parameters. For general problems, this can be achieved through a variety of algorithms with differing performances. [1, p. 693-717]

The "quality" of the model's function can be quantified with a metric which usually relates the error between the predicted and true outputs, $|\boldsymbol{y} - \hat{\boldsymbol{y}}|$, to a metric for the whole data set. In the case of a rational agent, the optimal behaviour is achieved when $\hat{\boldsymbol{y}} = \boldsymbol{y}$ for any input $\boldsymbol{X}$. In practice this is nearly impossible for most problems. However, we can find sets of parameters and families of functions which approach that kind of behaviour. These parameters must be learned by training the model on existing data. Consider a model, implemented with a function from some parametric family of functions, and a training set with N input-output pairs, $\{(\boldsymbol{X}_1, \boldsymbol{y}_1), ..., (\boldsymbol{X}_N, \boldsymbol{y}_N)\}$. The model's parameters will be randomly initialised and then the function can be used to predict outputs. The metric we use for quantifying the error in the predictions is called a loss function, and we aim to find a set of parameters, $\boldsymbol{\omega}$, which minimise this function. An optimisation algorithm is used to achieve this. [1, p. 693-717]

For a model to be useful, it needs to be able to generalise to broader environments and make accurate predictions using new, unseen data. Upon minimisation of the loss function for the training set, the performance of the model for novel data can be measured with a test set. This will have the same form as the training set, however the model is only exposed to the input vectors, $\boldsymbol{X}$, this time. The predictions will again be compared to the actual outputs and a metric is used to assess the performance. For a regression problem, in which the output is a continuous variable, a metric called the root mean squared error (RMSE) can be used. This is the average deviation of a prediction from the actual output. For a classification problem, where the output is one or more discrete variables, we can calculate the accuracy of the model. This is a computation of the percentage of correctly predicted outputs. [1, p. 693-717].

A number of assumptions underlie a model and hence affect the choice of a family of functions. The number of parameters and the type of function imply how much bias and variance the model has. A greater bias is caused by simplifying assumptions such as a linear relationship between inputs and outputs. For many problems such an assumption can negatively affect a model. As an example, a linear function could be used to approximate a non-linear relationship and we could hence observe "underfitting", where the model fails to learn most of the meaningful features from a training set. Greater variance occurs due to overly broad assumptions and too many parameters. In this case, a model will suffer from "overfitting", where it tries to fit noise and might learn meaningless features in the training set. Overfitting reduces a model's ability to generalise to novel data. Both underfitting and overfitting are effects we have to consider in supervised ML problems. There is no closed form solution for how much bias and variance we should engineer into a model to achieve optimal performance, which gives rise to the "bias-variance tradeoff" problem. For many models, the effect of overfitting tends to dominate and a technique called regularisation must be used to reduce it.

2.2 Loss Functions and Optimisation

Real world data is often noisy and thus means that a deterministic mapping, $\hat{\boldsymbol{y}} = f(\boldsymbol{X}; \boldsymbol{\omega})$, is difficult or impossible to find. The more fundamental interpretation is that the data has some underlying distribution $p(\boldsymbol{X})$. For a supervised learning problem, we usually want to estimate the conditional probabilities $p(\boldsymbol{y}|\boldsymbol{X})$ with a parametric model $p(\boldsymbol{y}|\boldsymbol{X}; \boldsymbol{\omega})$. The "distance" between two distributions, where a smaller distance implies greater similarity, can be quantified by the Kullback-Leibler (KL) divergence. A method called maximum likelihood estimation can be used to find a vector of parameters $\boldsymbol{\omega}$ for which the KL divergence is minimal. This method can lead us to a number of different loss functions for either continuous or discrete outputs [5, p. 129-133].

For a problem such as linear regression, a closed-form solution often exists for a set of parameters $\boldsymbol{\omega}$ which minimise the loss function for that problem. However, in a lot of cases there is no such solution and we must use optimisation algorithms to approximate the parameters. The graph of the loss function $J(\boldsymbol{X}; \boldsymbol{\omega})$ is assumed to be convex, which means it has a global minimum we can approximate. In practice this isn't always necessarily true. A method called gradient descent involves determination of the partial derivatives $\nabla_{\boldsymbol{\omega}} J(\boldsymbol{X}; \boldsymbol{\omega})$, which can be used to update the parameters. The new values are the result of "moving down" the path on $J(\boldsymbol{X}; \boldsymbol{\omega})$ with the steepest gradient for each parameter, and iterations of this process can be performed until convergence around the global minimum (or a local minimum in practice) is reached. Explicitly, for a given parameter ω_i,

$$\omega_i \longleftarrow \omega_i + \Delta\omega_i = \omega_i - \eta \frac{\partial J}{\partial \omega_i}, \tag{1}$$

where η is called the learning rate and determines the step size for the parameter updates. One of the biggest problems with standard gradient descent is that it becomes increasingly computationally expensive for larger data sets, since the loss function has to be calculated at each iteration. Stochastic gradient descent (SGD) randomly picks one data point to calculate the loss function at each step to reduce the computational expense. A methodology called "mini batch" is a compromise between gradient descent and SGD, where a small number of data points are randomly selected at each step. The size of the batches determines a balance between the precision of gradient descent and the speed of SGD [5, p. 80-95].

2.3 Neural Networks

The problem with a linear mapping $\boldsymbol{y} = f_{linear}(\boldsymbol{X};\boldsymbol{\omega})$ is that the underlying assumption is that the true mapping of the data is in fact linear. For many problems, the relationship between variables is non-linear and linear models thus cannot provide a good fit of the data. Neural networks (NNs) aim to learn the parameters $\boldsymbol{\omega}$ of a general function $\Phi(\boldsymbol{X};\boldsymbol{\omega})$ which is the result of applying a linear function to a transformed input. The input can be transformed by a series of N functions $f^N(f^{N-1}(...f^1(\boldsymbol{X})...))$, where each function is a layer of the network and performs a transformation on some vector (the vector does not necessarily have an equal number of dimensions to $\boldsymbol{X}$) which is the result of a transformation from a previous layer. The building blocks of a network are called units and there is a specific number of them for each layer. A unit will have connections from units in the previous layer and to those in the next. Every connection has a corresponding weight, and we can create a matrix of weights ω_{ij} where the (i,j) element is the weight for the connection from a unit i to a unit j in the next layer. For a set of units U, every unit has an output

$$\sigma_i = \phi(\sum_{k\in S}\omega_{ki}\sigma_k) = \phi(r_i), \tag{2}$$

where $S \in U$ is the set of units a layer behind unit i and ϕ is called an activation function. There is also a bias unit attached to every layer which has no incoming connections and a constant output. At each layer of a network the activation function uses the bias, unit outputs and weights from the previous layer to calculate new unit outputs for the current layer. We are concerned with feed-forward networks, for which the information flows in a single direction from an input layer through a series of activation functions to an output layer. The name “neural network” was given because the network is loosely inspired by neuroscience research into the brain, where units and weights respectively correspond to neurons and synapses [5, p. 164-172].

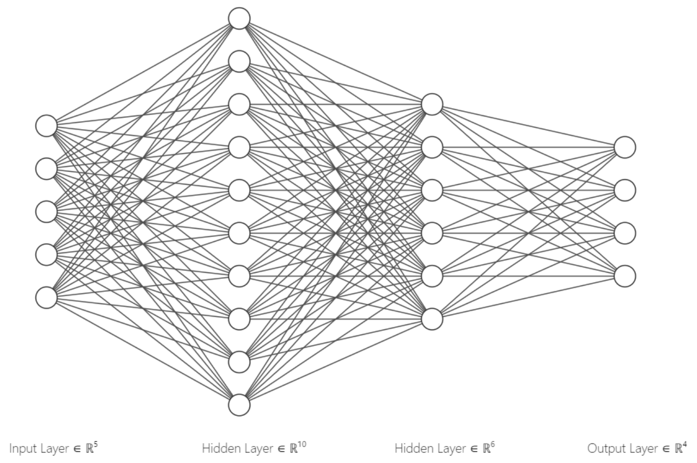


Figure 1: A diagram of a simple, fully-connected neural network with four output units. Every connection between two units has a corresponding weight. Adapted from [5].

The number of layers, N, is called the depth of the model. The width of each layer refers to the number of units it is composed of. The capacity of the model refers to the total number of parameters. For a model with a high capacity, the time required to compute an analytical expression for the gradients of the loss function, $\nabla_{\boldsymbol{\omega}} J$, will cause the total training time to drastically increase. An algorithm called "backpropagation" saves time by using the chain rule to evaluate gradients layer by layer, starting from the output layer and finishing at the input layer. We consider a loss function J which is a function of the set of all unit outputs $\{\sigma_i\}$ and connection weights matrices $\{\omega_{ij}\}$. Through the use of the chain rule for derivatives and a recursive expression, we can arrive at two equations

$$\frac{\partial J}{\partial \omega_{ij}} = \sigma_i \delta_j \tag{3}$$

and

$$\delta_j = \frac{\partial J}{\partial \sigma_j} \frac{\partial \sigma_j}{\partial r_j} = \begin{cases} \frac{\partial J}{\partial \sigma_j} \frac{d\sigma_j}{dr_j} & \text{if unit } j \in \text{output layer} \\ \left(\sum_{l \in L} \omega_{jl} \delta_l\right) \frac{d\sigma_j}{dr_j} & \text{if unit } j \notin \text{output layer} \end{cases} \tag{4}$$

where l is a set containing the units in layer after that which contains i. The weight update can be explicitly written as

$$\omega_{ij} \longleftarrow \omega_{ij} + \Delta\omega_{ij} = \omega_{ij} - \eta \sigma_i \delta_j, \tag{5}$$

which guarantees that the weight will always change in such a way as to decrease the loss function J [5, p. 175-200].

The learning rate η in equation 5 is part of a set of parameters called the hyperparameters, which define key parts of an NN architecture before it begins training. The "no free lunch" theorem in machine learning states that there is no one model which is optimal for every problem. Consequently, the optimal hyperparameters are often found through trial and error, although some algorithms exist which can search for the optimal values [5, p. 200-219].

NNs contain a large number of parameters relative to conventional alternatives and also non-linear transformations of data. This thus means that the graph of the loss function for a given network will likely possess a multitude of local minima and the property of non-convexity. Instead of searching for a global minimum, an NN will attempt to drive the loss function as low as possible. This will be repeated multiple times for the whole data set over what are called epochs. [5, p. 200-219].

2.4 Convolutional Neural Networks

Data which has a grid-like topology such as images, which can be represented by a 2D grid of pixels, can be processed by a special kind of network named a convolutional neural network (CNN). These networks have had great success in practical applications such as image classification and segmentation, speech recognition, natural language processing and related fields. At the base level a CNN is an NN which employs a convolution operation, in place of standard matrix multiplication, in at least one of its layers [5, p. 326-339].

NNs use a matrix of weights to connect every unit in one layer with every unit in the next layer. For image data, this would result in the connection of an input of thousands or millions of pixels, and thus units, with another layer. This is impractical for training since data sets often contain thousands of images. Additionally, this approach isn't even guaranteed to provide the best performance. CNNs learn meaningful features from images via what is known as a kernel, which is a matrix with dimensions typically much smaller than the input. Instead of attempting to learn features from single pixel inputs on an image with thousands or millions of pixels, we can instead use a kernel which spans tens of pixels or less to find features such as edges. An arbitrary number of kernels can be employed to find features in an image, and each kernel maps from the input to a feature map (another matrix). Kernels can be handcrafted to find specific features for specialised tasks, however the components of kernels in CNNs are weights, and hence each kernel learns a feature in an input which is useful for predicting an output. The addition of convolutional layers allows the CNN to learn higher level features in inputs. One simple sequence might involve using one layer to learn edge features and then another layers could learn to recognise shapes built from multiple edges [5, p. 326-339].

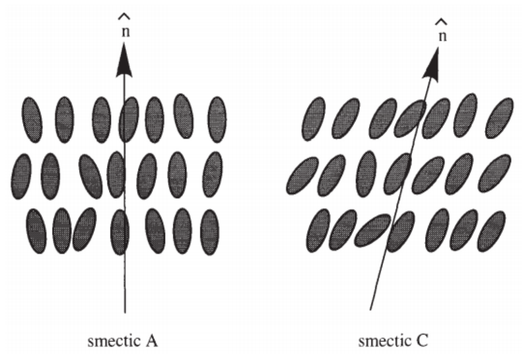
Along with the convolutional layer, CNNs also employ what is known as a pooling layer. This is another type of kernel which computes summary statistics, such as a max or average value, for areas equal to their kernel size on the input tensor they receive. The use of this layer is important as it creates a spatial invariance for small translations of features, which in turn improves generalisation of the network since the features in an unseen image generally won't be in the exact same places as those in the training set. Also, the pooling layer can reduce the dimensions of the tensor it operates on, which reduces the total training time [5, p. 326-339].

3 Liquid Crystals

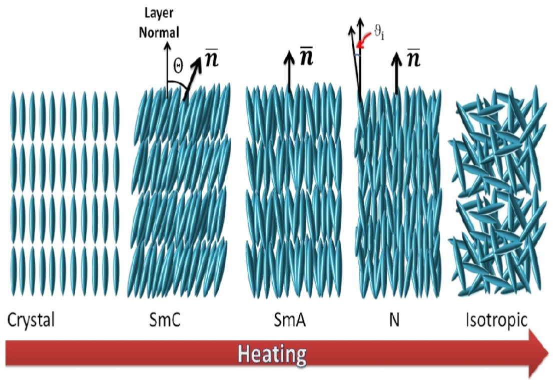
Liquids and crystals, two of the most common condensed matter phases, differ in that the molecules of a liquid are disordered and those of a crystal are ordered. The positions of molecules in a liquid are randomly distributed since they can diffuse throughout some volume. Their molecular axes, which are defined by symmetries of their structure, randomly change direction. This is in contrast to a crystal, where the positions are at specific lattice sites and at least one molecular axis will point in a specific direction for every molecule. Liquid crystal (LC) phases are more ordered than liquids but less ordered than crystals. Generally, the order in an LC is small relative to that of a crystal. This can be inferred from a comparison of the latent heat values for crystal-LC and LC-liquid transitions, which are 250 J/g and 5 J/g respectively [6].

The order of an LC can be quantified using two parameters: the positional and orientational order parameters. The former can be qualitatively described as the tendency for the centre of mass positions of the molecules to arrange into layers. An LC with higher positional order has more distinct layers and possibly a structure in the plane of each layer itself. The corresponding description for orientational order is the tendency of the molecular axis (or multiple axes) to point in some direction $\hat{\boldsymbol{n}}$ [6].

Thermotropic phases are constrained to a certain temperature interval for a given compound. Heating a compound increases the random thermal motion of molecules and thus reduces the order. For most compounds, at a cold enough temperature, a crystalline phase exists. As the temperature increases the LC will move through a sequence, which depends on the given compound, of phases until reaching an isotropic phase. In this final phase, molecules randomly diffuse



(a)



(b)

Figure 2: (a), which was taken from [6], shows two LC phases with high order for both position and orientation. (b), which was taken from [7], shows the effect of heating an LC.

and change orientation freely because there is no order. Figure 2 contains an example of such a sequence [6].

When a thermotropic LC, starting in the crystalline phase, is heated it is common to observe groups of smectic phases. The first group to appear could be soft crystal smectic phases, which are characterised by a high positional order that extends over long ranges. The molecules are arranged into layers and their distribution within each layer is a hexagonal lattice. The second group, which exists at a higher temperature, is the hexatic smectic group. These phases are structurally similar to those in soft crystal, however their positional order extends over a much shorter range. Finally, at even higher temperatures, we can observe the fluid smectic group. The fluid phases are characterised by the presence of orientational order and layers of molecules randomly distributed in the planes of their layers [8]. Figure 2 contains a diagram of the smectic A and C phases, which are both in the fluid group.

The nematic phase will usually be the first phase to be observed after the cooling of the isotropic state. This phase has orientational, but not positional, order. For compounds which exhibit chirality, the nematic phase can instead be called the cholesteric phase. It too has only orientational order, however the director $\hat{\boldsymbol{n}}$ periodically varies in direction which forms a helical structure [8].

4 Data Collection and Preparation

We needed to build a data set from scratch due to the lack of a prepared public data set or any previous work from students.

4.1 Extraction and Transformations

The initial data was extracted from the Instagram and YouTube accounts of a researcher named Vance Williams [9, 10]. The videos on these pages were downloaded as MP4 files. The rest of our data came from our supervisor Ingo Dierking. These videos were obtained from polarised microscopy (PM) and the heating or cooling of an LC compound. The training, validation and test data sets all needed to be in an image format for our classifier. To extract the frames from the videos and save them as images, we used VLC Player's scene filter function.

The videos from our supervisor were labelled with temperature ranges and compound names. We could thus, through use of a paper which was co-authored by our supervisor [11], sort the images into the correct classes. The images we had were named based on the video they were obtained from. This was done to ensure that images from a given video were not shared between the training set and any of the testing sets. This was necessary to avoid data leakage, which is where a model is effectively trained on some of the data it is being tested on. Leakage can create overly optimistic results and thus invalidate your findings [12].

Convolutional neural networks generally do not have specific requirements for the aspect ratio and resolution of the images they process. However, images in a data set will not necessarily have the same pixel dimensions. This is a problem because we have to design a neural network for a specific shape of a tensor input. The most commonly used compromise, to minimise the loss of important features in images, consists of cropping a square focused on the centre out of each image. We implemented this method for every image in our data set. Next, we considered the resolution of each image. Training on images with a higher resolution can allow a neural network to possibly learn more features and then generalise better to unseen data. One potential downside of a higher resolution is that the images then require significantly more training time. As a trade off between complexity of the image features and computational time, we chose pixel dimensions of $(256, 256)$.

The majority of the images we extracted from videos had a resolution of 2048x1088. We assumed that our neural networks would not need the full images, and instead would be provided with enough features from sub-images of each image. This would retain more data, and effectively expand our data set, for training as opposed to simply cropping a square and then resizing it. A script was created to crop six sub-images from each (2048x1088) image and then resize each sub-image to 256x256. Following this, every sub-image was converted from 3-channel colour mode to 1-channel greyscale mode. The sub-images were created by partitioning the original image into two rows and three columns.

The performance of a model can potentially be negatively affected by an imbalance of class sizes. This is because the model could be exposed to a disproportionate amount of training images from one or more classes and hence gain a bias towards those classes [13]. We attempted to approximately balance the classes for all data sets, except for where it would significantly reduce the size of the data set. Additionally, due to a lack of isotropic phase images, we created a script which used a technique called "salt-and-pepper" noise to generate pseudo isotropic images with a size of (256, 256). This was viable because every real isotropic image is simply a pattern of shades of grey, where the shade randomly changes throughout the image and thus creates the appearance of noise. The amounts of images we had for each phase in our overall data set are shown in figure 3.

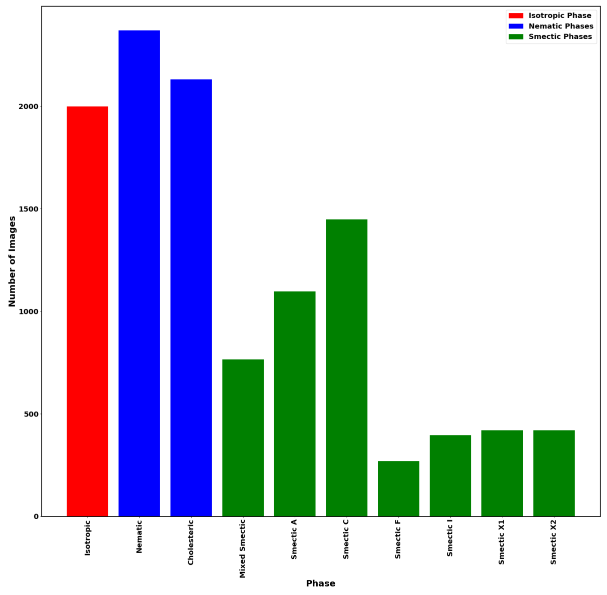


Figure 3: The number of images plotted against the phases.

4.2 Augmentations

The data set provided us with over 9,300 images to split into training, validation and test sets. Due to our use of either a 60-20-20 or 70-15-15 split, the training set would have a size of approximately 6,500 at the most. This is not necessarily large enough for a model to generalise to novel data successfully. Augmentations were used to effectively expand the data set without the addition of new data. This method works by applying a series of random transformations to each image in a batch during training.

The use of lots of image augmentations can massively increase training time when compared to the use of none. The optimal amount and choices of augmentations depends on the data set and the classification problem itself. Thus, they often need to be inferred from the nature of the data or found through trial and error. We attempted to systematically determine which group of augmentations results in the best performance from our models. Table 1 shows the groups we used.

5 Models

A set of models was created to train on the data that we obtained. The first six were simple sequential architectures, which were used to investigate the effect of different numbers of convolutional layers on model performance. Two architectures, named "ResNet50" and "Inception", with greater depth were used for the other two models.

5.1 Sequential Networks

A sequential architecture simply contains a sequence of layers through which information flows in a linear direction. Six models were created, each with a different number of convolutional layers. The naming convention was "n layers", where n represented the number of convolutional layers

No Augmentations	
Parameter	Value
rescale	1./255

Flip Augmentations	
Parameter	Value
rescale	1./255
horizontal flip	True
vertical flip	True

All Augmentations	
Parameter	Value
rescale	1./255
horizontal flip	True
vertical flip	True
rotation range	30
zoom range	0.2

Table 1: The three distinct groups of augmentations which were used. Each augmentation in a group is specified with a value for the corresponding parameter.

and $n \in \{1, 2, 3, 4, 5, 6\}$. All models were designed with a first layer that contained 32 kernels, where each had a size of $(3, 3)$, and every subsequent convolutional layer contained double the amount the previous one had. This expanded the number of feature maps and thus the feature space. We believed that a large number of high level features at the end of the model would be beneficial. Additionally, the rectified linear unit (ReLU) activation function was used.

Every convolutional layer was followed by a batch normalisation layer, which helps with regularisation and can reduce the number of training epochs required for convergence. Every batch normalisation layer was followed by a max pooling layer, which always had a pool and stride size equal to $(2, 2)$.

The output from the final convolutional layer in a network was always transformed by a global average pooling layer, which both helped to preserve spatial invariance of features and mapped every feature map from the final convolutional layer to a vector of scalar values. This vector was then fed through two fully connected, or “dense”, layers where the first had 256 units and the second had 128. The ReLU function and a dropout rate of 0.4, to improve regularisation, were used for both layers. At the end a dense layer formed the output, where the “SoftMax” activation function was used and the number of units was equal to the number of output classes available for the given problem. This number could be altered for different problems we approached.

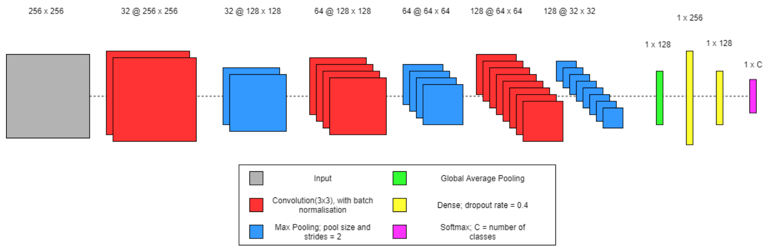


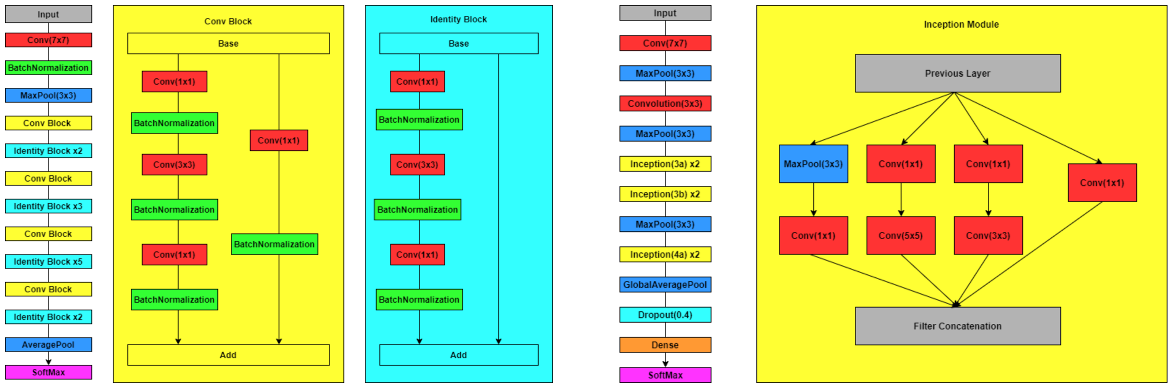
Figure 4: The basic architecture for the sequential model named “3 Layers”.

5.2 ResNet50 and Inception

Very deep neural networks are generally desired for specialised problems. In the case of sequential models, stacking on more and more layers will eventually cause what is known as the degradation effect. This is where the performance of the model will saturate and eventually decline. Although the effect is counter-intuitive, there are a few different potential causes and one of them is called the vanishing gradient problem. This is the failure of a network to propagate all of the useful gradient information from the output to the earlier layers. The backpropagation algorithm was briefly discussed in section 2.3 with equations 3, 4 and 5. It can be seen that, for a weight ω_{ij} connecting units i to j, there will be a multiplicative effect from the $\frac{\partial J}{\partial \sigma_j}$ term. For an activation function such as a sigmoid, $\frac{\partial J}{\partial \sigma_j}$ is restricted to the range [-1, 1] and thus the repeated multiplication of these terms when propagating gradient information to the earlier layers will cause the value of $\frac{\partial J}{\partial \omega_{ij}}$ to fall to zero. This means that the weights for early layers will update by a negligible amount, which can increase training time required for convergence and in the worst case will prevent a network from learning all of the meaningful features from some input. This effect can be almost eliminated through the use of the ReLU activation function, which has a constant $\frac{\partial J}{\partial \sigma_j} = 1$. Other factors exist and are hard to eliminate via simply tweaking standard architectures. For this reason, very specialised networks and ensembles of networks have been created by researchers to tackle these problems and create many-layered, stable models.

A “ResNet” is a residual NN. These networks use skip connections, which are general shortcuts which can bypass information from one layer to any layer after. These connections are implemented in ResNet50 as “identity blocks”. They form stages with the “convolutional blocks”, which define the main path. The connections neither introduce parameters nor computational complexity. They partially fix the vanishing gradient problem by allowing gradient information to flow through the shortcuts to earlier layers, so that these layers can continue to update their weights and hence continue to learn from some input. Figure 5 contains a diagram of the basic architecture. More details can be found in the paper “Deep Residual Learning for Image Recognition” [14].

The other architecture which was used to try and counter the vanishing gradient problem was based on the “Inception V3 model”. This architecture uses what is known as an inception module, which has branches of paths for information to flow through convolutional layers. The branches effectively increase the width of the module and thus the feature space. Also, they provide additional paths for gradient information to propagate through, which can help to reduce the vanishing gradient effect. The architecture we adapted used only a fraction of the total number of inception modules in Inception V3 in order to reduce training time. The basic building blocks are shown in figure 5 and specific details can be found in the paper “Going deeper with convolutions” [15].



(a) ResNet50

(b) Inception

Figure 5: (a) and (b) are respectively the basic architectures of ResNet50 and Inception.

5.3 Training and Callbacks

The creation of the models and the implementation of their subsequent training were both handled via the TensorFlow and Keras libraries [16, 17]. Google Colab, which provides access to a 12GB NVIDIA Tesla K80 GPU, was used to run the training scripts. GPU hardware is the current standard for training NN based models since their architecture can handle tensors more efficiently. Furthermore, we used a batch size of 32 for the training as this is a commonly used standard and batch sizes which are a factor of two make more effective use of the GPU’s memory. The “Adam” optimiser function was used to search for the optimal weights. Finally, the maximum number of epochs was set to 500.

Objects called “callbacks” were defined before the training began. They provided functionality for operations during training. Two common callbacks named “ModelCheckpoint” and “EarlyStopping” were used. The former was used to save the model which corresponded to the lowest validation loss and the latter to stop the training when the validation loss had not changed by more than a unit value for over 100 (patience) epochs. EarlyStopping can effectively stop the training before overfitting starts to occur. Another callback named “ReduceLROnPlateau” was used to reduce the learning rate with the same condition as EarlyStopping. The patience was instead set at 10 epochs, and the reduction of the learning rate assisted the convergence on a local minimum of the loss function. Additionally, we used a callback called “CSVLogger” to save each model’s training history as a CSV file.

6 Classification of 4 Phases

6.1 Data Set and Approach

The first problem we approached involved the classification of images as either smectic, cholesteric, nematic or isotropic. The smectic class was created from a mix of all of the different smectic phases. The size of the classes in each set is shown in table 2

Phase	Total	Train	Validation	Test
Isotropic	2200	1300	500	400
Nematic	2262	1300	491	471
Cholesteric	2100	1292	410	398
Smectic	2722	1300	500	922

Table 2: The number of images in each of the 4 phases for each set.

Each model was trained and then tested three times. A mean test accuracy and an uncertainty were computed for every model. Due to a low number of repeats, we decided to compute the uncertainty as half of the spread of the test accuracy values. All three groups of augmentations had their own repeats so that we could systematically test how the group affected model performance.

6.2 Results

The first results we inspected were the training times of the different models. It can be seen in figure 6 that the training time of a sequential model, for a given group of augmentations, increased by a small amount after the addition of a convolutional layer. The inception model's training time was slightly longer than that of the "6 Layer" model and ResNet50 took significantly longer to train than the other models. This was likely because it has a much higher number of convolutional layers compared to the others. Additionally, the "no" and "flip" augmentation groups had almost the same training time for a given model, while the "all" group's times were approximately a factor of 1.5 longer.

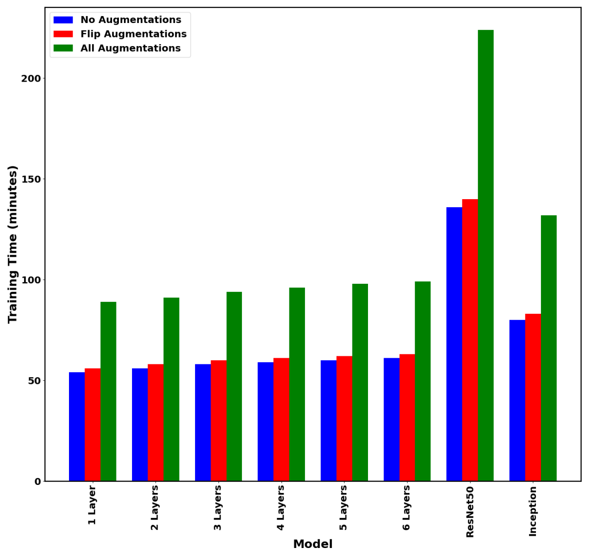
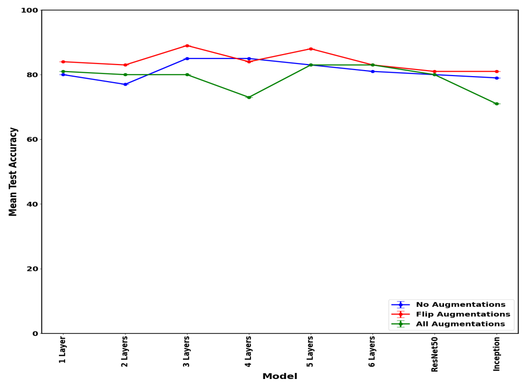
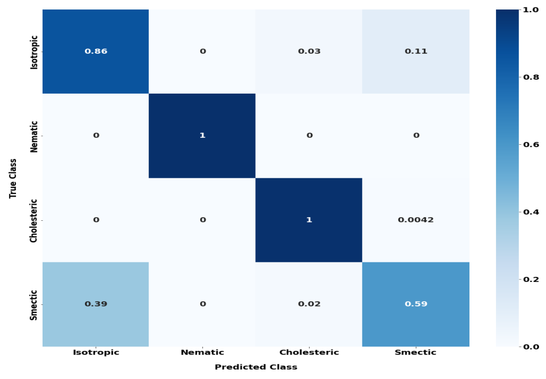


Figure 6: The training times for each model and group of augmentations.

Figure 7 contains a graph of the results from the repeated training and testing. The mean values of all of the models for each of the augmentation groups, from least to most augmentations, were 81%, 84% and 79% respectively. It was thus found that the models which were trained on data augmented by flip augmentations performed the best on average. The "all" augmentations group had the longest training times and also performed the worst out of the three groups. Due to this, it was decided that flip augmentations would be used for the other classification problems.



(a) Mean Test Accuracy for Different Augmentation Groups



(b) Confusion Matrix

Figure 7: (a) shows the mean test accuracy values we computed for different groups of augmentations. The error bars, which ranged in size from 4-6% on average for each group of augmentations, were omitted for clarity in the graph. (b) is the confusion matrix for the "3 Layers" model, which achieved the highest mean test accuracy value.

The model with the best overall performance was "3 Layers", which was trained on flip augmented data and achieved a test accuracy of $(89 \pm 4)\%$. The performance of the model, for the repeat with the median test accuracy, on individual classes is shown in figure 7. It can be seen that the model had a perfect accuracy for classifying nematic and cholesteric phases, and an accuracy of 86% for isotropic. The model performed poorly for the smectic phases, where it correctly classified them with an accuracy of only 59%. The smectic phases were incorrectly predicted as isotropic phases 39% of the time, which was almost the only source of error. This could have been because the smectic data set contained a moderate number of images with mainly dark pixels, which could create a resemblance to the isotropic phase and confuse the model. A larger data set, especially with a higher number and greater diversity of smectic images, will likely result in a better overall test set accuracy.

7 Classification of Smectic Phases

The overall data set contained an adequate amount of images to attempt to build classifiers for various groups of smectic phases. The same approach as described in section 6 was used to train and visualise results, with the exception of training times as we observed the ratios between the models to remain approximately constant.

7.1 Fluid, Hexatic and Soft Crystal Groups

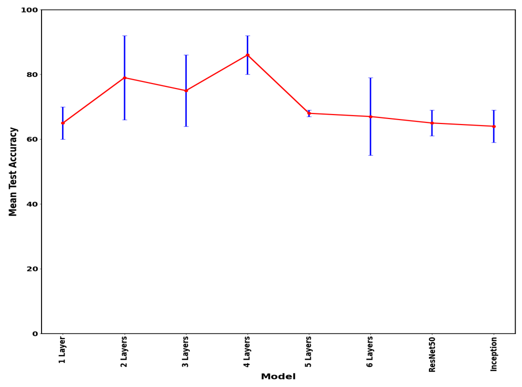
The data for smectic A and C was combined into a "fluid" group. Smectic F and I were incorporated into a "hexatic" group. Smectic X1 and X2 were used to create a "soft crystal" group.

The sizes of the groups in the different sets of data are shown in table 3.

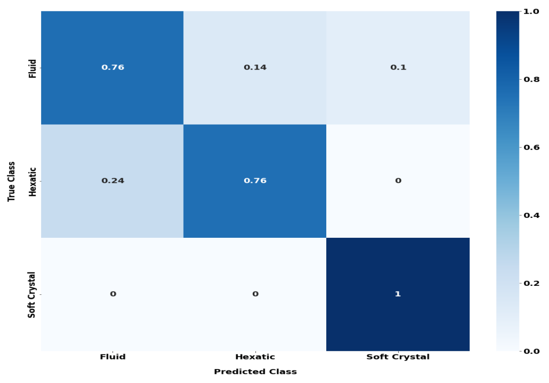
Phase	Total	Train	Validation	Test
Fluid Smectic	2520	1759	372	389
Hexatic Smectic	666	486	90	90
Soft Crystal Smectic	840	600	144	96

Table 3: The sizes of the sets for each of the fluid, hexatic and soft crystal groups.

Figure 8 contains the diagram for the mean test accuracy values for each model. The average uncertainty was 7%, and the models with a total of two, three and six convolutional layers had respective uncertainties of 13%, 11% and 13%. There didn't appear to be a clear reason as to why the uncertainties of some models were significantly bigger than others. More reliable measures of the uncertainty could be computed in future work if we train more repeats of the models. One possible way to train many repeats would involve the reduction of the maximum number of training epochs to a smaller number such as 50. This is because the lowest validation set loss value, which corresponded to the selected model, was observed to almost always occur within that range of epochs.



(a) Mean Test Accuracy



(b) Confusion Matrix

Figure 8: The performance of the models for the classification of the fluid, hexatic and soft crystal groups is shown in (a), and (b) is the confusion matrix for the "4 Layers" model.

The "4 Layers" network created the model with the best mean test set accuracy of $(86 \pm 6)\%$. Figure 8 shows the individual class accuracy values, where it can be seen that the model classified the soft crystal group with perfect accuracy and achieved 76% for the fluid and hexatic groups. The model appeared to incorrectly classify the fluid group as hexatic 14% of the time and hexatic as fluid with a rate of 24%. The reason for the second of the two errors could have been due to the imbalance of data towards the fluid group. This would create a bias towards that class during the training. The reason for the first of the two errors is unclear. However, based on (a) from figure 8, we can see that the test accuracy decreased for every model with a higher capacity than "4 Layers". This provided evidence that the models with a higher capacity than this model won't necessarily perform better. It is instead believed that the addition of data and a greater balance between the class sizes will improve the individual accuracy values for the fluid and hexatic groups.

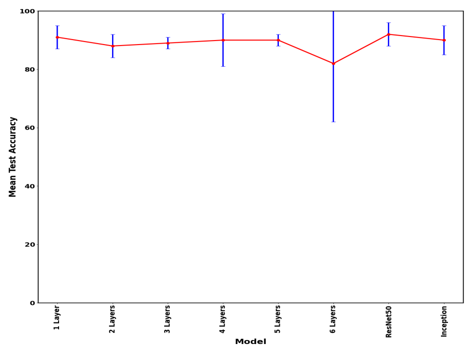
7.2 Smectic A and C

The smectic A and C phases are both ordered into layers, and they only differ by a tilted director, $\hat{\boldsymbol{n}}$. The total amount of data we had for the fluid group was significantly larger than the other classes in table 3. Also, the two separate phases which composed that group were approximately balanced in data size. Consequently, we created a binary classifier to attempt to distinguish between images for these two phases. The data set we used is shown in table 4.

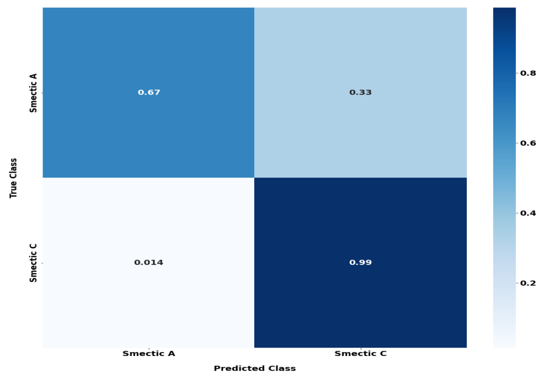
Phase	Total	Train	Validation	Test
Smectic A	1097	617	276	204
Smectic C	1448	884	282	282

Table 4: The data set used for smectic A and C classification.

It was found that the ResNet50 model had the best performance, with a test accuracy of (92±4)%. The results for all of the models are shown in figure 9, and it can be seen that almost every model achieved a mean test accuracy of at least 90%. This is likely because binary classification problems are generally easier to solve than general multi-class classification problems. In the case of the former, the model only needs to learn enough features to determine that an image is not one of the classes, or vice versa.



(a) Mean Test Accuracy



(b) Confusion Matrix

Figure 9: (a) contains the mean test accuracy values for the classification of smectic A and C. (b) is the confusion matrix for ResNet50.

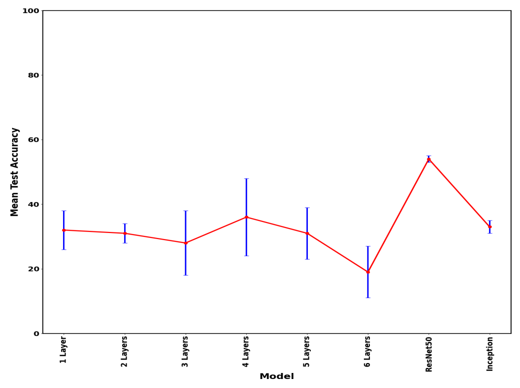
Although the overall test set accuracy was high for ResNet50, we can see in figure 9 that smectic A was incorrectly classified as smectic C 33% of the time. Table 4 shows that the number of smectic C images is approximately 30% greater than that of A. This slight imbalance and the moderate number of total images could be the reason for poor smectic A performance. It is hence believed that a larger and more balanced data set could produce an improvement.

7.3 6 Phases

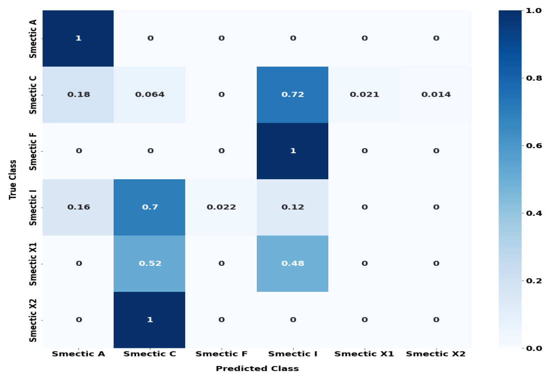
Our overall data set contained enough data for each of the six smectic phases to attempt to classify them. The data set we used for this problem is shown in table 5. All of the models performed poorly on the test set. The average of all of their test accuracy values was 33% and the model with the best performance was ResNet50, which achieved a test set accuracy of $(54 \pm 1)\%$.

Phase	Total	Train	Validation	Test
Smectic A	1097	617	276	204
Smectic C	1448	884	282	282
Smectic F	270	180	45	45
Smectic I	396	228	78	90
Smectic X1	420	258	96	66
Smectic X2	420	240	90	90

Table 5: The number of images in each of the six smectic phases for each set.



(a) Mean Test Accuracy



(b) Confusion Matrix

Figure 10: (a) and (b) are respectively the mean test accuracy graph for the classification of 6 smectic phases and the confusion matrix for ResNet50.

It can be seen in figure 10 that ResNet50 classified smectic A phases with perfect accuracy. However, it performed poorly for every other phase. Smectic C was incorrectly classified as smectic I 72% of the time, which was surprising because the former of the two phases contained the most images in the smectic data set. The large imbalance of data caused by smectic C appeared to have created a bias towards itself. This was inferred from the incorrect classification of the smectic I, X1 and X2 phases as smectic C a majority of the time.

It was hypothesised that the main source of the poor individual class accuracy values was the lack of data for every phase except smectic A and C. In future work, we will expand our smectic data set via the incorporation of images for the underrepresented phases. Additionally, we will test a greater range of high capacity networks, as ResNet50 had a larger capacity than every other model and outperformed the rest.

8 Summary of Joshua Heaton's Results

I obtained the results listed in sections 6 and 7 independently of my project partner Joshua Heaton. We collected and transformed the overall data set together and then decided to work in parallel on similar problems. This was done to increase productivity and vary the range of models used on different problems.

For the classification of 4 phases, the “2 Layers” sequential model, trained with flip augmentations, obtained a test accuracy of $(91 \pm 4)\%$. The flip augmentations group produced the models with the highest average test accuracy, and thus only that group was used for the other classification problems. The “4 Layers” model achieved the highest test set accuracy of $(97 \pm 1)\%$ for the binary classification of smectic A and C.

9 Conclusion

A range of CNNs were used for different classification problems. For the 4 phase classification, it was found that the sequential model with 3 convolutional layers performed the best with a test set accuracy of $(89 \pm 4)\%$. It was also concluded that the flip augmentations group produced the best model performances. The next problem was the classification of fluid, hexatic and soft crystal smectic phases. The sequential model with 4 convolutional layers achieved the highest test accuracy at $(86 \pm 6)\%$. Smectic A and C binary classifiers were created and ResNet50 achieved the highest test set accuracy of $(92 \pm 4)\%$. Finally, all of the models had poor performance for the classification of 6 smectic phases, where the best model was ResNet50 with $(54 \pm 1)\%$.

Confusion matrices were used to investigate test accuracy values for individual classes. The “3 Layers” model, trained on the data set with 4 phases, appeared to often classify smectic images as isotropic, and we thus concluded that a greater number and diversity of smectic images could help to prevent this from happening. The “4 Layers” model, that was trained to classify the three smectic groups, achieved an accuracy of 76% for each of the fluid and hexatic groups and a perfect accuracy for the soft crystal group. ResNet50 was observed to correctly classify smectic C images for the binary classification with almost perfect accuracy, however it achieved only 67% for smectic A. The individual class accuracy values for the classification of 6 smectic phases with ResNet50 were all close or equal to 0% apart from smectic A, which was classified with a perfect accuracy. Smectic I, X1 and X2 were incorrectly classified as smectic C more than 50% of the time. It was suggested that the significant imbalance of the data set in table 5 could have created a bias towards smectic C and hence negatively affected other class accuracy values.

There were no clear trends between model choice and test accuracy for the classification of 4 phases, smectic A and C and 6 smectic phases. Despite this, it was found that ResNet50 had the best performance for data sets with individual smectic phase classes. Thus, alongside a larger and more balanced data set, the investigation into networks with an even higher capacity than ResNet50 could be required to learn enough meaningful features to distinguish between individual smectic phases. Finally, for the classification of the fluid, hexatic and soft crystal groups it was observed that mean test accuracy generally increased from “1 Layer” up to “4 Layers”, whereafter the accuracy gradually decreased. This suggested that the networks with the highest capacities were suffering from overfitting, where the high level features they learned could have simply been noise in the images. Also, the addition of new hexatic group images could reduce the incorrect hexatic-fluid classification rate.

10 Future Work

We will first tackle the most obvious problem which is the data set sizes and class imbalances. Our supervisor will be able to provide us with new data for the underrepresented smectic phases. We believe that this will improve the test accuracy values for the smectic 6 phases and fluid, hexatic and soft crystal group classifiers. Furthermore, if images paired with continuous numerical data are obtained then we could approach several regression problems. The numerical data could correspond to a physical property of an LC such as the pitch length of a cholesteric phase.

It was stated in section 7.3 that an investigation into more high capacity networks could be beneficial. We will not be simply searching for networks with more parameters, as this is not guaranteed to improve performance and can cause overfitting. Specifically, we will use networks which have achieved a high test accuracy for the "ImageNet" data set. This is one of the most widely used standards for testing state of the art image classification models. The data set consists of 14 million images and 22 thousand individual classes [18]. Additionally, we could investigate entirely different types of NNs. One example is transformer networks which have shown promising results for computer vision problems, however the approach is still in its infancy [19].

References

[1] Russell, S.J., Norvig, P., *Artificial Intelligence: A Modern Approach.* Prentice Hall, third ed., 2010.

[2] Silver, D., Huang, A., Maddison, C. et al., "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, p. 484–489, 2016.

[3] Cruz, J. A., Wishart, D. S., "Applications of machine learning in cancer prediction and prognosis," *Cancer Informatics*, vol. 2, pp. 59–77, 2006.

[4] Senior, A.W., Evans, R., Jumper, J., Kirkpatrick, J., Sifre, L., Green, T. et al., "High Accuracy Protein Structure Prediction Using Deep Learning." In Fourteenth Critical Assessment of Techniques for Protein Structure Prediction (Abstract Book), 2020.

[5] Goodfellow, I., Bengio, Y., Courville, A., *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[6] Collings, P.J., Hird, M., *Introduction to Liquid Crystals Chemistry and Physics.* Taylor and Francis, first ed., 1997.

[7] Dierking, I., Al-Zangana, S., "Lyotropic liquid crystal phases from anisotropic nanomaterials," *Nanomaterials*, vol. 7, 2017.

[8] Dierking, I., *Textures of Liquid Crystals.* WILEY-VCH, 2003.

[9] `https://www.instagram.com/accounts/login/?next=/vance.williams/`. Accessed: 1/1/21.

[10] `https://www.youtube.com/channel/UCB8qnCxJbdsuXpQ5RbLNy3Q`. Accessed: 1/1/21.

[11] Schacht, J., Dierking, I., Gießelmann, F., Mohr, K., Zaschke, H., Kuczyński W., Zugenmaier, P., "Mesomorphic properties of a homologous series of chiral liquid crystals containing the -chloroester group," *Liquid Crystals*, vol. 19, pp. 151–157, 1995.

[12] Kaufman, S., Rosset, S., Perlich, C., "Leakage in data mining: Formulation, detection, and avoidance," *ACM Transactions on Knowledge Discovery from Data*, vol. 6, 2011.

[13] Krawczyk, B., "Learning from imbalanced data: open challenges and future directions," *Progress in Artificial Intelligence*, vol. 5, pp. 221–232, 2016.

[14] He, K., Zhang, X., Ren, S., Sun, J., "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[15] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A. et al., "Going deeper with convolutions," *arXiv preprint arXiv:1409.4842*, vol. 1409, 2014.

[16] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C. et al., "TensorFlow: Large-scale machine learning on heterogeneous systems." Software available from tensorflow.org.

[17] F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

[18] "Image Classification on ImageNet." `https://paperswithcode.com/sota/image-classification-on-imagenet`. Accessed: 3/1/21.

[19] Dosovitskiy, A. et al., "An image is worth 16x16 words: Transformers for image recognition at scale," 2020.