MPhys Lab Book Semester 1

James Harbon

Project Partner: Joshua Heaton

**Classification of Liquid Crystal Phases via Machine Learning**

The aim of the project is to create machine learning models which can correctly classify images of liquid crystal (LC) phases.

**Theory**

The phases of LCs (mesophases) are characterised by the type of ordering they possess. Positional order (presence of an ordered lattice structure) and orientational order (how much molecules generally point in the same direction) are both used to distinguish between phases. Additionally, order can be short-range (between molecules in close proximity to each other) or long-range (order on large, sometimes macroscopic, dimensions).

For the project, we focus on only thermotropic LCs. The phases for these LCs (thermotropic phases) occur in a certain temperature range. If the temperature rises too high, the random thermal motion of the molecules will destroy the ordering of the given LC phase. At a high enough temperature, most thermotropic LCs will have an isotropic phase. This phase is observed to possess fluid-like flow behaviour, an isotropic ordering (little or no long-range order) of position and orientation and also random motion of the molecules.

The nematic phase is commonly observed for LCs. In this phase, the rod-shaped organic molecules only exhibit orientational order, where they align their long axes along a certain direction. This order can be observed over long-range distances between molecules. Most nematic phases are uniaxial, meaning that there is a single long, preferred axis to align with. However, biaxial nematic phases have been observed, where the molecules align with an additional, secondary axis. Since there is no positional order, the centre of mass positions of the molecules are randomly distributed in the same manner as a liquid. Furthermore, the fluidity of the molecules is similar to that in the isotropic phase.

Smectic phases possess higher ordering than nematics, and thus occur at lower temperatures where the thermal motion can’t destroy the structure. Smectics are typically characterised by both the partial positional ordering of molecules into layers and the presence of orientational ordering. Smectic A, for example, consists of molecules arranged into layers, where the long axes of the molecules in a layer are aligned with the normal axis of that layer. Another phase, smectic C has the same positional ordering, however the orientation of the molecules in the layers is tilted slightly away from the normal axis of the layer. Many different smectic phases exist, where different types and degrees of positional and orientational ordering occur.

The chiral nematic phase possesses the property of chirality. The phase is often referred to as the cholesteric phase, due to its existence being first discovered in cholesterol derivatives. Over a specific distance, named the pitch length, the orientation of the molecules moves through a full 360 degree rotation.

Finally, at low enough temperatures, most LCs will form a conventional crystal. This phase is simply named crystalline.

Diagram

Description automatically generated

Figure - From (a) to (d): Nematic, smectic A, smectic C and cholesteric.

Diagram

Description automatically generated

Figure - An illustration of the phases an LC might move through. In general, different compounds will move through different phases as they are heated.

**Week 1 (6/10/20 - 13/10/20)**

We spent the first week reading literature on the use of machine learning for liquid crystal phase classification. We found a very limited amount of literature on the topic, which suggests that this could be a relatively new problem.

We read a paper titled “Learning physical properties of liquid crystals with deep convolutional neural networks” (<https://arxiv.org/pdf/2004.01691.pdf>). One of the problems they attempted to solve was the classification of liquid crystal phase images. They used a simple architecture and only classified isotropic and nematic images. Their model achieved almost perfect accuracy. The reason for this was most likely because their model for this problem was trained only on textures generated from monte carlo simulations. This means that their training data doesn’t vary enough for the model to generalise and the data won’t necessarily reflect real images of LC phases. Also, it is generally quite easy, even for a human eye, to distinguish between the two phases since isotropic images are typically very dark with spots of lighter shades distributed throughout (salt and pepper noise), whereas nematic images have some resemblance of a pattern. We will instead train our model on images from more than two classes. Furthermore, we will only use experimental data for our image set.

We found some YouTube videos involving LC phase experiments.

<https://www.youtube.com/channel/UCqb11FvjkkpD0V5h6p9c2Cg/videos>

<https://www.youtube.com/watch?v=c4FuNSUHAPU>

We also found images and videos which belong to a researcher named Professor Vance Williams. His uploaded data can be found on his YouTube channel and Instagram page.

<http://www.sfu.ca/chemistry/people/profiles/vancew/>

<https://www.youtube.com/channel/UCB8qnCxJbdsuXpQ5RbLNy3Q>

<https://www.instagram.com/accounts/login/?next=/vance.williams/>

VLC Player software’s scene filter function was used to extract frames from the videos and then save them as images. Any files already in the format of a raw image were simply downloaded and saved in our dataset.

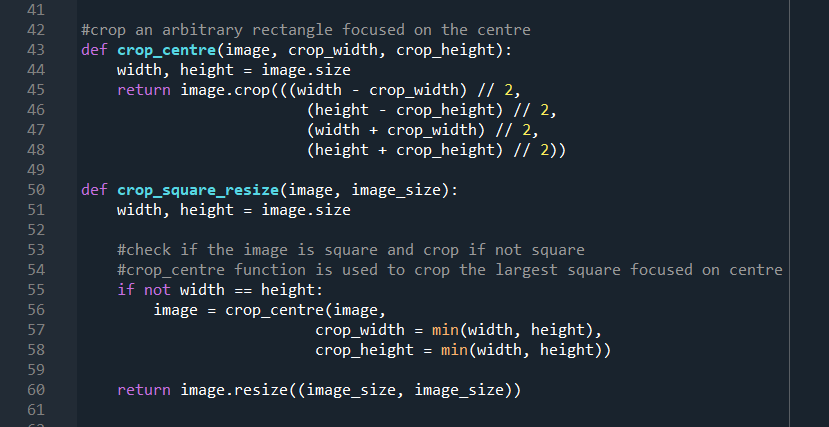
**Week 2 (13/10/20 - 20/10/20)**

**2.1 Creation of Data Transformation Scripts**

The second week was mostly spent creating scripts in Python to transform and pre-process our image data. I created a script called image\_transform\_split, which was designed to handle all of the image preprocessing that was required and could also split the data. The script is comprised of multiple functions which are integrated into bigger functions for different tasks. I first created three functions named load\_images, save\_images and make\_classes. These functions took advantage of the python standard library, os. The first two functions are self-explanatory, and the third function was used to create new folders to save the split dataset into.

A function named crop\_centre was created to crop a rectangle, with its centre aligned with that of the image, from the image. The Python Imaging Library (PIL) was used to implement the cropping since an image can be loaded as an Image (a class within PIL) object, which has a crop method defined. The dimensions and corner positions of the cropped image can be adjusted by specifying the coordinates of its vertices relative to that of the image. The parameters used by the function include an image (Image object), crop\_width and crop\_height.

Another function, crop\_square\_resize, was created to use crop\_centre to crop the largest square, centred on the middle, out of the image. The parameters are an image and an image\_size. The image\_size parameter determines the square dimensions you want to resize the cropped square image to. The width and height of the original image are first read by using the image.size attribute of the Image object. The function uses an if statement to check if the width is equal to the height (a square image). If not, then it crops the square out and resizes it. If it is already square, then it simply resizes it.



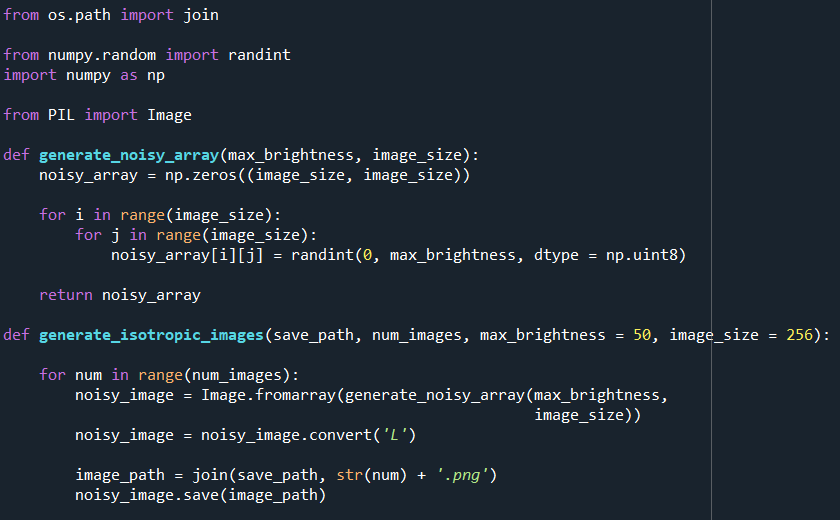
Finally, a function named transform\_split was created to integrate the crop\_square\_resize function with some code which can split the dataset into three classes: train, valid and test.

**2.2 Isotropic Image Generator**

A script named isotropic\_image\_generator was created to simulate and then save images of the isotropic LC phase. An isotropic phase image, created from polarized optical microscopy, simply looks like a noisy, black and white image. Due to this, it is a waste of time trying to collect or create labelled images of an isotropic phase. We can simply simulate the images using what is called ‘salt and pepper noise’.

A square numpy array (matrix) is initially created with dimensions (image\_size, image\_size), where image\_size is a parameter defined upon calling the function. The array is created with every entry equal to zero. A loop within a loop was then used to modify the value of each entry, or effectively each pixel, in the array. The value for each entry determines the brightness of each pixel of the image, and in general can range between 0 (black) and 255 (white). The values are selected randomly from a range of (0, max\_brightness), where max\_brightness is a parameter the function accepts. The max\_brightness was set by default as 50 since isotropic images are generally quite dark, and we want the image to be an accurate simulation of an actual isotropic phase. The function created to implement this technique was named generate\_noisy\_array.

After the creation of a function to generate “salt and pepper noise” images, the next step was to create a function which could repeat this process a predetermined number of times and then save all the generated images in a folder. The function accepts parameters save\_path (where you want to save the images), num\_images (how many images you want to generate), max\_brightness (set by default to 50 as explained previously) and image\_size (pixel dimensions of the generated images). An isotropic image is created using generate\_noisy\_array and a function from the PIL Image module called fromarray, which can convert a numpy array to an Image object. A method from the Image class named convert is used, with string “L” passed to the function as an argument, to ensure that the image has a single channel.



**2.3 First Dataset**

Along with the data that was found from YouTube and Vance Williams, we received some videos from our supervisor Ingo Dierking showing a nematic phase. We again used VLC Player to extract frames from these videos. All of the data collected up to this point formed our first dataset.

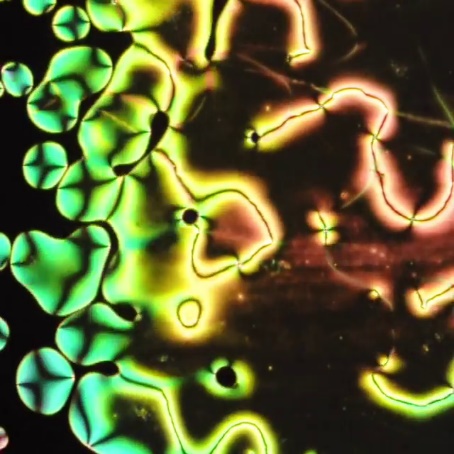
Below is an example raw image from each class.

Isotropic

A picture containing night sky

Description automatically generated

Nematic



Cholesteric

A picture containing text

Description automatically generated

Smectic

A picture containing colorful, dancer, decorated, several

Description automatically generated

The generate\_isotropic\_images function was first used to create 500 isotropic images. Then the transform\_split function was used to split the image classes into train, valid and test folders. An image size of 256x256 pixels was used.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Unedited Total | Train | Valid | Test |
| Isotropic | 500 | 300 | 100 | 100 |
| Nematic | 677 | 407 | 135 | 135 |
| Cholesteric | 531 | 383 | 106 | 106 |
| Smectic | 534 | 322 | 106 | 106 |

The usual procedure of creating an ImageDataGenerator object for each of the training and validation datasets was used. This object allows us to specify the augmentations we wish to apply to a generated batch of images during training.

|  |  |
| --- | --- |
| ImageDataGenerator Parameters | Parameter Values |
| rescale | 1./255 |
| shear\_range | 0.2 |
| zoom\_range | 0.2 |
| horizontal\_flip | True |
| vertical\_flip | True |
| rotation\_range | 30 degrees |

ImageDataGenerator objects have a method defined called flow\_from\_directory.

|  |  |
| --- | --- |
| flow\_from\_directory Parameters | Parameter Values |
| directory | path to the dataset |
| target\_size | (256, 256) |
| batch\_size | 32 |
| class\_mode | ‘categorical’ |
| shuffle | True |
| color\_mode | ‘grayscale’ |

**Week 3 (20/10/20 - 27/10/20)**

**3.1 Training and Callbacks**

A function named “train” was created in a script named “model\_training”. This function was created to be used in various notebooks or other scripts to reduce the amount of repeated code. For the compile method we used Adam as the optimiser, CategoricalCrossEntropy as the loss function and accuracy as the metric.

Within the function, five different Keras Callbacks were defined. Callbacks are used to perform functions during the training of a model. The first Callback is an object of a class we created named “EpochTime”, which uses Python class inheritance to create a class of a class (it is a class of the Callback class). The overall effect of this Callback is to store the time it takes for the current epoch to be completed in a list and then repeat this process for the remaining epochs, appending the times to the same list. At the end of training, the list is saved in a JSON file and the file is saved at a chosen location. This was done in case we wanted to inspect the times it takes for different models to train, since we can find the sum of the lists for different models which will give us the total training time for each model.

The second Callback is an object from the ModelCheckpoint class named “best\_checkpoint” and saves the model, along with the weights it has learned, to a specified location. It only saves the weights which correspond to the lowest validation loss, which is another way of saying the most confident model.

The third Callback is named “early\_stop” and is an object from the EarlyStopping class. It monitors how the validation loss changes from epoch to epoch, and if the loss does not change by more than a specified amount over a specific number of epochs, then the Callback causes the training to end. This can help to prevent the problem of overfitting to training and validation sets. The delta parameter was left at its default value of 1 and the patience, which determines how many epochs the Callback will wait for where the loss hasn’t changed by more than the delta value before stopping the training, was set to 100.

The fourth Callback is called “reduce\_lr”, which is an object from the ReduceLROnPlateau class. It monitors the validation loss, like early\_stop, and if it doesn’t change by more than the delta parameter over a number of epochs specified by the patience parameter, a parameter named “factor” is multiplied by the current learning rate to get the new learning rate. Another parameter named “min\_lr” is used to prevent the learning rate from reducing below a certain value.

The fifth Callback is an object from the CSVLogger class. At the end of the training, it simply creates a “.csv” file and saves it in a predetermined location. The columns in the csv file are: epoch (which epoch number the row corresponds to), accuracy (training accuracy value), loss (training loss value), lr (learning rate), val\_accuracy (validation accuracy value) and val\_loss (validation loss value).

**3.2 Model Evaluation Script**

After a model has been trained on a dataset we need to evaluate its performance on an unseen dataset. During the training it is evaluated on a validation set. However, the validation accuracy and loss influence the model selection and the model can thus be overfitted to a validation set and might not generalise very well.

We created a script which has a function named “generate\_test\_set” to create batches of images from the test set. A function named “test\_accuracy” evaluates the test accuracy and loss on this dataset and can then return either or both.

**3.4 Low and Medium Capacity Models**

Table

Description automatically generatedThe original plan involved the creation of three models named “low\_cap”, ”medium\_cap” and “high\_cap”. The “cap” refers to the capacity of the model, which in turn loosely refers to the total number of parameters of the model. Due to the results we found for the first two models, we decided that there was little reason to create the third model.

The first model is a basic sequential model. The details were illustrated in a graph using the plot\_model function from the Keras library.

The second model is again a sequential model and has a higher number of total parameters than the first model. The details were again illustrated in a graph.

Diagram

Description automatically generated

**3.5 Results for First Models**

Graphical user interface

Description automatically generated

The low\_cap model managed to achieve a peak validation accuracy of 99.89%. The test set accuracy was 100%.

Graphical user interface, chart

Description automatically generated

The medium\_cap model achieved a peak validation set accuracy of 99.95% and the test set accuracy was 100%.

Both of the first models achieved almost perfect accuracy on the validation and test sets. For a dataset with non-linear features and for such small models, this is unlikely to happen. We suspect that it is due to data leakage. The data was split into train, validation and test sets randomly with no regard for if the images looked similar in those datasets, since we only ensured that there were no duplicates from dataset to dataset. We need both more data and a stricter splitting approach for the data to more accurately evaluate how well the models generalise to unseen data.

**Week 4 (27/10/20 - 3/11/20)**

**4.1 New Data From Supervisor**

Our supervisor’s PhD student recorded some videos of LC phase transitions. There were multiple videos for different LC compounds. The compounds were either heated or cooled, and the change in temperature caused a given compound to start transitioning into a different phase. These videos were sent to us and we used VLC Player to save frames from these videos.

The LC compounds included in the videos were:

* 5CB
* 8CB
* M5
* M6
* M7
* M9
* M10

The compounds often did not change much during heating at certain parts of the video. Due to this we lowered the rate at which we saved frames from certain parts of the video rather than simply saving most of the frames.

Since most of the videos contained more than one phase, we had to sort the images into the correct classes. Our supervisor told us that 5CB only has a nematic phase and 8CB has a sequence of smectic A, nematic and then isotropic when heating the compound. For the M-series compounds, we were sent a paper titled “Mesomorphic properties of a homologous series of chiral liquid crystals containing the a-chloroester group”. This paper included a table containing phase transition temperatures for different LC phases in M-series compounds. Since the video file names contained a temperature range and an indication of whether the compound was being either heated or cooled, we could identify separate phases in the videos and sort the images accordingly.

**4.2 Splitting the New Dataset**

The naming of the image files included a letter at the start followed by numbers ranging from 1 to the total number of images from a given video. The letter allowed us to distinguish between images from different videos. As an attempt to reduce data leakage between training and validation/test sets, images from a given video were confined to a single dataset (one of training, validation, or test). This was done for all the different videos.

Our dataset size increased a lot after incorporating all the new images. There were 9,284 images in total.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Unedited Total | Train | Valid | Test |
| Isotropic | 2200 | 1300 | 500 | 400 |
| Nematic | 2262 | 1300 | 491 | 471 |
| Cholesteric | 2100 | 1292 | 410 | 398 |
| Smectic | 2722 | 1300 | 500 | 922 |

**4.3 Multiple Layered Sequential Architectures**

Six different sequential convolutional neural network architectures were created to train on our dataset. The naming convention was “n layers”, where n = (1, 2, 3, 4, 5, 6). A diagram of the “3 layers” network architecture is shown below.

Chart, box and whisker chart

Description automatically generated

Batch normalisation layers were applied to the output of every convolutional layer to improve regularisation. Dropout were used after each dense layer to also achieve this.

**Week 5 (3/11/20 - 10/11/20)**

**5.1 Results for 6 Sequential Models**

We trained our 6 sequential models on the dataset for 4 phase classification.

Below are our initial results for 4 phases and no augmentations.

|  |  |  |
| --- | --- | --- |
|  | Validation Accuracy/% | Test Accuracy/% |
| 1 layer | 67 | 69 |
| 2 layers | 73 | 72 |
| 3 layers | 74 | 75 |
| 4 layers | 85 | 82 |
| 5 layers | 65 | 68 |
| 6 layers | 76 | 74 |

Below are our initial results for 4 phases and flip augmentations.

|  |  |  |
| --- | --- | --- |
|  | Validation Accuracy/% | Test Accuracy/% |
| 1 layer | 72 | 70 |
| 2 layers | 72 | 73 |
| 3 layers | 80 | 81 |
| 4 layers | 88 | 89 |
| 5 layers | 84 | 85 |
| 6 layers | 77 | 78 |

Below are our initial results for 4 phases and all augmentations.

|  |  |  |
| --- | --- | --- |
|  | Validation Accuracy/% | Test Accuracy/% |
| 1 layer | 67 | 66 |
| 2 layers | 74 | 71 |
| 3 layers | 68 | 66 |
| 4 layers | 83 | 81 |
| 5 layers | 81 | 78 |
| 6 layers | 75 | 73 |

Based on the results, we decided that the flip augmentations group created the optimal balance between training time and test accuracy.

**5.2 Repeats**

We wanted to investigate the stability of the models for this dataset by running repeats of the training. Repeats 1, 2 and 3 are listed as R1, R2 and R3.

Below are our repeat results for 4 phases and no augmentations.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 Layer | 2 Layers | 3 Layers | 4 Layers | 5 Layers | 6 Layers | ResNet50 | Inception |
| Validation Acc R1/% | 75 | 77 | 79 | 74 | 78 | 73 | 75 | 79 |
| Validation Acc R2/% | 72 | 71 | 67 | 70 | 70 | 71 | 79 | 76 |
| Validation Acc R3/% | 67 | 73 | 81 | 80 | 81 | 65 | 70 | 80 |
| Validation Acc Mean/% | 71 | 74 | 76 | 75 | 76 | 70 | 75 | 78 |
| Validation Acc Uncertainty/% | 4 | 3 | 7 | 5 | 6 | 4 | 5 | 2 |
| Test Acc R1/% | 81 | 70 | 87 | 88 | 84 | 90 | 86 | 77 |
| Test Acc R2/% | 85 | 91 | 79 | 85 | 75 | 82 | 78 | 86 |
| Test Acc R3/% | 75 | 70 | 88 | 82 | 90 | 70 | 75 | 75 |
| Test Acc Mean/% | 80 | 77 | 85 | 85 | 83 | 81 | 80 | 79 |
| Test Acc Uncertainty/% | 5 | 10 | 5 | 3 | 8 | 10 | 6 | 6 |
| Training Time/minutes | 54 | 56 | 58 | 59 | 60 | 61 | 136 | 65 |

Below are our repeat results for 4 phases and flip augmentations.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 Layer | 2 Layers | 3 Layers | 4 Layers | 5 Layers | 6 Layers | ResNet50 | Inception |
| Validation Acc R1/% | 69 | 77 | 82 | 80 | 87 | 71 | 72 | 75 |
| Validation Acc R2/% | 74 | 74 | 86 | 85 | 83 | 76 | 76 | 70 |
| Validation Acc R3/% | 73 | 78 | 87 | 85 | 93 | 86 | 80 | 70 |
| Validation Acc Mean/% | 72 | 76 | 85 | 83 | 88 | 78 | 76 | 72 |
| Validation Acc Uncertainty/% | 3 | 2 | 3 | 3 | 5 | 8 | 4 | 3 |
| Test Acc R1/% | 84 | 82 | 83 | 84 | 93 | 83 | 81 | 79 |
| Test Acc R2/% | 88 | 80 | 88 | 86 | 88 | 80 | 82 | 80 |
| Test Acc R3/% | 80 | 87 | 95 | 83 | 82 | 87 | 80 | 84 |
| Test Acc Mean/% | 84 | 83 | 89 | 84 | 88 | 83 | 81 | 81 |
| Test Acc Uncertainty/% | 4 | 4 | 6 | 2 | 6 | 4 | 1 | 3 |
| Training Time/minutes | 56 | 57 | 59 | 60 | 61 | 62 | 140 | 66 |

Below are our repeat results for 4 phases and all augmentations.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 Layer | 2 Layers | 3 Layers | 4 Layers | 5 Layers | 6 Layers | ResNet50 | Inception |
| Validation Acc R1/% | 66 | 73 | 80 | 79 | 81 | 74 | 73 | 65 |
| Validation Acc R2/% | 69 | 72 | 76 | 77 | 81 | 66 | 72 | 68 |
| Validation Acc R3/% | 70 | 70 | 79 | 85 | 77 | 86 | 79 | 72 |
| Validation Acc Mean/% | 68 | 72 | 78 | 80 | 80 | 75 | 75 | 68 |
| Validation Acc Uncertainty/% | 2 | 2 | 2 | 4 | 2 | 10 | 4 | 4 |
| Test Acc R1/% | 81 | 89 | 77 | 70 | 83 | 81 | 80 | 77 |
| Test Acc R2/% | 75 | 82 | 80 | 80 | 83 | 77 | 83 | 70 |
| Test Acc R3/% | 86 | 70 | 83 | 70 | 84 | 91 | 78 | 65 |
| Test Acc Mean/% | 81 | 80 | 80 | 73 | 83 | 83 | 80 | 71 |
| Test Acc Uncertainty/% | 6 | 10 | 3 | 5 | 1 | 7 | 3 | 6 |
| Training Time/minutes | 89 | 91 | 94 | 96 | 98 | 99 | 224 | 104 |

The mean test accuracy values for each model and group of augmentations is shown below.

**Chart, line chart

Description automatically generated**

The “3 layers” model achieved the highest mean test accuracy with the flip augmentations applied. The confusion matrix is shown below.

**Chart

Description automatically generated**

**Week 6 (10/11/20 - 17/11/20)**

**6.2 ResNet50 Architecture**

The ResNet50 architecture was used to train on the data. This network employs residual, or skip, connections which can help to propagate gradient information from the final to the initial layer in a network which is very deep. Specific details of the architecture can be found in the paper “Deep Residual Learning for Image Recognition” (<https://arxiv.org/pdf/1512.03385.pdf>).

**6.3 Inception Architecture**

The “Inception V3” architecture was used. Specific details can be found in the paper “Going Deeper with Convolutions” (<https://arxiv.org/pdf/1409.4842.pdf>). We only used the first three inception modules in order to reduce training time.

**Week 7 (17/11/20 - 24/11/20)**

**7.1 Smectic 3 Phase Classifier**

A dataset was created for the classification of three smectic phase groups: fluid, hexatic and soft crystal.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Phase Group | Total | Train | Valid | Test |
| Fluid | 2520 | 1759 | 372 | 389 |
| Hexatic | 666 | 486 | 90 | 90 |
| Soft Crystal | 840 | 600 | 144 | 96 |

**7.2 Results for Smectic 3 Phase**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 Layer | 2 Layers | 3 Layers | 4 Layers | 5 Layers | 6 Layers | ResNet50 | Inception |
| Validation Acc R1/% | 61 | 65 | 61 | 63 | 59 | 65 | 58 | 61 |
| Validation Acc R2/% | 61 | 61 | 59 | 67 | 52 | 63 | 50 | 64 |
| Validation Acc R3/% | 61 | 61 | 61 | 70 | 73 | 80 | 71 | 61 |
| Validation Acc Mean/% | 61 | 62 | 60 | 67 | 61 | 69 | 60 | 62 |
| Validation Acc Uncertainty/% | 0 | 2 | 1 | 4 | 11 | 9 | 11 | 2 |
| Test Acc R1/% | 68 | 90 | 68 | 80 | 68 | 56 | 68 | 60 |
| Test Acc R2/% | 59 | 64 | 68 | 85 | 68 | 65 | 60 | 71 |
| Test Acc R3/% | 68 | 83 | 90 | 92 | 68 | 81 | 68 | 60 |
| Test Acc Mean/% | 65 | 79 | 75 | 86 | 68 | 67 | 65 | 64 |
| Test Acc Uncertainty/% | 5 | 13 | 11 | 6 | 0 | 13 | 4 | 5 |

**Chart, line chart

Description automatically generated**The mean test accuracy values are shown below.

The “4 layers” model achieved the highest mean test accuracy value. The confusion matrix is shown below.

**Chart, treemap chart

Description automatically generated**

**Week 8 (24/11/20 - 1/12/20)**

**8.1 Rebalanced Smectic 3 Phase Classifier**

Created a rebalanced version of the smectic 3 phase dataset to retrain models and observe if this version of the dataset produces better performance.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Phase Group | Total | Train | Valid | Test |
| Fluid | 1585 | 940 | 327 | 318 |
| Hexatic | 666 | 408 | 123 | 135 |
| Soft Crystal | 840 | 498 | 186 | 156 |

**8.2 Results for Smectic 3 Phase Rebalanced**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 Layer | 2 Layers | 3 Layers | 4 Layers | 5 Layers | 6 Layers | ResNet50 | Inception |
| Validation Acc R1/% | 49 | 51 | 51 | 47 | 58 | 35 | 71 | 51 |
| Validation Acc R2/% | 44 | 51 | 51 | 55 | 28 | 38 | 75 | 42 |
| Validation Acc R3/% | 51 | 51 | 51 | 50 | 50 | 52 | 66 | 51 |
| Validation Acc Mean/% | 48 | 51 | 51 | 51 | 45 | 42 | 71 | 48 |
| Validation Acc Uncertainty/% | 4 | 0 | 0 | 4 | 15 | 9 | 5 | 5 |
| Test Acc R1/% | 49 | 43 | 47 | 56 | 52 | 38 | 41 | 51 |
| Test Acc R2/% | 52 | 45 | 52 | 24 | 52 | 34 | 57 | 52 |
| Test Acc R3/% | 52 | 35 | 52 | 39 | 52 | 48 | 52 | 50 |
| Test Acc Mean/% | 51 | 41 | 50 | 40 | 52 | 40 | 50 | 51 |
| Test Acc Uncertainty/% | 2 | 5 | 3 | 16 | 0 | 7 | 8 | 1 |

**Week 9 (1/12/20 - 8/12/20)**

**9.1 Smectic A and C Classifier**

Created a dataset for the smectic A and C classes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Phase | Total | Train | Valid | Test |
| Smectic A | 1097 | 617 | 276 | 204 |
| Smectic C | 1448 | 884 | 282 | 282 |

**9.2 Smectic A and C Results**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 Layer | 2 Layers | 3 Layers | 4 Layers | 5 Layers | 6 Layers | ResNet50 | Inception |
| Validation Acc R1/% | 93 | 98 | 91 | 91 | 96 | 96 | 94 | 93 |
| Validation Acc R2/% | 92 | 98 | 99 | 98 | 100 | 94 | 92 | 94 |
| Validation Acc R3/% | 97 | 99 | 91 | 96 | 99 | 94 | 98 | 92 |
| Validation Acc Mean/% | 94 | 98 | 94 | 95 | 98 | 95 | 95 | 93 |
| Validation Acc Uncertainty/% | 3 | 1 | 4 | 4 | 2 | 1 | 3 | 1 |
| Test Acc R1/% | 91 | 84 | 88 | 90 | 92 | 60 | 95 | 95 |
| Test Acc R2/% | 94 | 91 | 92 | 99 | 90 | 100 | 92 | 90 |
| Test Acc R3/% | 87 | 90 | 88 | 82 | 88 | 85 | 88 | 85 |
| Test Acc Mean/% | 91 | 88 | 89 | 90 | 90 | 82 | 92 | 90 |
| Test Acc Uncertainty/% | 4 | 4 | 2 | 9 | 2 | 20 | 4 | 5 |

The mean test accuracy values are shown below.

Chart, line chart

Description automatically generated

**Chart, treemap chart

Description automatically generated**ResNet50 had the highest mean test accuracy. The confusion matrix is shown below.

**Week 10 (8/12/20 - 15/12/20)**

**10.1 Smectic 6 Phase Classifier**

Created a dataset for six different smectic phases.

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

**10.2 Smectic 6 Phase Results**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 Layer | 2 Layers | 3 Layers | 4 Layers | 5 Layers | 6 Layers | ResNet50 | Inception |
| Validation Acc R1/% | 33 | 33 | 34 | 67 | 50 | 61 | 33 | 52 |
| Validation Acc R2/% | 33 | 33 | 33 | 53 | 38 | 13 | 48 | 49 |
| Validation Acc R3/% | 33 | 33 | 46 | 20 | 34 | 27 | 46 | 55 |
| Validation Acc Mean/% | 33 | 33 | 38 | 47 | 41 | 34 | 42 | 52 |
| Validation Acc Uncertainty/% | 0 | 0 | 7 | 24 | 8 | 24 | 8 | 3 |
| Test Acc R1/% | 36 | 34 | 36 | 56 | 36 | 8 | 53 | 26 |
| Test Acc R2/% | 36 | 28 | 36 | 34 | 36 | 28 | 53 | 30 |
| Test Acc R3/% | 25 | 30 | 13 | 19 | 21 | 21 | 55 | 30 |
| Test Acc Mean/% | 32 | 31 | 28 | 36 | 31 | 19 | 54 | 29 |
| Test Acc Uncertainty/% | 6 | 3 | 12 | 19 | 8 | 10 | 1 | 2 |

The mean test accuracy values are shown in the graph below.

**Chart, line chart

Description automatically generated**

ResNet50 had the highest mean test accuracy. The confusion matrix is shown below.

Graphical user interface, application, Teams

Description automatically generated

**Week 11 (15/12/20 - 22/12/20)**

**11.1 Rebalanced Smectic 6 Phase Classifier**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Phase | Total | Train | Valid | Test |
| Smectic A | 770 | 449 | 165 | 156 |
| Smectic C | 815 | 491 | 162 | 162 |
| Smectic F | 270 | 180 | 45 | 45 |
| Smectic I | 396 | 228 | 78 | 90 |
| Smectic X1 | 420 | 258 | 96 | 66 |
| Smectic X2 | 420 | 240 | 90 | 90 |

**11.2 Results of Smectic 6 Phase Rebalanced**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 Layer | 2 Layers | 3 Layers | 4 Layers | 5 Layers | 6 Layers | ResNet50 | Inception |
| Validation Acc R1/% | 25 | 25 | 25 | 23 | 22 | 39 | 29 | 33 |
| Validation Acc R2/% | 26 | 25 | 25 | 26 | 44 | 38 | 53 | 35 |
| Validation Acc R3/% | 25 | 25 | 25 | 30 | 30 | 37 | 33 | 31 |
| Validation Acc Mean/% | 25 | 25 | 25 | 26 | 32 | 38 | 38 | 33 |
| Validation Acc Uncertainty/% | 0 | 0 | 0 | 4 | 11 | 1 | 12 | 2 |
| Test Acc R1/% | 26 | 23 | 26 | 27 | 21 | 12 | 43 | 33 |
| Test Acc R2/% | 26 | 37 | 27 | 40 | 27 | 22 | 21 | 43 |
| Test Acc R3/% | 26 | 43 | 27 | 40 | 27 | 30 | 38 | 36 |
| Test Acc Mean/% | 26 | 34 | 27 | 36 | 25 | 21 | 34 | 37 |
| Test Acc Uncertainty/% | 0 | 10 | 1 | 7 | 3 | 9 | 11 | 5 |

**Week 12 (22/12/20 - 29/12/20)**

The final week was spent writing the project report.

Next semester we will increase the size of our datasets, especially for the underrepresented phases. We will also investigate the performance of more CNN architectures and different types of networks such as transformer networks.