Deep learning applied to “Particle Picking” in Cryo-EM

Investigating a deep learning approach to classification of Cryo-EM images

Thomas Dybdahl - 20112976
Supervisor: Christian Storm

Master’s Thesis
Department of Computer Science
Science & Technology
Aarhus University
June 15. - 2018
Deep learning applied to “Particle Picking” in Cryo-EM

Investigating a deep learning approach to classification of Cryo-EM images

Thomas Dybdahl

Abstract

Machine learning techniques such as neural networks has long been a powerful tool for data analysis, and in later years it has been successfully integrated to assist in labour intensive analytical procedures. An example of such a procedure is the particle picking problem in Cryo-EM image analysis. A fully automated approach using deep learning has been presented in the paper [Wan16]. This report investigates a sub-problem arising in the automated approach regarding classification of images as containing either a centered particle or no particle. The effectiveness of multiple configurations of 3 types of neural networks (DNN, CNN, and spatial transformer) as well as simpler approaches to solving this problem were investigated on a data set generated from Cryo-EM images of the protein beta-galactosidase. These neural networks were implemented using Python and Tensorflow. The best performance was achieved by incorporating a spatial transformer module in a convolutional neural networks (CNN) achieving above 95% classification accuracy on isolated Cryo-EM sample images. However to fully investigate the effectiveness of the implemented models with regards to the fully automated approach, a larger data set containing other particles than beta-galactosidase, as well as images with non-centered particles is required.
Contents

1 Introduction .................................................. 7
  1.1 Problem introduction .................................... 7
  1.2 Data .......................................................... 9

2 Theory .......................................................... 11
  2.1 Neural Networks ........................................... 11
    2.1.1 Activation Function ................................... 12
    2.1.2 Fully Connected Multilayer Feed Forward Network ... 12
    2.1.3 Training .................................................. 13
    2.1.4 CNN ...................................................... 14
    2.1.5 Spatial Transformer Module ............................. 16

3 Implementation ............................................... 19
  3.1 Simple Solutions Implementation .......................... 20
  3.2 Neural Networks Implementation ........................... 21

4 Experiments .................................................... 25
  4.1 Experimental setup ....................................... 25
  4.2 Simple Pixel Value based .................................... 26
    4.2.1 Average Pixel Value ................................... 26
    4.2.2 Pixel Value Variance ................................... 26
    4.2.3 Number of pixels less than boundary value ............ 27
    4.2.4 Number of pixels below value in inner square .......... 28
  4.3 Neural Network based ....................................... 28
    4.3.1 Dense Neural Network ................................... 28
    4.3.2 CNN ...................................................... 31
    4.3.3 Spatial Transform + CNN ................................ 33

5 Discussion .................................................... 35
  5.1 Simple models .............................................. 35
    5.1.1 Average Pixel Value ................................... 35
    5.1.2 Pixel Variance .......................................... 35
    5.1.3 Less than boundary value ................................. 36
    5.1.4 Inner square ............................................ 36
5.2 Neural Networks ................................................. 36
  5.2.1 DNN ......................................................... 36
  5.2.2 CNN ......................................................... 37
  5.2.3 Spatial transformer ......................................... 37
5.3 Model comparison ............................................... 38
5.4 Model parameters - infinite possibilities ....................... 38
5.5 Data quality ..................................................... 39

6 Conclusion .......................................................... 41
  6.1 Summary ........................................................ 41
  6.2 Perspective and future works .................................. 41
    6.2.1 Initial generalization .................................... 42
    6.2.2 Iterative learning ....................................... 42
    6.2.3 Expectations ............................................ 42
    6.2.4 Expanding the problem .................................. 43

Table 1: Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AvgPix</td>
<td>Average pixel value</td>
</tr>
<tr>
<td>AvgVar</td>
<td>Pixel value variance</td>
</tr>
<tr>
<td>LessThan</td>
<td>Number of pixels less than boundary value</td>
</tr>
<tr>
<td>InnerSqu</td>
<td>Number of pixels less than boundary value in inner square</td>
</tr>
<tr>
<td>DNN</td>
<td>Dense neural network</td>
</tr>
<tr>
<td>FCMFFN</td>
<td>Fully connected multilayer feed forward network</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional neural network</td>
</tr>
<tr>
<td>CNN-Spat</td>
<td>References the spatial transformer network experiment</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Problem introduction

Using machine learning techniques to analyze data from scientific experiments has become more and more commonplace. These techniques are also usable to assist not only in data analysis, but when properly integrated also as a catalyst in time consuming data wrangling. An example of the latter is the particle picking process in single particle analysis of images obtained using Cryo-EM. The existing process is time consuming and data intensive, but machine learning has the potential to at least partly automate it and dramatically reduce time spent for researchers.

A Cryo-EM machine can yield high resolution images of single particles such as proteins. This technique can be used to determine the structure of these particles on an atomic scale. To obtain the images the first step of the process is to suspended multiple samples of the particle in a solvent. The machine then presses a droplet onto a micrograph that has been cooled by liquid nitrogen. This fixates the particles in a stable conformation such that an EM-image can be taken of the micrograph. An example of such an image can be seen in figure 1.1.

The next step in the process is locating where on this micrograph the particles are located, example given in figure 1.1, the so called ”Particle Picking” problem. Multiple different solutions to this problem exists. The conventional approach requires the scientist to manually pick out points on the micrograph. This process is tedious as the micrographs can contain multiple traces of the sample as well as introducing user bias and user error to the picking process. Solutions to this problem has been implemented using both supervised and unsupervised learning methods. The supervised learning methods requires that a large amount of labeled data from the same particle is readily available for training, while the unsupervised solutions are clustering algorithms that need
some reference as to what is representative of a particle in the current micrograph. Following the particle picking part further steps are needed to filter and clean up the data for the high resolution images.

How to refine the pipeline of this particle picking step has been investigated in [Wan16]. In this paper a pipeline for fully automated particle picking is described and tested. The solution described is a pipeline utilizing a pre-trained neural network. The model of the pipeline can be seen in figure 1.2. The model utilizes an iterative pipeline that use a network that has been pre-trained on examples of other samples of known particles. This approach will help identifying general shapes common to multiple samples. The pipeline is initiated by feeding the new micrograph containing the unknown particle to the network and use the results of the initial classification to re-train the network on the results of this round of classification. The purpose of this iterative approach is to teach the network to identify specific characteristics unique to this particle. When performed this process shows results comparable to that of human experts as documented in [Wan16].

The purpose of this project is, from a computer scientist’s point of view, to investigate different methods for identifying which pictures contain noise and which contains an image of the sample particle on data generated from Cryo-EM experiments. This project encompasses the process of a computer scientist facing an interdisciplinary challenge. The project started by contacting a molecular biologists who had a clear picture of the workflow and the problem areas associated with the “particle picking” problem, who supplied a working data set. For this project the investigation was limited to testing models that could be used as a classifier for particle picking. The data only contained examples
of 1 particle and as such the project was further limited to initially investigate models trained and tested on a single particle. In this limited area of investigation simple models of classification are compared with deep learning methods as proposed in [Wan16].

This report will introduce the basic theory of neural network architectures used in the experiments in chapter 2. The implementation details of the solutions will be sketched up in chapter 3. The experimental results and setup is presented in chapter 4. In chapter 5 the results presented in chapter 4 will be discussed and in chapter 6 the report will be summarized and the potential for future works is reflected upon.

## 1.2 Data

The data this project operates on consists of 13500 pictures taken using Cryo-EM of the protein Beta-galactosidase. The data set is balanced in positive and negative samples and was extracted from the cisTEM software. The data was split into a training set of 10800 images and a test set of 2700 images, both
sets were fully balanced in the amount negative and positive samples. Each image is represented by a 120x120 matrix containing values of electron densities. The electron density matrices are translatable to greyscale images for visualization but were processed using the non transformed values during training and testing. The images were labeled using the built in clustering algorithm. The algorithm requires an expert picking examples for the algorithm to calculate initial clustering values.

Some of the pictures does not contain any segment of Beta-Galactosidase but is pure background noise, these are considered the negative samples. The pictures containing Beta-Galactosidase are however very noisy, and as such not easy to identify by eye. In figure 1.3 two examples of the data samples are given. In figure 1.3a an example of a positive sample is given. Notice how the center is darker than the center of figure 1.3b which is an example of a negative sample.

![Figure 1.3: Images (a) containing Beta-Galactosidase and (b) not containing Beta-Galactosidase](image)
Chapter 2
Theory

The purpose of this section is to re-state the theory on which the classifier implementations are based. This provides a basic understanding of the theory and terminology which is used as a basis for the following discussion and comparison of results of experiments with multiple configurations of neural networks. A thorough introduction to the theory behind most network architectures can be found in [OG17]. This chapter will cover general neural network architecture and Convolutional Neural Network architecture based on [OG17] and provide a brief overview of the spatial transformer module.

2.1 Neural Networks

Neural Networks is an umbrella terminology branching over a class of mathematical models that are at its core inspired by how the neurons in the human brain are structured, hence the name "neural network". The strength of a neural network is the flexibility that it provides. Multiple configurations of networks and the ability to learn both simple and complex patterns provided with enough data makes them a powerful tool.

The basic building block of a neural network is an artificial neuron as shown in figure 2.1. This figure is an illustration of a neuron with incoming values from a previous layer and an outgoing activation value. Neural networks are organized in layers which consists of one or multiple neurons each. A neuron activation can be 0 or non-0. If the neuron activation value is non-0 it is said to be activated. The total value of the input sum to the neuron i is calculated as seen in equation 2.1 where $w_j$ is the weight on the edge from neuron j in the previous layer to neuron i, and $a_j$ references the output of the activation of neuron j in the previous layer (the output of the activation function).

\[ \text{inp}_i = \sum_{j=0}^{n} w_j \times a_j \]  

(2.1)
The output of the neuron is the result of the activation function $g_i$ acting on the input as demonstrated in equation 2.2, which is defined as the activation of neuron $i$, or $a_i$. The activation value is then passed further along the network to the neurons neuron $i$ is connected to.

$$a_i = g_i(i_{pi})$$

When training a neural network it is the weights in the network that are optimized. These weights are updated in such a way that the measured error of the network result is minimized. The details of this process will be explained later in this section.

### 2.1.1 Activation Function

The activation function is the function responsible for generating the neuron’s activation. Every neuron in the same layer contains the same activation function type. The functions are diverse and can be chosen to enforce specific properties. The Sigmoid function for example transforms values ranging from minus infinity to infinity into a scaled value between 0 and 1. The activation function used in the neural networks in this project is the Rectified Linear function. The rectified linear function stated in equation 2.3 only activates the neuron (returns a value other than 0) if incoming value is greater than 0 the relationship is linear.

$$f(x) = \max(0, x)$$

### 2.1.2 Fully Connected Multilayer Feed Forward Network

To introduce basic concepts behind neural network architecture the Multilayer Feed Forward Network is examined. This is one of the simplest neural network structures. Figure 2.2 shows an example of such an network. It consists of an input layer taking the input data, one or multiple hidden layers that are fully connected to the next layer. The last hidden layer is connected to an output layer. The network is in fact multiples of the simple neuron in figure 2.1.
organized in multiple layers. The number of layers in a network is referred to as the depth of the network with the width of a layer defined as the number of neurons in that layer. The name "Feed Forward" references to the property of the network, that each activation is always fed forward to the next layer. The Fully Connected Multilayer Feed Forward Network is also known as a "Dense Neural Network" or DNN. Dense refers to the property that the network is fully connected between each layer.

The input layer is the layer responsible for handling the input features. The hidden layers are the mathematical operators that will operate on the input features. The output layer is the layer that in a regression problem will output a predicted value based on the learned weights and in a classification problem will output an assigned probability to each of the classification classes. In a classification problem there will be an output node per class.

![MultiLayer Feed Forward Network](image)

Figure 2.2: MultiLayer Feed Forward Network. Green nodes (input layer). Blue nodes (hidden layer). Red nodes (output layer)

### 2.1.3 Training

With the basic structure of neural networks defined next step is to specify how the weights in the network are updated. The training can be accomplished using a back-propagation algorithm that uses some form of optimization algorithm to optimize the weights in the network. The back-propagation algorithm ensures that the optimizations are propagated throughout the network. Some simple terminologies for training networks are:

- **Batch size**: Size of each batch of data points
- **Batches**: The number of batches it takes to contain the data set with batches of size **Batch size**
- **Epoch**: One pass of the entire data set through the network (forward and backwards)

For training a neural network it is necessary to evaluate the quality of the network. For this we use a loss function. The loss is a quantifiable value.
indicating quality of the network compared to its ideal state. The purpose of training is then to minimize this function.

For a regression problem one such function could be the Mean Square Error loss. This function calculates the average of the squared distance between the label value and the predicted value as seen in equation 2.4

\[ L = \frac{1}{N} \sum_{i=1}^{N} (Y_{\text{pred}} - Y_{\text{lab}})^2 \]  

(2.4)

As the problem at hand is a classification problem we move onwards to the loss functions relevant to this problem. A widely used loss function here is the Softmax Cross-entropy function, which is also the function used in the implementation of this project. The Softmax function creates normalized probabilities from a collection of arbitrary values. We denote the output values from the neural network on input data point \( x_i \) as a score vector \( s = f(x_i) \). The loss of a data point \( x_i \) can then be calculated as seen in equation 2.5 which is the negative log of the normalized probability. (The max-part comes in play here as the output class chosen is the one with the highest normalized probability).

\[ L_i = -\log \left( \frac{e^{s_i}}{\sum_j e^{s_j}} \right) \]  

(2.5)

The network loss is then calculated as the average loss of every data point on the current network state.

With the loss function defined the next step is understanding how to update the weights in the network. The most used algorithm for this is the backpropagation algorithm. This algorithm operates in rounds, each of which consists of a forward pass and a backwards pass. The forward pass calculates all the activations and outputs with a given data point as input using the current weights in the network. Starting from the connections from the last hidden layer to the output layer the backwards pass calculates a "fractional error" using partial derivatives of the loss function. This "fractional error" is a measure of how much each neuron contributed to the measured error. The fractional error is propagated backwards throughout the entire network. These are then used to update the weights. The forward - backwards process is repeated until the algorithm meet the termination parameters set when initiated. This could be a number of epochs.

### 2.1.4 CNN

Convolutional Neural Networks is an architecture that is recognized as one of the most powerful architectures for image classification. One of the properties that makes it ideal for handling images as data points in comparison to the fully connected multilayer feed forward network is the number of weights that needs to be trained. With images as input the feature space is flattened to be
input into a FCMFFN requires height * width * depth input neurons. The CNN instead uses the property that images are defined this way to extrapolate higher order features which is then used for classification. A Convolutional Neural Network in general consists of:

1. **Input Layer** Neurons handling data input

2. **Feature Extraction Layers** Layers handling feature extraction. These takes the following structure:
   - (a) Convolution layer
   - (b) Pooling layer

3. **Classification Layer** A fully connected layer handling output classes - Analog to output layer from FCMFFN.

Increasing the depth of the convolution layers will increase the abstraction of the features to be detected. This is useful for breaking down highly abstract shapes to smaller and easier to recognize shapes.

### 2.1.4.1 Feature Extraction Layer

The Feature Extraction Layer performs the feature detection. This layer consists of 2 elements. The Convolution layer and the pooling layer. The convolution layer is the central concept of the Convolution Neural Network architecture. The feature extraction process is composed of:

- **Input** which can be either the original input image or the feature map output from earlier operations in the network
- **Filters** that is a set of matrices of the same size. The values in these filters are analog to weights on connections in the FCMFFN, and are the values optimized during training
- **Activation Map**. The resulting 2d matrix of a convolution process of 1 filter on the input
- **Activation Volume**. The output of the operation. This is a 3d matrix consisting of the stacked activation maps for each filter

Each filter results in a corresponding activation map. This map is calculated by sliding the filter along the input matrix and calculating the dot product of the filter on current covered area on the input as a activation value. All the activation values for the current filter are combined and results in the activation map for the current filter. The number of filters defines the depth of the resulting activation volume as demonstrated in figure 2.3.

The pooling layer will pool the values and apply the chosen pooling operation resulting in a new matrix. An example of this operation is the maxpool operator
that will pool the values in a sub-matrix of predetermined size and return the
maximum value of this sub-matrix. This is done using a sliding window along
the input matrix using a given step-size to determine how many indexes the
sub-matrix moves each step. This operation can be observed in equation 2.6.
With a step size larger than 1 the resulting matrix is reduced in size compared
to the original input matrix.

\[
\begin{pmatrix}
1 & 2 & 4 & 1 \\
3 & 1 & 9 & 9 \\
0 & 8 & 4 & 1 \\
6 & 2 & 3 & 6
\end{pmatrix}
\xrightarrow{\text{MaxPool}(\text{step}=2,\text{dim}(2,2))}
\begin{pmatrix}
3 & 9 \\
8 & 6
\end{pmatrix}
\] 

(2.6)

2.1.5 Spatial Transformer Module

The spatial transformer module as described in [Jad+15] is a way to add spatial
invariance to Convolutional Neural Networks. The spatial transformer module
works by defining a spatial transformation matrix. This matrix will then learn
how to apply the defined spatial transformation to input. The consequence of
this is the ability to recognize features and shapes from different angles in a 2d
image of a 3d object.

The spatial module is according to [Jad+15] a module that can be incorpo-
rated in the CNN architecture naively and improve image classification rates of
2d images of 3d objects. The spatial transformer module consists of 3 parts as
shown in figure 2.4.

The purpose of each of these parts:

- *Localization net* generates the parameters of transformation $\theta$. The lo-
calization net is typically a CNN with the restriction that the net has
a final output layer with the number of neurons needed for the desired
transformation.
Figure 2.4: Model of the spatial transformer module. Figure originates from [Jad+15]

- **Grid generator** creates a sampling grid. This grid is used to determine how the input should be sampled to generate the transformation.

- **Sampler** The sampler produces the transformed output map by applying the sampling grid to the input.

The grids are all differentiable and will therefore be trainable by use of back-propagation during the normal training of the networks.
Chapter 3
Implementation

Implementing the models required choices to be made regarding which languages and frameworks to be used. As a general implementation language for short data manipulation scripts python was chosen. Python gives the advantage of easy use of many libraries and frameworks relevant to Machine Learning problems. Tensorflow was chosen as the framework for implementing the neural network solutions. Tensorflow allows for using the custom estimator framework for high level implementation of Neural networks. Another benefit of using python and Tensorflow is the access to Tensorboard which enables live observations of loss and training accuracy during training. Tensorflow was set up to run on the machine GPU for increased speed during training. All implementations and supporting scripts and modules can be found at https://tinyurl.com/ParticlePicking. The data used and the trained models is not shared in this folder due to the size. However these can presented upon request. The spatial transformer module implementation used is the implementation made by [Dao16].

To ensure data handling was consistent for every test, a script for handling the MRC files and segmenting the data into training and test set was implemented. This module is the DataLoaderModule.py script with the significant parts listed in listing3.1

```
Listing 3.1: The data loader module

import numpy as np
import mrcfile

print("DataLoaderModule loaded")
with mrcfile.open('negativeSample.mrcs') as mrc:
    x_neg = np.copy(mrc.data, order='K')
    mrc.close()
    y_neg = np.ones(x_neg.shape[0], dtype=int)
with mrcfile.open('posSample.mrcs') as mrc:
    x_pos = np.copy(mrc.data, order='K')
```
dmin = mrc.header.dmin
dmax = mrc.header.dmax
mrc.close()
y_pos = np.zeros(x_pos.shape[0], dtype=int)

def get_balanced_dataset():
    balanced_train_data = np.concatenate((
        x_pos[0:5400],
        x_neg[0:5400]))
    balanced_train_labels = np.concatenate((
        y_pos[0:5400],
        y_neg[0:5400]))
xpos_size = x_pos.shape[0]
x_neg_test_size = x_neg[5400:].shape[0]
xpos_test_start_index =
xpos_size - x_neg_test_size
    balanced_test_data = np.concatenate((
        x_pos[xpos_test_start_index:],
        x_neg[5400:]))
    balanced_test_labels = np.concatenate((
        y_pos[xpos_test_start_index:],
        y_neg[5400:]))

return balanced_train_data,
balanced_train_labels,
balanced_test_data,
balanced_test_labels

3.1 Simple Solutions Implementation

Implementing the simple models was accomplished using standard python commands combined with the numpy library for ease of implementation. The simple implementations all follow the same structure "training" and "evaluating". Training the model is accomplished by creating representative values for positive and negative cases of training data as seen in listing 3.2. This example calculates the average value of each image. The averages are summed up in either "sum_pos" or "sum_neg" depending on the class the image belongs to. These sums are divided by the number of images in each class. This yields the average pixel value from each class from the training set. These average pixel values are used as the representative values for the corresponding class.

Listing 3.2: Calculating the average values

sum_pos = 0
for i in range(0,5400):

```
test = train_data[i,:, :]
sum_pos = sum_pos + np.mean(test)

sum_neg = 0
for i in range(5400, 10800):
    test = train_data[i,:, :]
    sum_neg = sum_neg + np.mean(test)

avg_neg = sum_neg / 5400
avg_pos = sum_pos / 5400
```

Evaluating the model is accomplished by calculating a representative test value for each data point in the test set. The label assigned is then the label of the class with a representative closest to the calculated test value. Example of this is seen in listing 3.3 where the test value is the average value of the matrix that makes up the test data point. Accuracy is then calculated by counting the discrepancies between the assigned labels and the known labels from the test set.

```
Listing 3.3: Calculating the test values
res_vect = np.zeros(eval_labels.shape[0])
errors = 0
for i in range(0, eval_data.shape[0]):
    mean = np.mean(eval_data[i,:, :])
    if abs(mean-avg_neg)<abs(mean-avg_pos):
        res_vect[i] = 1
    if(res_vect[i] != eval_labels[i]):
        errors = errors + 1
print("Accuracy: ",
(1-errors/eval_data.shape[0])*100)
```

Every simple model was implemented using this structure with extra helper functions calculating the representative values corresponding to the model. Implementing the K-fold validation for parameter search also follows this structure of training and validation.

### 3.2 Neural Networks Implementation

Implementing the neural networks makes use of the custom estimator framework from Tensorflow. All implementations are based on the CNN implementation from Tensorflows tutorial [Ten18].

Instantiating a custom estimator (listing 3.4) requires setting up an estimator with a model function defining the neural network and a path pointing to the save destination of the model. The term "model" refers to the neural network.
Listing 3.4: Setting up data and classifier

```python
train_data, train_labels, eval_data, eval_labels = dlm.get_balanced_dataset()
classifier = tf.estimator.Estimator(
    model_fn=nn_model_fn, model_dir=
    "/tmp/new/emdat_dnn_tiny_model_06")
```

Training the model is then handled by the estimator frameworks `train` function where we supply the variables shown below to the train operation seen in listing 3.5.

- The training data and connected labels.
- The batch size (which in all cases were set to 100).
- The number of epochs (50).
- Boolean value defining if the data is to be shuffled for training.

Evaluating the test set is equally streamlined, calling the frameworks `eval` function and supplying the test data as seen in listing 3.6

Listing 3.5: Training the classifier

```python
train_input_fn =
tf.estimator.inputs.numpy_input_fn(
    x={"x": train_data},
    y=train_labels,
    batch_size=100,
    num_epochs=50,
    shuffle=True)
classifier.train(
    input_fn=train_input_fn,
    steps=None)
```

Listing 3.6: Evaluating the classifier

```python
eval_input_fn =
tf.estimator.inputs.numpy_input_fn(
    x={"x": eval_data},
    y=eval_labels,
    num_epochs=1,
    shuffle=False)
eval_results = classifier.evaluate(input_fn =
eval_input_fn)
print(eval_results)
```

Setting up the model function (the neural network) requires setting up the input parameters, in this case matrices of 120x120x1 shown in listing 3.7.
Listing 3.7: Setting up input data to model function

```python
def nn_model_fn(features, labels, mode):
    """Model function for NN."""
    # Input Layer
    input_layer = tf.reshape(features['x'],
                            [-1, 120, 120, 1])
```

In listing 3.8 it is demonstrated how to set up a fully connected layer in the tensorflow framework. The setup requires:

- Defining the input to the layer
- The number of neurons in the layer (units)
- The activation function the neurons represent which in this case is the rectifier function

Dropout randomizes which neurons are active during training. This is defined by a probability that each neuron does not factor in the current round of training. Dropout is a form of regularization that should minimize the model overfitting to the training data.

Listing 3.8: Creating fully connected layer and setting up dropout

```python
dense1 = tf.layers.dense(inputs=input_layer,
                         units=28, activation=tf.nn.relu)
dense2_flat = tf.layers.flatten(dense1)
dropout = tf.layers.dropout(
    inputs=dense2_flat, rate=0.6,
    training=mode == tf.estimator.ModeKeys.TRAIN)
```

Predicting the labels is achieved by a logits layer. Each neuron in this output layer corresponds to a class (2 in this case). The neuron with the highest assigned value is chosen and the corresponding class label is the model’s prediction as seen in 3.9.

Listing 3.9: Setting up logits layer and calculating predictions

```python
logits = tf.layers.dense(inputs=dropout, units=2)
predictions = {
    "classes": tf.argmax(input=logits, axis=1)
}
```

In listing 3.2 the framework will simply return the predictions if the model mode is set to "predict".

```python
if mode == tf.estimator.ModeKeys.PREDICT:
    return tf.estimator.EstimatorSpec(mode=mode,
                                      predictions=predictions)
```
Calculating the loss is achieved by using the Tensorflow loss functions when passing the labels and the logits layer to the chosen loss function as seen in listing 3.10. The implementations in this project uses the softmax cross entropy function. To make the loss accessible for Tensorboard the loss is tagged as a summary scalar variable.

Listing 3.10: Setting up loss function and accuracy calculator

```python
# Calculate Loss (for both TRAIN and EVAL modes)
loss = tf.losses.sparse_softmax_cross_entropy(
    labels=labels,
    logits=logits)
accuracy = tf.metrics.accuracy(
    labels = labels,
    predictions = predictions["classes"])
tf.summary.scalar('Loss', loss)
```

The implementations used the standard Gradient Descent Optimizer supplied by the Tensorflow framework as seen in listing 3.11 to handle training.

Listing 3.11: Setting up the optimizer

```python
# Configure the Training Op (for TRAIN mode)
if mode == tf.estimator.ModeKeys.TRAIN:
    optimizer = tf.train.GradientDescentOptimizer(
        learning_rate=0.001)
    train_op = optimizer.minimize(
        loss=loss,
        global_step=tf.train.get_global_step())
    tf.summary.scalar('accuracy', accuracy[1])
return tf.estimator.EstimatorSpec(mode=mode,
    loss=loss, train_op=train_op)
```
Chapter 4

Experiments

The following section documents the process of exploring solutions to the specific problem of classifying the Cryo-EM images as containing either protein or noise. Throughout the sequence of experiments the complexity of the model used as a classifier increases.

The data set is handled by the DataLoaderModule.py ensuring consistent training and isolated test data for every experiment. The experiments consisted of training the specific model on the training set consisting of 10800 pictures, with 5400 images belonging to each class while evaluating on the test set as described in section 1.2. The purpose of these experiments is to explore the viability of multiple solutions measured by the accuracy they achieve on the test set. The results of these tests is presented in table 4.1. Observed in this table

<table>
<thead>
<tr>
<th>Name:</th>
<th>AvgPix</th>
<th>VarPix</th>
<th>LessThan</th>
<th>InnerSqu</th>
<th>DNN</th>
<th>CNN</th>
<th>CNN-Spat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acc:</td>
<td>55.85%</td>
<td>78.56%</td>
<td>75.52%</td>
<td>85.78%</td>
<td>92.26%</td>
<td>94.59%</td>
<td>95.41%</td>
</tr>
<tr>
<td>Error:</td>
<td>44.15%</td>
<td>21.44%</td>
<td>24.48%</td>
<td>14.22%</td>
<td>7.74%</td>
<td>5.41%</td>
<td>4.59%</td>
</tr>
</tbody>
</table>

the neural network solutions outperform the simple solutions with the DNN achieving an error rate 45.5% lower than the "InnerSqu" experiment. The CNN improves upon this result with a 30% lower error rate. The CNN-spat model achieves a further reduction of the error rate of 15% relative to the CNN model.

4.1 Experimental setup

The experimental setup is listed in table 4.2. This is the machine the experiments were performed on.
4.2 Simple Pixel Value based

The first set of experiments was designed as simple investigations on the distribution of the pixel values. The experiments attempt to classify the images based on simple models. This is to ensure that the use of simpler methods have been investigated before increasing model complexity.

4.2.1 Average Pixel Value

This experiment represents the simplest feasible solution in the sequence of experiments. The purpose of this experiment is to investigate the "lowest hanging fruit" solution. This model iterates through every data point in the training set. For each data point the average value of the matrix is calculated. This average pixel value is then added to a variable "sum_xx" where xx is an indicator of whether the matrix is a negative sample or a positive sample. Each of these variables is divided by the number of samples in each class to get an average of the values of each class. Classification of the pictures in the test set is performed by a simple distance comparison between the calculated average values in the training set and each data point in the test set’s average value individually.

Table 4.3: Accuracy of the Average Pixel Value test with the representative values for each class listed

<table>
<thead>
<tr>
<th>Accuracy:</th>
<th>55.85</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. Positive Samples:</td>
<td>-0.01527</td>
</tr>
<tr>
<td>Avg. Negative Samples:</td>
<td>0.00499</td>
</tr>
</tbody>
</table>

The accuracy of this experiment was found to be near 55% as shown in table 4.3. It was observed that the values used to represent the positive and the negative classes are extremely close, in comparison to the interval the individual pixel values span (which is -32.17 and 6.39 in the training set).

4.2.2 Pixel Value Variance

This experiment investigates how well the variance of the pixel values in each picture can be used as a simple form for classification. This is accomplished by
iterating through every data point in the training set. For each data point the variance of the matrix values is calculated and added to a variable "sum_xx" where xx is an indicator of whether the matrix is a negative sample or a positive sample. This sum is then divided by the number of data points belonging to the specific class, to get the average variance for each class, respectively. The test pictures are then classified by least difference comparisons between their variance and the average variance of either class.

Table 4.4: Accuracy of the Average Pixel Variance test with the representative values for each class listed

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>78.56</td>
</tr>
<tr>
<td>Avg. Var. Positive Samples</td>
<td>1.1411</td>
</tr>
<tr>
<td>Avg. Var. Negative Samples</td>
<td>0.9657</td>
</tr>
</tbody>
</table>

Seen in table 4.4 the accuracy of this method reaches 78.56% indicating that the pixel values do not diverge from their respective means to the same degree across both classes. The average variance within each class is however still close.

4.2.3 Number of pixels less than boundary value

The purpose of this experiment was to investigate if the 2 different classes on average is distributed unevenly with a relative majority of the pixels having a value less than a given boundary value. This was achieved by counting into a variable for each class, for every data point, the number of pixels less than the given boundary value. This sum was divided by the number of data points in each class, such that an average was found. To find the boundary value 5-fold cross validation was used, splitting the training set to find the boundary value that yields the largest validation accuracy for use in the final model. The test set was classified by least difference comparisons between the average number for the class and the number below the boundary value for the test data point. In table 4.5 the accuracy of the model is presented along the representative

Table 4.5: Accuracy of the Bound test with the representative values for each class listed and the chosen Boundary with associated validation acc.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>75.52</td>
</tr>
<tr>
<td>Validation Accuracy</td>
<td>80.37</td>
</tr>
<tr>
<td>Bound</td>
<td>-2</td>
</tr>
<tr>
<td>Avg. #Less than bound Pos</td>
<td>481.29</td>
</tr>
<tr>
<td>Avg. #Less than bound Neg</td>
<td>283.09</td>
</tr>
</tbody>
</table>

values for each class and the chosen boundary value. It was observed that there was a slight decrease in accuracy from the previous experiment.
4.2.4 Number of pixels below value in inner square

The inner Square experiment combines the knowledge that there seems to be a difference in the number of pixels below a certain boundary value with the notion that these are not evenly distributed throughout the matrix. The model establishes the size of an "inner matrix" and counts the number of values inside this matrix below the boundary value. From this number, the average percentage of values that are inside the inner matrix below the boundary value is calculated for each class. The experiment uses 5-fold validation to find the combination of size of the inner square and the boundary value that yields the highest validation accuracy. The test set was then classified by calculating this value for each point in the test set and classified by least difference comparisons. It was observed that the accuracy increases when the comparison of pixels below the boundary value is limited to the center of the image. This indicates that some patterns or features of interest may be present here.

Table 4.6: Accuracy of the "InnerSqu" test with the representative values for each class listed, the chosen Boundary and inner matrix dimensions with associated validation acc.

<table>
<thead>
<tr>
<th>Test Accuracy:</th>
<th>85.78</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation Accuracy:</td>
<td>91.73</td>
</tr>
<tr>
<td>Bound:</td>
<td>-1</td>
</tr>
<tr>
<td>Inner square dim.</td>
<td>31x31</td>
</tr>
<tr>
<td>Avg. #Less than bound Pos:</td>
<td>61.46</td>
</tr>
<tr>
<td>Avg. #Less than bound Neg:</td>
<td>40.95</td>
</tr>
</tbody>
</table>

4.3 Neural Network based

The neural network based experiments investigates different network types as well as different configurations for each network with the purpose of understanding how each configuration impacts the test accuracy. The networks being investigated are multiple versions of both Dense Neural networks (DNN) and Convolution Neural Networks (CNN) along with a version of a CNN with a Spatial Transform Module.

4.3.1 Dense Neural Network

Testing different configurations of a Dense Neural Network involves many variables. These experiments investigate a few of these by configuring a small 1 layer DNN, a small 2 Layer DNN, a wide 1 layer DNN, and a wide 2 layer DNN. Within each of the different networks, dropout regularization is used to observe how this influences the outcome in relation to each of the network configurations. The networks were trained using a batch size of 100 for 50 epochs using 4 different dropout ratios yielding the results seen in tables 4.7 to 4.10.
4.3.1.1 1 layer DNN

The 1 layer DNN is configured as:

1. Input Layer taking n 120x120 Matrices
2. Hidden layer with 28 nodes using the Rectified Liner Unit as activation function
3. Dropout layer with probability p of dropping a node
4. Output layer with 2 nodes, 1 for each class

Table 4.7: 1 layer DNN training and test Acc. for every tested drop out ratio

<table>
<thead>
<tr>
<th>Dropout</th>
<th>0%</th>
<th>20%</th>
<th>40%</th>
<th>60%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Acc:</td>
<td>96.98%</td>
<td>96.73%</td>
<td>96.52%</td>
<td>97.37%</td>
</tr>
<tr>
<td>Test Acc:</td>
<td>85.19%</td>
<td>77.19%</td>
<td>89.07%</td>
<td>77.04%</td>
</tr>
</tbody>
</table>

As seen in table 4.7 there are notable differences between the observed training accuracy and the observed test accuracy. This is indicative of the model overfitting to the training data. Increasing the dropout is an attempt to reduce this effect.

4.3.1.2 2 layer DNN

The 2 layer DNN adds a second layer to the first network to create a deeper network. The configuration of this network is:

1. Input Layer taking n 120x120 Matrices
2. Hidden dense layer with 28 nodes using the Rectified Liner Unit as activation function
3. Hidden dense layer with 14 nodes using the Rectified Liner Unit as activation function
4. Dropout layer with probability p of dropping a node
5. Output layer with 2 nodes, 1 for each class

Table 4.8: 2 layer DNN training and test Acc. for every tested drop out ratio

<table>
<thead>
<tr>
<th>Dropout</th>
<th>0%</th>
<th>20%</th>
<th>40%</th>
<th>60%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Acc:</td>
<td>96.5%</td>
<td>93.43%</td>
<td>96.56%</td>
<td>95.87%</td>
</tr>
<tr>
<td>Test Acc:</td>
<td>87.67%</td>
<td>92.26%</td>
<td>75.22%</td>
<td>82.41%</td>
</tr>
</tbody>
</table>

The 2 layer DNN exhibits overfitting on the training data which can be observed by comparing the test accuracy to the training accuracy as this shows
a large gap between these. With a dropout of 20% the model is able to generalize to the test data as it is observed that the gap between training and test accuracy is decreased.

### 4.3.1.3 1 layer wide DNN

The purpose of this experiment is to explore the effects of widening the initial hidden layer. In relation the the first single layer network we expect the model to reach a higher training accuracy as it should fit stronger to the training data. The configuration for 1 layer wide DNN is:

1. Input layer taking n 120x120 Matrices
2. Hidden dense layer with 128 nodes using the Rectified Liner Unit as activation function
3. Dropout layer with probability p of dropping a node
4. Output layer with 2 nodes, 1 for each class

Table 4.9 shows an increase in the measured training accuracy indicating the model’s increased ability to fit to the training data. The dropout regularization does not appear to have a significant effect on the results as the observed test accuracy appears to fluctuate.

Table 4.9: 1 layer wide DNN training and test Acc. for every tested drop out ratio

<table>
<thead>
<tr>
<th>Dropout:</th>
<th>0%</th>
<th>20%</th>
<th>40%</th>
<th>60%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Acc:</td>
<td>97.91%</td>
<td>98.28%</td>
<td>97.02%</td>
<td>97.81%</td>
</tr>
<tr>
<td>Test Acc:</td>
<td>89.59%</td>
<td>80.81%</td>
<td>89.79%</td>
<td>80.96%</td>
</tr>
</tbody>
</table>

### 4.3.1.4 2 layer wide DNN

The 2 layer wide DNN is a 2 layer DNN with more neurons in each hidden layer. A factor of 10 more neurons in each layer in comparison to the 2 layer DNN examined earlier. The purpose of this is to examine the behaviour of the DNN when size of the model increases in both depth and width. The configuration of the 2 layer wide DNN:

1. Input Layer taking n 120x120 Matrices
2. Hidden dense layer with 280 nodes using the Rectified Liner Unit as activation function
3. Hidden dense layer with 140 nodes using the Rectified Liner Unit as activation function
4. Dropout layer with probability p of dropping a node
5. Output layer with 2 nodes, 1 for each class

The results of this test can be found in table 4.10. Looking at the results

Table 4.10: 2 layer wide DNN training and test Acc. for every tested drop out ratio

<table>
<thead>
<tr>
<th>Dropout:</th>
<th>0%</th>
<th>20%</th>
<th>40%</th>
<th>60%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Acc:</td>
<td>97.02%</td>
<td>95.63%</td>
<td>96.52%</td>
<td>96.85%</td>
</tr>
<tr>
<td>Test Acc:</td>
<td>78.19%</td>
<td>90.26%</td>
<td>77.89%</td>
<td>78.41%</td>
</tr>
</tbody>
</table>

from this test it can be observed that reducing the models tendency to overfit is not achievable by only incorporating a dropout layer between the last hidden layer and the output layer, because the training accuracy and test accuracy exhibit large gaps with a ”top performance” still yielding a gap of 5%-points.

### 4.3.2 CNN

The Convolution Neural Networks is the standard solution to most image recognition and classification problems. The following experiments will explore how the CNN performs with dropout regularization and increasing depth of the network. Increasing the depth of a CNN raises the level of abstraction of features the model is able to estimate. This is expected to result in an increased test accuracy in a deeper CNN if the data contains more abstract shapes.

#### 4.3.2.1 1 layer CNN

The 1-layer CNN is the simplest CNN configuration tested in this project. This configuration uses 32 filters in the first layer and a 2x2 max-pooling. The configuration for 1 layer CNN:

1. Input Layer taking n 120x120 Matrices
2. Convolution layer with 32 filters and kernel of dimensions 5x5
3. Max pooling 2x2 with step size 2
4. Hidden dense layer with 1024 nodes using the Rectified Liner Unit as activation function
5. Dropout layer with probability p of dropping a node
6. Output layer with 2 nodes, 1 for each class

Results of the test are displayed in table 4.11. With dropout regularization this model seems able to compensate for over-fitting observing only a small gap between training and test accuracy with a dropout factor of 0.6.
Table 4.11: Test and training acc. for the 1 layer CNN

<table>
<thead>
<tr>
<th>Dropout:</th>
<th>0%</th>
<th>20%</th>
<th>40%</th>
<th>60%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Acc:</td>
<td>96.93%</td>
<td>96.87%</td>
<td>96.61%</td>
<td>95.41%</td>
</tr>
<tr>
<td>Test Acc:</td>
<td>88.74%</td>
<td>93.89%</td>
<td>88.44%</td>
<td>94.22%</td>
</tr>
</tbody>
</table>

4.3.2.2 2 layer CNN

The 2 layer CNN adds another layer of convolutions where the second convolution layer contains double the amount of filters used in the first convolution layer. This is done as an attempt to learn more abstract features from the data. The configuration of 2 layer CNN:

1. Input Layer taking n 120x120 Matrices
2. Convolution layer with 32 filters and kernel of dimensions 5x5
3. Max pooling 2x2 with step size 2
4. Convolution layer with 64 filters and kernel of dimensions 5x5
5. Max pooling 2x2 with step size 2
6. Hidden dense layer with 1024 nodes using the Rectified Linear Unit as activation function
7. Dropout layer with probability p of dropping a node
8. Output layer with 2 nodes, 1 for each class

Table 4.12: Test and training acc. for the 2 layer CNN

<table>
<thead>
<tr>
<th>Dropout:</th>
<th>0%</th>
<th>20%</th>
<th>40%</th>
<th>60%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Acc:</td>
<td>96.26%</td>
<td>95.94%</td>
<td>95.93%</td>
<td>95.37%</td>
</tr>
<tr>
<td>Test Acc:</td>
<td>84.67%</td>
<td>94.59%</td>
<td>78.81%</td>
<td>81.26%</td>
</tr>
</tbody>
</table>

Results of the test are displayed in table 4.12. This network in comparison with the 1 layer CNN is also able to close the gap between training accuracy and test accuracy. Only a small increase in test accuracy when compared to the 1 Layer CNN is observed.

4.3.2.3 3 layer CNN

The last CNN experiment adds further depth to the network by adding a convolution layer applying 128 filters for further abstraction of features. The configuration of 3 layer CNN:

1. Input Layer taking n 120x120 Matrices
2. Convolution layer with 32 filters and kernel of dimensions 5x5
3. Max pooling 2x2 with step size 2
4. Convolution layer with 64 filters and kernel of dimensions 5x5
5. Max pooling 2x2 with step size 2
6. Convolution layer with 128 filters and kernel of dimensions 5x5
7. Max pooling 2x2 with step size 2
8. Hidden dense layer with 1024 nodes using the Rectified Linear Unit as activation function
9. Dropout layer with probability p of dropping a node
10. Output layer with 2 nodes, 1 for each class

Table 4.13: Test and training acc. for the 3 layer CNN

<table>
<thead>
<tr>
<th>Dropout:</th>
<th>0%</th>
<th>20%</th>
<th>40%</th>
<th>60%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Acc:</td>
<td>94.91%</td>
<td>95.15%</td>
<td>95.17%</td>
<td>95.43%</td>
</tr>
<tr>
<td>Test Acc:</td>
<td>89.89%</td>
<td>93.44%</td>
<td>84.07%</td>
<td>93.81%</td>
</tr>
</tbody>
</table>

The results of testing this type of network is displayed in table 4.13. There is no increase in accuracy observed when increasing the depth from 2 convolution layers to 3. This result could indicate that beyond rough shapes no finer features are present in the image that can be used to decide between the 2 classes.

4.3.3 Spatial Transform + CNN

This section investigates how the spatial transformer module influences an earlier tested network. This is accomplished by creating a localization network using a 2 layer CNN with a hidden layer for feature detection and a dropout layer. This feeds into the spatial transform layer alongside the input layer for spatial transform. The output of this transform is then fed into a 2 layer CNN as tested earlier. The expectation is that this module will be better able to ascertain if the shape at the center is noise, ice or a warped representation of the sample molecule. The configuration of the CNN with Spatial Transformer is:

1. Input Layer taking n 120x120 Matrices
2. Convolution layer with 32 filters and kernel of dimensions 5x5
3. Max pooling 2x2 with step size 2
4. Convolution layer with 64 filters and kernel of dimensions 5x5
5. Hidden dense layer with 1024 nodes using the Rectified Linear Unit as activation function

6. Dropout layer with probability $p_1$ of dropping a node

7. The Spatial Transform layer taking as input the output of the previous layer and the input layer.

8. Convolution layer with 32 filters and kernel of dimensions 5x5 taking the output of the Spatial Transform layer as input.

9. Max pooling 2x2 with step size 2

10. Convolution layer with 64 filters and kernel of dimensions 5x5

11. Hidden dense layer with 1024 nodes using the Rectified Linear Unit as activation function

12. Dropout layer with probability $p_2$ of dropping a node

13. Output layer with 2 nodes, 1 for each class

The results of the spatial transform network test can be seen in Table 4.14. The test values indicate that keeping dropout rate low in the first dropout layer has the most significant influence on test accuracy. When combined with a high dropout in the second dropout layer the highest test accuracy is observed.

Table 4.14: The test accuracies of the CNN + spatial transformer with multiple dropout configurations

<table>
<thead>
<tr>
<th>Dropout p1(row), p2(col)</th>
<th>0%</th>
<th>20%</th>
<th>40%</th>
<th>60%</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>94.81%</td>
<td>91.04%</td>
<td>94.33%</td>
<td>95.33%</td>
</tr>
<tr>
<td>20%</td>
<td>84.07%</td>
<td>50%</td>
<td>50%</td>
<td>95.41%</td>
</tr>
<tr>
<td>40%</td>
<td>75.11%</td>
<td>90.33%</td>
<td>50%</td>
<td>50%</td>
</tr>
<tr>
<td>60%</td>
<td>89.59%</td>
<td>88.04%</td>
<td>87.11%</td>
<td>79.19%</td>
</tr>
</tbody>
</table>
Chapter 5

Discussion

This chapter will reflect on the results shown in chapter 4. The reported accuracies of the experiments will be compared and the reasoning behind the discrepancies will be discussed. In table 5.1 the highest achieved accuracy of each class of experiments is re-stated to refresh how the resulting accuracies compared.

Table 5.1: Table 4.1 reprinted for the readers convenience. The table shows the highest accuracy and lowest error rate for each model

<table>
<thead>
<tr>
<th>Name</th>
<th>AvgPix</th>
<th>VarPix</th>
<th>LessThan</th>
<th>InnerSqu</th>
<th>DNN</th>
<th>CNN</th>
<th>CNN-Spat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acc:</td>
<td>55.85%</td>
<td>78.56%</td>
<td>75.52%</td>
<td>85.78%</td>
<td>92.26%</td>
<td>94.59%</td>
<td>95.41%</td>
</tr>
<tr>
<td>Error:</td>
<td>44.15%</td>
<td>21.44%</td>
<td>24.48%</td>
<td>14.22%</td>
<td>7.74%</td>
<td>5.41%</td>
<td>4.59%</td>
</tr>
</tbody>
</table>

5.1 Simple models

5.1.1 Average Pixel Value

The simplest model "AvgPix" yielded only a 55% accuracy as per table 4.3. This result is not surprising given that the model classifies based on the average pixel value of each class and these values are nearly identical. The recorded accuracy, while close to 50%, is slightly higher than guessing with a 50-50 distribution but still unacceptable for practical use.

5.1.2 Pixel Variance

The natural progression from the "AvgPix" experiment was to investigate to which degree the pixels from each class deviates from the mean pixel value of the picture. This was investigated in the "VarPix" experiment. The results
listed in table 5.1 show a 78% accuracy, which is a significant improvement compared to "AvgPix". This result can be explained by the values shown in table 4.4 indicating that the images containing a particle has a higher variance than images that does not. This result is not surprising as we expect the images containing the particles to exhibit a sharper contrast between the boundaries of the particle and the background noise.

5.1.3 Less than boundary value
The "LessThan" experiment did not yield a higher accuracy when classifying the test set. In table 5.1 the accuracy is reported to be 75.52% even though the difference between the positive class and the negative class seems to be significant as seen in table 4.5, the validation accuracy of 80% seems to indicate that model has been over fitted to the training set (it does not generalize well to the test set). This could be explained by the fitting parameter "boundary value" which may not have taken the optimal value for this model when classifying the test set. Another possible explanation could be that the model is too simple to express the differences between the classes. A small expansion of this model would be to incorporate an upper bound, counting the pixels above this bound as well. The purpose of this is to investigate if the images containing particles on average contains more extreme values.

5.1.4 Inner square
The "InnerSqu" experiment yielded more promising. This model reported a 85.78% accuracy as seen in table 5.1, however with an even higher validation accuracy of 91.73%. It is not surprising that this model performs better as every image is centered on a particle if one exists in the image. This property will by its nature yield a high difference between positive and negative samples. The value used for classification could be altered to the average pixel value inside the inner square. The purpose of this change would be to investigate if this property is even more pronounced localized at the center, than given the entire image as in the "AvgPix" experiment. Another interesting extension would be to implement an upper boundary value such that the count now contains all extreme values above the upper and below the lower boundary. This may lead to stronger overfitting to the training set given that another fitting variable is introduced. However observing how these changes to the model influences test accuracy could be interesting.

5.2 Neural Networks

5.2.1 DNN
The DNN experiments resulted in a maximum accuracy of 92.26%. This result came from the 2 Layer DNN using 2 small hidden layers as reported in section 4.3.1.2. This configuration outperforms the 2-layer wide DNN by having
a higher degree of generalization with a 20% dropout ratio. The 1 layer wide DNN has a higher training accuracy but is strongly overfitted and does not show signs of improving with a higher dropout ratio as seen in table 4.9.

Overall no configuration of the DNN (layers, depth, dropout) seem to give a consistently strong test result, leading to the realization that the DNN architecture is not an optimal model to use as a classifier of Cryo-EM images. This is evident when compared to the "InnerSqu" model. An explanation of this would be that the "InnerSqu" model is designed to take advantage of the information that the data is images, where if a particle is present it is at the center. This ensures that the model only gives attention to the parts of the image that are relevant to this problem while the DNN will attempt to learn from the entire image.

5.2.2 CNN

The CNN class of experiments further improves the accuracy with the strongest configuration reaching a 94.59% for the 2 layer CNN as stated in table 5.1. The difference between the highest accuracy among the different CNN configurations is not large however. The 1 layer CNN reaches an accuracy of 94.22% while the 3 layer CNN reaches 93.81%. The intuition for this would be that beyond initial features detected in the first layer there are no discernible features to detect and as such increasing the depth of the CNN does not serve a purpose. The inability to recognize higher order features could be due to the low resolution of the images resulting in only rough shapes being recognized as features.

Comparing the results of the CNN tests to the DNN tests, the CNN test accuracy does not fluctuate to the same degree when adjusting the depth and width of the network. While the configurations of the CNN have more stable test accuracy than the DNN, the models still overfit to the training data for most configurations. The reason for this is that the CNN is better able to classify images than the DNN since the DNNs are designed to handle the data as a 2d matrix (with a 3rd dimension of channels) while the DNN handles the images as a basic input feature list. Unlike with the DNN the CNN is then able to utilize the information of how the pixel values clump together to form shapes and features. This is analogous to how the "InnerSqu" model draws information from how the images is expected to be oriented.

5.2.3 Spatial transformer

The Spatial Transform Network experiments required many test runs. This was due to the model’s difficulty with minimizing loss. Some of the tests were unable to converge to a reasonable loss and test accuracy and yielded only a 50% accuracy as disclosed in table 4.14. This model did however outperform the other models yielding a 95.41% accuracy with a 20% dropout in the spatial transform module localization network and a 60% dropout in the following
CNN. The increase in performance over the 2-layer CNN was a 15% decrease in error-rate. The smaller than expected increase in accuracy could be explained by the same reason, that increasing the number of CNN layers did not yield a significant increase in accuracy. With low resolution images the spatial variance between the particles might not be discernible and as such not give the model more information.

When looking at table 4.14 keeping dropout p1 to 0% evens out the test accuracies fluctuating from 90% to 95% when changing dropout p2. Changing dropout p1 decreases the network consistency, where it is observed that the test accuracy fluctuations increase. This indicates that overfitting during the spatial transform is not a issue.

5.3 Model comparison

When comparing the models it is evident that the CNN and CNN-spat outperforms the other solutions as seen in table 5.1. As discussed above the images may be in too low a resolution to learn higher abstraction shapes which limits the potential of these architectures in this specific problem. As stated in chapter 1 the end purpose of the model is the ability to capture general shapes from multiple particles, apply these generalizations to the first iteration of particle detection on a completely new particle, while afterwards improve on the classification with "self training" on the results of the initial classification. This adaptable approach to classification should make the spatial transformer a suitable candidate as it reduces the rotational variance of the particles.

The models have so far been compared based on the results of generalizing from a training set to a test set consisting of images of the same particle. The need for heavy generalization and the benefit of applying heavy dropout may be more urgent when training the networks on a training set initially containing different particles than the test set. This problem will benefit from the networks being able to generalize to a larger degree.

5.4 Model parameters - infinite possibilities

During the experimental phase many possibilities for experimental variation presented themselves. One might look at the effect the activation functions had on the outcome, considering the classification was a binary decision problem. Using a different optimization algorithm may also influence how well the model converges towards a minimum. The experiments mostly explored different architectures but even within these the room for experimentation is significant. Examples of these possibilities could be:

- Changing the filter size in CNN
- Changing filter counts in CNN
• Choosing a different pooling operator in CNN

• Changing the spatial transformation matrix

The experiments explored changes in the dropout ratio, depth, and width within each type of network. This was to control overfitting in the different models. The results from chapter 4 show that the models are able to fit the data with training accuracies reaching around 95%. Adding dropout may lower this but improve generalization of the model. What was observed however was that the resulting test accuracy did not always improve when increasing the drop-out, and adding too much dropout seemed to decrease the performance of the model.

5.5 Data quality

The observed test and training accuracies are strongly influenced by the data source. The original data set is not a "golden" data set with 0% error. This means that both the test set and the training set contains false positives and false negatives that will undoubtedly influence the results. These data impurities stem from the process of generating the data set. This process introduces bias when the "expert" is picking reference points that are to be fed to the clustering algorithm. The algorithm itself does not guarantee 100% correct labeled output, while any errors made during the initial picking will influence the quality of the output.

The highest achieved training accuracy was 98% for the wide DNN. This would be surprising if we had a pure training set since we expect the wide DNN to overfit the training data without any form of regularization. This is especially true when considering the DNN handles the 120x120 images as 14400 separate input features and only 10800 training examples are supplied breaking with the "best practice" of having less features than training examples.
Chapter 6

Conclusion

6.1 Summary

This report has depicted the experimental process of particle classification from problem specification to result evaluation. During this process it became evident that to better evaluate the models more data was needed - this is elaborated upon in the next section. The models were implemented and compared using the data available. The CNN based solutions proved to outperform the other architectures with the spatial transform module improving on the result. The surprising effectiveness of 85% of the "InnerSquare" simple solution was a testament to the invariant that if an image contained a particle, the image was centred around the image. As such the result may not have been so surprising. In conclusion the CNN-Spat proved itself to be the most effective at making the distinction between particle containing and non containing images by obtaining a 95% accuracy with only 50 epochs of training. This conclusion comes with the caveat of added difficulty during training. This is a reference to the experimental section where multiple training runs were not able to minimize the loss function as shown in table 4.14.

6.2 Perspective and future works

Looking forward the obvious next step is acquiring more Cryo-EM data. Obtaining more data from different particles would allow for a more thorough experimental process with regard to the specific data pipeline shown in figure 1.2. Given multiple data sets of different particles answering the question generalization from known particles to unknown particles as touched on in chapter 5 becomes possible. This is a problem that if answered will deepen the understanding of how to optimize the data pipeline stated in [Wan16] and shown in figure 1.2. Having access to data consisting of two additional unique particles would allow for the following experiments described in sections 6.2.1 and 6.2.2
6.2.1 Initial generalization

The natural extension of the experiments performed in this project is testing how well the models initially generalize from training on known particles to classification on unknown particles. With a data set consisting of images each containing one of 3 different particles and images containing no particle. This line of experiments should split the data in 5 sets each belonging to class "particle A", "particle B", "particle C", "empty training", and "empty test". These experiments should be executed as follows:

- For each of the particles build a training set consisting of the other particle data sets and the "empty training" set
- Train every model on the training set and evaluate the accuracy of the model prediction on the test set consisting of the data connected to the chosen particle and the "empty test" data
- Repeat for every particle

The purpose of this line of experiments is to understand which models handle the initial generalization of detecting an un-trained particle when compared to empty images. This understanding should help create a baseline to measure how much the effect of further training on the results from the initial classification has.

6.2.2 Iterative learning

This line of experiments would expand on the "Initial Generalization" experiment. These experiments would follow the flow of image 1.2, utilizing an iterative training approach. After each round of classification, the resulting classifications would be used to re-train the model with the result as new training set. The purpose of this is to take a generalized model and through iterative training teach the model to recognize more specifically the current particle in relation to background noise.

6.2.3 Expectations

The 2 experiments described above would be interesting to compare. Primarily to investigate how much is won by iterating over the training on the output of the classification results. This would help in understanding which model is best equipped to take advantage of this process. Another interesting investigation is the effect of how similar the particles appear when the resolution is as low as the images shown in 1.3. If the particles are similar, either due to inherent resemblance or forced similarity by low resolution images, the expectation is that only minor improvement will be observed when re-training in the "Iterative learning" line of experiments mentioned above.
6.2.4 Expanding the problem

This project has been concerned with classifying images that either contain a particle at the center, i.e. the positive case or contains no particle, i.e. the negative case. This however does not fully represent the problem associated with particle picking. The process proposed in [Wan16] utilizes a sliding window technique as demonstrated in figure 6.1. As a consequence of this the classification problem of each window position increases in complexity from that of this project. The objective will not have changed from identifying window positions that contain a particle at the center. However the negative case now covers window positions either not containing a particle or containing a non-centered particle as shown in figure 6.2. This picture simulates the possible cases the sliding window can assume on a Cryo-EM grid. The cases can then be described as:

- A positive case and a negative case, where the negative case covers non-centered particles in addition to empty images
- A positive case, a negative case, and an additional “misplaced” case increasing the number of classes present in the problem to 3 from the initial 2 covered in this project

If time permitted, investigating how the models described in this project performed on data with these characteristics would be relevant to practical application of the models.

Figure 6.1: Image created by merging sample images from the supplied Cryo-EM data set. Illustrating a sliding window.
Figure 6.2: Image created by merging sample images from the supplied Cryo-EM data set. Green squares indicate windows centered on particles. Blue squares indicate windows not containing any part of a particle. Red squares indicate windows that contain some part of particle but is not centered on the particle.
Bibliography


