MANCHESTER 1824
The University of Manchester

Developing a convolutional neural network for the classification of liquid crystal textures and phase transitions

Semester one MPhys report submitted in partial fulfillment of the requirements for the degree of

MPhys Physics

in the Department of Physics and Astronomy

YEAR OF SUBMISSION

2021

Jason Dominguez

ID 10154707

THE UNIVERSITY OF MANCHESTER

Abstract

Different convolutional neural network (CNN) architectures were trained for the classification of isotropic, nematic, cholesteric and smectic liquid crystal phase textures. Of the models trained, an eight-layer CNN and an Inception CNN were applied to unseen liquid crystal texture transition test videos. The eight-layer model achieved 69.8% accuracy on a validation set of 736 unseen textures and the Inception model achieved 95.9% on 1546 unseen textures.

On five videos of the 8CB liquid crystal, the models detected nematic-smectic transitions to occur at $(33 \pm 1)^{\circ}$C. This was consistent with the known transition temperature of 33.6°C. For seven videos of 8CB and the M6 liquid crystal, with transitions involving the isotropic, cholesteric and smectic phases, the models accurately identified transitions involving the isotropic phase. However, the models were unable to accurately detect transitions involving the cholesteric and smectic phases. The similarity in the performance of these two models, despite the difference in validation accuracy achieved, indicated that the validation set of textures used was not an accurate predictor of performance on the test videos.

The same Inception model architecture was also trained for differentiating between fluid smectic and hexatic phase textures. This model achieved 98.2% accuracy on a validation set of 216 images. Despite this high accuracy, when applied to two unseen transition videos, the model did not correctly identify the transition between fluid smectic and hexatic phases.

Finally, improvements to the work done in the project so far were suggested, such as a different method for validation models during training.

Contents

1 Introduction

1.1 Background and Motivation

Liquid crystals (LCs) are fluids with partial order, between that of the isotropic and crystalline phases [1]. Whilst perfect crystals have full three-dimensional order and isotropic fluids have none, LC phases have orientational order and sometimes one-, two- or three-dimensional positional order [2].

Figure 1 shows a non-exhaustive categorization of the different LC phases most relevant in this project. Two main types of LCs are thermotropic, whose phases change with temperature, and lyotropic, whose phases change with concentration [2]. Beyond this, thermotropic LCs can be divided by molecular shape, such as rod-like, known as calamitic, and type of order present. The Nematic (N) LC phase is distinguished by the presence of long-range orientational order, with no long-range positional order [1]. This is shown in Figure 1 with the molecules pointing along the director, $\hat{n}$. For molecules with no center of inversion symmetry (chirality) [3], the director can vary throughout the LC. This makes it chiral nematic, known as cholesteric, (N*). Smectic LCs has some form of positional order. The fluid smectic phases, SmA and SmC, have one dimensional positional order. Molecules arrange into layers, with SmC having an additional tilt of molecules with respect to the normal of these layer [1]. These are shown in Figure 1. The hexatic smectic phases, SmF and SmI, also have short-range positional order with the layers formed.

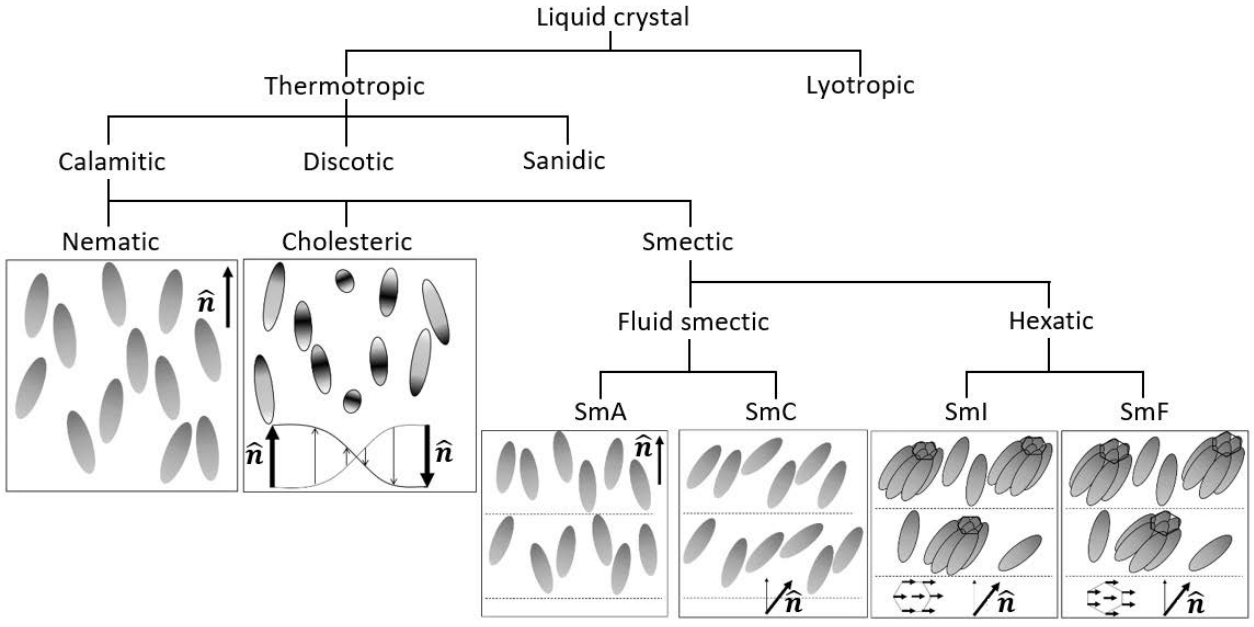


Figure 1: Diagram showing the categorization and molecular ordering of the N, N*, SmA, SmC, SmI and SmF phases. For N*, the darker part of the oval indicates orientation direction. Dotted lines indicate the boundary of smectic layers. Figure modified from [1] and [2].

When new compounds are formed, it is important to study their properties. This includes which LC phases the compound exhibits, if any, and at what temperatures these phases occur. For

example, in the application of LCs to liquid crystal displays (LCDs), the optical properties and operational temperature of the LC, which depend on the phase and transition temperatures respectively, are critical to the performance of the device [4]. Two leading techniques for determining the properties of LCs are polarizing-optical microscopy (POM) and differential scanning calorimetry (DSC).

LCs exhibit anisotropy. This causes light to propagate at different speeds through a LC, depending on its polarization direction [2]. POM utilizes this property of LCs to produce images known as textures [1]. In POM, polarized light is transmitted through a LC. It then has to pass through a polarizer, perpendicular to the polarization direction of the incident light [5]. Anisotropy changes the polarization of the transmitted light, so that the light passes through the crossed-polarizer and is detected. Alternatively, isotropic fluids leave the polarization unaffected, resulting in a black texture. Different LC phases produce different characteristic textures, some of which are shown in Figure 2. From the observation of these textures and how they change with temperature, the phase transitions of a LC can be identified [5]. However, accurate identification of phases and transitions from these textures requires expertise. This is important when the phase becomes more ordered, as differences between phases become more subtle [1].

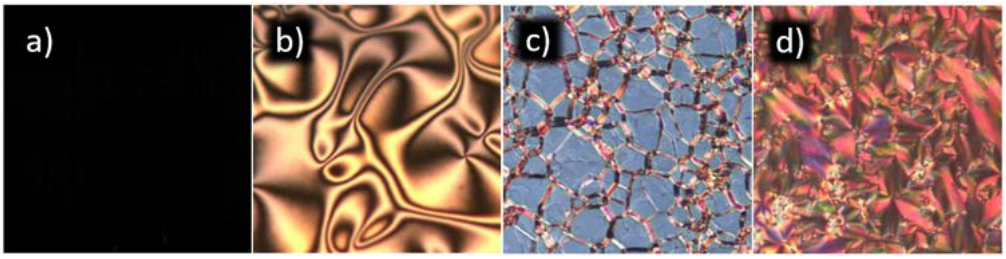


Figure 2: a) An example of the texture produced for an isotropic fluid. b) Schlieren texture of the nematic phase. c) 'Oily streaks' texture of a cholesteric LC. d) Texture of the SmC phase. All images were taken by members of the Soft Matter and Liquid Crystals group at the University of Manchester [6].

DSC, unlike POM, is only used for determining phase transition temperatures and does not obtain information about the phases of a LC [2]. When a LC undergoes a phase transition, there is an associated change in enthalpy and entropy. This corresponds to a change in the order of the system [7]. DSC indirectly detects this change as a function of temperature [7]. An example of a graph obtained for LC phase transitions is shown in Figure 3.

Currently, POM and DSC can be utilized together. POM provides phase and transition temperature information and DSC provides further transition information. This project follows on from the work done by Sigaki et al. [9], who used machine learning to accurately identify the isotropic and nematic phases from their textures. A machine learning model capable of running with accessible computational power, which can identify liquid crystal phases and transitions, would be useful alongside the current leading techniques. This model would provide benefits over POM in that it would not require extensive LC experience and benefits over DSC in that it

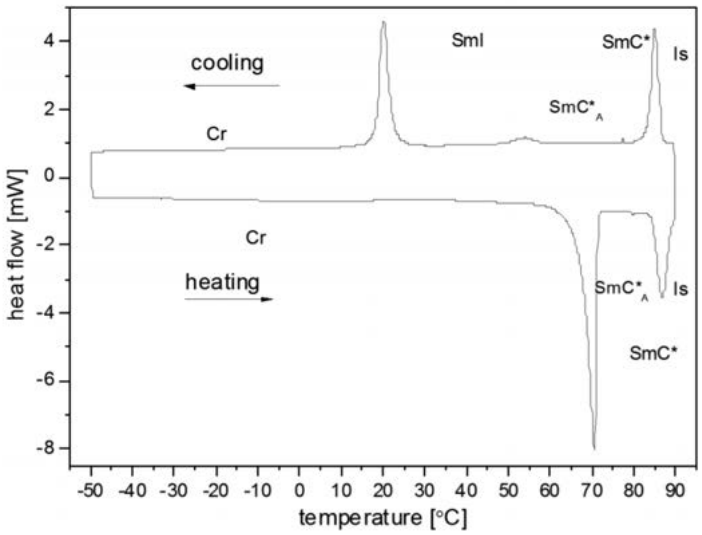


Figure 3: An example of a heat flow against temperature graph obtained using DSC. Peaks and troughs indicate transitions. Figure taken from [8].

would be able to provide information about the phase, as well as the transition temperature.

1.2 Aims and Objectives

The original aim of this project was to apply machine learning techniques to the task of classifying LC textures. The objectives for achieving this aim were to: build up a dataset of LC textures to use, choose one or more machine learning algorithms that would be suitable for the classification, implement/train a machine learning algorithm/model for the classification, and evaluate the success of the model on unseen LC textures.

In the early stages of this project, this aim was extended from the classification of individual LC texture images to the frame by frame classification of LC texture videos. With this change, the aim of the project was to apply machine learning techniques to the classification of LC texture videos to determine the identity and temperature of LC phase transitions. The objectives for achieving this new aim remained the same, however the trained model was to be evaluated on LC texture videos.

2 Theory

2.1 Machine learning

2.1.1 Supervised Machine Learning and Deep Learning

In supervised machine learning there is a set of data, known as training data, for which the output is known [10]. Using this, the role of supervised learning is to learn the mapping from the

inputs to the known outputs [11]. This mapping is then used on unseen data. A simple example of this is linear regression, whereby x coordinates and their corresponding y coordinates are known and a mapping, or linear fit, is learned. This can map new x coordinates to y values. Within the field of machine learning, deep learning is the current leading technique, being held largely responsible for the rapid and most powerful advances in machine learning [11].

Artificial neural networks (ANNs) are an example of deep learning. These consist of layers of 'neurons'. The connection of many neurons, which perform different computations, allows for the ANN to model a complex function [12]. The complexity of this function is increased by increasing the number of neurons and or layers in the network [10]. Training an ANN follows the form: forward propagation, backward propagation, weight updates and repetition. For each neuron, i, in the layer l of an ANN, there are associated weights, $\boldsymbol{W}_i^{[l]}$ [12]. In forward propagation, the ANN is presented with one or more data examples, represented by a feature vector, $\boldsymbol{x}$. Each element of $\boldsymbol{x}$ is taken as the input for each neuron of the first layer. For images, $\boldsymbol{x}$ could be the pixel values flattened from a matrix to a column vector. With the input, each neuron, i, computes

$$\sum_{k=1}^{n} W_{i,k}^{[l]} x_k + W_{i,0}^{[l]} \tag{1}$$

[10]. A non-linear function is then applied to this value, with a common choice being the rectified-linear-unit (ReLU) function. The outputs of neurons in one layer are the input to each neuron in the next layer. This continues until the output layer. Each neuron in this layer computes an output, or prediction, $y_{\text{pred},i}$. With the output of the network, the loss is calculated. For classification problems, a commonly used loss function is the categorical cross-entropy loss, which is

$$\mathcal{L} = -\sum_{i}^{N} y_i \log(y_{\text{pred},i}) \tag{2}$$

[10] for a single example input. Here, the sum is over each output neuron of the network, the number of prediction classes N, and y is the expected output. For an output layer with three neurons, an output could be $[0.2, 0.1, 0.7]$. An example of an expected output could be $[0, 0, 1]$, representing the label of the third class in this three-class classifier. The backpropagation algorithm then uses this loss to compute the derivatives used to update the weights via a gradient-based optimization algorithm, such as gradient descent,

$$\boldsymbol{W}^{[l]} := \boldsymbol{W}^{[l]} - \alpha \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^{[l]}} \tag{3}$$

[12], where α is a predefined learning rate. The Adam algorithm is a commonly used variant of gradient descent which has been shown to reach convergence when minimizing the loss function sooner than standard gradient descent [13]. The entire process is repeated until the loss, which is inversely related to accuracy, is lowered to convergence.

2.1.2 Bias-Variance Tradeoff and Generalizability

When applying deep learning to a dataset it is common practice to have a training set and test set. The methodology is that a model is trained using the training set examples and then applied to the test data to test if the model generalizes to unseen data [11].

One problem faced in the training of a model is the bias-variance tradeoff, also known as under or overfitting. A model can overfit to the training data, meaning it has high variance and low bias. This is seen by a high training accuracy and a lower test accuracy [10]. Figure 4 shows an example of this, where a model has learned to recognize the noise in training data. This will not generalize well to unseen data. Alternatively, if the training accuracy lower than desired, then the model is said to underfit the training data - low variance, high bias. The ideal model will achieve a test accuracy as high as possible [11].

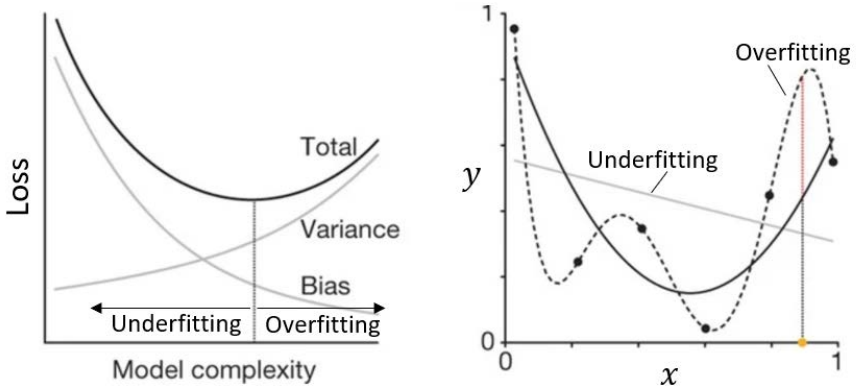


Figure 4: Graphs showing the effects of under and overfitting and how loss can indicate this. Figure edited from [14].

A validation set of data is often used during training. This is a small portion of the training data which is held-out and not used for updating weights. For every epoch, which is an iteration through all the training data, the accuracy on the validation set is computed. Monitoring this accuracy can identify when the model is overfitting training data, as the validation accuracy will be lower than the training accuracy. Using this information, hyperparameters of the model, such as number of layers, can be changed [11].

Some methods for reducing overfitting include: reducing the model complexity (i.e. number of layers or parameters of the model), using dropout layers, or getting a wider range of training data [10]. Dropout layers are used in an ANN, during training, to randomly ignore a set number of neurons each epoch. This reduces model complexity during training. In the example of image classification, it is common to use image augmentation. This is where images are randomly selected, each epoch, to have some processing applied. By making images look different, such as through flipping them, it is effectively like using new images to train the network [10].

2.2 Convolutional Neural Networks

Figure 5 shows the LeNet-5 architecture [15], an example of a convolutional neural network (CNN). CNNs are a type of ANN which perform better for image classification problems. They include convolutional and subsampling layers, before the output is flattened into a vector and passed through an ANN. As shown in Figure 6, convolutional layers have filters of a certain size and perform convolutions over the entire area of the input. Rather than having weights to update, convolutional layers update the parameters in the filters [10].

Subsampling, or pooling, layers, such as max pooling, look at a certain sized area of the input and output the maximum value in that area [10]. This is shown in Figure 6. There are no trainable parameters in pooling layers, so they help reduce the computational cost of a CNN.

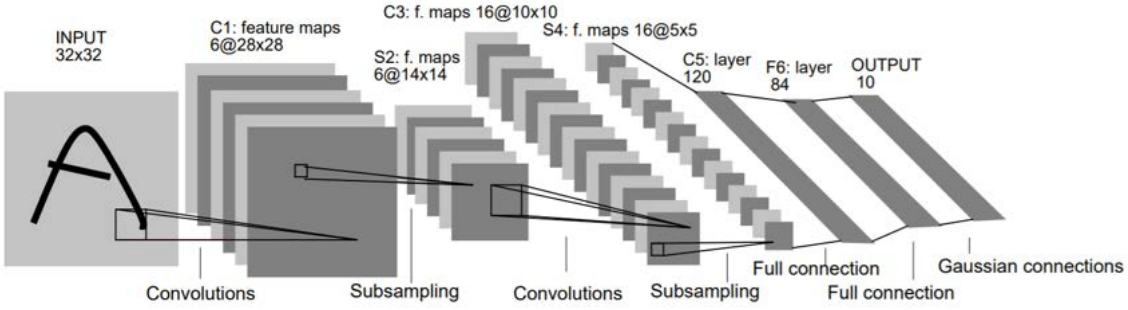


Figure 5: Diagram of the LeNet-5 CNN architecture, with the number of filters and filter size, or number of neurons shown for each layer. Figure taken from [15].

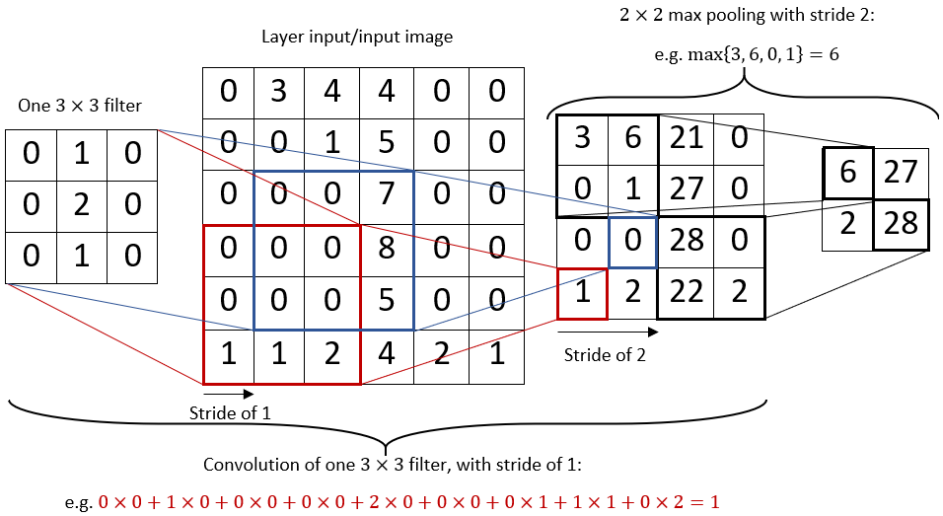
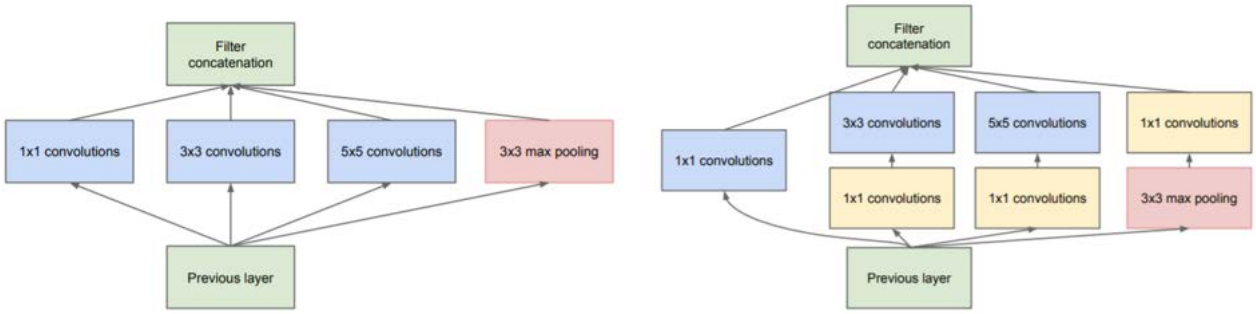


Figure 6: Diagram showing an example of a convolution operation, followed by max pooling. The stride of each indicates the distance between adjacent pooling or convolution operations on the input. A single convolution calculation is shown in red, where each element of the filter multiplies the corresponding element of a section of the input image.

2.3 Inception Networks

In 2014, researchers at Google designed GoogLeNet, an Inception network, which achieved the highest accuracy in the 2014 ImageNet image classification competition [16]. Inception networks replace the convolutional layers in CNNs with Inception modules, as shown in Figure 7.



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 7: Diagrams of the Inception module taken from [16].

The Inception module utilizes layers with different properties, such as filter size, or type, in parallel. One benefit of this is that it removes the need to guess what layer should be used. Instead, by having different layers in parallel, the network can learn which layer is best suited, through the training process. Having different layers in parallel is very computationally expensive, so to mitigate this, 1×1 convolution layers are used to reduce the number of computations performed, as shown in Figure 7.

3 Collection and Preparation Of Texture Images

Before the implementation of any supervised deep learning model, POM texture images and transition videos needed to collected and images grouped by phase. The complete dataset obtained so far in the project consists of 11773 textures of different phases, as shown in Table 1, as well as 107 POM videos of various phase transitions.

Table 1: Number of images for each phase in the complete dataset of textures

Isotropic	Nematic	Cholesteric	Unspecified Smectic	SmA	SmC
1981	2813	2132	766	1284	1448
SmE	SmF	SmI	SmX1	SmX2	Crystalline
105	270	396	420	420	504

Figure 8 shows a summary of the image preprocessing done for this project. Most images used in this project came from POM phase transition videos of the 5CB, 8CB, M6-10 and D5-8 LCs, taken by members of the Soft Matter and Liquid Crystals group at the University of Manchester [6]. To get images from the videos, the open-source VLC media player [17] was used to extract

the frames of the videos. Frames were captured every five to twenty frames depending on how fast changes occurred in the video, as having many, almost identical, images would take up more storage space. This have little benefit for a deep learning model. The images obtained from these videos had a resolution of 2048×1088. As this was a large resolution, the images were then split into six 682×544 images, increasing the total number of different images in the dataset to that shown in Table 1. This resolution still allowed key features of the textures to be identified. The collection and processing of images from videos using VLC media player is shown in Appendix A.

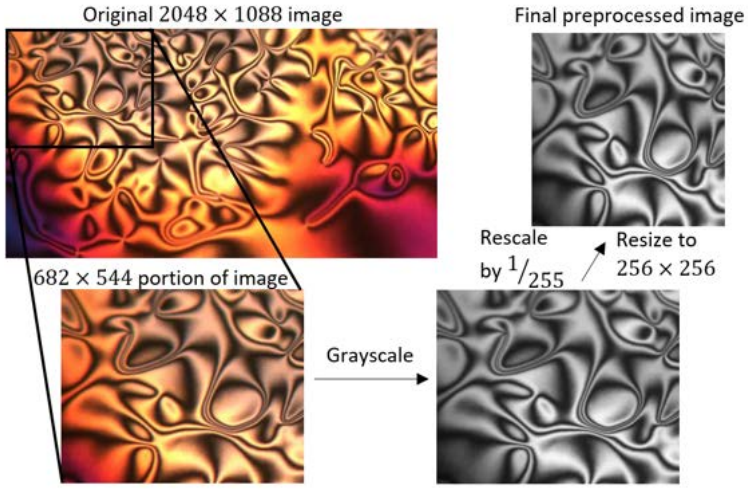


Figure 8: Diagrams of all the images preprocessing steps done before use in a deep learning model, on an example nematic texture.

All images of a single phase were grouped in the same folder in storage. The videos, from which the images were extracted, were given a LC name and temperature range. Using this information along with the known phase transition temperatures of the LCs, the phase of the images could be identified. Images of the LCs M6-10 and D5-8 were grouped using literature [18] and [19] respectively. Images from the 5CB and 8CB LCs were able to be identified visually as they exhibited visually distinct isotropic, nematic and smectic phases.

A smaller proportion of images were acquired from images and videos posted on Instagram and YouTube, courtesy of Vance Williams of the Department of Chemistry at Simon Fraser university [20] [21]. The phases of these images obtained from these sources were identified via the captions of the videos or photos and had resolutions between 320×360, 480×480, 640×640, 640×720 or 720×720.

For this project, the Keras deep learning framework [22] in Python was used to implement deep learning models. In all these models, real-time preprocessing was done via Keras. Images were converted to grayscale. This was done because texture, rather than color, determines the phase. Also, it can save computation time. As there were images of different sizes, images were also resized to 256×256. This was chosen as it is square and the dimensions are powers of two. Both of these are common practices with CNNs. Also, this size was small enough to reduce

computation time, without obscuring texture details. Finally, the images were rescaled by $\frac{1}{255}$, so that pixel values were between 0 and 1. This is required for CNNs to work correctly.

4 Data Selection and Training Image Classification Models

4.1 Isotropic, Nematic, Cholesteric and Smectic Phase Classification

4.1.1 Method

The first attempt at applying machine learning to LC phase transitions was done for phase transitions involving the isotropic, nematic, cholesteric or smectic phases, as these are the easiest to distinguish visually. To achieve this, an isotropic, nematic, cholesteric and smectic image classifier needed to be trained.

The first model was created early in the project, before all the images had been collected. Before training a model, it was necessary to create training and validation image sets and a test set of phase transition videos. The videos selected for the test set were those of the 8CB and M6 LCs, as these involved a mix of different transitions between the phases. To avoid data leakage, where a model has effectively 'seen' the answer before, any images extracted from these videos were not included in the training or validation sets. Images from a single video were not split between training and validation, again to avoid data leakage. Also the sets were made balanced, with a roughly equal number of images of each phase, by deleting similar images from over-represented phases. This was to avoid bias towards predicting one phase.

Table 2 shows the first dataset used for classification of these phases. The validation set was chosen to be roughly 15% of the total training and validation set size, to allow for enough images to train the model.

Table 2: Number of training and validation images for each texture in the first dataset used in this project.

Phase	Training images	Validation images
Isotropic	666	186
Nematic	597	166
Cholesteric	670	191
Smectic	674	193

As this was early in the project, only one model was trained on this dataset. This model was an eight-layer, LeNet-5-inspired, model, with dropout layers and random flipping, rotation and zooming augmentation, to attempt to reduce overfitting of the model. The model architecture is shown in Appendix B.2.1. The model was trained using a default adam optimizer and a categorical-crossentropy loss, using Keras, which are common choices in image classification problems.

Later in the project, once all the images in Table 1 had been obtained via the use of the preprocessing described in Section 3, a new training and validation dataset was made. This can be seen in Table 3. The same test videos were chosen as for the previous model. The training-validation split was also approximately 15% validation images and the same precautions for avoiding data leakage were followed. Due to the relatively large number of some textures, for example 3299 smectic images, excess images were deleted in order to create a more balanced dataset, as above. Using this dataset, Inception networks and CNNs of varying layers were trained, to find the model architecture with the best validation accuracy. Each model was trained using a default adam optimizer and categorical-cross entropy loss function. Training was done using a Google Colaboratory [23] GPU, allowing for quicker training times compared to using a standard CPU.

Table 3: Number of training and validation images for each texture in the second dataset used in this project.

Phase	Training images	Validation images
Isotropic	1446	259
Nematic	1367	392
Cholesteric	1547	352
Smectic	1499	543

4.1.2 Results

For the first model trained for this classification problem, the training and validation accuracy and loss were computed after each training epoch and plotted, as shown in Figure 9. There is a lot of fluctuation in the validation accuracy and loss. This is possibly due to the relatively small size of the validation set, where a change in prediction for a small number of images can have a larger effect on the accuracy and loss. Overall, this shows that the validation accuracy reached its maximum value at epoch 86 with a value of 69.8%. However, the validation loss is seen to increase. This is usually a sign of overfitting to the training images, also shown by the training accuracy being much higher than the validation accuracy. Figure 9 also shows the confusion matrix for this trained model for the validation set images. The ideal model has 1 for every entry in the diagonal, indicating that each image was predicted correctly. From Figure 9 it can be seen that this model mislabelled 75.15% of the validation smectic phases as nematic, indicating that it had not learned to distinguish the two.

Table 4 shows the results of training different model architectures for sixty epochs on the training and validation split shown in Table 3. From this table, it can be seen that, for CNN models, increasing the number of layers increased the validation accuracy. Overall, Inception networks achieved higher accuracy, with InceptLC-V4 model achieving the highest validation accuracy, of 95.9%. With a training time per epoch of 90 s, the total training time was approximately 1.5 hours. The model also showed less overfitting to the training data than most

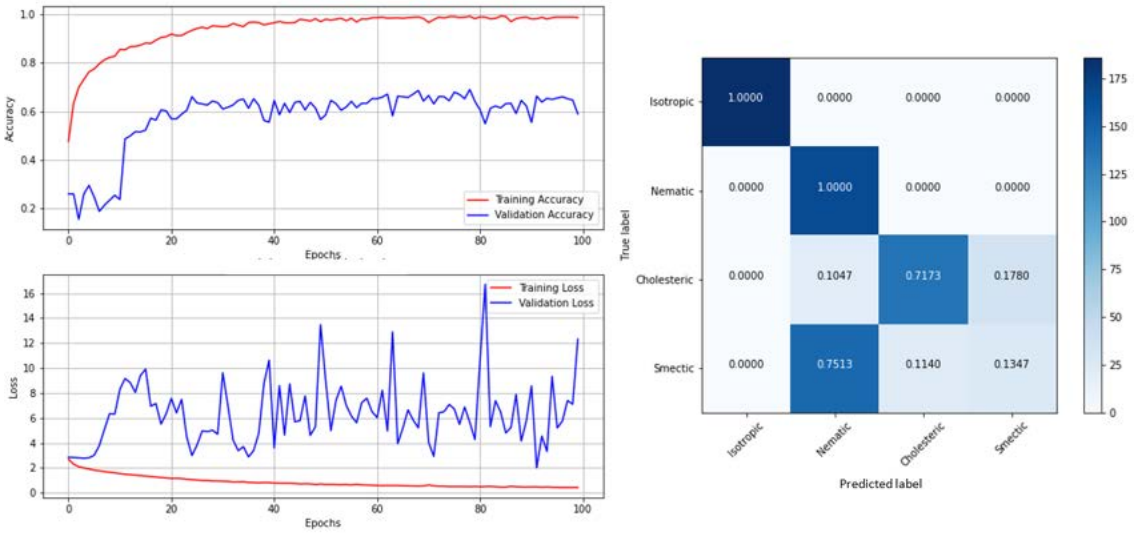


Figure 9: A graph showing the training and validation accuracy and loss for each training epoch. The confusion matrix on the right shows a comparison between the saved model's predictions on the validation set images and the true image labels. The blue bar indicates number of images.

models. This can be seen in Figure 10, as its loss is mostly decreasing. The confusion matrix for this model, shown in Figure 10, shows that all phase textures were correctly predicted with a high accuracy. Overall, this model showed a more accurate performance on the unseen validation images than the previous model, shown by Figure 9.

Table 4: Results for different model used. Max val acc is the maximum validation accuracy of three training repeats of each model. Range is the range of these three results. Other values correspond to the max val acc epoch. For layout, layers are: CP = convolution-max pooling, FC = ANN, 0.5D = 0.5 dropout, P = average pooling, I = Inception module, F = flipping augmentation. Numbers indicate repeated layers.

Model name	Layout	Max val acc (%)	Range (%)	Val loss	Train acc (%)	Epoch
ConvLC-V1	2(CP)-FC	75.4	4.8	1.24	100	7
ConvLC-V2	F-2(CP)-FC-0.5D	74.5	7.2	0.64	65.6	18
ConvLC-V3	3(CP)-FC	81.4	8.3	1.00	99.1	3
ConvLC-V4	F-3(CP)-FC-0.5D	82.9	7.7	0.72	88.3	23
ConvLC-V5	4(CP)-FC	82.5	4.4	7.33	97.7	4
ConvLC-V6	F-4(CP)-2(FC-0.5D)	89.5	6.6	0.43	87.1	45
InceptLC-V1	CP-I-P-FC	90.8	5.2	0.57	93.4	57
InceptLC-V2	CP-I-P-FC-0.5D	88.0	6.3	0.37	87.9	46
InceptLC-V3	CP-2(I)-P-FC	95.9	5.6	0.29	98.9	42
InceptLC-V4	F-CP-2(I)-P-FC-0.5D	95.9	5.2	0.22	98.6	57

Both the InceptLC-V4 Inception model and the first model trained for this part of the project were saved at their epoch of highest validation accuracy for use in the video classification part of the project. The InceptLC-V4 model architecture is shown in Appendix B.3.1.

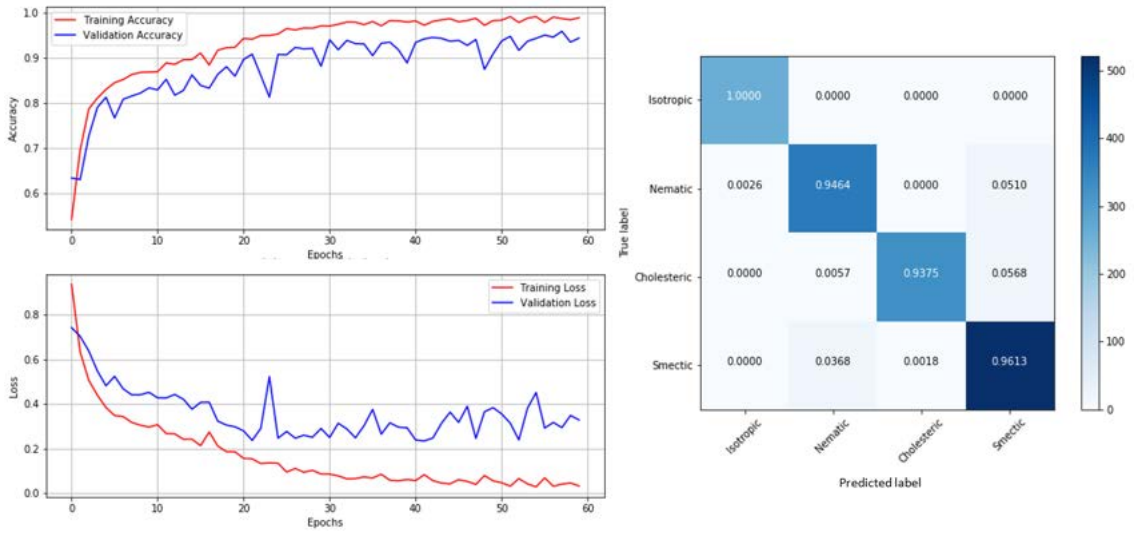


Figure 10: A graph showing the training and validation accuracy and loss for each training epoch. The confusion matrix is shown on the right. The blue scale bar indicates number of images.

4.2 Fluid Smectic and Hexatic Phase Classification

4.2.1 Method

For creating a classifier to identify transitions between fluid smectic and hexatic phases, the first step was to again create an image classifier. Because of the small number of fluid smectic and hexatic images, relative to other phases, a dataset containing only fluid smectic and hexatic images was created. This was to avoid having to distinguishing among many phases, including fluid smectic and hexatic. The training set consisted of 777 fluid smectic textures and 516 hexatic textures. The validation set consisted of 114 and 102 fluid smectic and hexatic textures, respectively. Fluid smectic images comprised of a roughly equal number of SmA and SmC images, whereas the hexatic images comprised of SmF and SmI images with an approximate ratio of $2:3$, due to the availability of images from each phase. As described in Section 4.1.1, the same methods to balance the dataset and avoid data leakage were followed. Only two test videos were chosen, to avoid further limiting the number of images available for the training and validation sets.

4.2.2 Results

Figure 11 shows the accuracy and loss curves for the InceptLC-V4 model, applied to this classification problem. At epoch 46, the highest validation accuracy, 98.2%, was achieved. The model was saved at this epoch. After this epoch, the loss starts to increase, indicating that overfitting was starting to occur. From the confusion matrix for the saved trained model, shown in Figure 11, it can be seen that predictions of unseen images for each class were highly accurate.

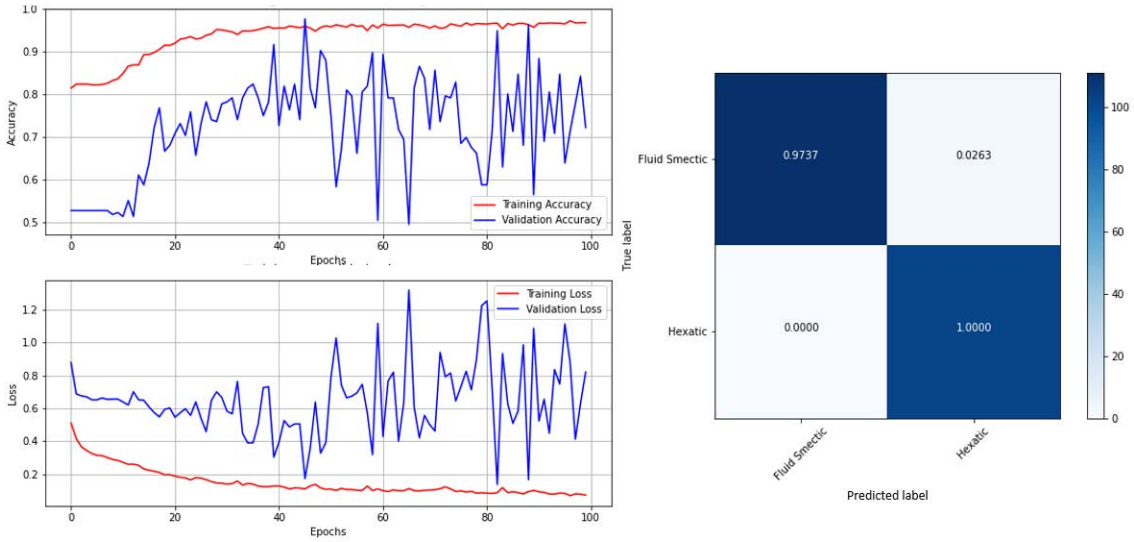


Figure 11: A graph showing the training and validation accuracy and loss for each training epoch. The confusion matrix is shown on the right. The blue scale bar indicates number of images.

5 Applying Trained Models to Phase Transition Videos

5.1 Method

For classifying videos of LC phase transitions, a model, trained for the phase classification of LC texture images, was applied to each frame of a video. This generated a phase prediction for that frame. The prediction and its estimated confidence were then written onto the frame and all the labelled frames were then stitched together. This created a completely labelled phase transition video. The OpenCV library in Python was used to perform: frame extraction, resizing to 256×256, converting to grayscale, labelling and stitching. The script written for this is shown in Appendix B.1.

The estimated confidence of a prediction depended on the model. For a multiclass classification model, such as that used in Section 4.1, a single prediction would be a vector, for example $[0.2, 0.65, 0.1, 0.05]$, with all elements adding to 1. This example would give the prediction of class two, as the second element of the prediction vector is the maximum. For the classifier in Section 4.1, this would have corresponded to a nematic prediction. For this prediction, the value of 0.65 would be taken as the estimated prediction confidence. For a binary classifier, such as that used in Section 4.2, only a single value between 0 and 1 is predicted. If this value rounds to 1, the prediction will be the second class. The estimated confidence will be taken as the value, for example 0.6. If the values rounds to 0, the first class will be predicted. The estimated confidence of this prediction would be the value taken away from 1. For example, a prediction of 0.4 would have an estimated confidence of 0.6 for the first class.

As well as labelling videos with the predicted phase and estimated confidence, the videos were labelled with the temperature. From this, graphs of phase prediction confidence against temperature were plotted to give a graphical representation of the predicted phase transition.

Most videos obtained in this project were titled with their initial temperature, T_{initial}, and either their final temperature, T_{final}, or the rate of temperature change, $\frac{\Delta T}{\Delta t}$. A positive $\frac{\Delta T}{\Delta t}$ indicated heating the LC and vice versa for cooling. Using this information, along with the frames-per-second, fps, or number of frames in the video, n_{tot}, the temperature interval between frames, Δ, was calculated as

$$\Delta = \frac{T_{\text{final}} - T_{\text{initial}}}{n_{\text{tot}}}, \tag{4}$$

if T_{final} was known, or

$$\Delta = \frac{1}{\text{fps}} \frac{\Delta T}{\Delta t}, \tag{5}$$

if $\frac{\Delta T}{\Delta t}$ was known. In both cases, the temperature associated with each frame, $n = 1, 2, ..., n_{\text{tot}}$, was calculated as

$$T_n = T_{\text{initial}} + n\Delta. \tag{6}$$

5.2 Results

5.2.1 Isotropic, Nematic, Cholesteric and Smectic Phase Classification

Table 5 shows a summary of the performance of the first model from Section 4.1 on the test videos. A video was considered correctly labelled if the correct phases were identified and the detected transition temperature was consistent with values from literature. 'Partially correctly labelled' refers to videos where the model either correctly identified one, but not all transitions, or mislabelled a phase cholesteric instead of nematic. The justification for the last exception is that, in practical setting, the chirality of a LC if often known. Therefore, if a nematic phase is predicted for a chiral LC, it can be assumed to be cholesteric and vice versa.

Table 5: Table showing how test videos were classified by the first trained model, for different phase transitions.

-	I↔ N	I↔ N*	N↔ Sm	N*↔ Sm	I↔ N*↔ Sm
Correctly labelled	1	0	5	0	0
Partially correctly labelled	1	2	0	0	1
Incorrectly labelled	1	0	0	2	0
Total videos	3	2	5	2	1

As seen from Table 5, all five videos of transitions between nematic and smectic phases were correctly classified. Figure 12 shows the confidence against temperature graph for one of these transition videos. From this graph, the transition temperature is predicted to be $(33 \pm 1)^{\circ}$C, which is consistent with 33.6°C from the literature [24]. The uncertainty in this result is due to the absolute temperature uncertainty of the device used for cooling or heating the LCs. The performance of this model for these transitions was unexpected, as it had mislabelled many smectic textures as nematic in the validation set. This shows that the characteristics of textures in the validation set may not have been representative of the textures in the test set videos.

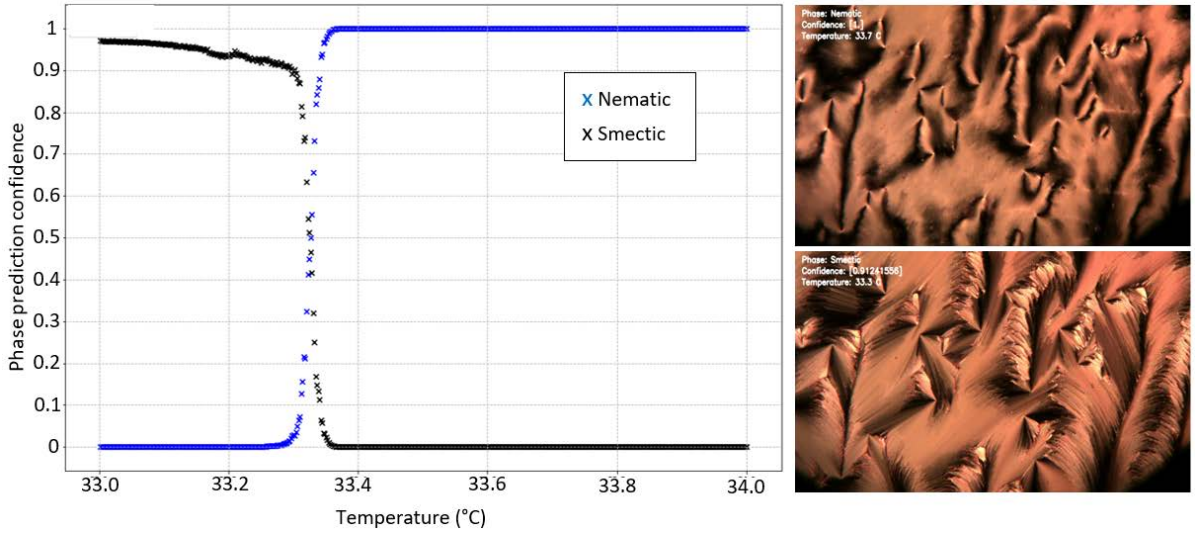


Figure 12: Figure showing the phase prediction confidence against temperature graph for a transition between the nematic and smectic phase. Alongside this are screenshots from the labelled video, either side of the transition, to show the texture differences.

Figure 13 shows the 'partially correctly labelled' isotropic to nematic transition, labelled as an isotropic to cholesteric transition. The transition temperature in this graph is $(41 \pm 1)°$C, with the uncertainty chosen as explained above. This is consistent with the literature value of $40.5°$C [24].

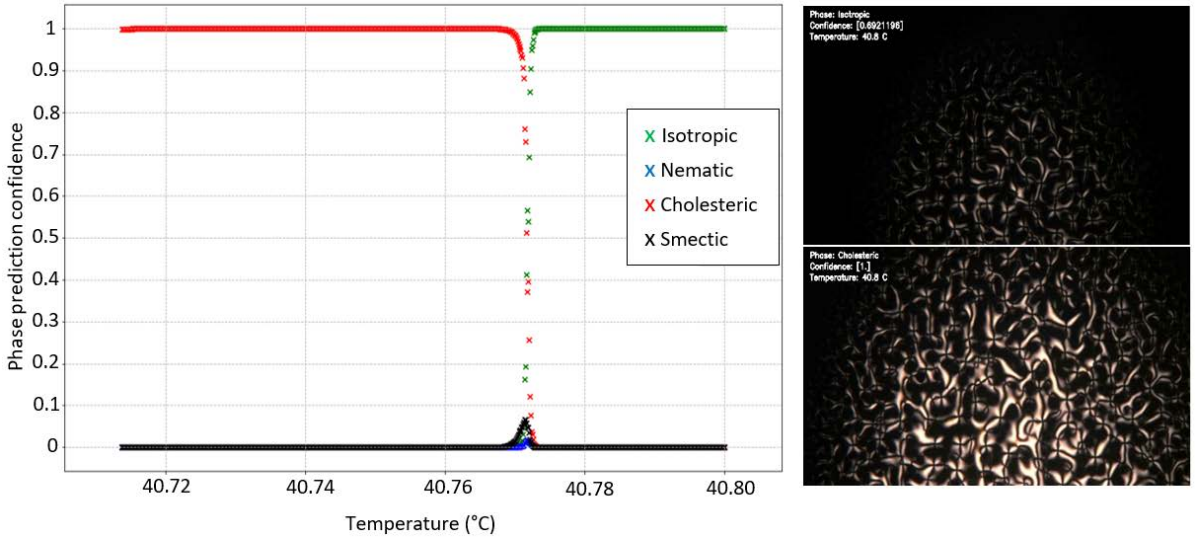


Figure 13: Figure showing the phase prediction confidence against temperature graph for an isotropic to nematic transition. Alongside this are screenshots from the labelled video, either side of the transition, to show the texture differences.

The results of applying the model to an isotropic to cholesteric transition are shown in Figure 14. The isotropic to cholesteric transition was correctly identified, at $(171 \pm 1)°$C. This uncertainty was chosen as the cross-over region in phase prediction confidence, shown closer in the bottom left graph of Figure 14, spreads approximately $1°$C either side of the cross-over between phase predictions. This transition temperature is not consistent with the literature value of approximately $190°$C [19]. However, there is a possible systematic uncertainty with this

predicted transition temperature. The video was labelled to be from $181°$C to $154°$C. However, the video starts of with a black, isotropic, texture, shown in Figure 14, even though $181°$C is below the M6 LC's I-N* transition temperature. Visually, however, the video was labelled accurately as the texture changed from isotropic to cholesteric. As seen from Figure 14, the model then incorrectly detects a transition to the smectic phase after a region which is relatively uncertain between cholesteric and smectic.

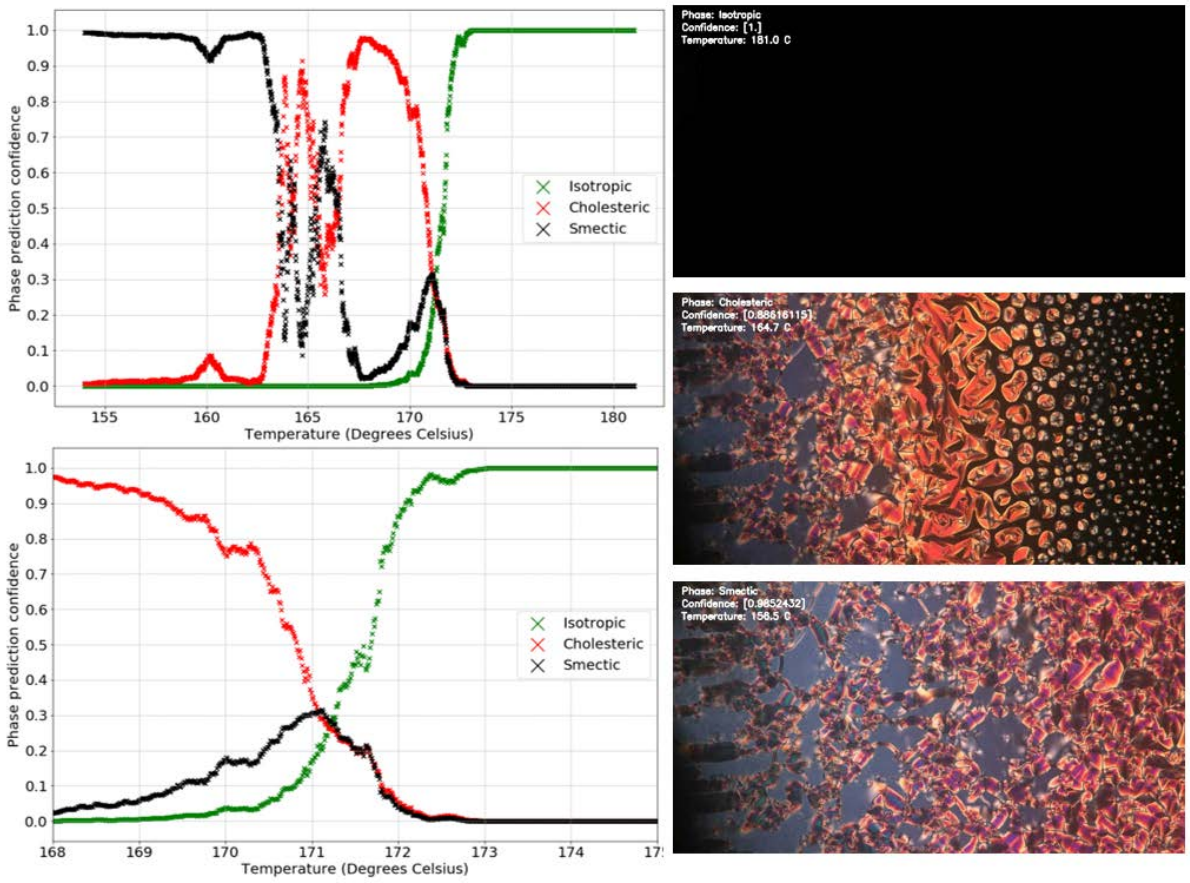


Figure 14: Figure showing the phase prediction confidence against temperature graph for a isotropic to cholesteric transition. Alongside this are screenshots from the labelled video showing how the cholesteric phase was misrecognised as smectic. A graph showing the identified I-N* transition region is shown in the bottom left.

Overall, this model performed well on the 8CB phases, especially involving N-Sm transitions. However, the model performed less well with the M6 videos, identifying N*-Sm transitions where there were none, or not identifying them when they were present. Another drawback of this model was the identification of a homeotropic nematic phase as isotropic. The homeotropic nematic phase, also called the psuedo-isotropic phase, is where the optic axis is along the line of sight of the image, so it appears black. The model likely did not notice that it was a nematic phase due to learning that a dark image corresponds to the isotropic phase. As no homeotropic images were included in the training process, this performance was expected. The graphs for all video results obtained using this model are shown in Appendix C.1.

Even though the InceptLC-V4 model achieved a higher validation accuracy, the overall

performance of this model was similar to that of the above model. Again, this is a sign that the validation set used was not representative of textures in the test set videos. However, three of the nematic-smectic transition videos were labelled less precisely. An example of this, for the same transition video shown in Figure 12, is shown in Figure 15. Here, cholesteric is predicted around the transition, before changing to the correct phase, less precisely finding the transition. Another difference between the performance of this model and the one above was that this model was able to recognise the homeotropic nematic phase video. This improvement was due to homeotropic images being included in the training and validation sets for this model. The extent to which it can be said that this model has learned to recognise the homeotropic nematic phase is limited due to only one video being tested. It is unclear at this stage whether the model just recognises low-brightness textures, which are not black isotropic textures, as nematic. The graphs for all video results obtained using this model are shown in Appendix C.2.

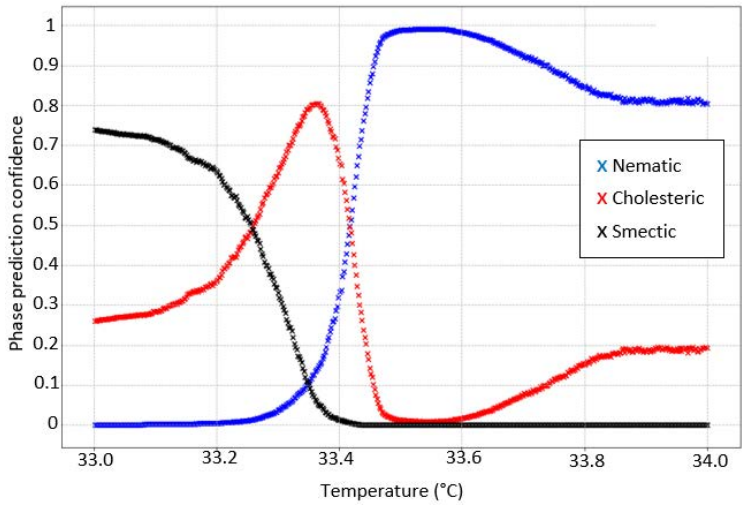


Figure 15: Figure showing the phase prediction confidence against temperature graph for a transition between the nematic and smectic phase, using the InceptLC-V4 model.

Despite both models having accurately labelled some transitions, rather than showing that these models themselves have learned what the transitions look like, this shows that CNNs have the potential to learn these transitions. This is because the test sets are too small for the models to be generalizable at this stage.

5.2.2 Fluid Smectic and Hexatic Phase Classification

Figure 16 shows the results of applying the model from Section 4.2 to one of the two test videos. It can be seen that no transition was detected. This was true for both videos. However, there is a change in prediction confidence. In Figure 16, there is a peak at $(95 \pm 1)°$C. The uncertainty is again from the absolute temperature uncertainty of the device used for cooling or heating the LCs. This is consistent with the known SmC-SmI transition temperature of $92.9°$C [18] within three standard deviations. Despite the model achieving a high validation accuracy, test videos were incorrectly labelled. A possible explanation for this is that the model was only trained on

a relatively small number of fluid smectic and hexatic images. Also, the validation set was small, meaning that, despite achieving a high validation accuracy, the small sample size is unlikely to represent the full variety of fluid smectic and hexatic textures that there are. However, some sign of a phase transition can be gained from the graphs produced, analogous to using DSC, with no information about which phases the transition was between. The results for both videos labelled by this model are shown in Appendix C.2.6.

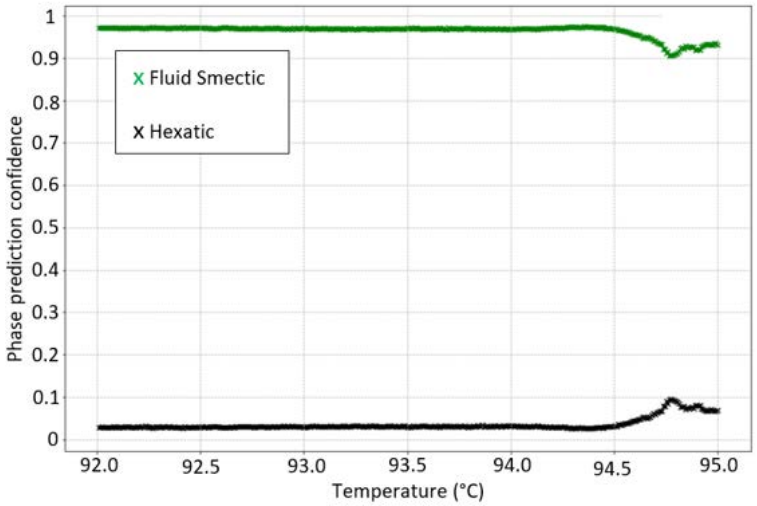


Figure 16: Figure showing the phase prediction confidence against temperature graph for a transition between the fluid smectic and hexatic phases.

6 Conclusion and Future Work

So far in the project, CNN and Inception models were trained to identify transitions involving: isotropic, nematic, cholesteric and smectic phases; and fluid smectic and hexatic phases. For these classification problems, trained Inception models achieved over 90% validation accuracy.

For identifying transitions involving isotropic, nematic, cholesteric and smectic phases, the performance of the Inception model and a CNN model, which achieved 69.8% validation accuracy, were compared. The two models accurately identified the N-Sm transitions of the 8CB LC, predicting phase transition temperatures consistent the known values. However, the Inception model was less precise for some of these transitions, incorrectly identifying a cholesteric phase around the transition, before resolving to the correct phase. Transitions of the M6 LC were less accurately identified, with both models either missing transitions between the cholesteric and smectic phases, or detecting transitions where there were none.

Despite the high validation accuracy of the model applied to identifying fluid smectic-hexatic transitions, neither of the two test videos were accurately identified. A limitation of this part of the project was the small number of images and videos available, meaning less images to train and validate on and less videos to test on to accurately determine how well the model

could perform for the task. However, the fluid smectic-hexatic model did show some signs of the transition in both test videos.

The main drawback of the results obtained so far is that, despite models achieving a high validation accuracy, they do not perform as well on the unseen videos. One improvement that could be made to the work done so far in the project would be to collect more images and videos. More images would enable a wider variety of characteristics of each phase texture to be learned by the model, helping it to generalize to unseen images and videos. More videos would also provide a more rigorous test of the models. So far, results do not conclusively show whether a model has learned the characteristics of a particular transition, or just for the few cases tested.

As obtaining more data is time consuming and not always possible, another improvement would be to attempt a different method of validating models during training. No improvement in labelling videos was seen between the higher and lower validation accuracy models. This shows that the validation images chosen did not represent the unseen video textures accurately. The validation method used so far was a single hold-out set. Another method is k-fold cross validation. In this, a model is trained several times, each time with different images being chosen to be in training and validation sets. This could improve results as a model can be validated on a wider range of textures. This could lead to a more generalizable model which could label unseen test videos more accurately. A final improvement would be to train all models for different choices of test videos. It cannot be determined from a small set of randomly chosen test videos whether the model is inaccurate, or whether the transitions are hard to identify and vice versa.

An extension to this project would be to look at different transition, for example, more difficult to distinguish phases, such as between SmA and SmC. Alternatively, a model which can distinguish among many phases, such as I, N, N*, SmA, SmC, SmE, SmF and SmI, could be attempted. More images would be needed for some of these phases, so that the number of examples is comparable other phases.

References

[1] I. Dierking, *Textures of Liquid Crystals*. Weinheim: Wiley-VCH, 2003.

[2] J.-G. An, S. Hina, Y. Yang, M. Xue, and Y. Liu, “Characterization of liquid crystals: A literature review,” *Reviews on Advanced Materials Science*, vol. 44, pp. 398–406, 2016.

[3] P. J. Collings, “Phase structures and transitions in thermotropic liquid crystals,” in *Handbook Of Liquid Crystal Research*, P. J. Collings and J. S. Patel, Eds. Oxford: Oxford University Press, 1997, ch. 4, pp. 99–124.

[4] R. H. Chen, *Liquid Crystal Displays: Fundamental Physics and Technology*, ser. Wiley-SID series in Display Technology. Hoboken, New Jersey: Wiley, 2011.

[5] S. J. Cowling, *Optical Microscopy Studies of Liquid Crystals*. American Cancer Society, 2014, ch. 9, pp. 1–38. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9783527671403.hlc012

[6] “Soft matter: Liquid crystals,” https://softmatter-dierking.myfreesites.net/group.

[7] K. Binnemans, “Chapter 254 - lanthanidomesogens,” in *Including Actinides*, ser. Handbook on the Physics and Chemistry of Rare Earths, J.-C. G. Bünzli and V. K. Pecharsky, Eds. Elsevier, 2013, vol. 43, pp. 1 – 158. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780444595362000015

[8] L. Kolek *et al.*, “Studies of phase diagram of a liquid crystal with 4-[2-(3-fluorophenyl)ethyl]biphenyl core of molecules,” *ACTA Physica polonica A*, vol. 122, no. 2, 2012.

[9] H. Y. D. Sigaki *et al.*, “Learning physical properties of liquid crystals with deep convolutional neural networks,” Apr 2020.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016.

[11] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA: The MIT Press, 2015.

[12] T. M. Mitchell, *Machine learning*. New York: WCB/McGraw-Hill, 1997.

[13] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2017.

[14] J. Lever, M. Krzywinski, and N. Altman, “Model selection and overfitting,” *Nature Methods*, vol. 13, no. 9, pp. 703–4, 2016.

[15] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[16] C. Szegedy *et al.*, “Going deeper with convolutions,” *CoRR*, vol. abs/1409.4842, 2014. [Online]. Available: http://arxiv.org/abs/1409.4842

[17] “VideoLAN VLC media player,” https://www.videolan.org/.

[18] I. Dierking, F. Giesselmann, J. Ksserow, and P. Zugenmaier, “Properties of higher-ordered ferroelectric liquid crystal phases of a homologous series,” *Liquid Crystals*, vol. 17, no. 2, pp. 243–261, August 1994.

[19] J. Schacht *et al.*, “Mesomorphic properties of a homologous series of chiral liquid crystals containing the alpha-chloroester group,” *Liquid Crystals*, vol. 19, no. 2, pp. 151–157, August 1995.

[20] “Instagram: vance.williams,” https://www.instagram.com/vance.williams/.

[21] “YouTube: Vance Williams,” https://www.youtube.com/c/VanceWilliamsSFU.

[22] “Keras API,” https://keras.io/.

[23] “Google Colaboratory,” https://colab.research.google.com/.

[24] S. H. Lee, H. Yong Song, K. Hyun, and J. Hyup Lee, “Nonlinearity from ft-rheology for liquid crystal 8cb under large amplitude oscillatory shear (laos) flow,” *Journal of Rheology*, vol. 59, no. 1, pp. 1–19, 2015. [Online]. Available: https://doi.org/10.1122/1.4901288

Appendices

A Image Collection and Preprocessing

A.1 Using VLC Media Player to Extract Video Frames

The following figures outline the steps necessary for using VLC media player to extract images from videos.

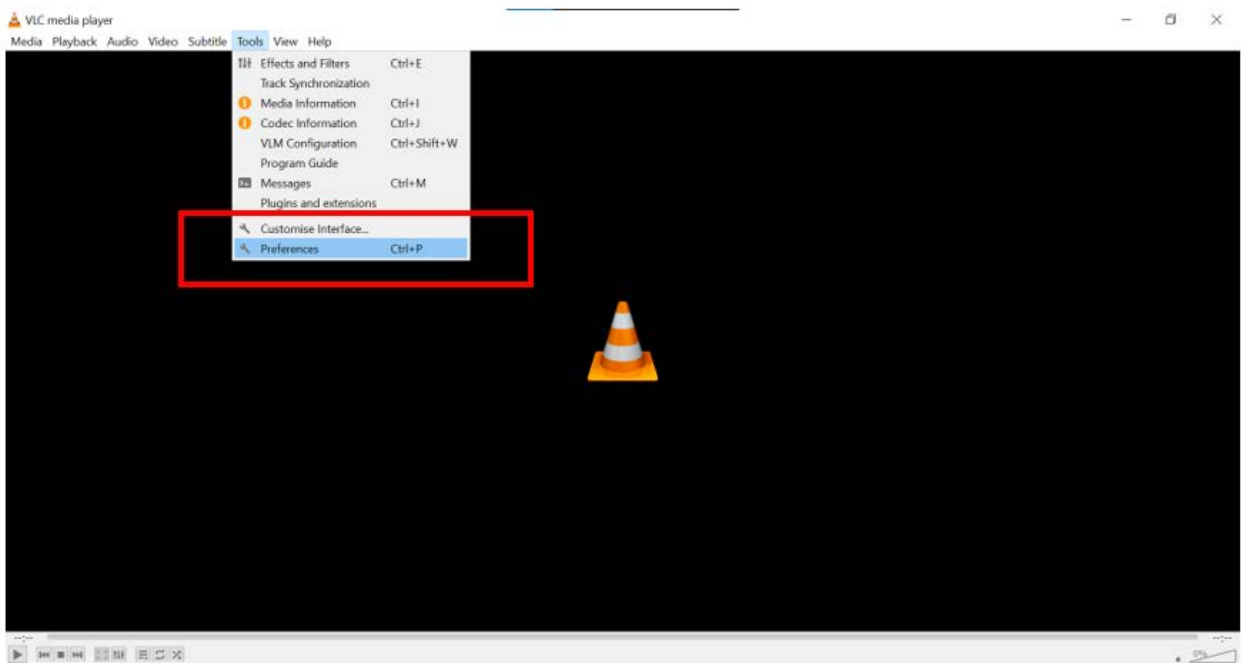


Figure 17: Caption

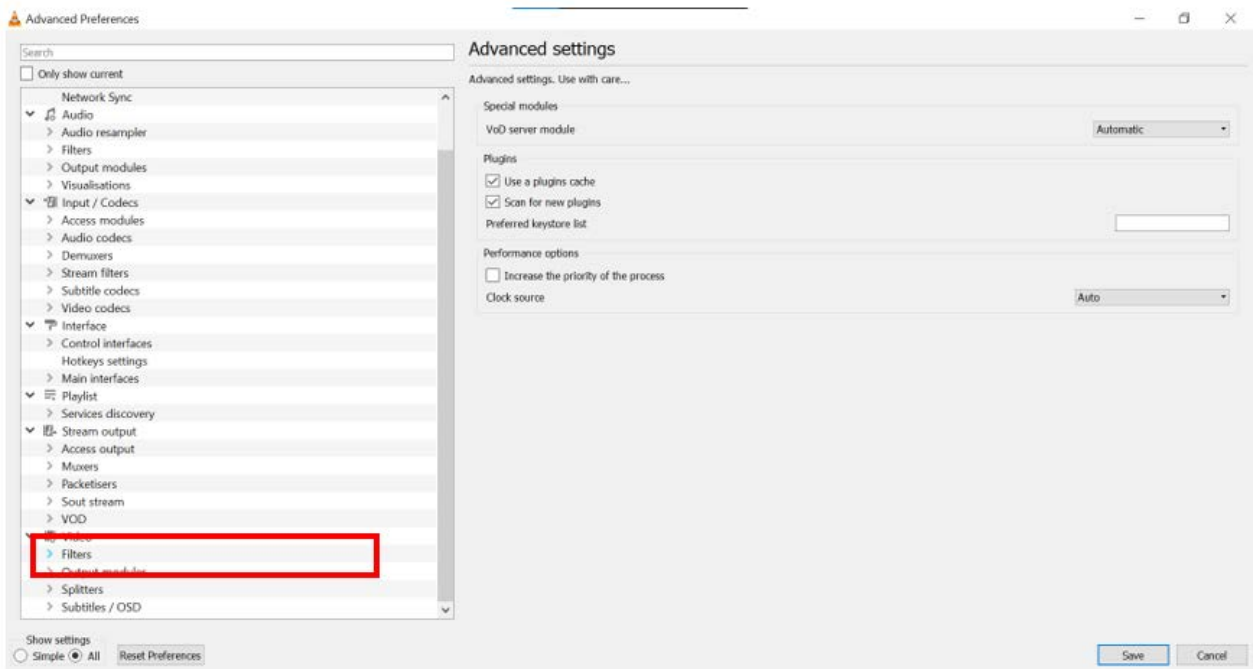


Figure 18: Caption

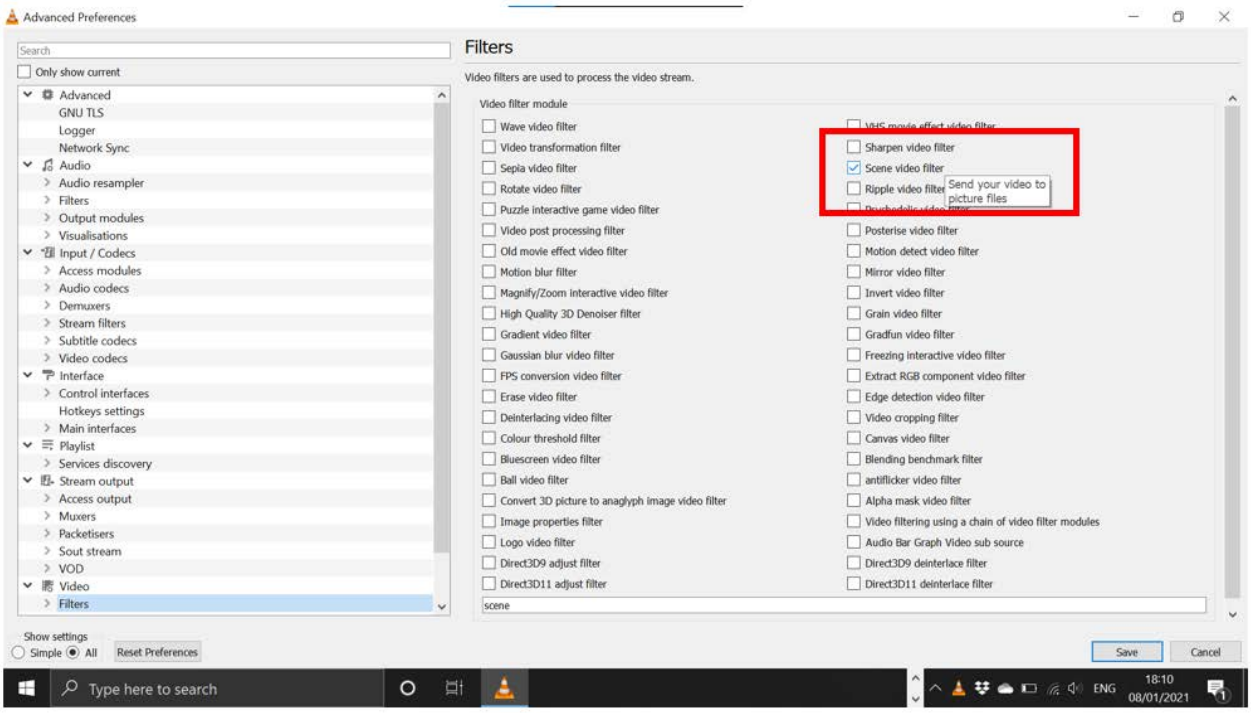


Figure 19: Caption

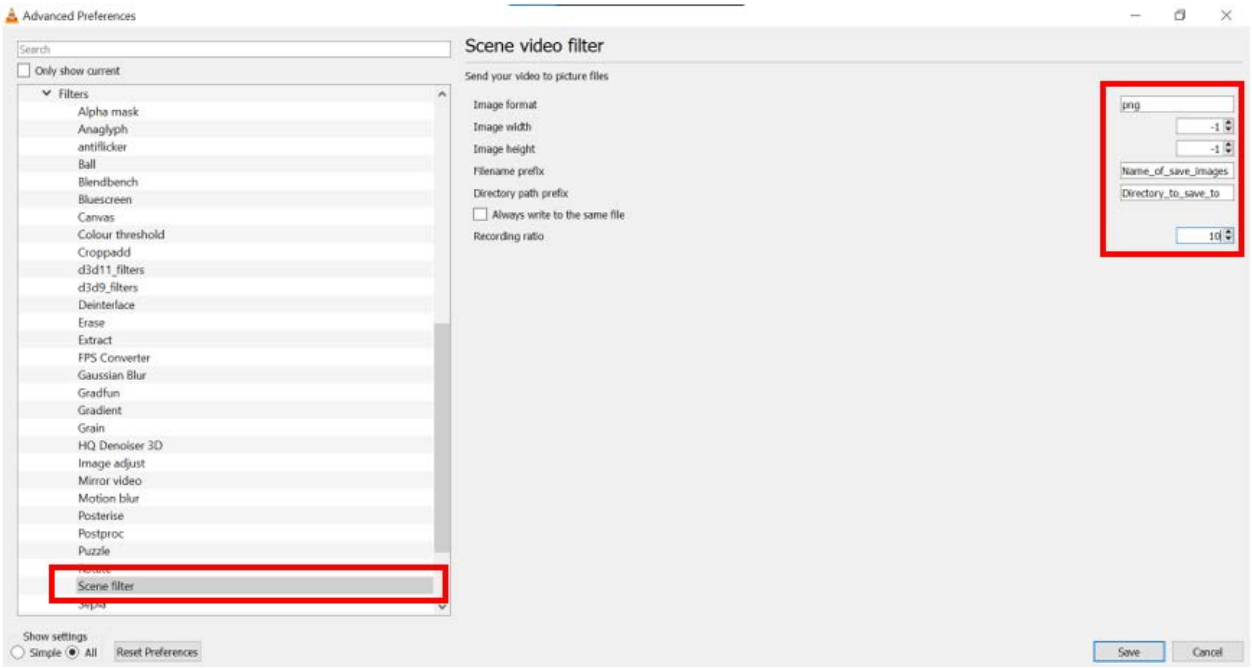


Figure 20: Caption

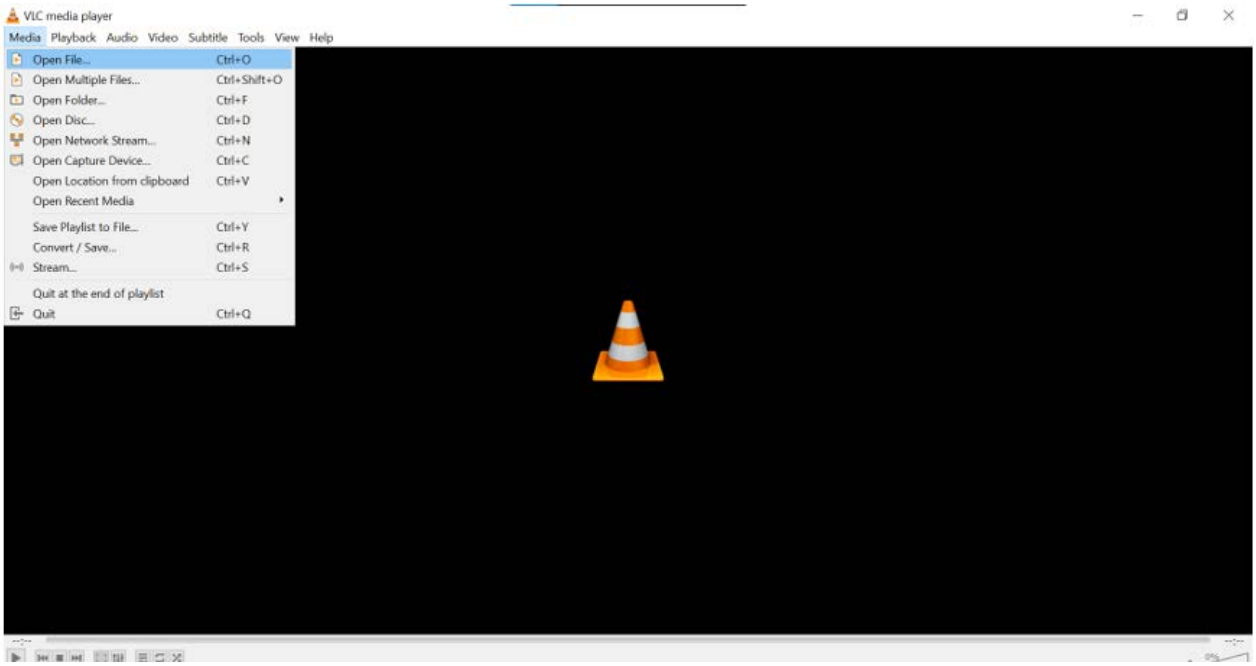


Figure 21: Caption

A.2 Script for Splitting Images

Below is the script written for splitting images as described in Section 3.

```
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 27 20:29:02 2020

@author: Jason
```

```
"""

import numpy as np
import os
from PIL import Image
from pathlib import Path

def image_split_and_save(image_directory, target_image_directory,
                         target_size=(544,682)):
    """
    Given a directory of images, this function takes each image and splits it
    into as many non-overlapping images of target_size shape as will fit. It
    then saves each new, split image into the target_image_directory.
    New saved image has a name of the format "name-00001" for the first
    image split from image called name

    Parameters
    ----------
    image_directory : string
        Directory destination containing all the images that are required to be
        split.
    target_image_directory : string
        Directory destination where the split images will be saved.
    target_size : tuple, optional
        The dimensions that you require the original image to be split into.
        The default is (200,200).

    Returns
    -------
    None.

    """
    directory = Path(image_directory)
    target_height, target_width = target_size

    n_image = 1
    for file in directory.iterdir():
        image = Image.open(file)
        image_array = np.asarray(image)
        fname, ext = os.path.splitext(file)
        fname = fname.replace(image_directory, "")

        height = image_array.shape[0]
        width = image_array.shape[1]
        n_h = height // target_height
        n_w = width // target_width

        split_image_number = 1
        for h in range(n_h):
            start_height = target_height * h
            for w in range(n_w):
                start_width = target_width * w
                split_image_array = image_array[
                    start_height:start_height + target_height,
                    start_width:start_width + target_width, :]
                split_image = Image.fromarray(split_image_array)
                save_path = (target_image_directory
                             + fname
                             + "-%05d"%(split_image_number)
```

```
                                + ext)
                split_image.save(save_path)
                split_image_number += 1
        print("Image %d has been split.\n" %(n_image))
        n_image += 1

    return None


if __name__ == "__main__":
    image_directory = "C:\\Users\\Jason\\Documents\\Unsplit_image_dir\\"
    target_image_directory = "C:\\Users\\Jason\\Split_image_dir\\"
    image_split_and_save(image_directory, target_image_directory)
```

B Code for Training Models and Labelling Transition Videos

The training of models using Keras was done using Google Colaboratory. Through this, models could be trained on 12 GB nVIDIA GPUs. Compared to standard CPUs, GPUs can reduce training time by a factor of ten. Alos, as the datasets in this project were less than the available RAM, images could be cached, further decreasing training time.

B.1 Video Labelling Class using OpenCV

Below is the script containing a Python class used for applying trained Keras models to transition videos, labelling them with predicted phase, estimated confidence and temperature. It was also used for storing results as a csv file, as well as plotting phase prediction confidence against temperature graphs.

```
# -*- coding: utf-8 -*-
"""
Created on Wed Dec 16 17:39:39 2020

@author: Jason
"""


# Liquid crystal video phase labeller

# Given a model trained for LC texture phase classification
# this script will label a given video with the determined
# phase at each frame and output a phase prediction
# confidence graph against temperature

import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import cv2
import pandas as pd


class PhaseLabeller:
    """
    Class for loading a trained model, using that model to get phase
    prediction and a label a video and save data as csv for plotting graphs.
```

```
    """

    def __init__(self):

        self.model = None
        # Initialize a dict which will be filled
        # with phase prediction confidences and temperatures
        self.confidence_dict = {"Temperature":[]}
        self.phase_list = []


    def get_model(self, model_load_dir, phase_list):
        """
        Loads a trained Keras model which will be used to label a video.

        Parameters
        ----------
        model_load_dir : String
            Folder where saved Keras model is stored.
        phase_list : List
            List of phase for which the model is trained to predict.

        Returns
        -------
        None.

        """
        # Load in the trained model
        self.model = tf.keras.models.load_model(model_load_dir)
        for layer in self.model.layers:
            layer.trainable = False
        self.model.summary()

        self.phase_list = phase_list
        # Create empty lists to store prediction accuracy for each phase
        for phase in phase_list:
            self.confidence_dict[phase] = []



    def label_video(self, vid_path, vid_save_path,
                    start_temp, end_temp=None,
                    temp_rate_per_sec=None):
        """
        Labels a video with the predicted phase, phase prediction confidence
        and temperature.

        Parameters
        ----------
        vid_path : String
            File path of video which is to be labelled.
        vid_save_path : String
            File path where labelled video will be saved.
        start_temp : Float
            Start temperature of video in degrees C.
        end_temp : Float, optional
            End temperature of video in degrees C. The default is None.
        temp_rate_per_sec : String, optional
            Rate of temperature change for the video (per second).
```

```
            The default is None.

        Returns
        -------
        None.

        """
        # Get the video to be labelled
        vid_stream = cv2.VideoCapture(vid_path)
        writer = None
        width = int(vid_stream.get(cv2.CAP_PROP_FRAME_WIDTH))
        height = int(vid_stream.get(cv2.CAP_PROP_FRAME_HEIGHT))
        fourcc = int(vid_stream.get(cv2.CAP_PROP_FOURCC))

        # Relate frames of videos to certain temperature
        fps = vid_stream.get(cv2.CAP_PROP_FPS)
        frame_count = int(vid_stream.get(cv2.CAP_PROP_FRAME_COUNT))
        #vid_duration = frame_count/fps
        if end_temp != None:
            temp_step_per_frame = (end_temp - start_temp)/frame_count
        else:
            temp_step_per_frame = temp_rate_per_sec/fps

        i = 0
        while True:
            (grabbed, frame) = vid_stream.read()
            if not grabbed:
                break

            output_vid = frame.copy()
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            #frame = cv2.resize(frame, (256, 256)).astype("float32") / 255
            frame = cv2.resize(frame, (256, 256)).astype("float32")

            # Get phase prediction for frame
            # Needs to be changed if using a binary classification model
            pred = self.model.predict(np.expand_dims(frame, axis=(0,3)))
            y_predict = np.argmax(pred, axis=1)
            confidence = pred[0][y_predict]
            for n, phase in enumerate(self.phase_list):
                # Store data for plotting
                self.confidence_dict[phase].append(pred[0][n])
            label = self.phase_list[int(y_predict)]

            # Get temperature of frame
            temp = start_temp + i*temp_step_per_frame
            self.confidence_dict["Temperature"].append(temp)
            i += 1

            # Label frame
            phase_text = "Phase: {}".format(label)
            phase_confidence_text = "Confidence: {}".format(confidence)
            temp_text = "Temperature: %.1f C"%(temp)
            cv2.putText(output_vid, phase_text, (35,50),
                        cv2.FONT_HERSHEY_SIMPLEX,
                        1.25, (255, 255, 255), 5)
            cv2.putText(output_vid, phase_confidence_text, (35,100),
                        cv2.FONT_HERSHEY_SIMPLEX,
                        1.25, (255, 255, 255), 5)
```

```
            cv2.putText(output_vid, temp_text, (35,150),
                        cv2.FONT_HERSHEY_SIMPLEX,
                        1.25, (255, 255, 255), 5)

            if writer is None:
                # Must have FFMPEG install for this to work
                writer = cv2.VideoWriter(vid_save_path, fourcc, fps,
                                         (width, height), True)
            writer.write(output_vid)
            cv2.imshow("Output_vid", output_vid)

            key = cv2.waitKey(1) & 0xFF
            if key == ord("q"):
                break
        writer.release()
        vid_stream.release()
        cv2.destroyAllWindows()



    def to_csv(self, save_dir):
        """
        Stores phase prediction and temperature data for labelled video in a
        csv file for later re-plotting

        Parameters
        ----------
        save_dir : String
            File directotory where csv will be saved.

        Returns
        -------
        None.

        """
        columns = ["Temperature"] + self.phase_list
        temp_confidence_data = pd.DataFrame(data=self.confidence_dict,
                                            columns=columns)
        temp_confidence_data.head()
        temp_confidence_data.to_csv(save_dir)


    def conf_temp_plot(self, phase_plot_list, color_list, save_path,
                       x_tick_range=None, start_temp=None, end_temp=None):
        """
        Plots phase prediction confidence graph against temperature.

        Parameters
        ----------
        phase_plot_list : List
            List of string phase names to plot.
        color_list : List
            List of colors to for plotting the associated phase in
            phase_plot_list
        save_path : String
            File path where graph will be saved.
        x_tick_range : np.array, optional
            Specific temperature range to plot between. The default is None.
        start_temp : Float, optional
```

```
            Value of temperature at which to start on x axis.
            The default is None.
        end_temp : TYPE, optional
            Value of temperature at which to end on x axis.
            The default is None.

        Returns
        -------
        None.

        """
        plt.figure(figsize=(15,10))
        temp = self.confidence_dict["Temperature"]
        for n, phase in enumerate(phase_plot_list):
            phase_conf = self.confidence_dict[phase]
            color = color_list[n]
            plt.scatter(temp, phase_conf, c=color, marker="x", label=phase)

        plt.xlabel("Temperature (Degrees Celsius)")
        if x_tick_range != None:
            plt.xticks(x_tick_range)
        if start_temp != None and end_temp != None:
            plt.xlim(start_temp, end_temp)
        plt.ylabel("Phase prediction confidence")
        plt.yticks(np.arange(0, 1.1, 0.1))

        plt.legend(loc="best")
        plt.grid(linestyle="--")

        plt.savefig(save_path)
```

B.2 The First Model

B.2.1 Model Diagram

The diagram below is the model architecture for the first model trained in this project.

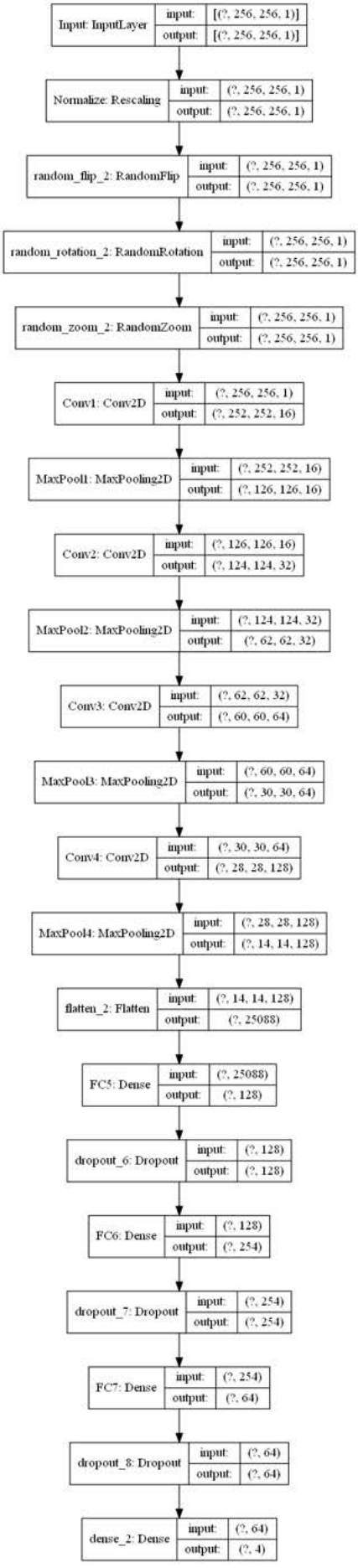


Figure 22: The first CNN model architecture. ? refers to the number of images, which can be arbitrary.

B.2.2 Script for Training

Below is the script used to implement and train the first CNN model in Keras. This was run using Google Colaboratory, with liquid crystal images stored on Google Drive.

```
#!/usr/bin/env python
# coding: utf-8

# # **CNN LC phase classifier**


from google.colab import drive
drive.mount('/content/drive')

file_title = "First CNN"

import numpy as np
import tensorflow as tf
from keras import Model
from keras import layers
from keras.preprocessing import image_dataset_from_directory
from keras.callbacks import ModelCheckpoint
from keras.utils.vis_utils import plot_model

# Get necessary scripts from google drive
import sys
sys.path.insert(0,"/content/drive/MyDrive/")
from Model_metric_plotter_saver import (plot_metrics,
                                        save_history_to_csv,
                                        confusion_matrix_plot)

# Define where images are stored
img_directory = "/content/drive/My Drive/I-N-Chol-Sm_dataset-balanced"

train_dir = img_directory + "/Train"
val_dir = img_directory + "/Val"

# These lines of code allow for the datasets to be loaded in batch_size at
# a time from files
image_size=(256,256)
train_dataset = image_dataset_from_directory(train_dir,
                             labels="inferred",
                             label_mode="categorical",
                             color_mode="grayscale",
                             batch_size=64,
                             image_size=image_size,
                             shuffle=True
                         )
val_dataset = image_dataset_from_directory(val_dir,
                             labels="inferred",
                             label_mode="categorical",
                             color_mode="grayscale",
                             batch_size=64,
                             image_size=image_size,
                             shuffle=False # Set to false so confusion matrix
                                           # can be plotted
                         )

# These next steps enable the dataset to be stored in cache (if it can fit)
# meaning that after data is loaded in on the first epoch, it will
```

```
# be loaded in much faster
AUTOTUNE = tf.data.experimental.AUTOTUNE

train_dataset = train_dataset.cache().prefetch(buffer_size=AUTOTUNE)
val_dataset = val_dataset.cache().prefetch(buffer_size=AUTOTUNE)


# Define model layers
# Input/preprocessing layers
input_layer = layers.Input(shape = (256,256,1), name="Input")
X = layers.experimental.preprocessing.Rescaling(1./255,
                                                name="Normalize")(input_layer)
# Augmentation layers
X = layers.experimental.preprocessing.RandomFlip()(X)
X = layers.experimental.preprocessing.RandomRotation(0.2)(X) # rotate by 0.2*2pi
X = layers.experimental.preprocessing.RandomZoom(0.2)(X)

# Conv layers
X = layers.Conv2D(16, kernel_size=(5,5), activation="relu", name="Conv1")(X)
X = layers.MaxPooling2D(2,2, name="MaxPool1")(X)

X = layers.Conv2D(32, kernel_size=(3,3), activation="relu", name="Conv2")(X)
X = layers.MaxPooling2D(2,2, name="MaxPool2")(X)

X = layers.Conv2D(64, kernel_size=(3,3), activation="relu", name="Conv3")(X)
X = layers.MaxPooling2D(2,2, name="MaxPool3")(X)

X = layers.Conv2D(128, kernel_size=(3,3), activation="relu", name="Conv4")(X)
X = layers.MaxPooling2D(2,2, name="MaxPool4")(X)

# Fully connected layers
X = layers.Flatten()(X)

X = layers.Dense(units=128, activation="relu", name="FC5")(X)
X = layers.Dropout(0.5)(X)

X = layers.Dense(units=254, activation="relu", name="FC6")(X)
X = layers.Dropout(0.5)(X)

X = layers.Dense(units=64, activation="relu", name="FC7")(X)
X = layers.Dropout(0.5)(X)

# Output layer
X = layers.Dense(units=4, activation="softmax")(X)

# Save a diagram of the model
model = Model(input_layer, X, name=file_title)
model.summary()
model_save_path = "/content/drive/My Drive/" + file_title + ".png"
plot_model(model, to_file=model_save_path,
           show_shapes=True, show_layer_names=True)

metrics = ["accuracy"] # Accuracy is the metric of interest while trainin the
                       # model
# This is where to save a trained model
model_save_dir = "/content/drive/My Drive/" + file_title + "_saved_model"

model.compile(optimizer="adam",
              loss="categorical_crossentropy",
```

```
              metrics=["accuracy"])
# Use a checkpoint to save a most at the epoch of best validation accuracy
checkpoint = ModelCheckpoint(model_save_dir,
                             monitor = "val_accuracy",
                             save_best_only = True, mode="max")
# Train the model for 60 epochs and return the data each epoch in history
# for plotting later
history = model.fit(train_dataset,
                    validation_data=val_dataset,
                    epochs=60,
                    callbacks=[checkpoint],
                    verbose=1
                   )

save_dir = "/content/drive/My Drive/" + file_title

metrics = ["loss", "accuracy"]
# Plot the training history of the model and save
training_plot_save_path = save_dir + "_training_history"
plot_metrics(history, training_plot_save_path, metrics)

# Save training history to csv
csv_data_save_path = save_dir + "_training_history.csv"
save_history_to_csv(history, csv_data_save_path, metrics)

# Evaluate model on validation data and save the confusion matrix
# Get best saved model
trained_model = tf.keras.models.load_model(model_save_dir)
# Get predictions and true labels
predictions = trained_model.predict(val_dataset, verbose=1)
y_pred = np.argmax(predictions, axis=1)
y_true = np.argmax(np.concatenate([label for image, label in val_dataset],
                                  axis=0), axis=1)
# Plot and save the confusion matrix
target_names = ["Isotropic", "Nematic", "Cholesteric", "Smectic"]
confusion_mat_save_dir = save_dir + "_confusion_matrix"
confusion_matrix_plot(y_true, y_pred, target_names,
                      save=True, save_path=confusion_mat_save_dir)

# Applying the trained model to test video
from lc_video_phase_labeller import PhaseLabeller

model_load_dir = "C:/Users/Jason/Documents/University/Year_4/Saved_model/"
phase_list = ["Isotropic", "Nematic", "Cholesteric", "Smectic"]
vid_file = "N-I/"
vid_name = "nematic 5cb on glycerol-2-Yellow_35.6C_cooling"
vid_path = "C:/Users/Jason/Documents/" + vid_file + vid_name + ".avi"
save_dir = "C:/Users/Jason/Documents/University/"
vid_save_path = save_dir + vid_name + "_labelled.avi"
start_temp = 35.6
end_temp = None
temp_rate = 0.1 # rate of temperature decrease in seconds

phase_vid_labeller = PhaseLabeller()
phase_vid_labeller.get_model(model_load_dir, phase_list)

phase_vid_labeller.label_video(vid_path,
                               vid_save_path,
                               start_temp,
```

```
                                end_temp=end_temp,
                                temp_rate_per_sec=temp_rate)

# Save labelling prediction and temperature data in csv
save_path = save_dir + vid_name + ".csv"
phase_vid_labeller.to_csv(save_path)

# Plot and save the prediction confidence against temperature graph
save_path = save_dir + vid_name + ".png"
#phase_plot_list = ["Isotropic", "Nematic", "Cholesteric", "Smectic"]
phase_plot_list = ["Isotropic", "Nematic"]
#color_list = ["g", "b", "r", "k"]
color_list = ["g", "b"]
phase_vid_labeller.conf_temp_plot(phase_plot_list,
                                  color_list,
                                  save_path)


save_path = save_dir + vid_name + "-zoomed_in.png"
phase_vid_labeller.conf_temp_plot(phase_plot_list,
                                  color_list,
                                  save_path,
                                  start_temp=40.5,
                                  end_temp=41.5)
```

B.3 The InceptLC-V4 Model

B.3.1 Model Diagram

Figure 23: InceptLC-V4 model architecture. ? refers to the number of input images, which can be arbitrary.

B.3.2 Script for Training

Below is the script used to implement and train the InceptLC-V4 model in Keras. This was run using Google Colaboratory, with liquid crystal images stored on Google drive.

```
#!/usr/bin/env python
# coding: utf-8

# # **InceptionNet LC phase classifier**


from google.colab import drive
drive.mount('/content/drive')

file_title = "InceptLC-V4"

import numpy as np
import tensorflow as tf
from keras import Model
```

```
from keras import layers
from keras.preprocessing import image_dataset_from_directory
from keras.callbacks import ModelCheckpoint
from keras.utils.vis_utils import plot_model

# Get necessary scripts from google drive
import sys
sys.path.insert(0,"/content/drive/MyDrive/")
from Model_metric_plotter_saver import (plot_metrics,
                                        save_history_to_csv,
                                        confusion_matrix_plot)

# Define where images are stored
img_directory = "/content/drive/My Drive/I-N-Chol-Sm_dataset-balanced"

train_dir = img_directory + "/Train"
val_dir = img_directory + "/Val"

# These lines of code allow for the datasets to be loaded in batch_size at
# a time from files
image_size=(256,256)
train_dataset = image_dataset_from_directory(train_dir,
                            labels="inferred",
                            label_mode="categorical",
                            color_mode="grayscale",
                            batch_size=64,
                            image_size=image_size,
                            shuffle=True
                        )
val_dataset = image_dataset_from_directory(val_dir,
                            labels="inferred",
                            label_mode="categorical",
                            color_mode="grayscale",
                            batch_size=64,
                            image_size=image_size,
                            shuffle=False # Set to false so confusion matrix
                                          # can be plotted
                        )

# These next steps enable the dataset to be stored in cache (if it can fit)
# meaning that after data is loaded in on the first epoch, it will
# be loaded in much faster
AUTOTUNE = tf.data.experimental.AUTOTUNE

train_dataset = train_dataset.cache().prefetch(buffer_size=AUTOTUNE)
val_dataset = val_dataset.cache().prefetch(buffer_size=AUTOTUNE)


def inception_module(x,
                     filters_1x1,
                     filters_3x3_reduce,
                     filters_3x3,
                     filters_5x5_reduce,
                     filters_5x5,
                     filters_pool_proj,
                     name=None):
    # Defines the inception module
    conv_1x1 = layers.Conv2D(filters_1x1, (1, 1),
                             padding='same',
```

```
                              activation='relu')(x)
    # 1x1 dimension reduction before 5x5 convolution
    conv_3x3 = layers.Conv2D(filters_3x3_reduce, (1, 1),
                             padding='same',
                             activation='relu')(x)
    conv_3x3 = layers.Conv2D(filters_3x3, (3, 3),
                             padding='same',
                             activation='relu')(conv_3x3)
    # 1x1 dimension reduction before 5x5 convolution
    conv_5x5 = layers.Conv2D(filters_5x5_reduce, (1, 1),
                             padding='same',
                             activation='relu')(x)
    conv_5x5 = layers.Conv2D(filters_5x5, (5, 5),
                             padding='same',
                             activation='relu')(conv_5x5)

    pool_proj = layers.MaxPooling2D((3, 3), strides=(1, 1),
                                    padding='same')(x)
    pool_proj = layers.Conv2D(filters_pool_proj, (1, 1),
                              padding='same',
                              activation='relu')(pool_proj)

    output = layers.concatenate([conv_1x1,
                                 conv_3x3,
                                 conv_5x5,
                                 pool_proj], axis=3, name=name)

    return output


# Define model layers
# Input/preprocessing layers
input_layer = layers.Input(shape = (256,256,1), name="Input")
X = layers.experimental.preprocessing.Rescaling(1./255,
                                                name="Normalize")(input_layer)
X = layers.experimental.preprocessing.RandomFlip()(X)

# Conv layers
X = layers.Conv2D(16, kernel_size=(5,5), activation="relu", name="Conv1")(X)
X = layers.MaxPooling2D(2,2, name="MaxPool1")(X)

# Inception layers
X = inception_module(X,
                     filters_1x1=64,
                     filters_3x3_reduce=96,
                     filters_3x3=128,
                     filters_5x5_reduce=32,
                     filters_5x5=64,
                     filters_pool_proj=64,
                     name='Inception2')

X = inception_module(X,
                     filters_1x1=64,
                     filters_3x3_reduce=96,
                     filters_3x3=128,
                     filters_5x5_reduce=32,
                     filters_5x5=64,
                     filters_pool_proj=64,
                     name='Inception3')
```

```
133
134 # Fully connected layers
135 X = layers.GlobalAveragePooling2D()(X)
136 X = layers.Dense(units=128, activation="relu", name="FC4")(X)
137 X = layers.Dropout(0.5)(X)
138
139 # Output layer
140 # (change to units=1 and activation="sigmoid" for binary classifier)
141 X = layers.Dense(units=4, activation="softmax")(X)
142
143 # Save a diagram of the model
144 model = Model(input_layer, X, name=file_title)
145 model.summary()
146 model_save_path = "/content/drive/My Drive/" + file_title + ".png"
147 plot_model(model, to_file=model_save_path,
148            show_shapes=True, show_layer_names=True)
149
150 metrics = ["accuracy"] # Accuracy is the metric of interest while trainin the
151                        # model
152 # This is where to save a trained model
153 model_save_dir = "/content/drive/My Drive/" + file_title + "_saved_model"
154
155 model.compile(optimizer="adam",
156               loss="categorical_crossentropy",
157               metrics=["accuracy"])
158 # Use a checkpoint to save a most at the epoch of best validation accuracy
159 checkpoint = ModelCheckpoint(model_save_dir,
160                              monitor = "val_accuracy",
161                              save_best_only = True, mode="max")
162 # Train the model for 60 epochs and return the data each epoch in history
163 # for plotting later
164 history = model.fit(train_dataset,
165                     validation_data=val_dataset,
166                     epochs=60,
167                     callbacks=[checkpoint],
168                     verbose=1
169                    )
170
171 save_dir = "/content/drive/My Drive/" + file_title
172
173 metrics = ["loss", "accuracy"]
174 # Plot the training history of the model and save
175 training_plot_save_path = save_dir + "_training_history"
176 plot_metrics(history, training_plot_save_path, metrics)
177
178 # Save training history to csv
179 csv_data_save_path = save_dir + "_training_history.csv"
180 save_history_to_csv(history, csv_data_save_path, metrics)
181
182 # Evaluate model on validation data and save the confusion matrix
183 # Get best saved model
184 trained_model = tf.keras.models.load_model(model_save_dir)
185 # Get predictions and true labels
186 predictions = trained_model.predict(val_dataset, verbose=1)
187 y_pred = np.argmax(predictions, axis=1)
188 y_true = np.argmax(np.concatenate([label for image, label in val_dataset],
189                                   axis=0), axis=1)
190 # Plot and save the confusion matrix
191 target_names = ["Isotropic", "Nematic", "Cholesteric", "Smectic"]
```

```
confusion_mat_save_dir = save_dir + "_confusion_matrix"
confusion_matrix_plot(y_true, y_pred, target_names,
                      save=True, save_path=confusion_mat_save_dir)

# Applying the trained model to test video
from lc_video_phase_labeller import PhaseLabeller

model_load_dir = "C:/Users/Jason/Documents/University/Year_4/Saved_model/"
phase_list = ["Isotropic", "Nematic", "Cholesteric", "Smectic"]
vid_file = "N-I/"
vid_name = "nematic 5cb on glycerol-2-Yellow_35.6C_cooling"
vid_path = "C:/Users/Jason/Documents/" + vid_file + vid_name + ".avi"
save_dir = "C:/Users/Jason/Documents/University/"
vid_save_path = save_dir + vid_name + "_labelled.avi"
start_temp = 35.6
end_temp = None
temp_rate = 0.1 # rate of temperature decrease in seconds

phase_vid_labeller = PhaseLabeller()
phase_vid_labeller.get_model(model_load_dir, phase_list)

phase_vid_labeller.label_video(vid_path,
                               vid_save_path,
                               start_temp,
                               end_temp=end_temp,
                               temp_rate_per_sec=temp_rate)

# Save labelling prediction and temperature data in csv
save_path = save_dir + vid_name + ".csv"
phase_vid_labeller.to_csv(save_path)

# Plot and save the prediction confidence against temperature graph
save_path = save_dir + vid_name + ".png"
#phase_plot_list = ["Isotropic", "Nematic", "Cholesteric", "Smectic"]
phase_plot_list = ["Isotropic", "Nematic"]
#color_list = ["g", "b", "r", "k"]
color_list = ["g", "b"]
phase_vid_labeller.conf_temp_plot(phase_plot_list,
                                  color_list,
                                  save_path)


save_path = save_dir + vid_name + "-zoomed_in.png"
phase_vid_labeller.conf_temp_plot(phase_plot_list,
                                  color_list,
                                  save_path,
                                  start_temp=40.5,
                                  end_temp=41.5)
```

C Video Transition Labelling Results

This section shows all the phase prediction confidence against temperature graphs obtained so far in the project. They are grouped in sections by the model used for labelling the videos and the phase transitions present in the videos. The videos are all available on YouTube, with links for each video in the caption of the corresponding graph.

C.1 The First Model

C.1.1 Isotropic and Nematic Transitions

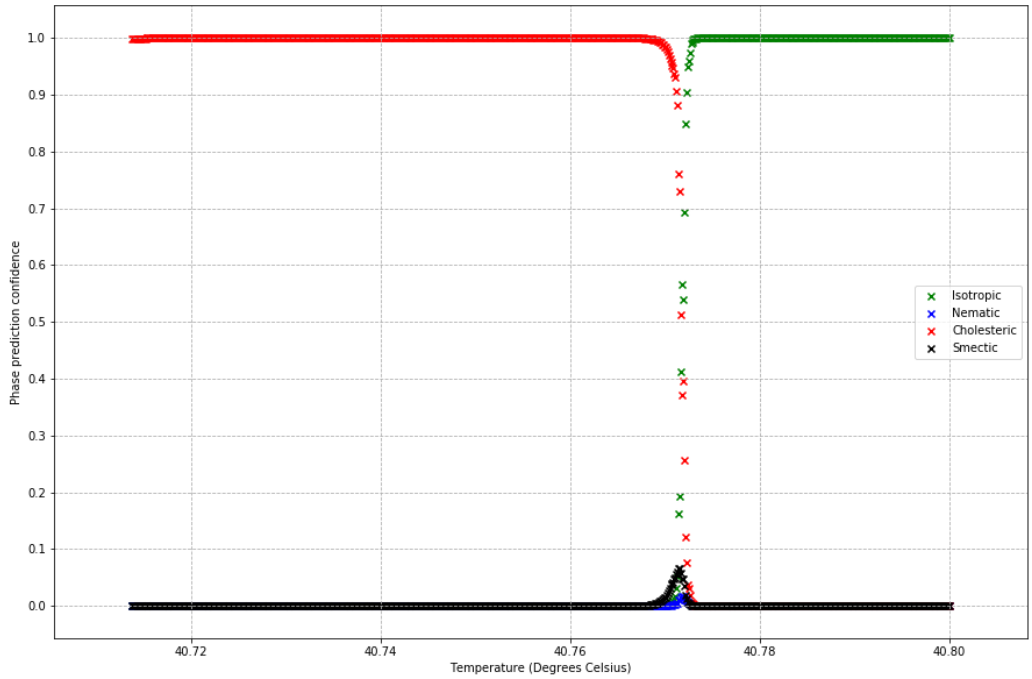


Figure 24: Phase prediction confidence against temperature graph for a video of an isotropic to nematic transition, from 40.8C cooling at 0.1 C/min, for an 8CB liquid crystal texture. This transition was labelled by the first CNN model. It was incorrectly labeled as an isotropic to cholesteric transition. Labelled video available at: https://youtu.be/58zLkkoQgwI.

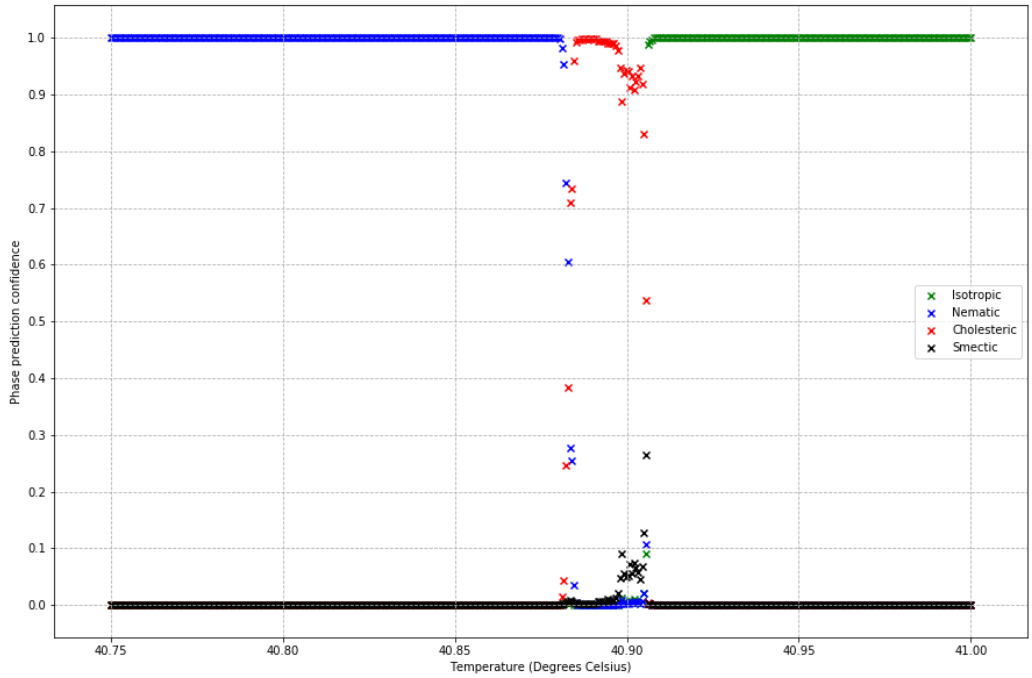


Figure 25: Phase prediction confidence against temperature graph for a video of an isotropic to nematic transition, from 41 C cooling at 0.5 C/min, for an 8CB liquid crystal texture. This transition was labelled by the first CNN. It was initially labeled as an isotropic to cholesteric transition, but then it was corrected to be nematic. Labelled video available at: https://youtu.be/xKO6FHayn6A.

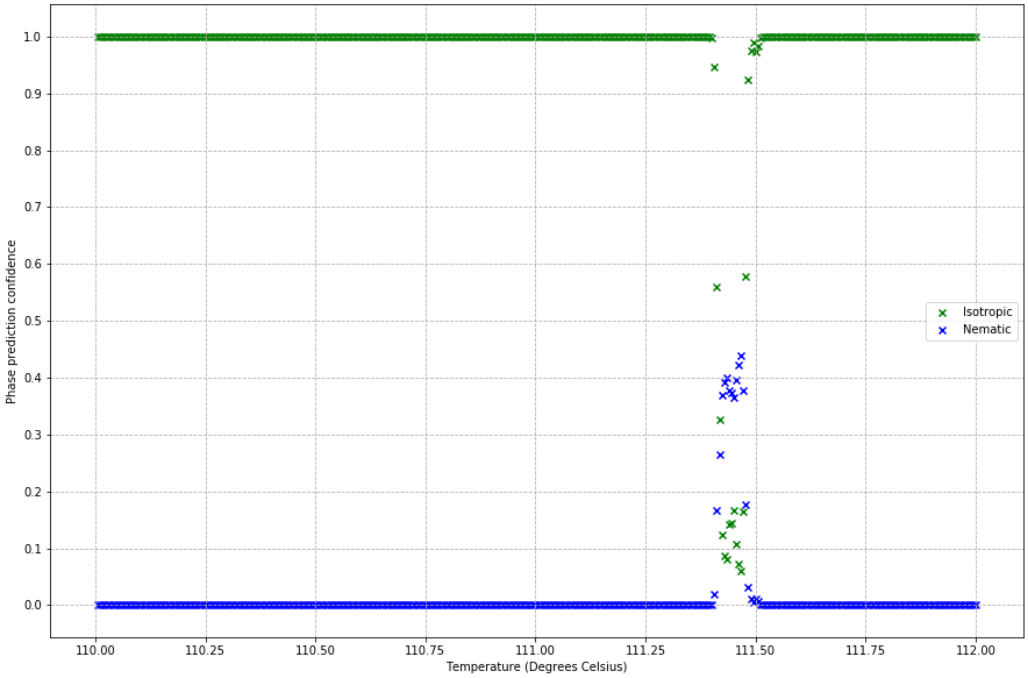


Figure 26: Caption

C.1.2 Isotropic and Cholesteric Transitions

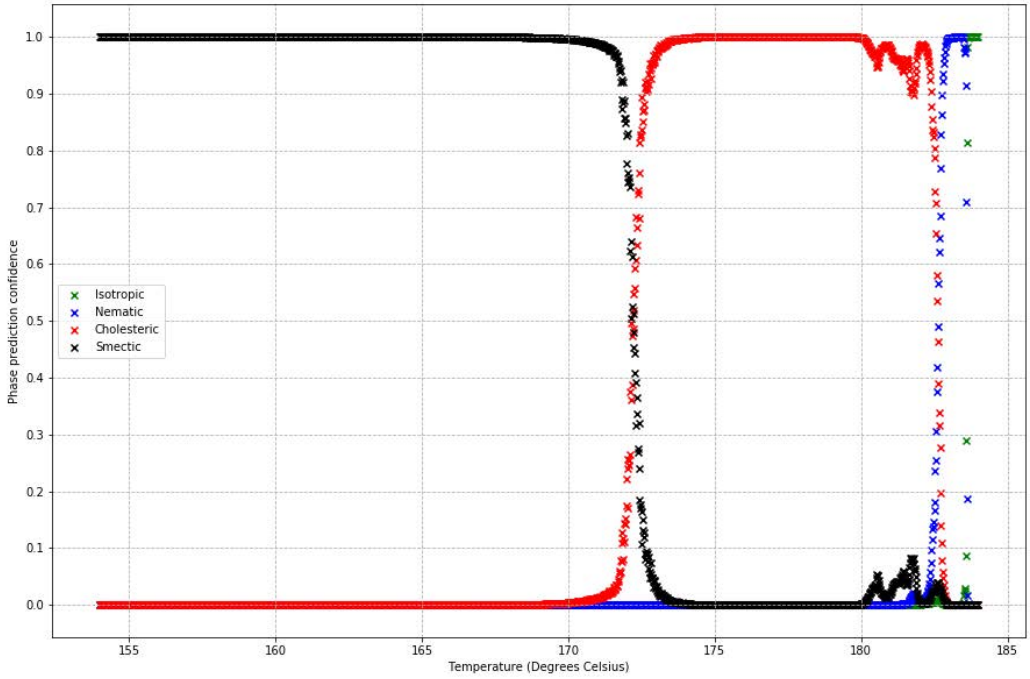


Figure 27: Phase prediction confidence against temperature graph for a video of a cholesteric to isotropic transition, from 154 C to 184 C, for an M6 liquid crystal texture. This transition was labelled by the first CNN. In this video a smectic phase was incorrectly identified. Labelled video available at: https://youtu.be/mqdwaWz0XVg.

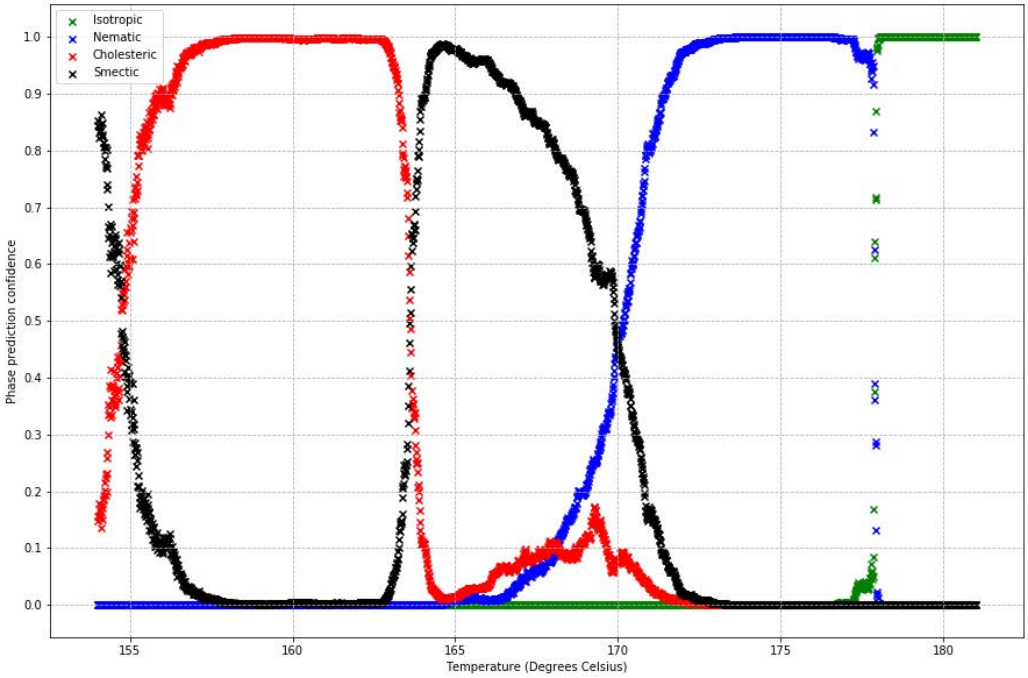


Figure 28: Phase prediction confidence against temperature graph for a video of an isotropic to cholesteric transition, from 181 C to 154 C, for an M6 liquid crystal texture. This transition was labelled by the first CNN. The isotropic to cholesteric was correctly identified, but cholesteric to smectic transitions were detected when they were none. Labelled video available at: https://youtu.be/R6hED0ZdEBU.

C.1.3 Nematic and Smectic Transitions

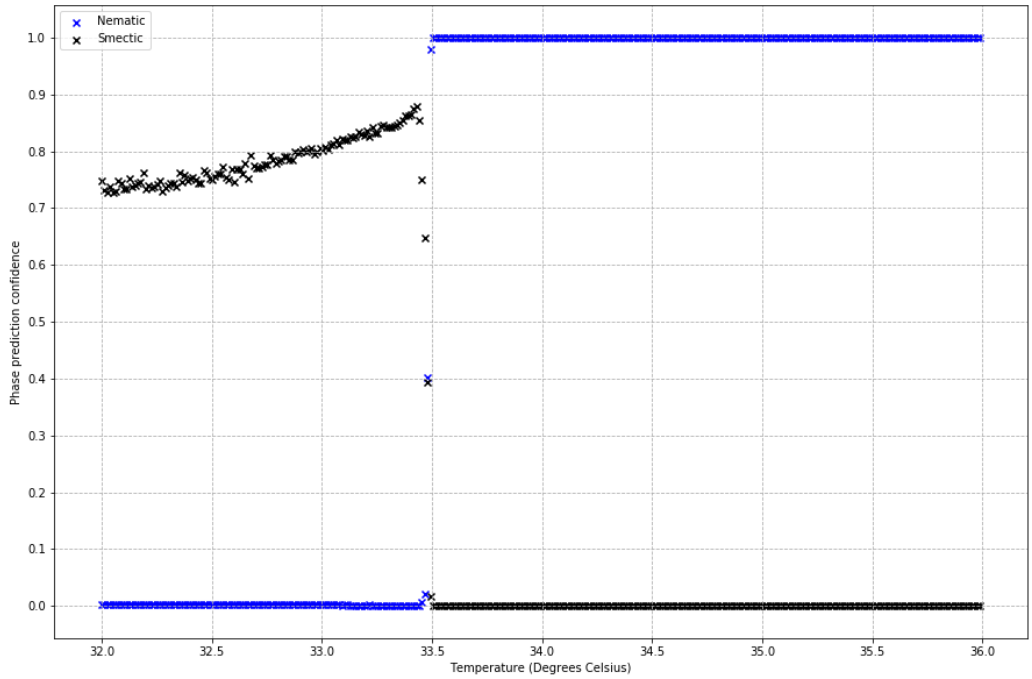


Figure 29: Phase prediction confidence against temperature graph for a video of a smectic to nematic transition, from 32-36C, for an 8CB liquid crystal texture. This transition was correctly labelled by the first CNN. Labelled video available at: https://youtu.be/ZJqcnqZ0jDI.

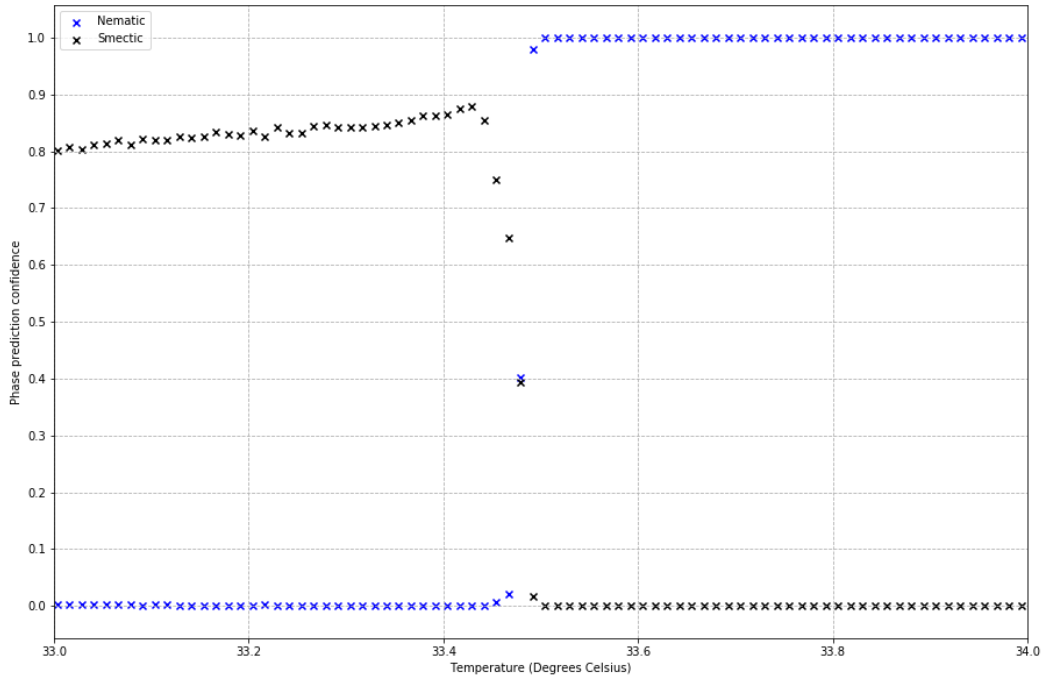


Figure 30: Phase prediction confidence against temperature graph for the same video of a smectic to nematic transition, from 32-36C, for an 8CB liquid crystal texture, as above. This graph shows a close-up of the transition.

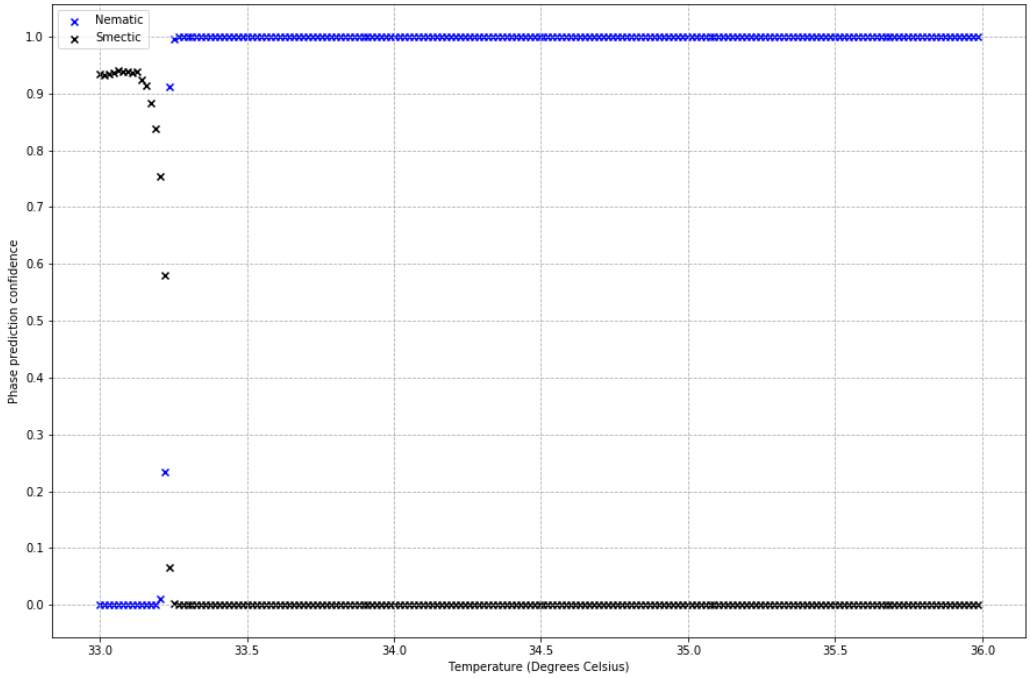


Figure 31: Phase prediction confidence against temperature graph for a video of a smectic to nematic transition, from 33-36C, for an 8CB liquid crystal texture. This transition was correctly labelled by the first CNN. Labelled video available at: https://youtu.be/fe1zxjhD3z0.

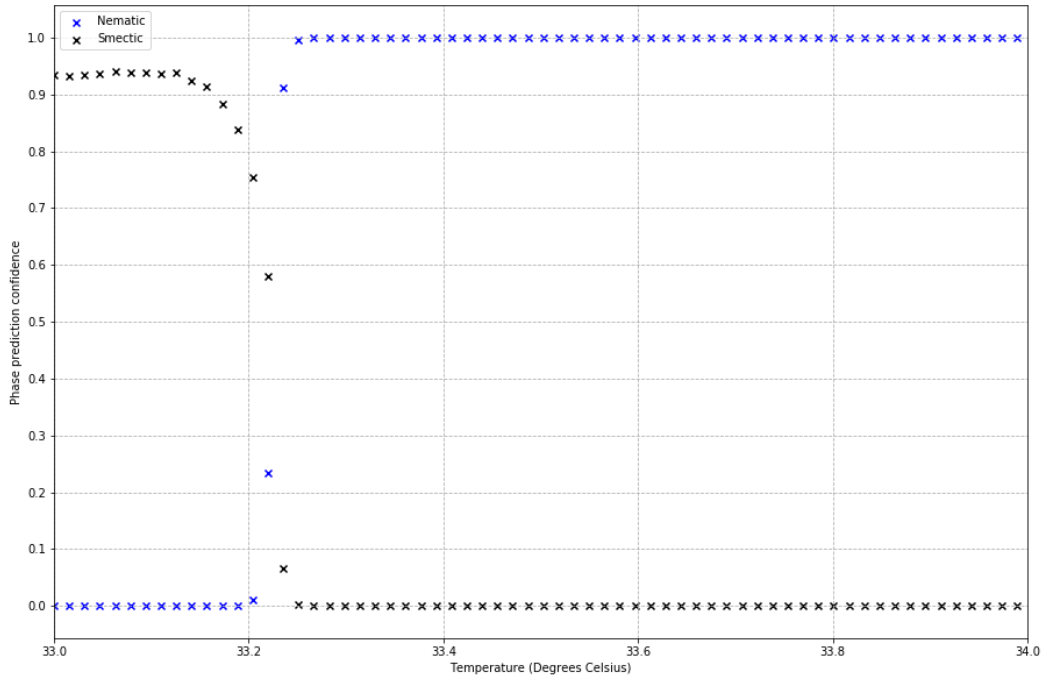


Figure 32: Phase prediction confidence against temperature graph for the same video of a smectic to nematic transition, from 33-36C, for an 8CB liquid crystal texture, as above. This graph shows a close-up of the transition.

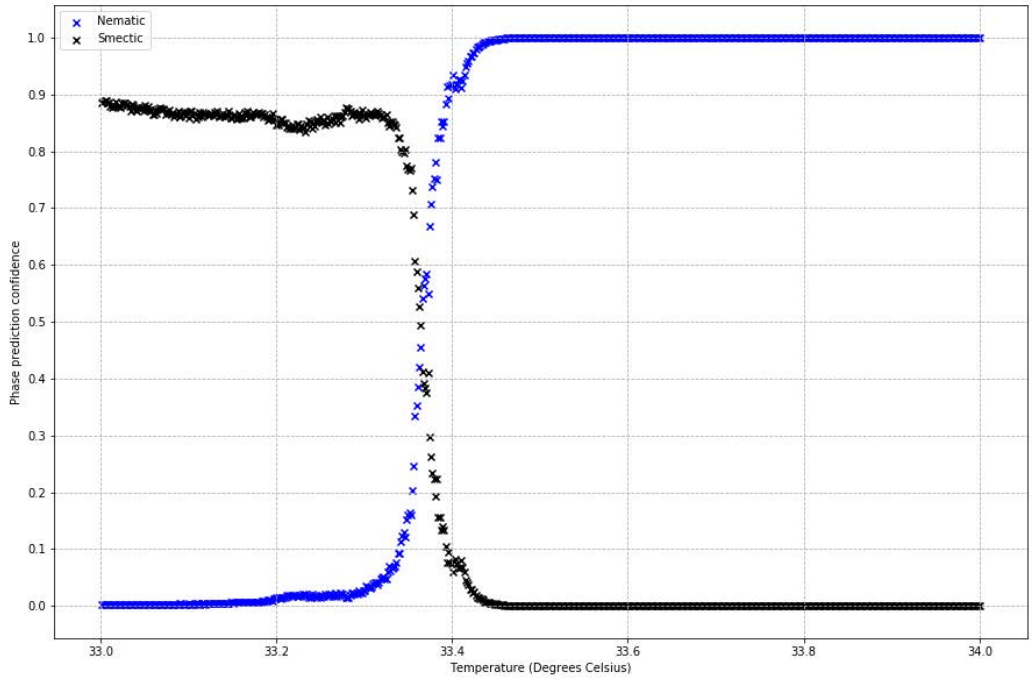


Figure 33: Phase prediction confidence against temperature graph for a video of a nematic to smectic transition, from 34-33C, for an 8CB liquid crystal texture. This transition was correctly labelled by the first CNN. Labelled video available at: https://youtu.be/mE049dcD4bo.

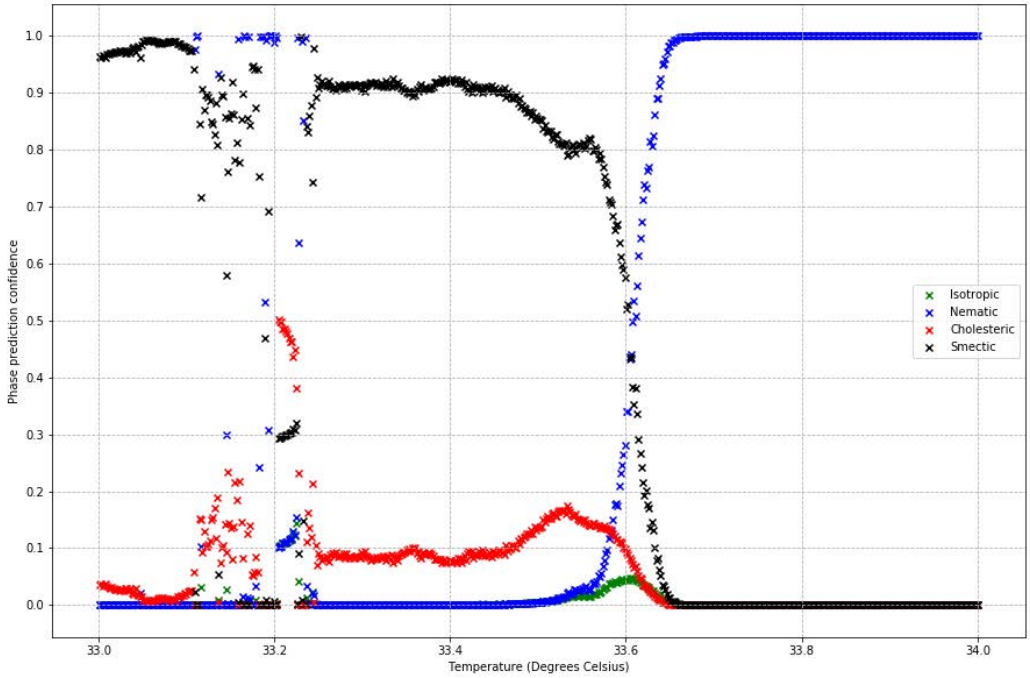


Figure 34: Phase prediction confidence against temperature graph for a video of a nematic to smectic transition, from 34-33C, for an 8CB liquid crystal texture. This transition was correctly labelled by the first CNN. Labelled video available at: https://youtu.be/h2YIncBL1Po.

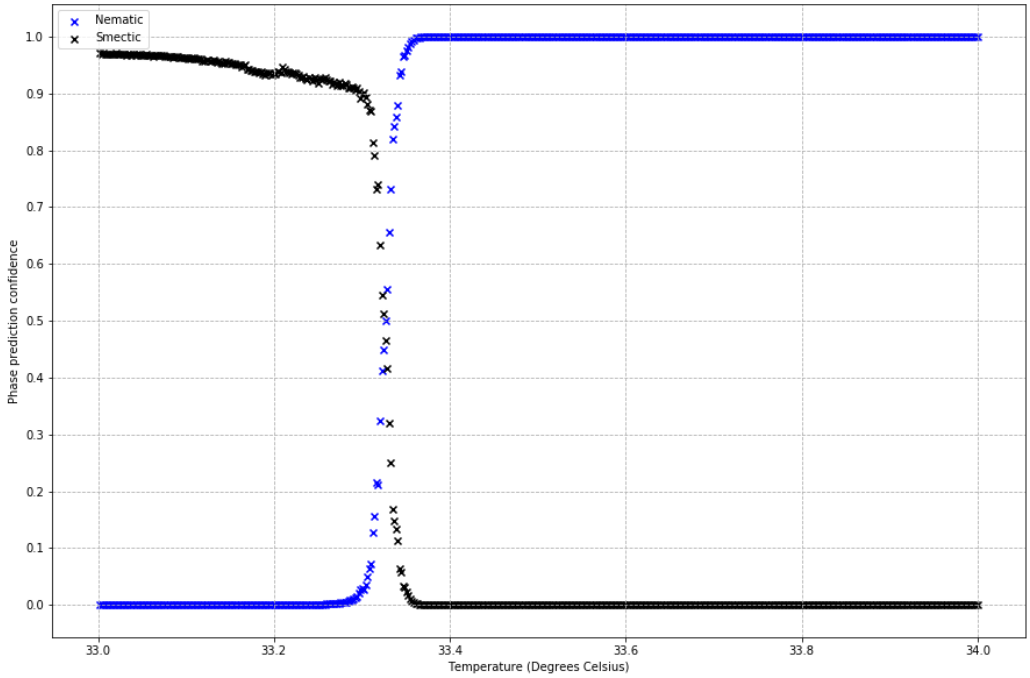


Figure 35: Phase prediction confidence against temperature graph for a video of a nematic to smectic transition, from 34-33C, for an 8CB liquid crystal texture. This transition was correctly labelled by the first CNN. Labelled video available at: https://youtu.be/xoROW7Cxlmg.

C.1.4 Cholesteric and Smectic Transitions

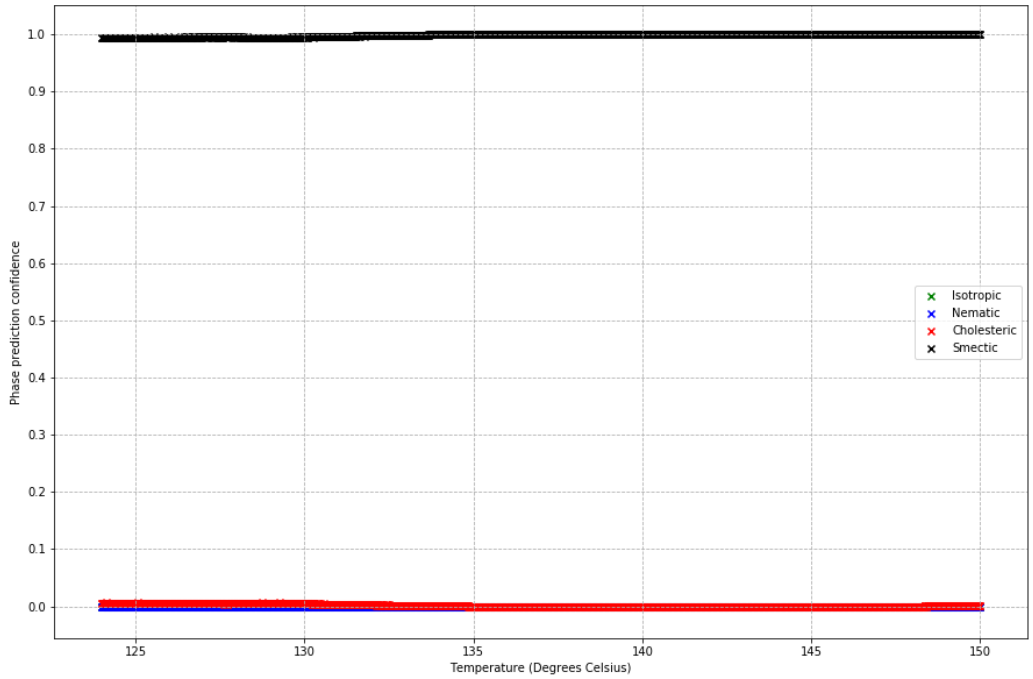


Figure 36: Phase prediction confidence against temperature graph for a video of a smectic to cholesteric transition, from 124-150C, for an M6 liquid crystal texture. This transition was incorrectly labelled by the first CNN. Labelled video available at: https://youtu.be/cakDgNUFBB8.

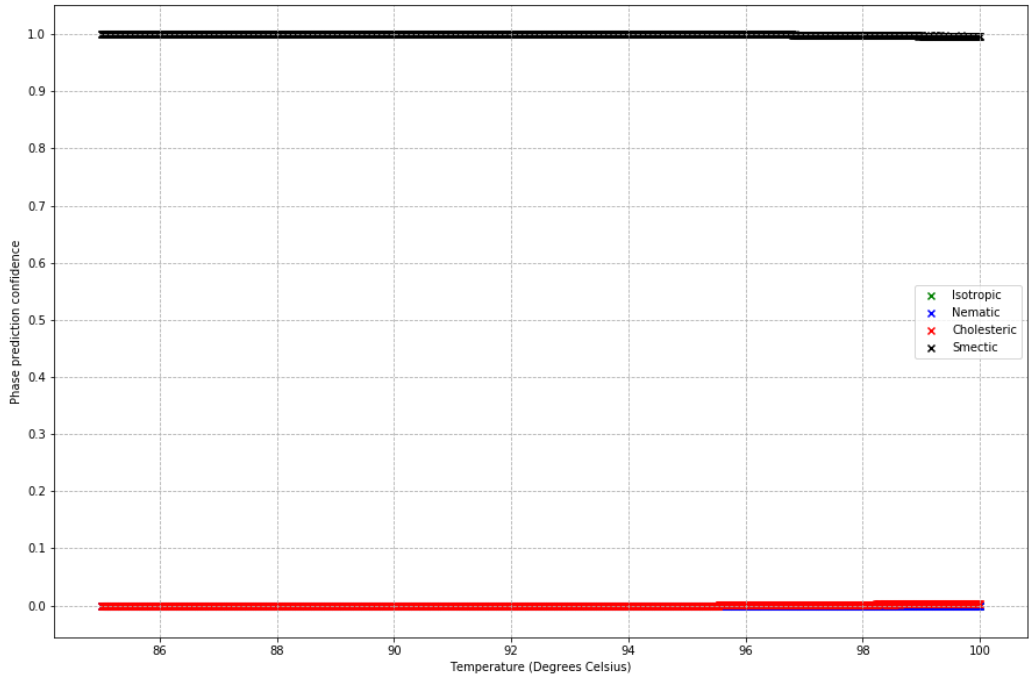


Figure 37: Phase prediction confidence against temperature graph for a video of a cholesteric to smectic transition, from 100-85C, for an M6 liquid crystal texture. This transition was labelled by the first CNN. The cholesteric to smectic transition was slightly late to be detected and then multiple transitions were detected, which did not occur. Labelled video available at: https://youtu.be/J0DAnK6iXTI.

C.1.5 Isotropic to Cholesteric to Smectic Transition

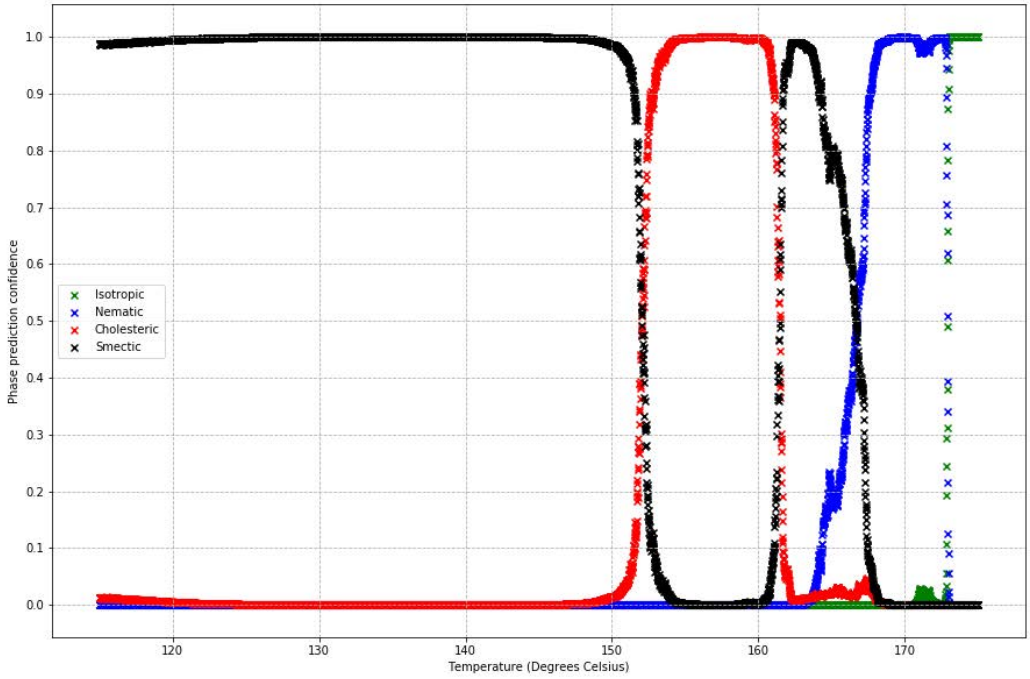


Figure 38: Phase prediction confidence against temperature graph for a video of an isotropic to cholesteric to smectic transition, from 175 C to 115 C, for an M6 liquid crystal texture. This transition was labelled by the first CNN. The isotropic to cholesteric was correctly identified, but a cholesteric to smectic transition was detected too early. Labelled video available at: https://youtu.be/mNB0Etym7Sk.

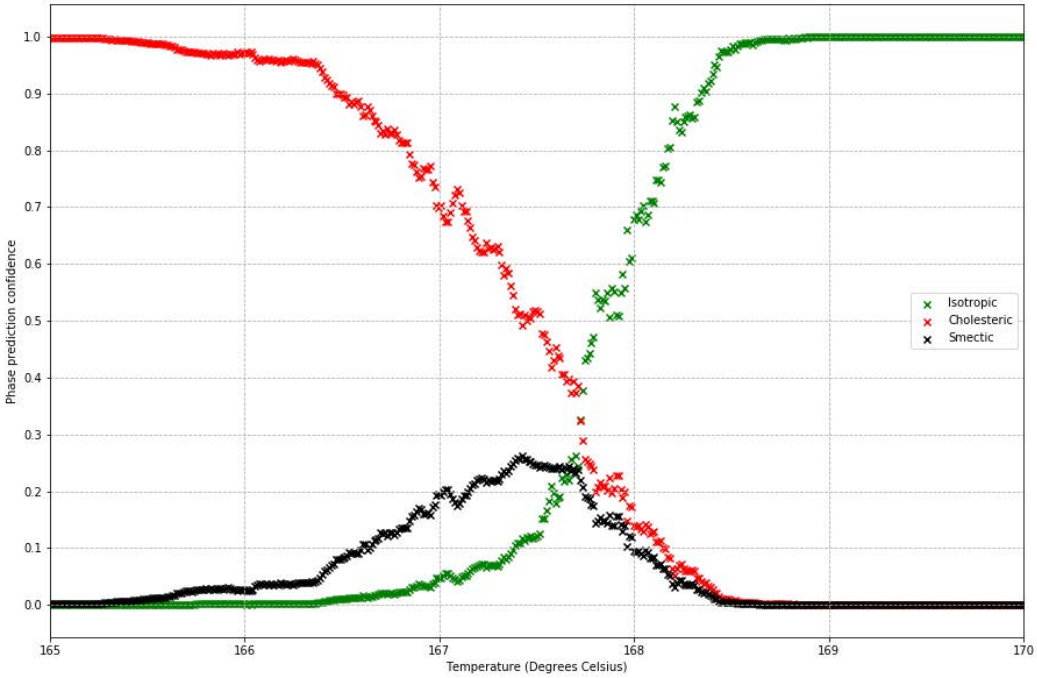


Figure 39: Phase prediction confidence against temperature graph for the same video of an isotropic to cholesteric to smectic transition, from 175 C to 115 C, for an M6 liquid crystal texture, as above. This graph shows a close-up of the isoptropic-cholesteric transition.

C.2 The InceptLC-V4 Model

C.2.1 Isotropic and Nematic Transitions

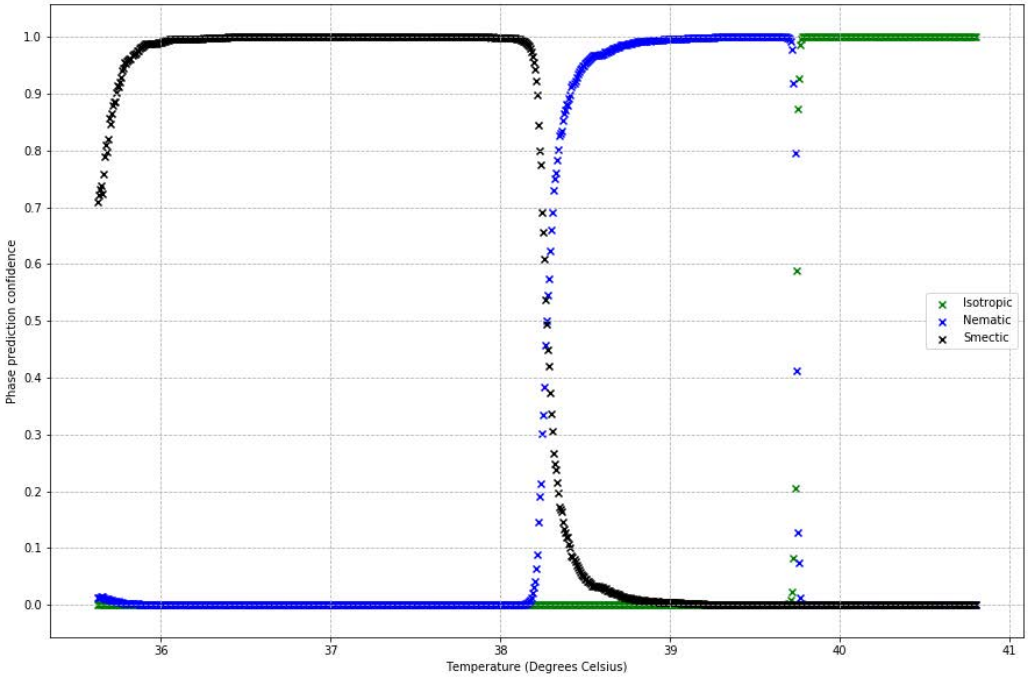


Figure 40: Caption

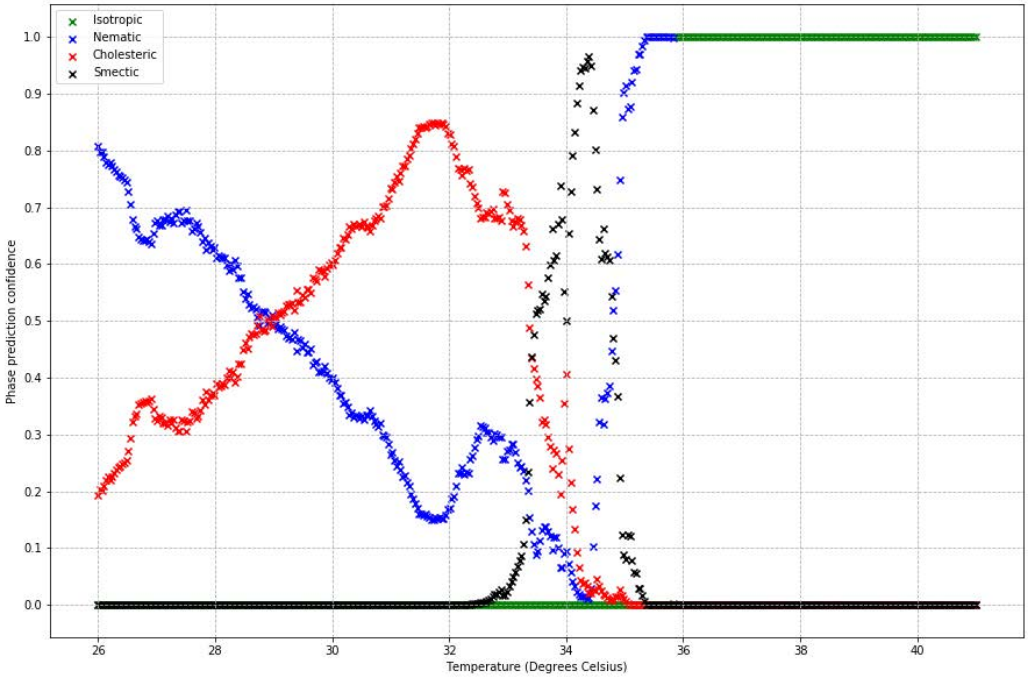


Figure 41: Caption

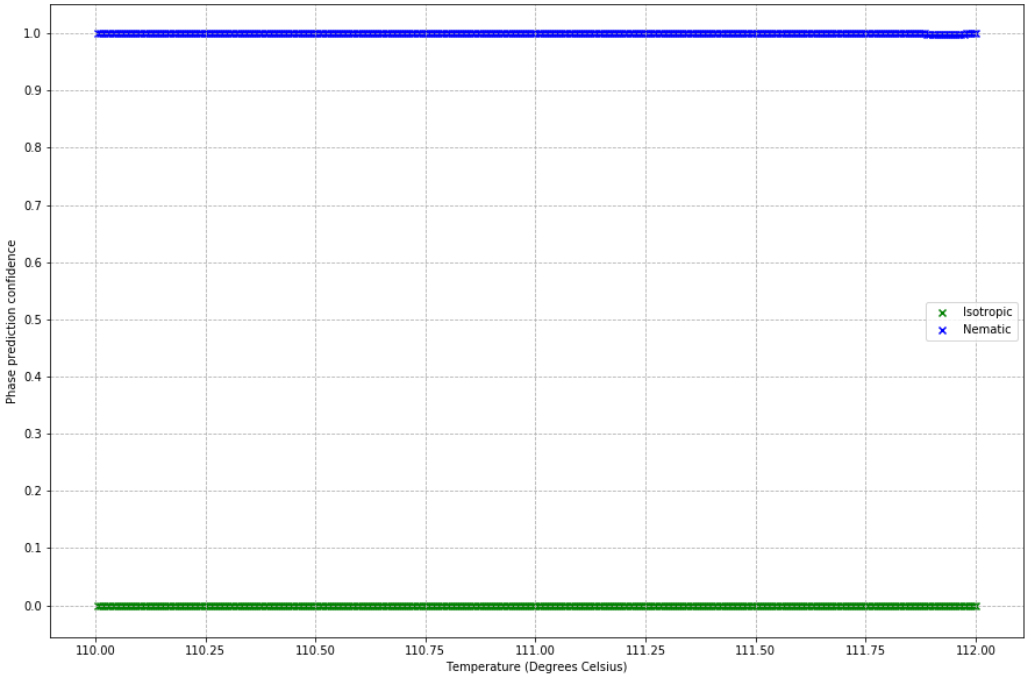


Figure 42: Caption

C.2.2 Isotropic and Cholesteric Transitions

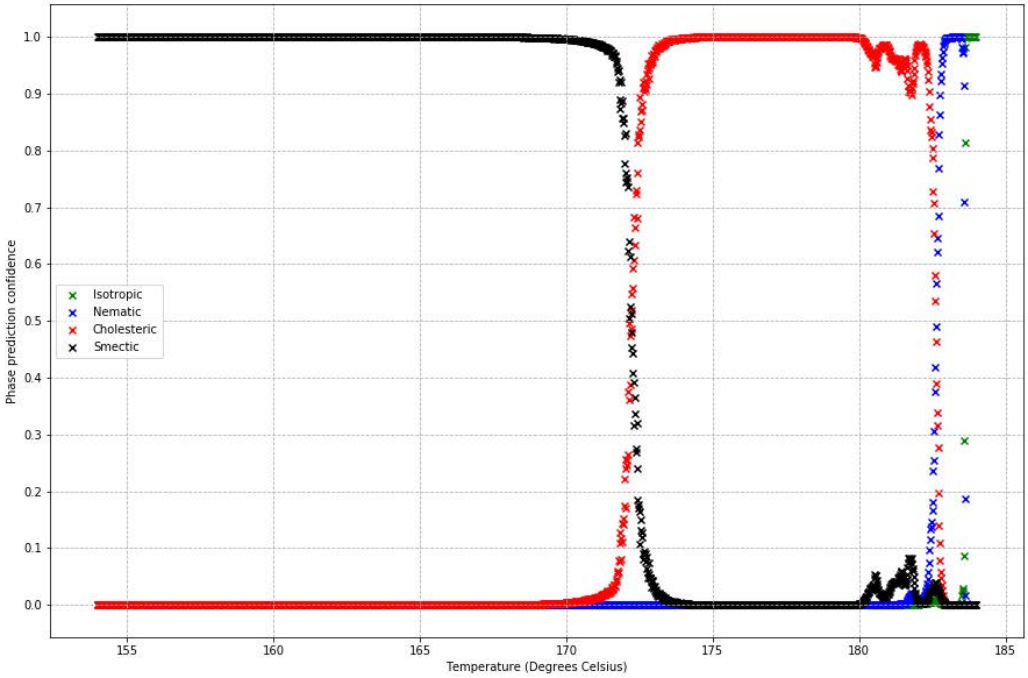


Figure 43: Caption

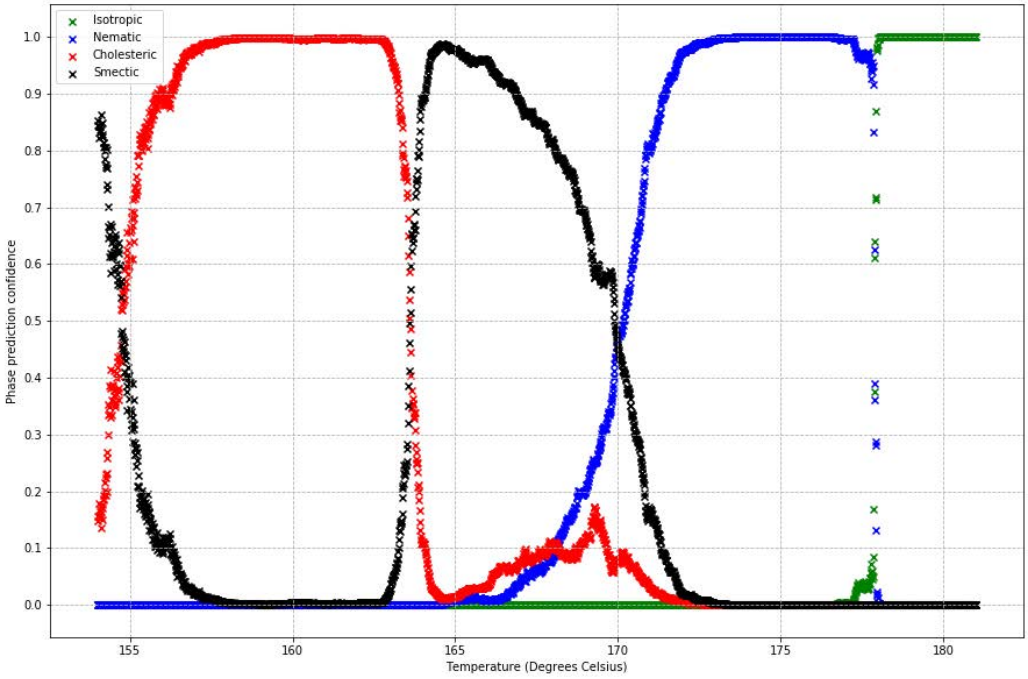


Figure 44: Caption

C.2.3 Nematic and Smectic Transitions

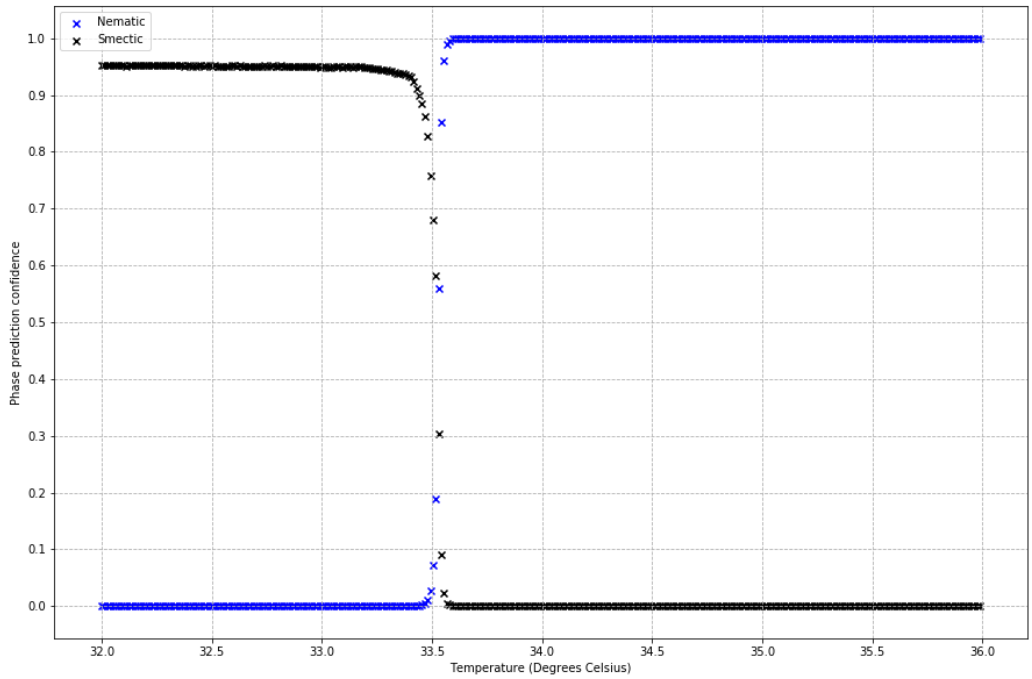


Figure 45: Caption

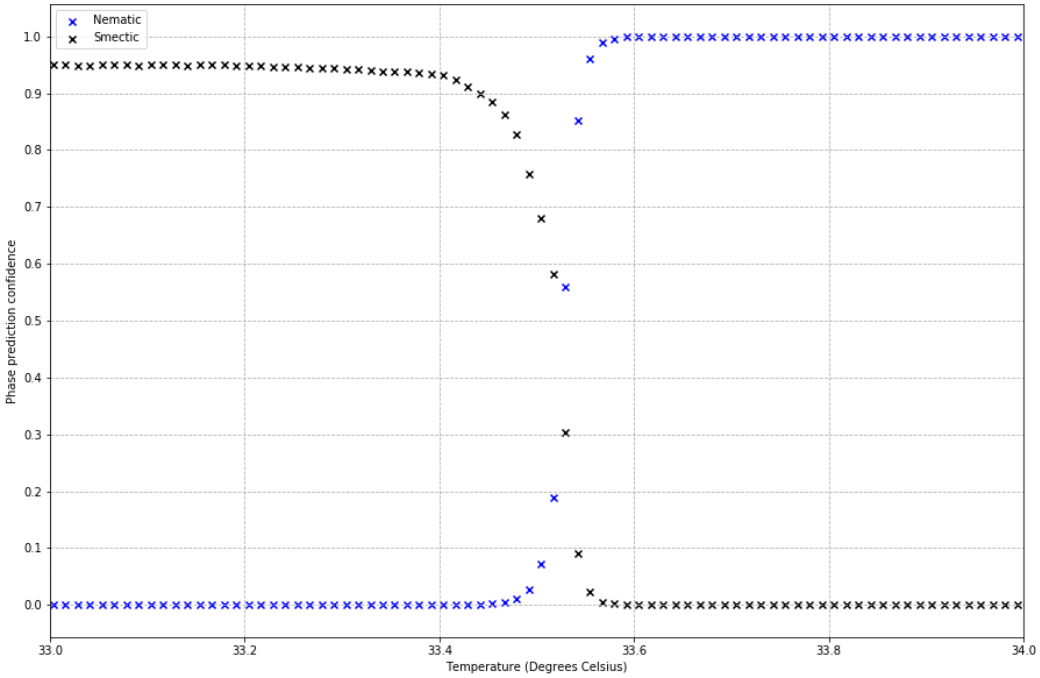


Figure 46: Caption

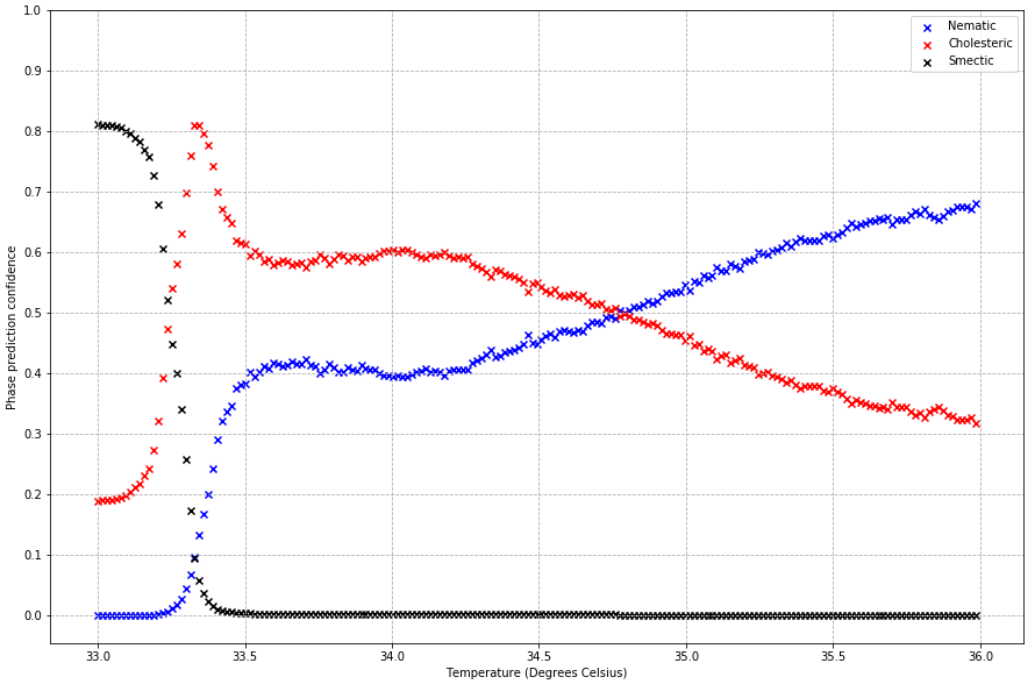


Figure 47: Caption

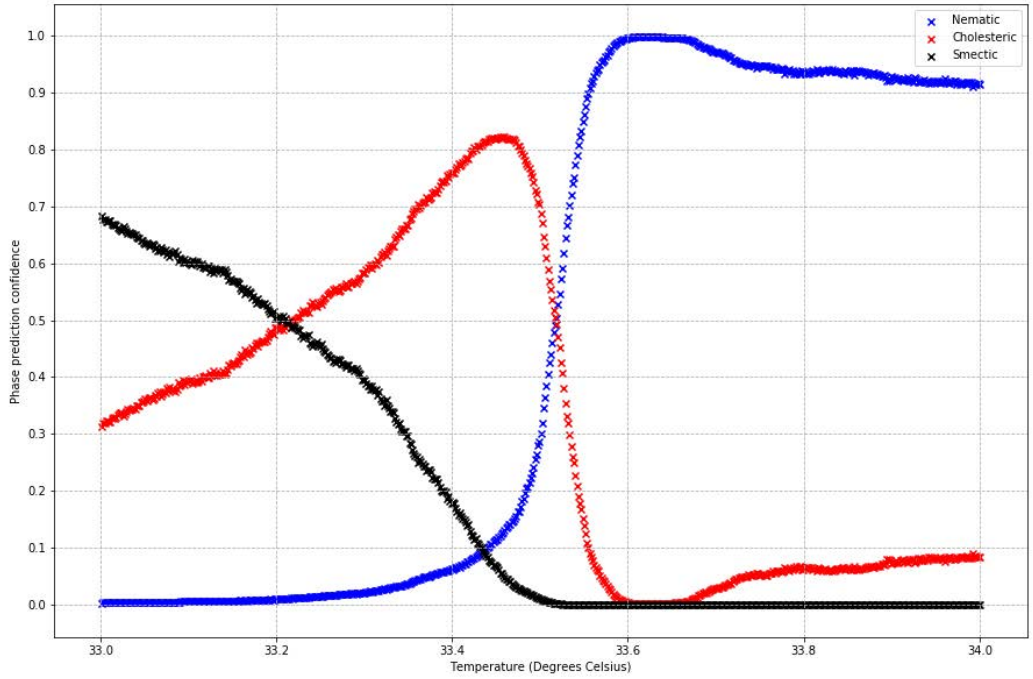


Figure 48: Caption

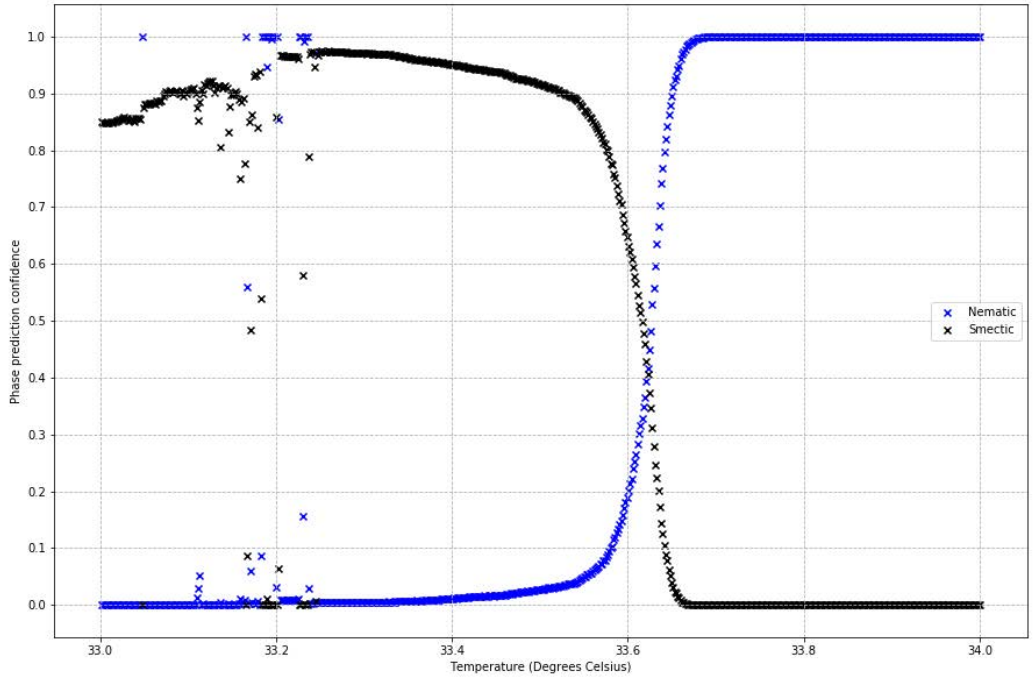


Figure 49: Caption

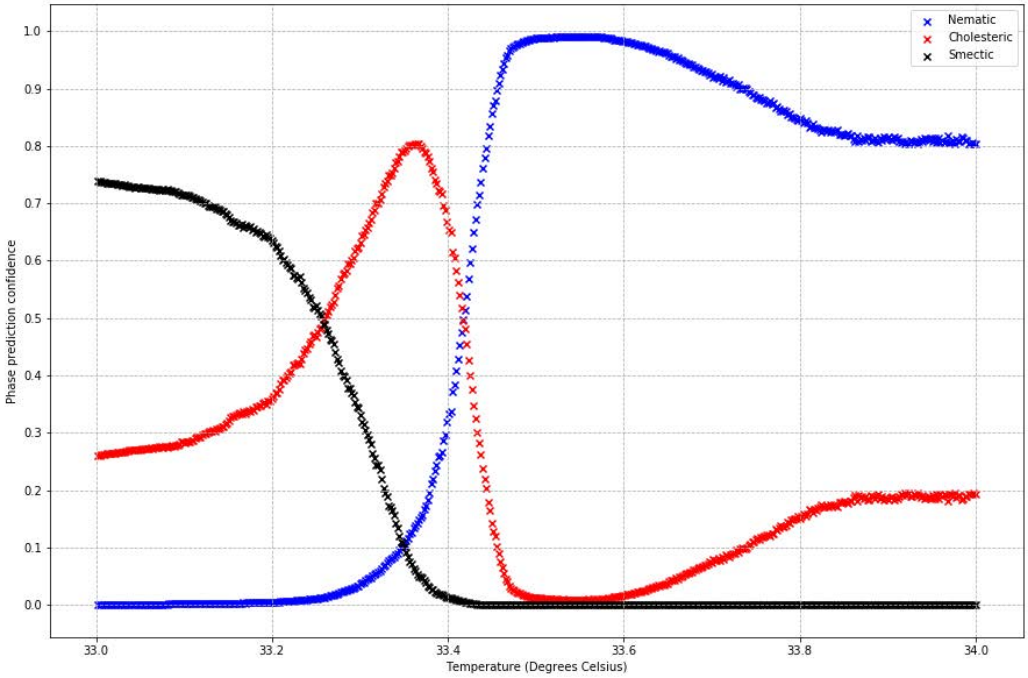


Figure 50: Caption

C.2.4 Cholesteric and Smectic Transitions

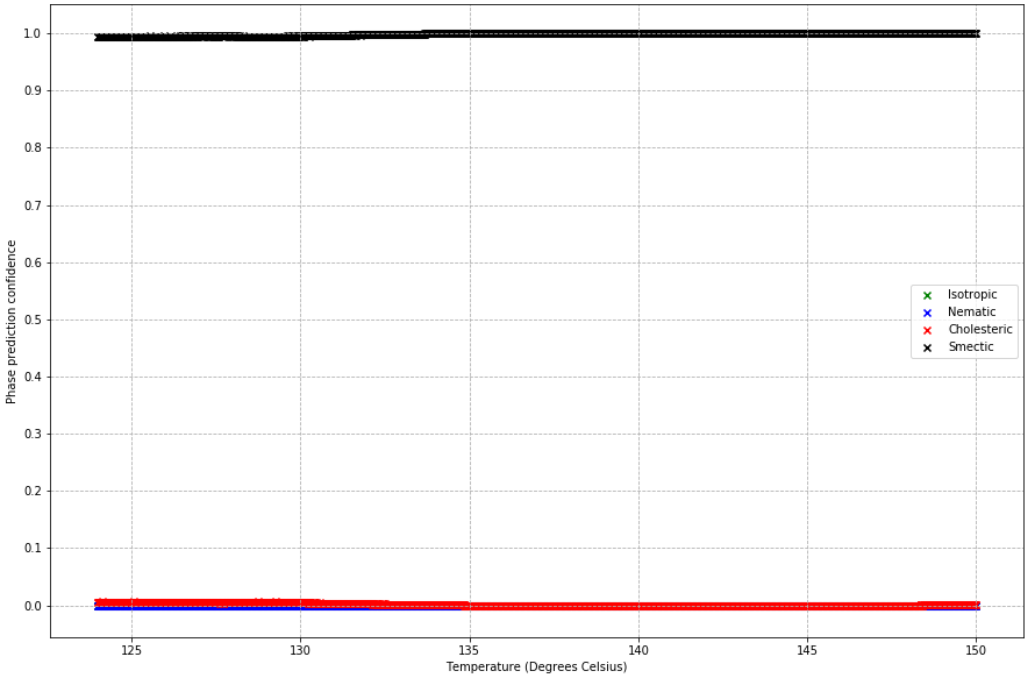


Figure 51: Caption

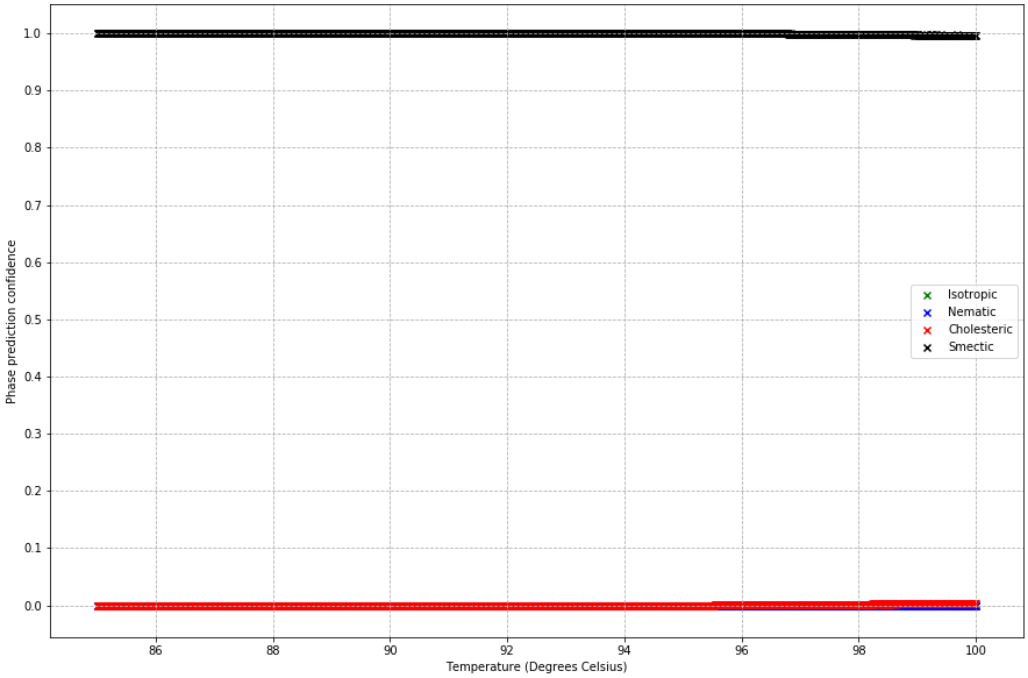


Figure 52: Caption

C.2.5 Isotropic to Cholesteric to Smectic Transition

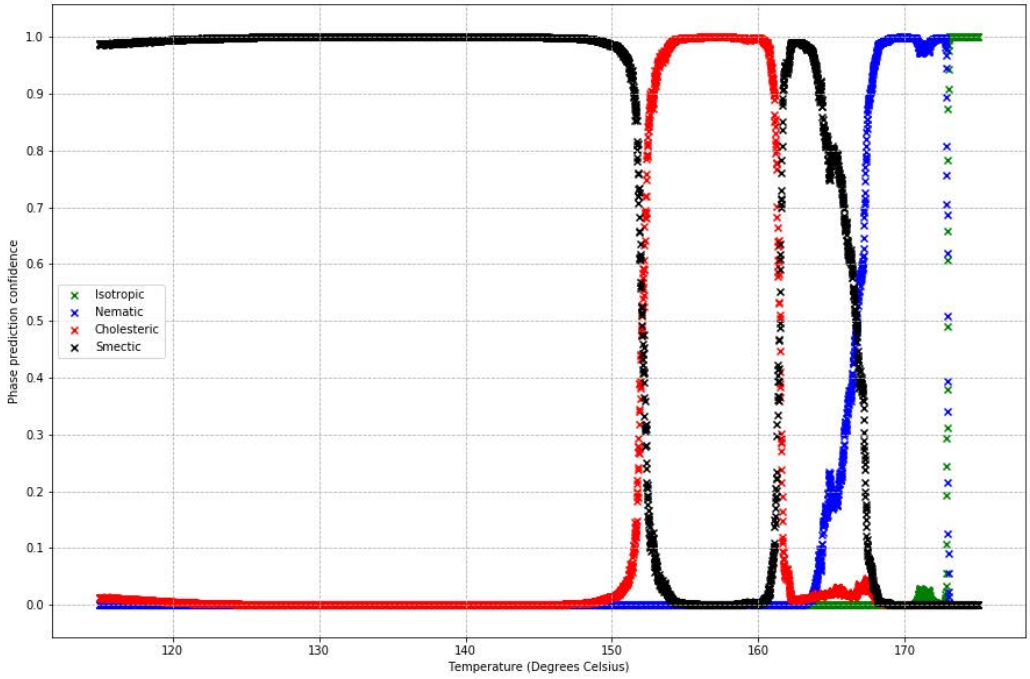


Figure 53: Caption

C.2.6 Fluid Smectic and Hexatic Transitions

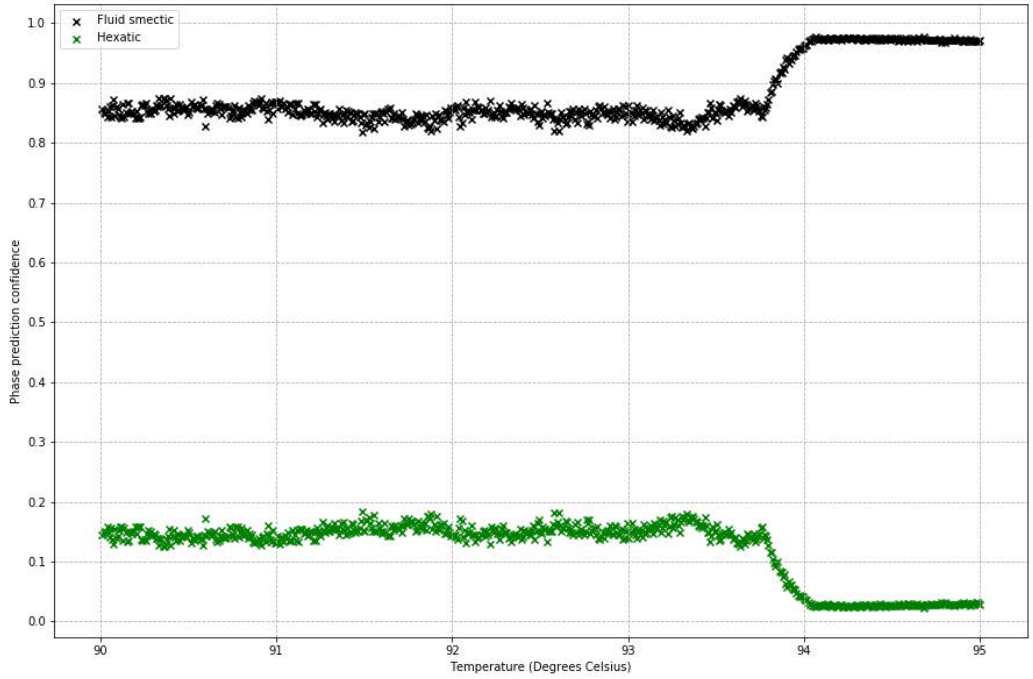


Figure 54: Caption

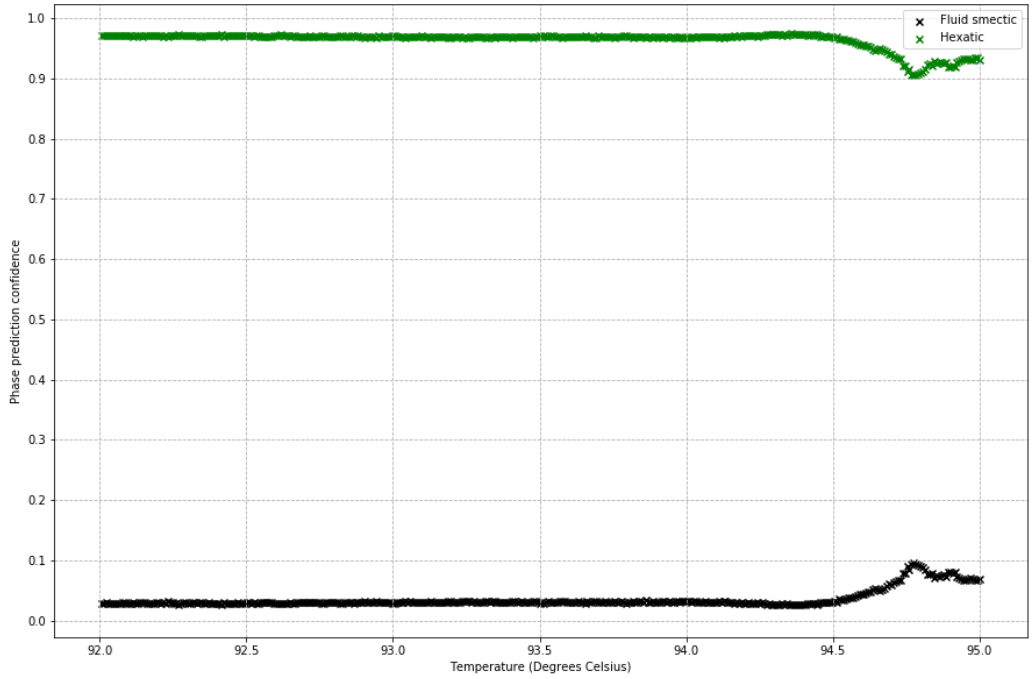


Figure 55: Caption