Classification of liquid crystal textures using convolutional neural networks

DOI:
10.1080/02678292.2022.2150790

Document Version
Accepted author manuscript

Classification of liquid crystal textures using convolutional neural networks

Ingo Dierking, Jason Dominguez, James Harbon, Joshua Heaton*

Department of Physics and Astronomy, The University of Manchester, Oxford Road, Manchester M13 9PL, UK

Abstract

We investigate the application of convolutional neural networks (CNNs) to the classification of liquid crystal phases from images of their experimental textures. Three CNN classifier model types (Sequential, Inception and ResNet50) are tuned and trained on five individual phase group datasets. The complete dataset includes images of the cholesteric phase, chiral fluid smectic A and C phases and hexatic smectic I and F phases, all extracted from polarised microscopy videos of various liquid crystalline compounds. Three binary classification tasks, each including two liquid crystal phases, provide the foundational demonstration of CNN model viability. The average test set accuracies obtained are approximately (95 ± 2)%. More complex multi-phase datasets are also created and investigated, with a three-phase cholesteric, fluid smectic, and hexatic smectic set, in addition to a set containing all five individual phases. The average test set accuracies for these classification tasks are (85 ± 2)%.

1. Introduction

Machine learning (ML) is the term assigned to a wide range of computer algorithms that use data to automatically improve their performance on a specific task. These tasks can take various forms, including decision-making, pattern recognition, and prediction [1]. A sub-field of ML known as deep learning generally consists of applying large-scale multi-layer neural networks, a type of algorithm inspired by the structure of the brain, to tasks involving highly complex abstractions of data. Such intensive algorithms typically require vast quantities of data and powerful computational resources to be trained effectively, with the advantage that they do not require any manual feature extraction [2]. With the recent explosion in availability of such data and sophisticated computing technology, deep learning has seen a surge in interest and application among several fields [3]. Computer vision is one such field that has been impacted greatly. Convolutional neural networks (CNNs), a type of neural network suited particularly well to grid-based data, have proven extremely successful in the tasks of image classification, segmentation, and object detection [4].

There are many thousands of documented liquid crystal (LC) compounds, with each displaying a certain sequence of identifiable phases between that of a liquid and solid [5]. Commonly, polarised microscopy (PM) is used to capture images of the textures produced by LC phases for identification by eye [5]. Literature on ML for LC phase classification is sparse, with most studies focusing on the extraction of physical properties of LCs using simulated texture data [6, 7, 8, 9], or other means [10, 11, 12, 13]. Of most relevance is the work by Sigaki et al., in which they utilise CNNs to classify simulated isotropic and nematic phases to high accuracy [6].

Here, we prepare a novel dataset of LC texture images captured by PM, spanning multiple phases of all orders. Subsequently, we apply CNN classifier models to various phase groupings, probing the limits of attainable model accuracy.

1.1 Liquid crystals

Liquid crystal phases are characterised by the orientational and positional order of the molecular arrangement. In general thermotropic LCs, as used in this study, become more ordered with decreasing temperature [5]. The order and overall structure of the LC phase determines its optical properties, in particular its birefringence. This enables images of the textures of a liquid crystal to be obtained by polarised microscopy, in which the sample is placed between two crossed polarisers.

At sufficiently high temperatures, thermotropic LCs take the form of a fully isotropic liquid with no structural order and hence no birefringence, resulting in completely dark textures. Upon cooling, they will display at least one partially ordered phase before reaching the fully crystalline stage. In between, the lowest order LC phase exhibits solely orientational order of the long molecular axis, called the nematic (N) phase, where the molecules are aligned along a particular direction called the director and are still free to translate as in a liquid. Compounds with chiral molecules may instead display the cholesteric (N*) phase, which is the same as the nematic phase except with helical variation of the director in a direction perpendicular to the long molecular axis. Layered positional order is introduced in the smectic phase (Sm), which is divided into three

distinct phase groupings. The orientation of the director further categorises these groupings. The fluid smectic phases exhibit 1D positional order, thus the formation of layers but with no positional order within these layers. The orientation of the director with respect to the layer planes determines whether the phase is smectic A (SmA), in which it is perpendicular, or C (SmC) otherwise. The smectic B (SmB), I (SmI), and F (SmF) phases are placed into the hexatic smectic (HSm) group, with short-range hexagonal order within the smectic layers, so called bond-orientational order. The soft crystal phases differ in that the molecules within layers exhibit long-range positional order [5].

This investigation uses CNNs to classify experimental polarising microscopy textures from thermotropic chiral LC compounds [14], including the phases N*, SmA*, SmC*, SmI*, and SmF*. Figure 1 gives example textures of each of these phases.

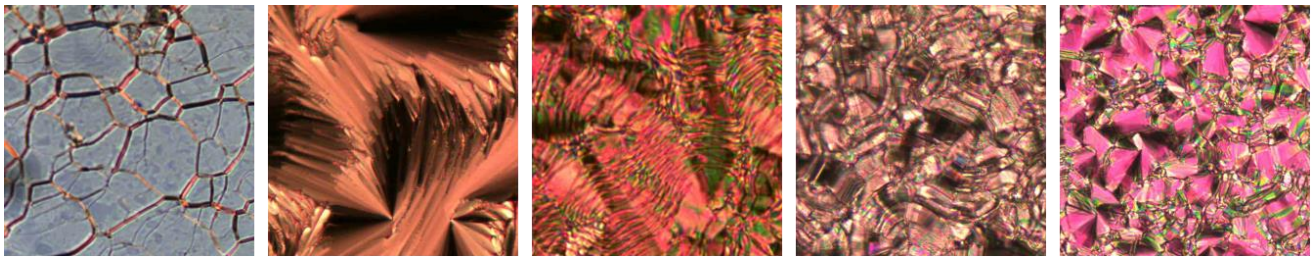


Figure 1: Samples from the LC texture dataset, from left to right: cholesteric, chiral smectic A, C, I and F.

1.2 Supervised machine learning

Machine learning algorithms can in general be categorised as supervised, unsupervised, or reinforcement learning [1]. The ML implementations of this study are purely supervised learning algorithms. In this case, the ML model is defined as a function, parametrised by learned values ϑ, that maps an input data sample x containing various features to an output predicted label $\hat{y}$ [1],

$$\hat{y} = f(x;\vartheta). \tag{1}$$

$\hat{y}$ can take various forms depending on the specific task, for example regression, in which the model attempts to predict a continuous value given the input data, or classification, in which it predicts a category to which the input sample belongs [1]. A supervised model attempts to learn appropriate ϑ values for the mapping using a set of training data, consisting of pairs, i, of input examples and their corresponding true labels, $\{x^{(i)},y^{(i)}\}$. The model takes a sample $x^{(i)}$ and produces an output label prediction $\hat{y}^{(i)}$ according to Equation 1 [1]. A chosen cost function $J(y, \hat{y};\vartheta)$ is then evaluated, which generally provides a measure of the divergence of the model outputs from the true labels. The parameters are then updated with a particular optimisation algorithm in order to minimise the cost. Successful training results in a model that is effective in producing accurate predictions for new unseen data samples [1, 2]. Fixed parameters that define the precise form of f, as well as training configurations, are known as hyperparameters [1, 2].

When deciding on the form of a supervised model, its capacity is of great importance, which can be thought of as the size of the model. A low-capacity model with few trainable parameters

may not extract or infer enough features from the training data to form accurate predictions. This problem is called underfitting. On the other hand, too high a capacity will cause the model to learn many features that are specific to the training data and therefore it may not perform well when inferring on new unseen examples. This is referred to as overfitting. The model's hyperparameters must, therefore, be properly tuned to ensure it does not overfit or underfit. In addition to limiting model capacity, regularisation techniques can be applied to help prevent overfitting and improve generalisation [1, 2].

Measurement of a supervised model's performance is critical in determining how well it will generalise to new unseen data, often done by evaluating a metric on a dataset of samples. For classification tasks a common metric is the percentage of samples to which the model assigned the correct class label [1]. The training dataset can be split into three subsets: training, validation, and test. The training set contains the data samples used to update the trainable parameters of the model. The validation set is used to tune the hyperparameters of the model. Evaluation of a trained model on the training and validation sets can reveal if the model has underfitted or overfitted. In the former case a low performance on both sets will be observed, whereas for the latter a high performance on the training set and low on the validation set will occur [1, 2]. Hyperparameters can then be adjusted accordingly before retraining the model. After a satisfactory model configuration and validation performance are reached, it is evaluated on the as-of-yet unseen test set to give an indication of the model's generalisation error [1, 2].

1.3 Neural networks

1.3.1 Layers

Neural networks are a type of ML algorithm that pass input data through a series of layers, each containing a number of units. Every unit of a layer is connected in a particular way to the units of the previous layer. Units can take various forms including ones with or without trainable parameters, and specific layers are engineered so as to identify, manipulate, and propagate data features from the input to the output of the network [15]. In a fully-connected, or dense, layer the input to each unit is the outputs of all units of the previous layer. The inputs are multiplied by the unit's learned weight parameters and summed together with a learned bias parameter to calculate the unit's output [2, 15]. An activation function can then be applied to the output of each unit of the layer to introduce non-linearity to the network. Such non-linearities are essential in allowing a neural network to act as a universal function approximator, enabling it to perform highly complex inference on input data [16]. A dense layer can hence be represented in matrix form as

$$O = A(WI + B), \tag{2}$$

where O is the vector containing the outputs for the layer, W is the matrix of layer weights, I is the vector of layer inputs, B is the vector of bias parameters, and A is the activation function applied element-wise [2, 15].

Convolutional layers take grid-based input data such as two-dimensional images, and convolve it with a kernel of trainable parameters. The kernel has a width and height smaller than that of the input. The kernel is moved iteratively over the input, with the distance moved each iteration

known as the stride of the convolution [17]. At each step the kernel's parameters are multiplied with aligned input values, with the results summed together to produce the final output value. An activation function can be applied to this output value. The complete layer output is formed as a grid containing the ordered output values from the convolution. The size of the output grid depends on the dimensions of the kernel and the stride of the convolution, as these together determine the number of convolution steps in each direction [17]. The input and output of the layer can have a third dimension, for example with the three colour channels of an RGB image. In this case, the kernel will have a different set of parameters for each channel. This number of channels is set by a hyperparameter known as the number of filters and can be increased from input to output by stacking the results from multiple kernels [17]. A convolutional layer's padding refers to the behaviour of the kernel at the edges of the input. When the kernel is confined completely within the input space it is called valid padding. Same padding is when the kernel extends beyond the input space such that each value is visited by the kernel the same number of times, with kernel values outside the space multiplied by zero. For a stride of one in both directions, same padding will result in an output with the same shape as the input, and valid padding will result in dimensionality reduction [2, 17]. An example of a convolution operation is detailed in Figure 2.

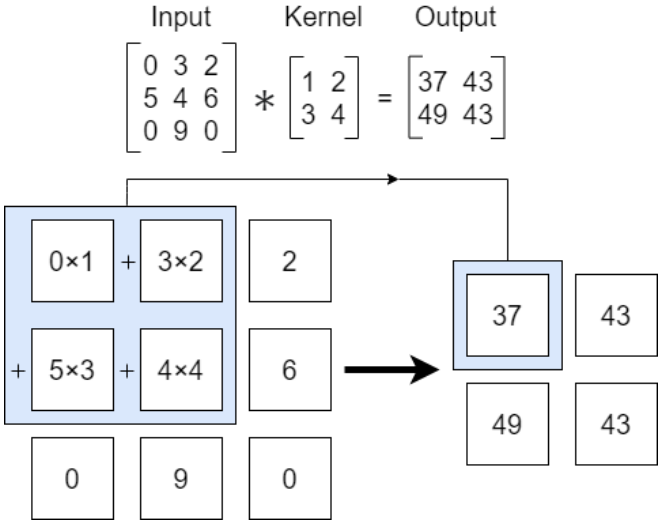


Figure 2: Adapted from [2]. An example convolution operation with one channel. The 3 × 3 input is convolved, denoted by ∗, with a 2 × 2 kernel with stride 1 × 1 and valid padding, to produce a 2 × 2 output matrix.

The main advantage of convolutional layers over dense layers is the greatly reduced computational and memory cost, since they require far fewer parameters. The kernel parameters are shared over the entire input, which can also improve regularisation. Each kernel can be thought of as learning to extract a particular feature from the input to be passed on to the next layer, such as edges of objects in an image [2].

Pooling layers are often used after convolutional layers to reduce the dimensions of the network [2]. They can be thought of as a convolutional layer with a kernel that does not have any trainable parameters and instead performs a specific operation. This could be, for example, taking the maximum value from the aligned input values at each step, known as max pooling. Average pooling takes the arithmetic mean of the aligned input values [17]. For pooling layers

the kernel size is instead known as the pool size, and the pooling operation is applied to each input channel individually. Global pooling refers to the case in which the pool size is equal to the input size, which results in a vector output with one value for each of the input channels [17]. Pooling layers are utilised to reduce the overall size and computational cost of the network whilst providing it with some invariance to translations of the input [2, 17].

A common choice of activation function for dense and convolutional layers is the rectified linear unit (ReLU), defined for unit output z as

$$A(z) = \max(0,z), \tag{3}$$

which partially models the behaviour of neurons in the brain [18]. It has the advantage of introducing non-linearity without the potential for vanishing or exploding gradients when training the network [2, 18]. For classifier neural networks, the final output layer is a dense layer with a number of units equal to the number of classes. The softmax activation function is applied, which converts the output of each unit into a probability that the input sample belongs to the unit's corresponding class [2]. For output unit i, this is defined as

$$\sigma(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}, \tag{4}$$

where N is the total number of classes. The final output of the network is generally taken as the class with the greatest assigned probability [2].

1.3.2 Training

Before training begins, a neural network's trainable parameters are randomly initialised from a particular distribution such as a normal distribution [2]. A training step is performed by first selecting a "minibatch" containing a set number, called the batch size, of random samples from the training set of data. The samples are passed through the network and a loss function is evaluated for each output [2]. For classifier models, a widely used loss function, derived from the Kullback–Leibler divergence and maximum likelihood estimation, is the categorical cross-entropy, defined as

$$L(p) = -\log p \tag{5}$$

where p is the model's output probability that the sample belongs to the true labelled class [19]. The cost function is then calculated as the average of the loss for all samples in the minibatch [2]. An algorithm called backpropagation is then applied to calculate the gradient of the cost function with respect to the trainable parameters, $g = \nabla_\theta J(y,\hat{y};\vartheta)$. Backpropagation, in summary, applies the chain rule of differentiation sequentially from the final network layer going backwards to the input layer, extracting the gradients at each unit output [2, 20]. A chosen optimisation algorithm is then applied to update the parameters, with a simple example being stochastic gradient descent [2]. In this case, the parameters are updated against the gradient as

$$\vartheta \leftarrow \vartheta - \alpha g, \tag{6}$$

where α is a hyperparameter called the learning rate, which modulates how much the parameters are adjusted with each step [20]. This act of descending the cost function aims to reduce the loss

when the model is evaluated on future samples, and in doing so improves its accuracy [2]. Minibatches are sampled without replacement until the entire training set has been observed by the model, completing an epoch of training [2].

1.3.3 Regularisation methods

There are numerous methods of regularising neural networks in order to avoid overfitting [2]. Here the details are provided for methods utilised in this study.

Dataset augmentation aims to effectively increase the overall number of samples in the training set by performing transformations on the samples when they are selected for a minibatch, with the result taking the same label as the base sample. In the case of image data, alterations can be applied randomly and can include flipping the image, rotations, translations, and magnification within certain ranges. When used appropriately, augmentations are a powerful and simple way to improve model generalisation [2].

Another simple yet highly effective regularisation method is early stopping. During training the model's performance on the validation set, usually simply the cost evaluated on the entire set, is recorded after every training epoch. Training stops if the cost has not decreased by more than a tolerance value after a certain number of epochs, defined by the patience hyperparameter. This helps greatly with regularisation because the model can overfit the training set if trained for too many epochs [2, 21].

Dropout is a regularisation technique in which some unit outputs in a layer are multiplied by zero with a set probability, called the dropout rate, with random units selected each training update step. This can be viewed as training multiple sub-models with shared parameters, and it has the effect of reducing the neural network's sensitivity to noise [22].

For a layer with batch normalisation, after calculating the layer output values for each sample in a minibatch, the outputs are rescaled by subtracting the minibatch mean for each unit output and dividing by the standard deviation. The result for each unit is then rescaled linearly with extra learnable parameters. For output z of a layer's unit this change is represented as

$$z \leftarrow \gamma \left(\frac{z - \mu}{\sigma} \right) + \beta, \tag{7}$$

where γ and β are the extra learnable parameters, μ is the mean and σ is the standard deviation of the unit's output over the minibatch. For future computations on single samples, during training running averages of the means and standard deviations are recorded. Batch normalisation improves model stability whilst training and provides a regularising effect by introducing a form of noise [23].

2 Methodology

2.1 Data preparation

All LC texture image data used has been obtained from PM videos of LCs, labelled by compound and temperature range. Two homologous series of chiral compounds were investigated which are discussed in detail in the literature [24, 25]. Textures of specific phases were accumulated

from all homologues which exhibited these phases. The software VLC Media Player [26] is used to extract image frames from the videos, and they are classified according to the LC phase displayed at the point of extraction. The raw images have a resolution of 2048 × 1088. They are split into six smaller images of size 682 × 544 without compromising the features displayed by each image. The images are then cropped to square 544 × 544 before being scaled down to the model input size of 256 × 256 and converted to greyscale with pixel value range zero to one. This input size is selected based upon results of studies relating to the performance of different machine learning models, changing input image size, number of CNN layers and number of inception blocks [14]. Further, images were converted to black and white to reduce the computational cost and to avoid models developing a dependence on colour, while the actual focus lies on the texture.

In construction of the training, validation, and test data sets, images of the same phase that also come from the same video are not divided between any of the three sets. This is to minimise potential data leakage, which is observed when samples in the training set are highly similar to samples in the validation or test sets, artificially inflating model accuracy on the validation or test set [27]. The distribution of the complete dataset over all LC phases is presented in Figure 3. From

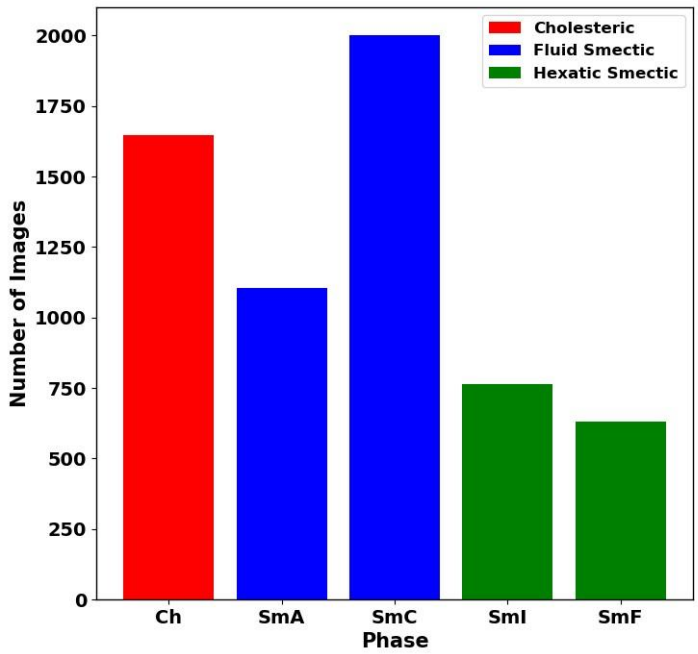


Figure 3: The number of images in the complete dataset belonging to each phase, with a total of 6,978 images.

this we construct five individual model training datasets, split by video in an approximate ratio of 3:1:1 into training to validation to test set image count. Videos are selected randomly from the training set to be moved into the validation and test sets until the target split ratio is approximately met. The five cases of different phase groupings studied, include three binary (two-phase) sets and two multi-phase sets. The names of each dataset and the phases they

include are summarised in Table 1, with the specific distributions of the data in each set presented in Table 2.

Table 1: The LC phases contained in each dataset.

Dataset	ChSm	SmAC	SmIF	ChSm2	ChSm4
Phases	Ch, Sm	SmA, SmC	SmI, SmF	Ch, FSm, HSm	Ch, SmA, SmC, SmI, SmF

Table 2: Image count and distribution for each LC phase class.

Phase	Ch	Sm	FSm	HSm	SmA	SmC	SmI	SmF
Training	1148	3598	2110	1488	722	1388	918	570
Validation	245	853	481	372	180	301	198	168
Test	253	881	515	366	204	311	210	162
Totals	1646	5332	3106	2226	1106	2000	1326	900

2.2 CNN Models

In this study we use three different types of CNN architectures, with each built from dense, convolutional, and pooling layers. All convolutional layers use a stride of 1×1, same padding, and ReLU activation, and all dense and convolutional layers have batch normalisation. A standard convolutional layer has kernel size 3 × 3, and a standard pooling layer refers to a max pooling layer with pool size and stride of 2 × 2 and same padding.

2.2.1 Sequential architecture

The simplest and lowest capacity models used are Sequential CNNs. They consist of a series of standard convolutional layers, each followed by a standard pooling layer. The number of channels is doubled with each successive convolutional layer. The final convolutional layer in the network is followed instead by global average pooling, which is proceeded by two dense layers, first with a number of units equal to the output of the average pooling, and second with half that amount. Both dense layers have ReLU activation and dropout rate 0.5. The final layer is the dense classification output with a number of units equal to the number of classes and a softmax activation. Figure 4 displays a diagram of a Sequential model with three convolutional layers.

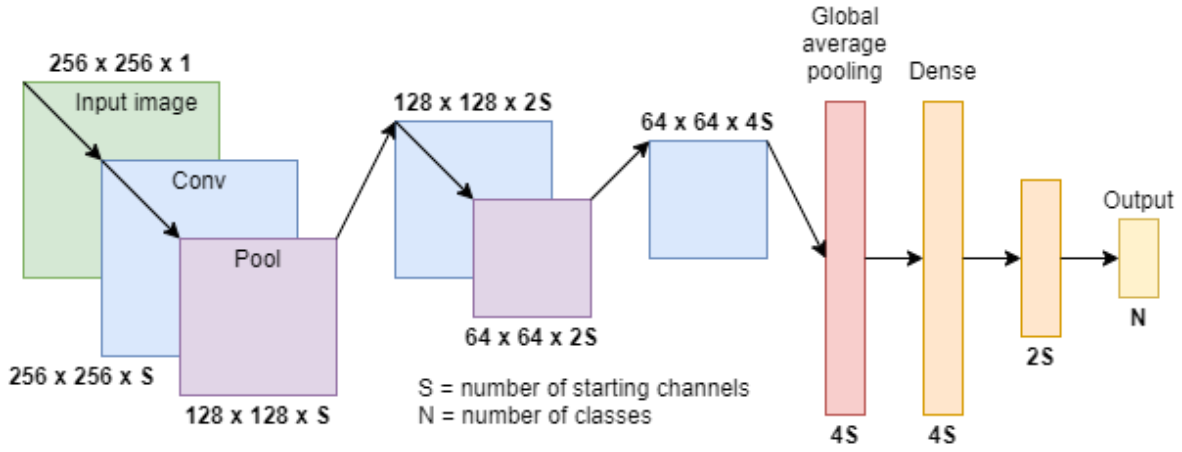


Figure 4: Schematic of a Sequential network with three convolutional layers. "Conv" refers to a standard convolutional layer and "Pool" to a standard pooling layer. Layer output shape is given in bold next to each layer.

2.2.2 Inception architecture

A set of models based on Google's Inception CNN architecture contain modules called Inception blocks, which have parallel convolutional layers with different kernel sizes sharing inputs and outputs [28]. The structure of an Inception block is detailed in Figure 5. Our downsized Inception networks begin with a convolutional layer with kernel size 7 × 7, followed sequentially by a standard pooling layer, a convolutional layer with kernel size 1×1, a standard convolutional layer, and a standard pooling layer. The output of this pooling layer is then fed into a series of one or more Inception blocks. The output of the final Inception block is followed sequentially by an average pooling layer with pool size and stride 5×5 and valid padding, a standard convolutional layer, and finally the same output dense layer structure as the Sequential models starting with global average pooling. Similar to the Sequential models, the number of channels doubles with each convolutional layer, aside from inside the Inception blocks, in which the number of channels is halved from the block input and then kept constant. The output concatenation has four times the channels as they are stacked from each branching layer.

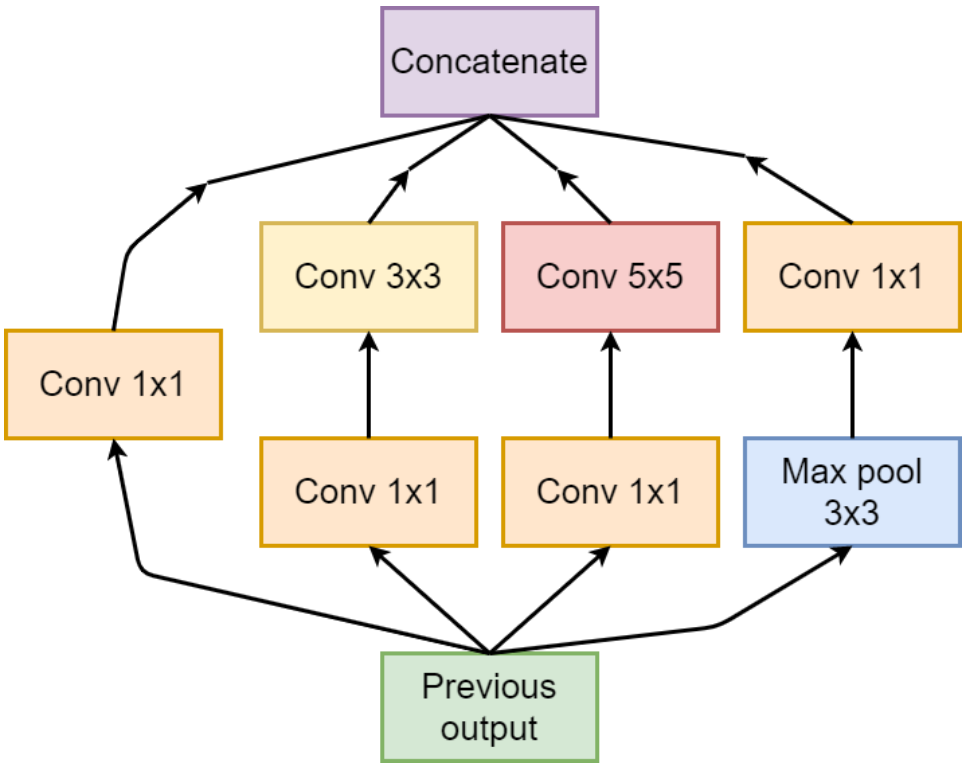


Figure 5: Diagram of a single inception block, adapted from [28]. "Conv 1x1" represents a convolutional layer with kernel size 1×1 and "max pool 3x3" represents max pooling with pool size and stride of 3 × 3. The branching architecture with varying kernel sizes attempts to extract features of varying sizes from the input [28].

2.2.3 ResNet50 architecture

The ResNet models, first implemented in 2015 by Kaiming He *et al.*, aim to tackle the problem of vanishing gradients in CNNs with many layers [29]. They do this by adding skip connections to the network, which feed the ouputs of layers early in the network to later layers, in conjunction with the standard sequential layer inputs. The specific model we utilise in this study is called ResNet50, owing to it having a total of 50 layers [29]. A diagram of the architecture is presented in Figure 6. This is an extremely high-capacity model owing to the number of layers and channels within each layer, amounting to more than 25.5 million trainable parameters [29].

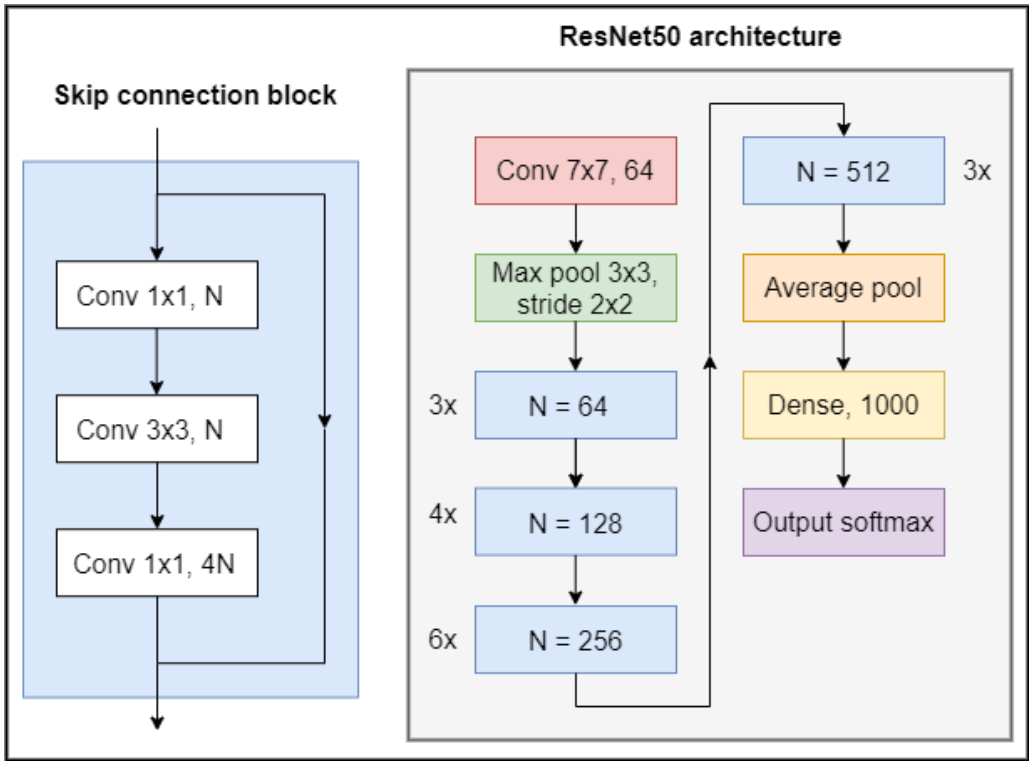


Fig.6: The architecture of ResNet50, adapted from [29]. "Conv 1x1, N" is a convolutional layer with kernel size 1 × 1 and *N* channels. All convolutional layers have stride of 1 × 1. The input to a skip connection block is put through three convolutional layers, and the output of this is concatenated with the original input. The factors next to each skip connection block represent a series of that number of blocks.

2.3 Model training configurations

We use the deep learning libraries TensorFlow and Keras to build and train all models [30, 31]. Model training is powered by NVIDIA CUDA, either on an NVIDIA RTX 2060 graphics card or cloud-based using Google Colaboratory [32, 33]. All models use the categorical cross-entropy loss function and are updated with the Adam optimiser with variable learning rate and other fixed hyperparameter settings of $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-7}$[14, 34]. Early stopping is applied to all models, with a patience of 25 epochs and based on validation set cost. Each model is trained for a maximum of 50 epochs if early stopping is not activated before. The final saved model parameters correspond to the epoch of training with the lowest validation set cost. Random flipping in both the horizontal and vertical directions is applied to the minibatches of images throughout training [14]. These are the only augmentations used. Model accuracy on a dataset is evaluated as the percentage of correctly classified images in the set.

2.4 Model tuning

For each dataset, we train and tune Sequential, Inception and ResNet50 classifier models with the aim of maximising accuracy when evaluating on the test set. The hyperparameters we choose

to vary include batch size and learning rate for all model types. For the Sequential models we also vary the number of convolutional layers and starting channels, and for the Inception models we vary the number Inception blocks and starting channels. The ResNet50 architecture is fixed. For every configuration tested we train the model ten times, recording the validation and test set accuracies at the end of each run. The model's parameters are reset between each run. The mean accuracies for the configuration are then calculated along with the standard deviations. Selection of hyperparameters is based on trial and error combined with grid-search methods. In a grid-search lists of values are specified for two or more hyperparameters, and models are trained with all possible combinations of the values [2].

3 Classification tasks and results

Here, the results for the models of each type achieving the highest mean test set accuracies for each phase classification task are presented. Architecture details are listed as number of convolutional layers, number of starting channels for Sequential models, number of blocks, and number of starting channels for Inception models. The error bars on the results summary graphs for each dataset represent the standard deviation of the test set accuracy. Model types are abbreviated as "Seq" for Sequential, "Inc" for Inception, and "RN50" for ResNet50. The y-axis scale is fixed at 50 to 100 percent to aid comparison of the results between each dataset.

The test set mean and standard deviation confusion matrices are given for the best model type and configuration in each case. Confusion matrix values represent the fraction of test set samples with a particular true label that the model assigned a particular predicted label. For the best performing model configurations, the confusion matrices are calculated for all ten individual trained models, and the mean and standard deviation are calculated for each value.

3.1 Binary classifiers

The binary classifiers predict which of two phases a LC texture is displaying. These serve as foundational investigations for model viability before attempting to build more complex multiphase classifiers. The simplest of such binary classifiers is to distinguish between the isotropic phase and a liquid crystal phase. This task is obviously quite trivial and all models achieved a 100% success rate, so this will not be discussed here in any more detail.

3.1.1 ChSm (cholesteric – smectic)

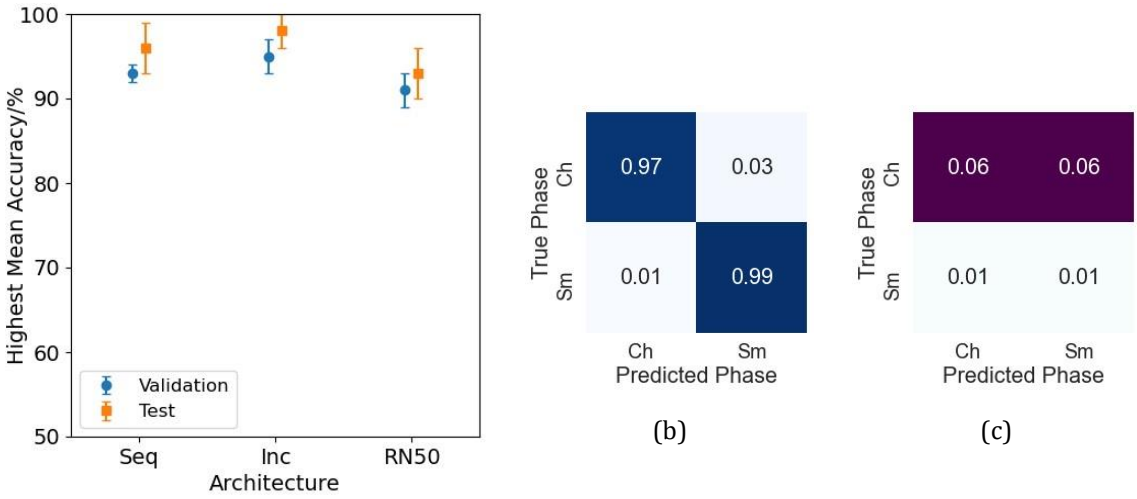
Models in this first non-trivial binary phase classification task attempt to differentiate between the cholesteric phase and all smectic phases in the complete dataset. The final results for the tuned models are displayed in Figure 7 and the model configurations and accuracies are given in Table 3. All three model types achieve over 90% mean test accuracy, with Inception the highest and ResNet50 the lowest. In the latter case we presume that the extremely high capacity of ResNet50 leads to some overfitting, despite usage of early stopping. Overall, the variances in accuracy suggest good stability in training the models, with no standard deviations greater than 3%. The mean test accuracies are higher than validation in every case, however, the differences

are small. This trend is most likely due to the relatively small size of the dataset, which could result in elevated sensitivity to the exact choice of videos to include in each set.

Table 3: Best results and corresponding model configuration details for the ChSm dataset. For the number of parameters, k represents multiples of one thousand and m represents one million.

ChSm	Sequential	Inception	ResNet50
Mean test accuracy/%	96 ± 3	98 ± 2	93 ± 3
Mean validation accuracy/%	93 ± 1	95 ± 2	91 ± 2
Architecture details	3, 64	1, 16	N/A
Batch size	16	16	16
Learning rate	5×10^{-5}	1×10^{-4}	1×10^{-4}
Trainable parameters	470 k	497 k	25.5 m

The mean confusion matrix in Figure 7b for the Inception model shows that the inaccuracies in general stem from misidentifying the cholesteric phase as smectic, at 3%. This is likely due to



(a) (b) (c)

Figure 7: (a) Mean test and validation set accuracies for the best model of each type trained on the ChSm dataset. (b) Mean test set confusion matrix for the ChSm Inception model. (c) Standard deviations of test set confusion matrices for the ChSm Inception model.

the large imbalance in the dataset in favour of the smectic class, which has approximately three times the number of samples. Such an imbalance may induce some bias in the model towards the larger class. In addition, the models are most unstable when processing cholesteric samples, as demonstrated by the standard deviation confusion matrix in Figure 7c, with 6% for both true cholesteric values. Again, this could be due to the class imbalance. There may also be some cholesteric images with features that are similar to some smectic ones. Overall, the ChSm task is

successful in demonstrating that the CNN models can easily learn to distinguish between features of two broad liquid crystal classes, phases with only orientational order and those with orientational and positional order.

3.1.2 SmAC

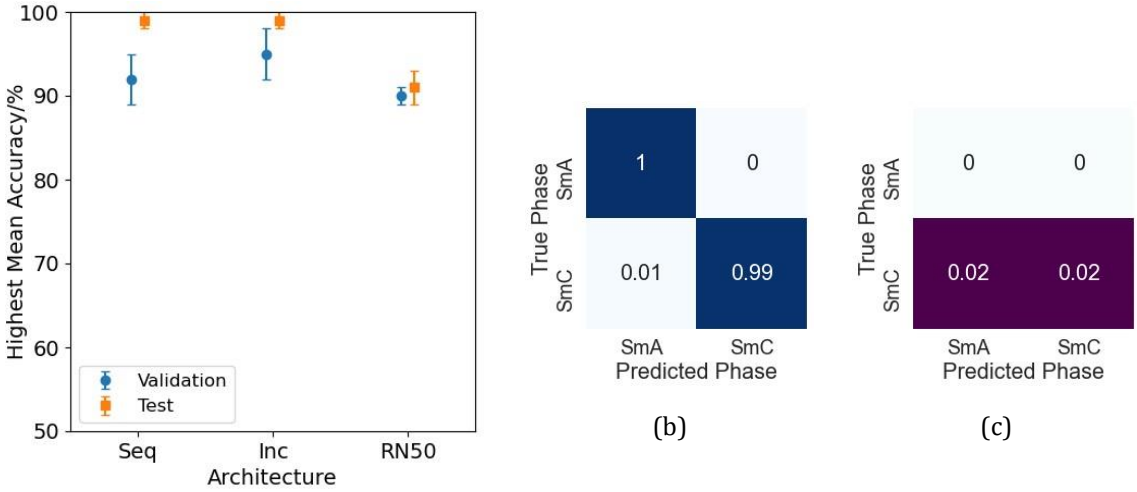
The fluid smectic A and C phases share rather similar textural features because the difference in order between the phases is subtle [5], smectic C being the tilted version of smectic A, separated by a 2nd order phase transition. One could, therefore, infer that the binary classification task on the SmAC dataset should be more challenging. Nevertheless, initial tests suggested that this is not the case, as the best model trained on a similar smectic A and C dataset achieved (97 ± 1)% mean test set accuracy [14]. We now attempt to verify and improve this result with the more robust method of performing ten training runs per model configuration as opposed to just three in the initial tests. The obtained results for the models tuned to the SmAC dataset are presented in Figure 8 and Table 4. Both the Sequential and Inception models achieve an extremely high accuracy on the test set, with ResNet50 trailing behind by approximately 8%. Again, the validation accuracies are lower for all models.

The mean confusion matrix for the Inception model in Figure 8b shows that on all ten training runs, the model is 100% accurate in identifying the smectic A phase in the test set. The only confusion is an average rate of 1% misidentification of smectic C as A, despite the slight imbalance of the dataset with almost two times more smectic C samples than A. The results of the initial tests for smectic A and C have thus been reinforced and improved upon, demonstrating the suitability of CNNs for cases with subtle differences between image features.

Table 4: Best results and corresponding model configuration details for the SmAC dataset

.

SmAC	Sequential	Inception	ResNet50
Mean test accuracy/%	99 ± 1	99 ± 1	91 ± 2
Mean validation accuracy/%	92 ± 3	95 ± 3	90 ± 1
Architecture details	4, 8	2, 2	N/A
Batch size	16	16	16
Learning rate	1×10^{-4}	1×10^{-4}	1×10^{-5}
Trainable parameters	31 k	34 k	25.5 m



(a) (b) (c)

Figure 8: (a) Mean test and validation set accuracies for the best model of each type trained on the SmAC dataset. (b) Mean test set confusion matrix for the SmAC Inception model. (c) Standard deviations of test set confusion matrices for the SmAC Inception model.

3.1.3 SmIF

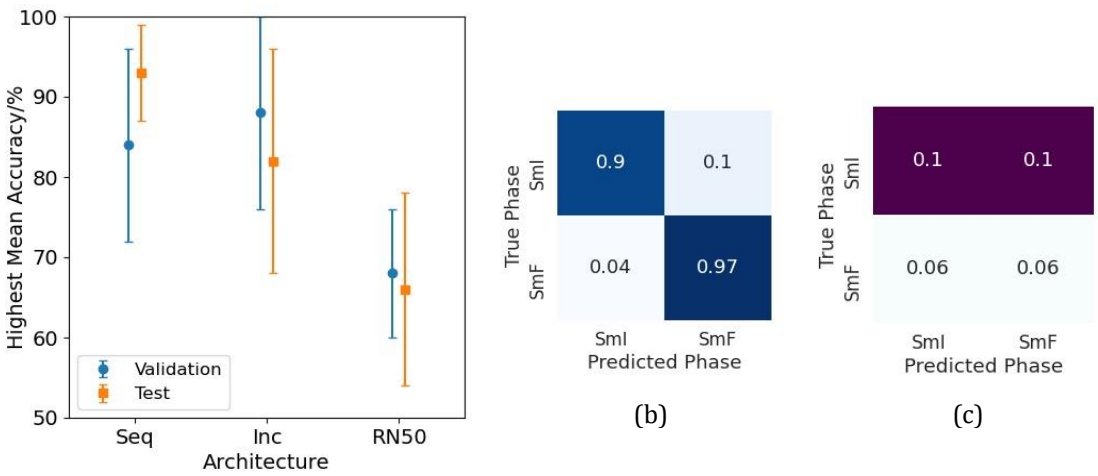
The hexatic smectic I and smectic F binary classification task is another one in which the phases display similar features due to similar underlying structure [5], with both hexatic phases being tilted, only differing in their tilt direction to the apex and the side of the hexagon, respectively. The results and model configurations for the SmIF dataset are displayed in Figure 9 and Table 5. Overall, the model performance is somewhat lower than for the previous datasets, with the highest mean test set accuracy achieved with the Sequential architecture, at (93 ± 6)%. In addition, there are large variances in accuracy for SmIF. This is most likely a result of the relatively small size of the dataset, which could cause instability in training the models as there are not enough samples to consistently learn the correct characteristic features for each phase. The poorer performance and high variance could also suggest that the features distinguishing smectic I and F are more subtle than in, for example, the case of smectic A and C, although by experience this does not seem to be the case. The performance of ResNet50 is particularly low.

This is in part because the training time is longer than for the sequential and inception models, which allows for only fewer tuning attempts.

Figure 9b shows the mean test set confusion matrix for the SmIF Sequential model. From this we see that the largest source of inaccuracy lies in the true smectic I sample predictions, with 90% mean test accuracy as opposed to 97% for smectic F. The two classes are well balanced in this case, which provides no explanation for the inaccuracy. It may instead perhaps stem from

Table 5: Best results and corresponding model configuration details for the SmIF dataset.

SmIF	Sequential	Inception	ResNet50
Mean test accuracy/%	93± 6	82± 14	66± 12
Mean validation accuracy/%	84± 12	88± 12	68± 8
Architecture details	3,128	2, 8	N/A
Batch size	16	16	16
Learning rate	1×10^{-4}	1×10^{-4}	1×10^{-5}
Trainable parameters	1.9 m	538k	25.5 m



(a) (b) (c)

Figure 8: (a) Mean test and validation set accuracies for the best model of each type trained on the SmIF dataset. (b) Mean test set confusion matrix for the SmIF Sequential model. (c) Standard deviations of test set confusion matrices for the SmIF Sequential model.

images in the dataset that have been taken near the point of a phase transition between smectic I and F, which could result in some overlap of important features from each phase in the image. Therefore, the model could be exhibiting a bias towards smectic F if its confidence identifying smectic F features is greater than for smectic I. The standard deviation confusion matrix in Figure 9c shows that true phase smectic I accuracy is also more unstable than smectic F, which provides further evidence that the model is less confident in extracting the features of smectic I.

Nevertheless, the performance on the SmIF dataset is surely satisfactory in that the models perform substantially better than randomly guessing. However, there could be room for

improvement by expansion of the dataset with more videos from smectic I and F phases, which would provide more samples for the models to learn a more accurate interpretation of the subtle textural features.

3.2 Multi-phase classifiers

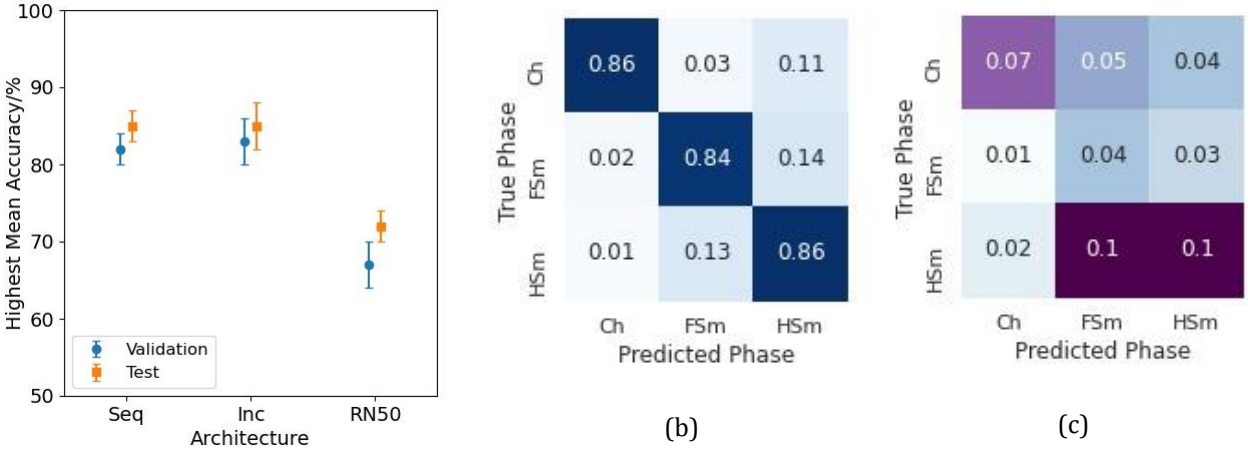
Having demonstrated success in binary LC phase classification, we now progress to the more challenging tasks of classifying more than two phases at a time.

3.2.1 ChSm2

The ChSm2 dataset contains all phases in the complete dataset, (i) cholesteric, and (ii) with smectic A and C grouped into the fluid smectic class and (iii) smectic I and F grouped into the hexatic smectic class. (The hexatic smectic class contains more images than the entire SmIF dataset because images from one video displaying the hexatic smectic phase could not be further labelled as smectic I or F). Figure 10 and Table 6 summarise the results and model configurations for this dataset. We obtain a similar performance from the Sequential and Inception models, with both reaching 85% mean test accuracy. ResNet50 lags behind again on this task, most likely for similar reasons of overfitting and fewer attempts at tuning, as discussed above. With no standard deviations greater than 3%, model training and accuracy is relatively stable for the ChSm2 task. The overall performance is lower in comparison to the binary datasets, which can be explained by the increased complexity and size of the task with a greater number of phases and, subsequently, more features to learn.

Table 6: Best results and corresponding model details for the ChSm2 dataset.

ChSm2	**Sequential**	**Inception**	**ResNet50**
Mean test accuracy/%	85 ± 2	85 ± 3	72 ± 2
Mean validation accuracy/%	82 ± 2	83 ± 3	67 ± 3
Architecture details	3, 64	1, 8	N/A
Batch size	16	16	16
Learning rate	5×10^{-5}	1×10^{-4}	1×10^{-4}
Trainable parameters	470 k	125 k	25.5 m



(a) (b) (c)

Figure 10: (a) Mean test and validation set accuracies for the best model of each type trained on the ChSm2 dataset. (b) Mean test set confusion matrix for the ChSm2 Sequential model. (c) Standard deviations of test set confusion matrices for the ChSm2 Sequential model.

Displayed in Figure 10b is the mean test set confusion matrix for the Sequential model. Surprisingly, the true cholesteric samples are most often mislabelled as hexatic smectic, at 11%. This is counter-intuitive because the hextic smectic phase has a higher order than even the fluid smetic, which would normally result in greater similarity between the cholesteric and fluid smectic phases rather than between cholesteric and hexatic smectic. There is also generally high confusion between the fluid and hexatic smectic phases, and relatively high instability in accuracy for true hexatic smectic samples, when interpreting Figure 10c. These results can be explained by the similarity of texture features over all fluid and hexatic smectic phases (fan-shaped textures only), with more data and possibly a more rigorous tuning required to increase accuracy further. Nevertheless, phases can be predicted with an accuracy of approximately 85%, which is a decent result when considering the size of the relatively small dataset.

3.2.2 ChSm4

At last, the final dataset, ChSm4, contains five classes representing all individual phases of the complete dataset, cholesteric as well as smectic A, C, I and F. This is the most ambitious and challenging of all the phase classification tasks investigated in this study, requiring the models to learn numerous subtle features to distinguish between a closely related series of LC phases. The results from extensive tuning are presented in Figure 11 and Table 7. We again obtain good performance on the test set from the Sequential and Inception models, at 87% and 86% mean accuracy respectively, with ResNet50 performing worse by approximately 20%. Of note is the particularly poor mean validation accuracy for all model types, with none reaching 70%. This is again indicative that the final result is sensitive to the specific choice of videos placed in each sub-dataset, likely due to the small overall size of the complete dataset.

Table 7: Best results and corresponding model configuration details for the ChSm4 dataset.

ChSm4	Sequential	Inception	ResNet50
Mean test accuracy/%	87 ± 3	86 ± 4	66 ± 7
Mean validation accuracy/%	65 ± 2	68 ± 5	60 ± 5
Architecture details	3, 128	2, 4	N/A
Batch size	16	16	16
Learning rate	1×10^{-5}	1×10^{-4}	5×10^{-4}
Trainable parameters	1.9 m	135 k	25.5 m

Upon constructing the mean test set confusion matrix for the Sequential model, Figure 11b depicts the main source of inaccuracy. The true smectic I samples are mislabelled as smectic C 55% of the time on average, with a large standard deviation of 19% as seen in Figure 11c. This suggests that there may be overlap in the model's learned features between the smectic C and I phases. We presume that this overlapping feature is in fact the helical superstructure exhibited by both phases, due to the chirality of the system. This is manifested in the textures of chiral smectic C as an equidistant line pattern, which is largely retained in chiral smectic I, but later partially vanishes for chiral smectic F. The model is largely successful in identifying all other phases, each correctly predicted approximately 90% of the time with smectic A correct 100% of the time. The accuracy on true smectic F samples has a high standard deviation of 15%, similarly to smectic I. These results combined provide insight into the inaccuracies of the ChSm2 task, which likely also suffers from confusion between smectic C and I samples.

The success of the models applied to the ChSm4 dataset is notable. However, training on a vastly expanded dataset would allow model accuracy and stability to be improved. This would require longer training times because larger model capacities would be required to extract more meaningful features from the data. A more balanced dataset with better representation for the smectic A, I and F phases would possibly also be beneficial.

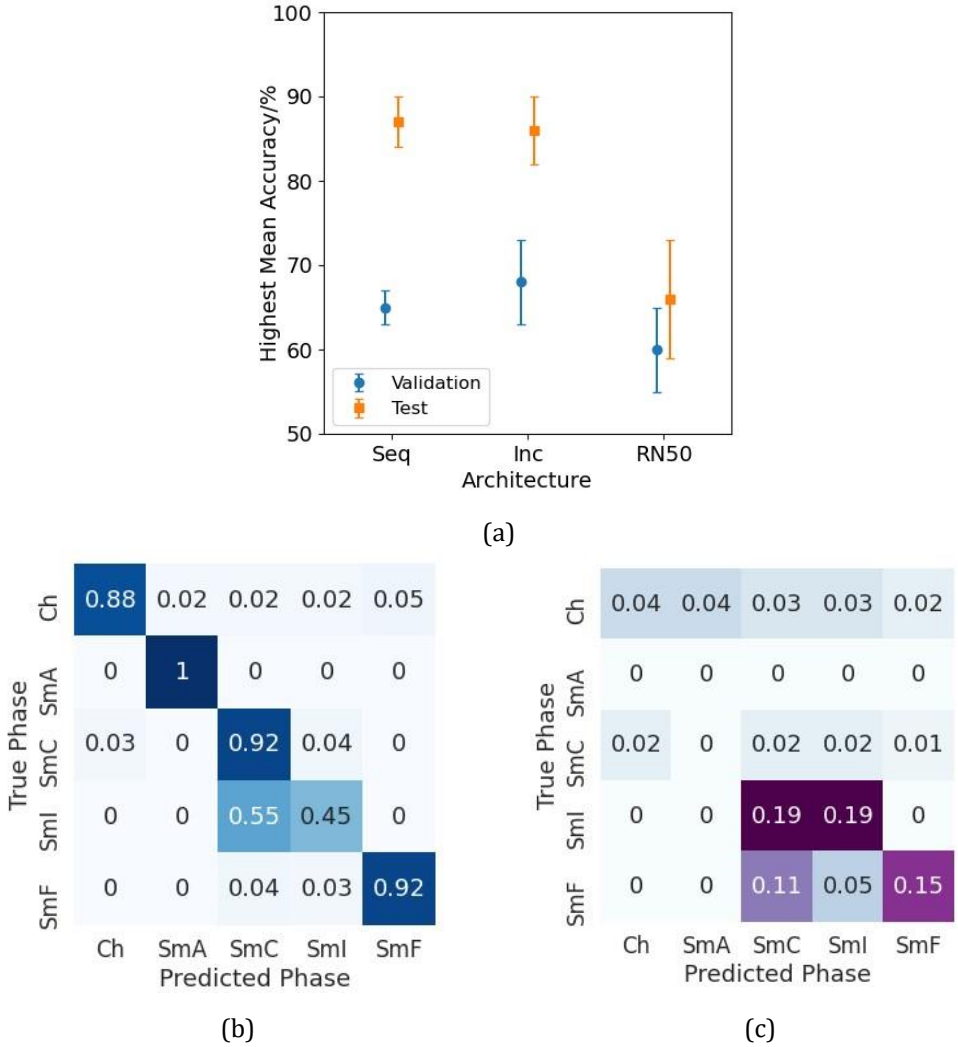


Figure 11: (a) Mean test and validation set accuracies for the best model of each type trained on the ChSm4 dataset. (b) Mean test set confusion matrix for the ChSm4 Sequential model. (c) Standard deviations of test set confusion matrices for the ChSm4 Sequential model.

4 Conclusions

In this study we have investigated various datasets of textures of liquid crystals from binary ((i) cholesteric – fluid smectic; (ii) smectic A – smectic C; (iii) smectic I – smectic F) to multi-phase systems ((iv) cholesteric – fluid smectic – hexatic smectic and (v) cholesteric – smectic A – smectic C – smectic I – smectic F) involving up to five different phases. We have trained and tuned different machine learning models, Sequential, Inception and ResNet50, to a high average test set accuracy. A summary of the best results for each phase grouping is presented in Table 8. Binary systems displayed an average test set accuracy of about 95% and higher, while multi-phase systems were accurate to approximately 85%. Since these results were obtained on real liquid crystalline systems with experimentally determined textures, such accuracies are quite remarkable for the relatively small and at times unbalanced test sets. Especially in the light that all studies up to date were carried out with respect to the quite trivial distinction between isotropic and nematic

phases, and often even on idealised computer-generated textures, which explains their high prediction accuracies as compared to this study of experimentally determined textures of largely different smectic phases.

Table 8: Results for the best tuned model for each prepared dataset. Binary classifiers (i) ChSm: cholesteric – fluid smectic; (ii) SmAC: smectic A – smectic C; (iii) SmIF: smectic I – smectic F. Multi phase classifiers (iv) ChSm2: cholesteric – fluid smectic – hexatic smectic and (v) ChSm4: cholesteric – smectic A – smectic C – smectic I – smectic F.

Dataset	**No. of phases**	**Best model type**	**Mean test accuracy/%**
ChSm	2	Inception	98 ± 2
SmAC	2	Inception	99 ± 1
SmIF	2	Sequential	93 ± 6
ChSm2	3	Sequential	85 ± 2
ChSm4	5	Sequential	87 ± 3

Overall the Sequential models have proved most successful, achieving the highest mean test accuracies in the three most complex classification tasks. The inception models follow closely behind with comparable results in all cases. ResNet50 has performed the worst on all datasets. However, it did produce mean test accuracies above 90% on the ChSm and SmAC datasets. The generally poor performance suggests the capacity of the model is too high and overfitting to the training set has occurred. Furthermore, the size of ResNet50 results in training times much longer than for the other models[14]. For this reason, fewer hyperparameter tuning cycles have been performed.

Throughout the investigation, dataset size has been the key limiting factor. The results could potentially be improved by further balancing and expanding the dataset with more samples. Particularly, including PM videos from an extended range of compounds would be beneficial in allowing the models to learn more general textural features from each phase, improving accuracy when applied to new unseen samples. An ambitious goal for potential future investigations would be to establish a vastly expanded dataset spanning more phases, such as frustrated phases (Blue Phases, BP, and twist grain boundary phases, TGB), or even soft crystals. Larger models such as ResNet50 would be necessary to obtain high accuracies on more complex classification tasks. Alternatives to CNNs could be investigated, such as the promising Transformer network architectures which have seen success in computer vision tasks involving extremely large datasets [35].

References

[1] K. Murphy, *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[3] A. Shrestha and A. Mahmood, “Review of deep learning algorithms and architectures,” *IEEE Access*, vol. 7, pp. 53040–53065, 2019.

[4] A. Voulodimos, N. Doulamis, A. Doulamis, E. Protopapadakis, and D. Andina, “Deep learning for computer vision: A brief review,” *Intell. Neuroscience*, 2018.

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

[6] H. Y. D. Sigaki *et al.*, “Learning physical properties of liquid crystals with deep convolutional neural networks,” *Scientific Reports*, vol. 10, p. 7664, 2020.

[7] H. Y. D. Sigaki, R. F. de Souza, R. T. de Souza, R. S. Zola, and H. V. Ribeiro, “Estimating physical properties from liquid crystal textures via machine learning and complexity-entropy methods,” *Phys. Rev. E*, vol. 99, p. 013311, 2019.

[8] E. N. Minor *et al.*, “End-to-end machine learning for experimental physics: using simulated data to train a neural network for object detection in video microscopy,” *Soft Matter*, vol. 16, p. 1751, 2020.

[9] M. Walters, Q. Wei, and J. Z. Y. Chen, “Machine learning topological defects of confined liquid crystals in two dimensions,” *Phys. Rev. E*, vol. 99, p. 062701, 2019.

[10] F. Leon, S. Curteanu, C. Ta, L. Lin, and N. Hurduc, “Machine learning methods used to predict the liquid-crystalline behavior of some copolyethers,” *Molecular Crystals and Liquid Crystals*, vol. 469, 2007.

[11] C. Butnariu, C. Lisa, F. Leon, and S. Curteanu, “Prediction of liquid-crystalline property using support vector machine classification,” *Journal of Chemometrics*, vol. 27, no. 7-8, pp. 179–188, 2013.

[12] H. Doi, K. Takahashi, K. Tagashira, J. Fukuda, and T. Aoyagi, “Machine learning-aided analysis for complex local structure of liquid crystal polymers,” *Scientific Reports*, vol. 9, 2019.

[13] T. Inokuchi, R. Okamoto, and N. Arai, “Predicting molecular ordering in a binary liquid crystal using machine learning,” *Liquid Crystals*, vol. 47, no. 3, pp. 438–448, 2020.

[14] I. Dierking, J. Dominguez, J. Harbon and J. Heaton, to be published

[15] S. Haykin, *Neural Networks: A Comprehensive Foundation*. 2 ed., 1998.

[16] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, pp. 359–366, 1989.

[17] H. H. Aghdam and E. J. Heravi, *Guide to Convolutional Neural Networks: A Practical Application to Traffic-Sign Detection and Classification*. Springer Publishing Company, Incorporated, 1st ed., 2017.

[18] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” *Proc. Mach. Learn. Res*, vol. 15, pp. 315–323, 2011.

[19] D. Kline and V. Berardi, "Revisiting squared-error and cross-entropy functions for training neural network classifiers," *Neur. Comp. App.*, vol. 14, pp. 310–318, 2005.

[20] S. Amari, "Backpropagation and stochastic gradient descent method," *Neurocomputing*, vol. 5, pp. 185–196, 1993.

[21] C. Bishop, "Regularization and complexity control in feed-forward networks," in *Proceedings International Conference on Artificial Neural Networks ICANN'95*, vol. 1, pp. 141–148, EC2 et Cie, 1995.

[22] N. Srivastava *et al.*, "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, 2014.

[23] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning*, vol. 37, pp. 448—-456, JMLR.org, 2015.

[24] I. Dierking, F. Gießelmann, J. Kußerow, and P. Zugenmaier, "Properties of higher-ordered ferroelectric liquid crystal phases of a homologous series," *Liquid Crystals*, vol. 17, pp. 243–261, 1994.

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

[26] VideoLan, "Vlc media player." https://www.videolan.org/vlc/index.html, 2006.

[27] S. Kaufman *et al.*, "Leakage in data mining: Formulation, detection, and avoidance," *ACM Trans. Knowl. Discov. Data*, vol. 6, pp. 556–563, 2012.

[28] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[30] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th* {*USENIX*} *Symposium on Operating Systems Design and Implementation (*{*OSDI*} *16)*, pp. 265–283, 2016.

[31] A. Gulli and S. Pal, *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[32] S. Cook, *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. Morgan Kaufmann Publishers Inc., 1st ed., 2012.

[33] E. Bisong, *Google Colaboratory*, pp. 59–64. 2019.

[34] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 2014.

[35] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. Khan, and M. Shah, “Transformers in vision: A survey,” *ArXiv*, 2021.