Multi-phase classification of liquid crystal textures using convolutional neural networks

Joshua Heaton
10133722

Department of Physics and Astronomy, The University of Manchester

MPhys project report

Project performed in collaboration with James Harbon
Supervisor: Dr Ingo Dierking

May 6, 2021

Abstract

Contents

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 (DL) 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, DL 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 individual 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 is used to capture images of the textures produced by LC phases for identification by eye [5]. Literature on machine learning 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].

In this project, we prepare a novel dataset of LC texture images captured by polarised microscopy (PM), spanning multiple phases of all orders. Subsequently, we apply CNN classifier models to various phase groupings, probing the limits of attainable model accuracy. The work expands on and consolidates that of the first semester report [14], in which we demonstrated the viability of CNNs in some simple LC phase classification tasks. This report will provide a brief overview of LC phases, supervised ML, and neural networks, with further detail found in the first report [14]. Details of the models used will then be provided, followed by a presentation of the results when they are applied to each of the prepared datasets. Summary conclusions and the limitations of the study will then be discussed.

2 Background principles

2.1 Liquid crystals

Liquid crystal phases are characterised by the positional and orientational order of the molecular arrangement. In general, the phase of lyotropic LCs depends on the concentration of the sample in a solvent whilst thermotropic LCs, studied in this project, 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 perpendicularly aligned polarisers. Polarised light incident on the set-up is altered in accordance with the current phase of the LC sample, producing characteristic features in the

resulting image [5].

At sufficiently high temperatures, thermotropic LCs take the form of a fully anisotropic liquid with no structural order and hence birefringence, resulting in completely dark PM textures. Upon cooling, they will display at least one higher ordered phase before reaching the fully crystalline stage. Of lowest order, just orientational, is the nematic (N) phase, in which the molecules are aligned along a particular axis, called the director, and are still free to move around as in a liquid. Compounds with chiral molecules may instead display the cholesteric (Ch) phase, which is the same as the nematic phase except with helical variation of the director. 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 (FSm) phases have no positional order within molecular 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 structures generating positional order within the layers. The soft crystal phases differ in that the layers show long-range positional order [5].

This project uses CNNs to classify PM textures from thermotropic chiral LC compounds, including the phases Ch, SmA, SmC, SmI, and SmF.

2.2 Supervised machine learning

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

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

$\hat{\boldsymbol{y}}$ 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 that the input data sample belongs to [1]. A supervised model attempts to learn appropriate $\boldsymbol{\theta}$ values for the mapping using a set of training data, consisting of pairs, i, of input examples and their corresponding true labels, $\{\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\}$. The model takes sample $\boldsymbol{x}^{(i)}$ and produces an output label prediction $\hat{\boldsymbol{y}}^{(i)}$ according to Equation 1 [1]. A chosen cost function $J(\boldsymbol{y}, \hat{\boldsymbol{y}}; \boldsymbol{\theta})$ 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 as such 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, of great importance is its capacity, which can be thought of as the size of the model. A low capacity model with few trainable parameters may not be able to extract or infer enough features from the training data to form good predictions, known as underfitting. On the other hand, too high a capacity will cause the model to learn meaningless or overly-fine features from the training data and 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 over or under fit. 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. As an example, for classification tasks a common metric is the percentage of samples that the model assigned to the correct class [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].

2.3 Neural networks

2.3.1 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, allowing it to perform highly complex inference on input data [16]. A dense layer can hence be represented in matrix form as

$$\boldsymbol{O} = A(\boldsymbol{W}\boldsymbol{I} + \boldsymbol{B}), \tag{2}$$

where $\boldsymbol{O}$ is the vector containing the outputs for the layer, $\boldsymbol{W}$ is the matrix of layer weights, $\boldsymbol{I}$ is the vector of layer inputs, $\boldsymbol{B}$ 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. The number of channels 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].

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 work in an analogous way to convolutional layers, however, the kernel has no trainable parameters to convolve 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 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 as, for unit output z,

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

which is inspired by 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 greatest assigned probability [2].

2.3.2 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, $\boldsymbol{g} = \boldsymbol{\nabla}_\theta J(\boldsymbol{y}, \hat{\boldsymbol{y}}; \boldsymbol{\theta})$. 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

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \boldsymbol{g}, \tag{6}$$

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 improve 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].

2.3.3 Regularisation methods

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

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 well, 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 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 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].

3 CNN Models

In this project 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.

3.1 Sequential

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 number of channels in the previous convolutional layer, and second with half this. 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.

3.2 Inception

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 [24]. The structure of an inception block is detailed in Figure 1. Our simplified 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. Similarly 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.

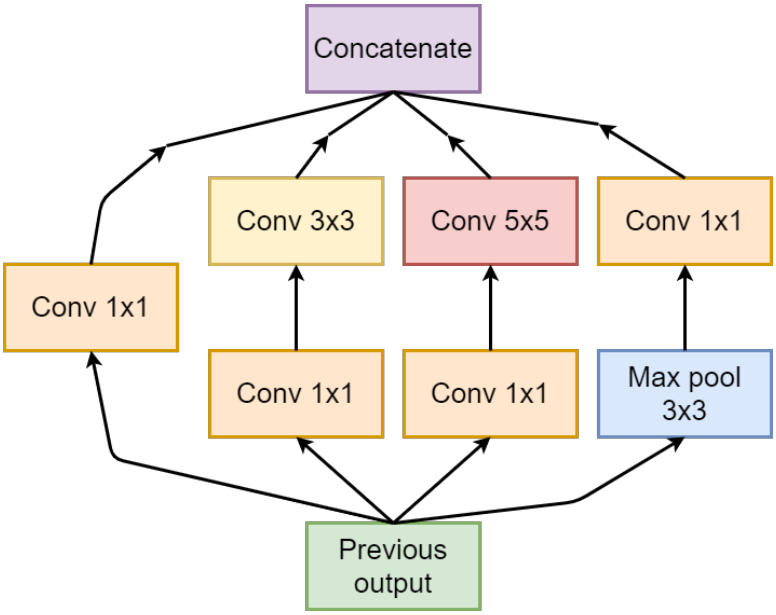

Figure 1: Diagram of a single inception block, adapted from [24]. "Conv 1x1" represents a convolutional layer with kernel size 1×1 and "max pool 3x3" represents max pooling with pool size and stride of 3×3. The branching architecture with varying kernel sizes attempts to extract features of varying sizes from the input.

3.3 ResNet50

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 [25]. 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 project is called ResNet50, owing to it having a total of 50 layers [25]. A diagram of the architecture is presented in Figure 2. This is an extremely high capacity model owing to the number of layers and channels within each layer, amounting to more than 23 million trainable parameters [25].

4 Methodology

4.1 Data preparation

All LC texture image data used has been obtained from polarised microscopy videos of LCs, labelled by compound and temperature range. If not provided, the phases displayed in the videos are identified using project supervisor Ingo Dierking's co-authored papers on homologous LC series [26, 27]. The software VLC Media Player [28] 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 on 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.

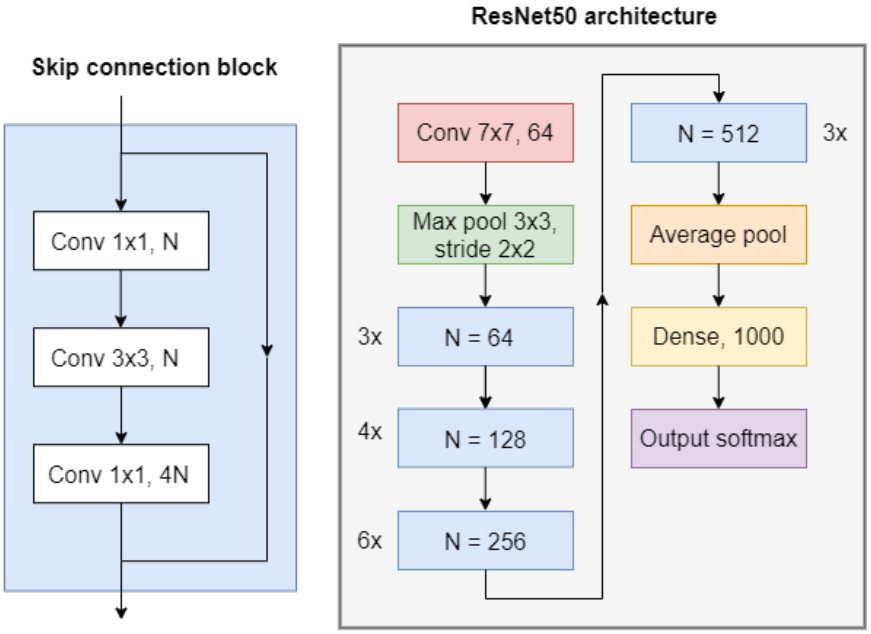

Figure 2: The architecture of ResNet50, adapted from [25]. "Conv 1x1, N" is a convolutional layer with kernel size 1×1 and N channels. All convolutional layers have stride of 1×1. The input to a skip connection block is put through three convolutional layers, and the output of this is concatenated with the original input. The factors next to each skip connection block represent a series of that many blocks.

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 when samples in the training set are highly similar to samples in the validation or test sets, artificially inflating perceived model accuracy [29].

The distribution of the complete dataset over all LC phases is presented in Figure . From this we construct five individual model training datasets, split by video in an approximate ratio of 3:1:1 training to validation to test set count. The five phase groupings includes three binary, or 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 Appendix .

Table 1: The LC phases contained in each dataset.

Dataset	ChSm	AC	IF	ChFluHex	ChACIF
Phases	Ch, Sm	SmA, SmC	SmI, SmF	Ch, FSm, HSm	Ch, SmA, SmC, SmI, SmF

4.2 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 are updated with the Adam

optimiser, detailed in the first semester report, 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 30 epochs and based on validation set cost. The final saved model parameters correspond to the epoch of training with the lowest validation set cost. Based on investigations of the previous semester, random flipping in both the horizontal and vertical directions are 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.

4.3 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 ten models, recording the validation and test set accuracies at the end of 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].

5 Summary of previous work

The work of the first semester focussed on applying sequential, Inception and ResNet models to three different LC phase classification tasks. These included a four-phase set with isotropic, nematic, cholesteric, and smectic, a binary set with smectic A and C, a general smectic set including fluid smectic, hexatic smectic, and soft crystal, and a six-phase smectic set including smectic A, C, I, F, and two soft crystal phases. The final models were trained three times each and the mean accuracy over all three runs calculated for the test sets. The error was calculated as half the range in accuracy. The results are summarised in Table 2. It was

Table 2: First semester mean test set percentage accuracies for final models and phase classification tasks.

Dataset	**No. of phases**	**Best model**	**Test accuracy/%**
Four-phase	4	Sequential 2 layers	91 ± 4
Smectic A and C	2	Sequential 4 layers	97 ± 1
General smectic	3	ResNet50	92 ± 4
Six-phase smectic	6	ResNet50	54 ± 1

concluded that the poor performance in the more complex six-phase smectic task was a result of limited dataset size [14].

6 Classification tasks and results

Here, the results for the models of each type achieving the highest test set accuracies for each phase classification task are presented. 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.

6.1 Binary classifiers

6.1.1 ChSm

6.1.2 AC

6.1.3 IF

6.2 Multi-phase classifiers

6.2.1 ChFluHex

6.2.2 ChACIF

7 Conclusions

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*, vol. 2018, 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] J. Heaton, "Classification of liquid crystal textures using machine learning," 2020.

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

[25] 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.

[26] 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, pp. 243–261, 1994.

[27] J. Schacht, I. Dierking, F. Gießelmann, K. Mohr, H. Zaschke, W. Kuczyński, 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.

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

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

[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.

Appendices