UNIVERSITY OF BIRMINGHAM

DEPARTMENT OF PHYSICS AND ASTRONOMY

Using Machine Learning to Investigate Rotations in HAADF-STEM Images of Large Nanoparticle Structures

Joe Laurie

Supervisor: Wolfgang Theis

Friday 5th April, 2024

Abstract

The aim of this project is to label the orientation angle of a nanoparticle based on its projection, using a purely artificial dataset which simulates results produced by High-Angle Annular Dark Field Scanning Transmission Electron Microscopy. Two methods using stereo imaging were trialed. One method used a convolutional neural network to find the angle of each shape and compare their difference against the known stereo angle which separated the images. The model was successful with images of squares as input data for stereo angles from 3° to 35°. With a 10° stereo angle, it achieved an accuracy of 100% at predicting the angle within 2 degrees of the truth value. For this stereo angle, the mean error between the predicted and truth angles was 0.038° with a standard deviation of 0.441°. The model was unable to train with 1D projections of 2D shapes, and did not attempt to train on 2D projections of 3D shapes. The second method used a modified U-Net structure to try and reproduce the stereo partner image by use of the primary image and the stereo angle. It was not successful. The possible reasons are discussed within. The models were made and trained in Python using Keras and Tensorflow [21]. The data was also made in Python.

1 Background & Aim

Nanoparticles, tiny structures ranging from 1 to 100 nanometers in size, hold immense promise across diverse fields due to their unique properties[9][23]. These minuscule entities exhibit distinct physical, chemical, and biological characteristics, differing significantly from their bulk counterparts[15]. From targeted drug delivery systems in medicine to enhancing catalytic reactions in materials science, nanoparticles' versatility is reshaping industries. Nanoparticles' unique properties are caused by: high surface area to volume ratio; low size, making quantum effects more prominent; defects; and structure/shape. Their low size, and highly varied shape, makes the study of nanoparticles a challenging prospect.

Traditionally, electron microscopes are utilised for atomic scale imaging because the electrons used have shorter wavelengths than the photons of the visible spectrum used in optical microscopy, providing greater resolution[4][16][25]. Improvements in electron microscopy have brought about significant advancement in nanoscale imaging and analysis across various scientific fields, with increased resolution in images of atomic and biological structures providing greater insight into the workings of matter. High-angle annular dark-field imaging (HAADF) is a technique used in STEM[13][12] that measures electrons incoherently scattered by the sample at high angles relative to the incident beam. Rutherford scattering from atomic nuclei causes the technique to be sensitive to both the Z value, and number, of atoms in the beam path[25]. Sample thickness therefore corresponds to a higher number of scattered electrons. The images produced show the projection of the sample, along the beam axis, with the thickness in grey-scale. The image appears like a silhouette of the sample, with thick and dense regions being brighter while thin parts are darker, figure 1.1. Their sensitivity to structural differences makes them particularly suitable for analysing the structure of nanoparticles.

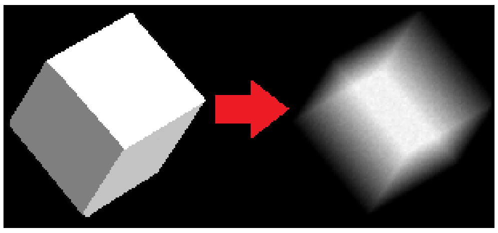


Figure 1.1: A representation of a cubic nanoparticle and its projection.

Supervised machine learning[20][11] is a method of machine learning where a model is trained on a dataset containing labelled target values for each input value. During training, the model learns to map input features to correct output labels, which are known as the ground truth. The model's parameters are adjusted iteratively to minimize a loss function[22][24], which measures the difference between the model's predictions and the actual target values. In the inference phase, a trained model can make predictions on unseen data by extrapolating patterns learned during its training phase. Supervised learning is mostly used for classification and regression tasks, the former involving categorisation of input data into one of several predefined classes or categories. The output value is categorical, meaning it represents a class or label. For example, identifying handwritten alphanumerical digits. Regression tasks a model with predicting a continuous numerical value based on input features. The output value is quantitative, making it useful for predicting numerical values, such as temperature, pricing or angles.

The primary aim of this project is to develop a method for identifying the angle at which a nanoparticle is oriented from the output of an electron microscope alone. If there was an easy way to find the labels then the project would not be necessary. However, the lack of a labelled dataset creates an unfortunate problem when trying to source a dataset for the model to train on, as the ground truth cannot be accessed by the model for calculating loss[22][24]. In other words, it is impractical to use a dataset where the labelled truth value of each input image is the angle of orientation. Instead, the model will be trained on stereoimages, where the stereo angle can serve as a proxy for the ground truth.

Stereoimaging involves capturing two images of the same object or scene from slightly different perspectives. These images, known as a stereo pair, mimic the binocular vision of human eyes, enabling depth perception. In the context of neural network training, stereoimaging provides essential data for supervised learning, allowing networks to infer spatial relationships and orientations without explicit angle information. The stereo angle, or the known angular offset of the images within a stereo pair, allows the network to map the angle space of the object it is imaging by comparing similar images in the context of their known separation. The stereo angle can be assumed to be 10° unless otherwise stated. Stereo imaging enhances the network's ability to analyze and interpret complex structures and shapes accurately. In order to analyse said images, a specific form of neural network is required.

Convolutional Neural Networks (CNNs) are a class of deep learning models designed for visual processing tasks. Inspired by the structure of the visual cortex, CNNs employ convolutional layers to extract hierarchical representations from input data through the application of convolutional filters[3]. Convolution allows a network to effectively learn spatial hierarchies of features, making them highly proficient in tasks like image recognition, object detection, and image classification. CNNs have revolutionized computer vision and are widely used in diverse applications, from autonomous vehicles to medical imaging. The project has tested two ways of using CNNs to determine the orientation angle of nanoparticles in input images.

In machine learning, loss measures the discrepancy between predicted and actual values during training. It quantifies how well a model's predictions match the ground truth labels. Common loss functions include mean squared error (MSE) for regression tasks and cross-entropy for classification tasks[24]. By the minimisation of loss through optimization algorithms like gradient descent, a model learns to make more accurate predictions and improve its performance on unseen data.

Created by Olaf Ronneberger, Philipp Fischer, Thomas Brox in 2015 and published in the paper "U-Net: Convolutional Networks for Biomedical Image Segmentation"[18], The U-Net architecture[19][10] is a type of convolutional neural network commonly used for image segmentation tasks, particularly in biomedical image analysis. Similar to autoencoders, it consists of a contracting path (encoder) to capture context and a symmetric expanding path (decoder) for precise localization. U-Nets excel in tasks demanding accurate and detailed segmentation, such as identifying specific organs or structures in medical images with high precision and reliability. An important feature, which separates them from simpler autoencoder networks, are the skip connections. Due to their symmetry, U-Nets are able to concatenate information from layers in the contracting path to corresponding layers in the expanding path of the same dimension. These connections, for which the structure gets its name, allow the preservation of fine grain details during the upsampling process and retention of spacial information in the original image.

Tomography is an imaging technique used to create cross-sectional images of objects by reconstructing their internal structures from a series of projections taken from different angles[5][7]. By taking an inverse Radon transformation, most commonly done by filtered back projection, it is possible to reconstruct the original object's attenuation or electron scattering profile [1]. The technique involves passing each projection through a filter to enhance contrast and remove artifacts, then back-projecting the filtered projections onto a common plane. By summing these back-projections from all angles, a 2D or 3D image representing the object's internal structure with high spatial resolution is obtained. This is often vital in medical diagnostics and material science.

Now that the key topics have been introduced, the aim of this project can be explained. The primary objective of this research project is to produce a neural network model that can detect the orientation angle of a nanoparticle imaged by a high angle annular dark field scanning transmission electron microscope. The training of the machine learning model must be done without use of the truth values, the correct angles for each image. In order to do this, the model will use stereo images to extract the angle information from the known stereo angle. Unfortunately, this cannot be done immediately, because there are many complications to the process. So, first, the model must be validated by success in simpler tasks. These tasks will be designed to expose the model to the aforementioned complications in stages so that solutions to the problems they cause may be found. The first stage will be to recognise the angle of a shape before it has been projected. Two ideas for a model were had, and the first task was used to test them in comparison. They both use two of the three pieces of available information to guess the third, those being two stereo images and the stereo angle. This is where the project began...

2 Methods

2.1 Data

Construction of a suitable training dataset was done in Python. Although the dataset was modified at different stages of the project, it followed the same general structure. Namely, to create 64 by 64 arrays and populate them with 1s and 0s to represent the pixels showing a target sample and background from an electron microscope. The sample was defined by taking cuts at certain distances from a chosen central coordinate, to define the edges of the shape, and filling the internal region with 1 values. The central coordinate, size and orientation of these samples were randomised to increase the total number of possible pixel configurations, which increases the size of the available dataset. A note - radians have been used as the unit of angle measurement in this project, although this report will frequently refer to degrees to aid readability. If the model trains too much, it will memorise the dataset and be unable to generalise its solutions to unseen data; this is known as overfitting [17]. Overfitting can be prevented by having a large and varied dataset. There are only so many orientations than be photographically expressed by pixels that are either on or off, so the variation of the sample size allows a larger variety of images to be fed into the model when training. The data also has noise applied to it (values between 0 and 1 were added to the array). There is a uniform random noise applied across the whole sample to represent the carbon black background layer that supports the sample in the real electron microscope.

The data is passed through a preprocessing step where the image is rolled to the center of the frame, by measure of its center of mass and offset on the axes of the image, and the noise is removed to attempt to increase the signal to noise ratio and "clean" the dataset. This makes the learning process easier as the model does not need to pay attention to the extremities of the image, they will remain unpopulated with data (they will be zero valued).

Using an artificial dataset means the data comes with an associated ground truth; the orientation angle used to generate the image can be kept in the data and used later for validation of the model's predictions. This offers a large advantage over lab data as it can be used to debug the code and also can be used for testing methods and ideas when designing the models. It is also economically unfeasible to use experimental data. I have been advised it would cost in the order of £10,000 for a day's use of an automated STEM, which would be a unjustifiable cost just to find, while training, that a different stereo angle would have been better.

2.2 The Difference Model

The first of the two methods used, the 'Difference Model', tries to calculate the stereo angle. Both images are passed into the same convolutional neural network which is tasked with outputting the orientation angle. It is a submodel within the larger model which takes the outputs of the CNNs and calculates their difference, hence the name. This is then compared to the stereo angle to calculate the loss. For the Difference Model, the loss used was MSE.

The convolutional neural network, pictured in figure 2.1, is built from twelve layers. The first nine layers are three repeated blocks of: a convolution layer, a max pooling layer, and a dropout layer[2][8]. The convolutional layers are intended to extract features by passing filters over the input images, and then create outputs called feature maps. The max pooling layers reduce the image size, and so may be thought of as compressing the data. This is done by taking maximal values (as opposed to multiplying matrices, as done for convolution) within filters, which are passed across the image inputs. The dropout layers randomly set inputs to zero, to prevent overfitting[17]. They reduce the reliance on individual neurons within the network and prevent the network from forming unhelpful habits when it finds local solutions. After these nine layers, the image size has been reduced from one 64*64 to sixteen 6*6 images known as feature maps. They are then flattened; the layered 2D tensors are reshaped into a single, 1D tensor, without adjusting the values. This allows the network to connect to its next two dense layers. These are fully connected layers and extract all the data learnt from the image into the final two output values.

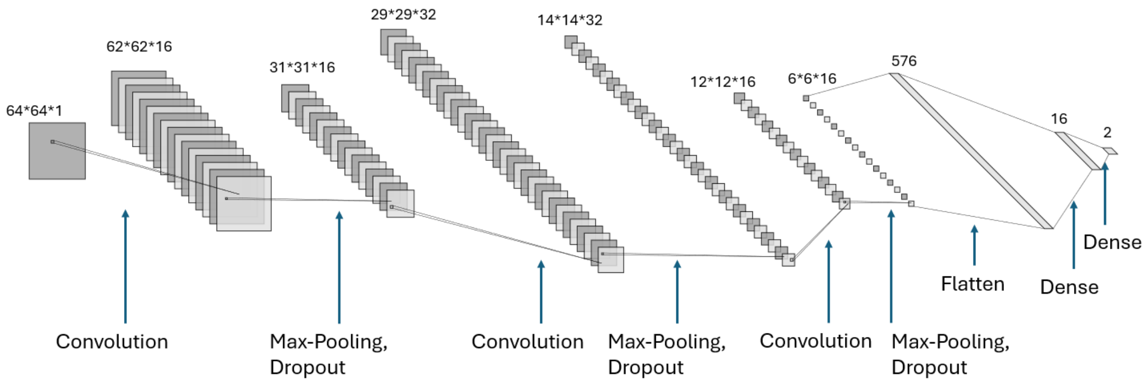
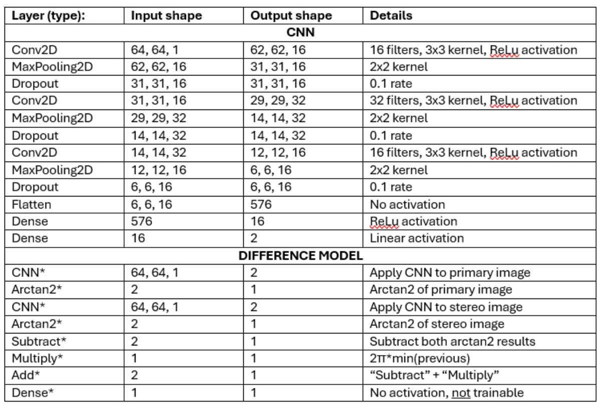


Figure 2.1: The structure of the convolutional neural network sub model used in the Difference Model for 2D input data.



Layer (type):	Input shape	Output shape	Details
		CNN	
Conv2D	64, 64, 1	62, 62, 16	16 filters, 3x3 kernel, ReLu activation
MaxPooling2D	62, 62, 16	31, 31, 16	2x2 kernel
Dropout	31, 31, 16	31, 31, 16	0.1 rate
Conv2D	31, 31, 16	29, 29, 32	32 filters, 3x3 kernel, ReLu activation
MaxPooling2D	29, 29, 32	14, 14, 32	2x2 kernel
Dropout	14, 14, 32	14, 14, 32	0.1 rate
Conv2D	14, 14, 32	12, 12, 16	16 filters, 3x3 kernel, ReLu activation
MaxPooling2D	12, 12, 16	6, 6, 16	2x2 kernel
Dropout	6, 6, 16	6, 6, 16	0.1 rate
Flatten	6, 6, 16	576	No activation
Dense	576	16	ReLu activation
Dense	16	2	Linear activation
		DIFFERENCE MODEL	
CNN*	64, 64, 1	2	Apply CNN to primary image
Arctan2*	2	1	Arctan2 of primary image
CNN*	64, 64, 1	2	Apply CNN to stereo image
Arctan2*	2	1	Arctan2 of stereo image
Subtract*	2	1	Subtract both arctan2 results
Multiply*	1	1	2π*min(previous)
Add*	2	1	“Subtract” + “Multiply”
Dense*	1	1	No activation, not trainable

Figure 2.2: Full details of the Difference Model.
"Layers" marked with an asterisk are not layers, but are rather the steps used in the model.

2.3 U-Net Model

Unlike the first method, the second does not use both images as input. Instead, a U-Net structure is used to try and recreate the stereo-partner image. The first image is passed as input to the network, and is reduced in size through convolution and maxpooling layers[8] in the contracting path of the U-Net. This allows the network to find the orientation of the shape, much like in the other model. At this point, the stereo angle is added to the calculated orientation angle to produce the orientation angle of the second shape. This new angle is then used to generate an image of the stereo partner. The estimated value moves up the expanding path of the U-Net and, through convolution and upsampling, is formed into a 64*64 pixel image. This output is then checked against the labelled truth value, which is the stereo partner image. The loss is calculated as the mean square error in pixel values between the two arrays. Once trained, it is possible to separate the contracting path and use this as a trained convolutional neural network for the prediction of angles on unseen data.

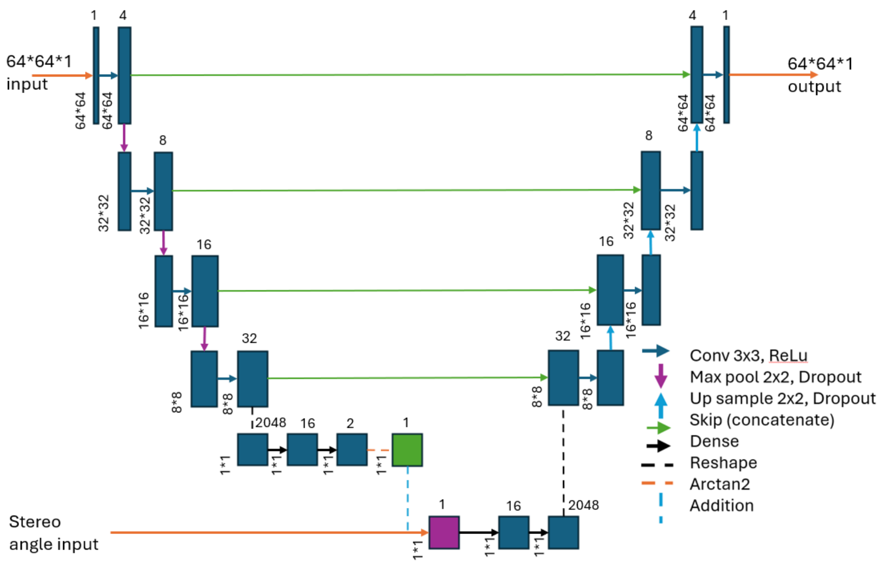
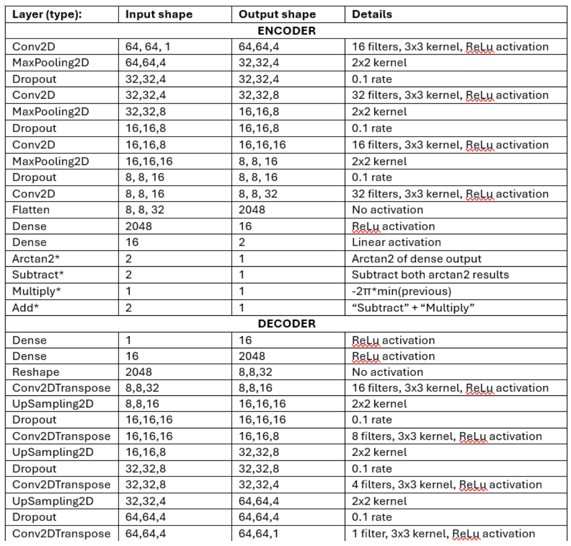


Figure 2.3: U-Net model structure. The contracting path goes down from the input image to the green box, which indicates the desired output of an orientation angle. The expanding path then goes upward from the purple box, which is the calculated angle of the stereo partner, to the output image in the top right.



Layer (type):	Input shape	Output shape	Details
		ENCODER	
Conv2D	64, 64, 1	64,64,4	16 filters, 3x3 kernel, ReLu activation
MaxPooling2D	64,64,4	32,32,4	2x2 kernel
Dropout	32,32,4	32,32,4	0.1 rate
Conv2D	32,32,4	32,32,8	32 filters, 3x3 kernel, ReLu activation
MaxPooling2D	32,32,8	16,16,8	2x2 kernel
Dropout	16,16,8	16,16,8	0.1 rate
Conv2D	16,16,8	16,16,16	16 filters, 3x3 kernel, ReLu activation
MaxPooling2D	16,16,16	8, 8, 16	2x2 kernel
Dropout	8, 8, 16	8, 8, 16	0.1 rate
Conv2D	8, 8, 16	8, 8, 32	32 filters, 3x3 kernel, ReLu activation
Flatten	8, 8, 32	2048	No activation
Dense	2048	16	ReLu activation
Dense	16	2	Linear activation
Arctan2*	2	1	Arctan2 of dense output
Subtract*	2	1	Subtract both arctan2 results
Multiply*	1	1	-2π*min(previous)
Add*	2	1	“Subtract” + “Multiply”
		DECODER	
Dense	1	16	ReLu activation
Dense	16	2048	ReLu activation
Reshape	2048	8,8,32	No activation
Conv2DTranspose	8,8,32	8,8,16	16 filters, 3x3 kernel, ReLu activation
UpSampling2D	8,8,16	16,16,16	2x2 kernel
Dropout	16,16,16	16,16,16	0.1 rate
Conv2DTranspose	16,16,16	16,16,8	8 filters, 3x3 kernel, ReLu activation
UpSampling2D	16,16,8	32,32,8	2x2 kernel
Dropout	32,32,8	32,32,8	0.1 rate
Conv2DTranspose	32,32,8	32,32,4	4 filters, 3x3 kernel, ReLu activation
UpSampling2D	32,32,4	64,64,4	2x2 kernel
Dropout	64,64,4	64,64,4	0.1 rate
Conv2DTranspose	64,64,4	64,64,1	1 filter, 3x3 kernel, ReLu activation

Figure 2.4: Full details of the U-Net Model. The encoder is the separable model for finding the angle. "Layers" marked with an asterisk are not layers, but are rather the steps used in the model.

2.4 Reading Angles & Rotational Symmetry

In order to test the models, they were first given a simpler task. Instead of analysing projections of 3D shapes, the models were trained on 2D shapes, namely squares. This was done to validate the general structure of the model, with a much easier task. This task is simpler as the image contains spatial information from which the orientation can be extracted, such as the gradient of the shape's edges. The shapes were generated as described in section 2.1, with their rotation being confined to the plane of the image. This results in a 'top down' view of 2d structures, which is equivalent to images of a prysm taken along the principle axis (where the cross section does not change).

The models encountered their first major problem here, in the continuity and periodicity of the angle mapping. In order to assign values to the angle of each shape, the models need to create a one to one mapping of the input images and outputs angles. As a shape is rotated around the zero position, its orientation can quickly change from 359° to 1°. This makes sense to any person, but confuses a model as it does not understand the discontinuity. The problem is brought to the fore with stereo angles. If the primary image is at 355° and its partner is at 5°, this no longer appears to be a separation of 10°, but rather of -350°. This led to the initial training of the models being successful in the majority of its angle range, but breaking down as it approached the upper and lower limits. This problem is worsened by the use of cyclic shapes. For the moment, I will make no distinction between cyclic and dihedral shapes, although this distinction will become important later. A square is of a four-fold rotation group, D_4, meaning it has four lines of symmetry and (importantly) four equivalent angles for each orientation. In other words, the regions $0-90°$, $90-180°$, $180-270°$ and $270-360°$ are identical. The solution to this problem is to use trigonometric functions to describe the angle. Initially, the CNNs were outputting a single value (assumed to be the angle). This was then changed to be two values. By taking the arc-sine and arc-cosine of these values, it forces the models to predict the sine and cosine of the angles. This means the CNNs can produce a continuous mapping of an output value for range of input values, and therefore the range of ground truth angles.

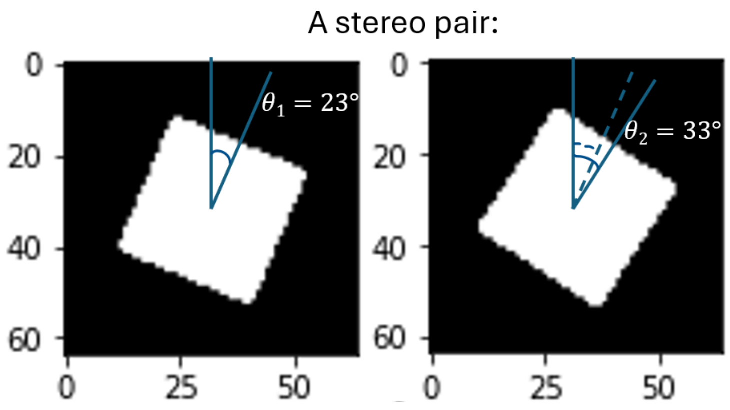


Figure 2.5: A stereo pair. Two squares taken from a "top down" view. $\theta_1 = 23°$, $113°$, $203°$ etc so the images are not unique for the whole range of [$0°, 360°$].

For the first model, the CNN is a separable sub-model which produces two outputs, S & C, these are assumed to be: $sin(n\theta)$ and $\cos(n\theta)$. The coefficient before θ is used to create n identical zones in the angle space, where n is the cyclic order of the shape (the number of rotational symmetries, or the order of rotational symmetry). For a regular n sided polygon, the shape has n rotational symmetries. The network is trained in a larger model which first applies the CNN to both input images, then finds the difference between each angle. It does this by first calculating $\theta = \frac{1}{n} arctan2(\frac{sin(n\theta)}{cos(n\theta)})$ for each angle, then subtracting them. The value of n can be found for a given shape programmatically, with a trial and error approach. A keen eyed reader may notice that this does not fully solve the problem, it is still possible to find the value of the second angle to be less than the first, if the stereo angle crosses the boundary between two equivalent regions in the angle space of the shape. This would result in a stereo angle that is negative. For a square (or any four-fold symmetric shape), the $10°$ stereo angle used in training causes the model to fail to train on angles from $80-90°$. This is around ten percent of the angle range. The erroneous calculation of the stereo angle will be $-80°$, as the model has predicted the second angle to lie in the zone before the first. This is close to ten times the correct value, except negative. These two facts cause the average predicted value to fall close to zero. With further training, model gravitates towards predicting zero for all values. To solve this problem, the final step of calculating the stereo angle is to check which predicted angle is lower and, if it is not the first, to add $\frac{360}{n}°$, where n is once again the order of the cyclic shape. In the case of a square, this is $90°$. For the second model, the contracting path (which is a very similar, separable, CNN) also outputs two values: S & C, equivalent to $sin(n\theta)$ and $\cos(n\theta)$. It then uses the same formula to calculate θ. At this point, it diverges. After adding the stereo angle, if this new value is greater than $\frac{360}{n}°$, $\frac{360}{n}°$ is subtracted to find the angle of the second image.

2.5 Inferring Angles

After a demonstrating success on the first task of reading angles and rotational symmetries, and showing that a solution could be found to the associated problems, the first model was reworked to be used on projected images. Please see section 3.2 to understand why the U-Net was not applied to this step of the process. The next challenge for the model was to train without the ability to directly read the angle from the image, this meant the use of projected data. To assess the viability of the model, it was trained on a simpler dataset. This involved taking two dimension shapes, as before, and then projecting them into a 1D array. Summing the arrays along their y-axis left only an x-axis of values, with each pixel now having the value of the sum of the y-axis for a given x-axis pixel. At this point, dihedral shapes, or shapes with mirror planes through them, presented a new problem. As can be seen in figure 2.6 below, these shapes do not produce a unique projection for every orientation. A trained model cannot take a single image and know what angle it corresponds to, some information is being lost in the projection step that cannot be recovered.

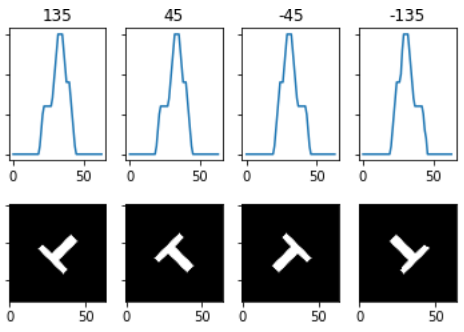


Figure 2.6: Projections of a 'T' shape at $135°$,$45°$,$-45°$,$-135°$
. Below are the original shapes and above are the corresponding projections.

Instead of using squares, the image projected was the 's' shape found in figure 2.7. This was a shape designed so that it did not have any symmetries after being projected. This was tested by using a sinogram, figure 2.7.

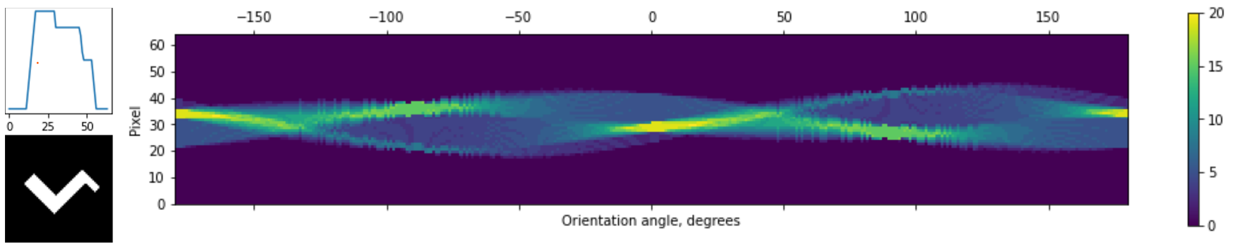


Figure 2.7: LEFT: 'S' shape. The shape below was projected along its y axis to produce the linear intensity above. Each x-axis value in the projection is the total number of white coloured pixels along the y-axis at each corresponding x-axis value below.
RIGHT: Sinogram of 'S' shape, made using ground truth labels with one data point at each x value.

In HAADF STEM, a sinogram is a visual depiction of the intensity of electrons scattered by a specimen at various angles. Each column in the sinogram represents a distinct angle of electron scattering, while each row corresponds to a specific position along the detector array. In our case these are the orientation angles and pixels of the image. Sinograms are crucial for reconstructing high-resolution images of the specimen, using mathematical algorithms like filtered back projection [1]. By constructing a sinogram using the ground truth labels (known orientation angles), it is possible to check if the projections are unique for each angle. As can be seen in figure 2.7, there are no two orientations with the same output projection. The instability in the data is caused by the pixelation of the projected images. By summing along the y-axis of the arrays, the stepping of diagonal lines in the source images has been transferred to the projections. The sinogram above produces a

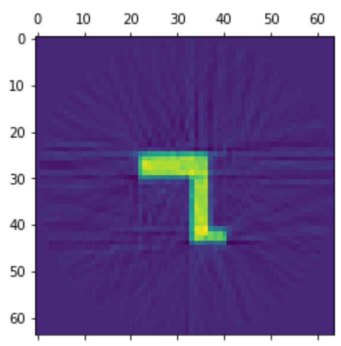


Figure 2.8: Sample reconstruction using inverse radon transformation of data from the sinogram 2.7. The noise around the sample is due to the defects in the sinogram caused by pixel jumping in the 'S' shape images.

reconstruction of the sample image after filtered back projection as below, figure 2.8. This was done using the skimage.transform.iradon function from Scikit-Image. The Difference Model had to be modified to work with 1D input data. This largely constituted changing the 2D layers to their 1D equivalents[8]. It also resulted in a smaller number of dense connections toward the end of the model. The model has the same form, which can be seen by comparing figure 2.1 and figure 2.9.

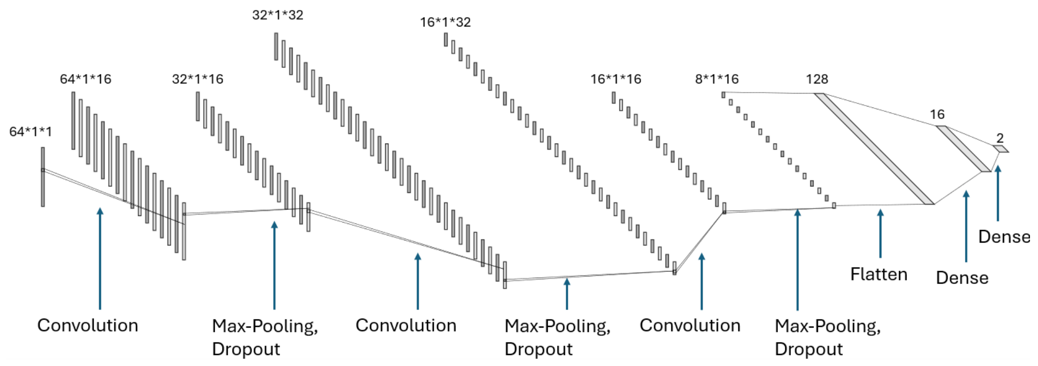
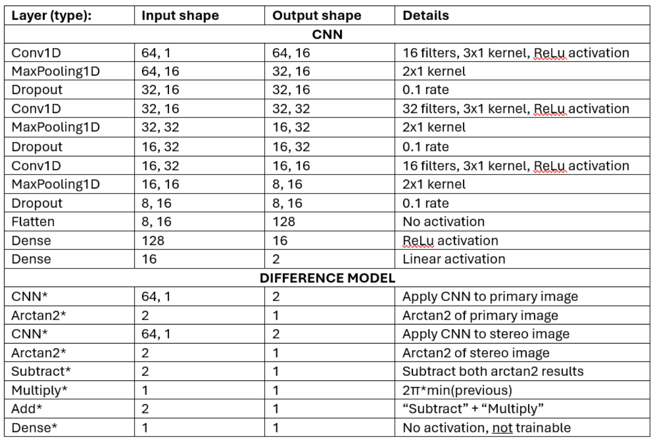


Figure 2.9: The structure of the convolutional neural network sub model used in the Difference Model for projection input data.



Layer (type):	Input shape	Output shape	Details
		CNN	
Conv1D	64, 1	64, 16	16 filters, 3x1 kernel, ReLu activation
MaxPooling1D	64, 16	32, 16	2x1 kernel
Dropout	32, 16	32, 16	0.1 rate
Conv1D	32, 16	32, 32	32 filters, 3x1 kernel, ReLu activation
MaxPooling1D	32, 32	16, 32	2x1 kernel
Dropout	16, 32	16, 32	0.1 rate
Conv1D	16, 32	16, 16	16 filters, 3x1 kernel, ReLu activation
MaxPooling1D	16, 16	8, 16	2x1 kernel
Dropout	8, 16	8, 16	0.1 rate
Flatten	8, 16	128	No activation
Dense	128	16	ReLu activation
Dense	16	2	Linear activation
		DIFFERENCE MODEL	
CNN*	64, 1	2	Apply CNN to primary image
Arctan2*	2	1	Arctan2 of primary image
CNN*	64, 1	2	Apply CNN to stereo image
Arctan2*	2	1	Arctan2 of stereo image
Subtract*	2	1	Subtract both arctan2 results
Multiply*	1	1	2π*min(previous)
Add*	2	1	"Subtract" + "Multiply"
Dense*	1	1	No activation, not trainable

Figure 2.10: Full details of the Difference Model, after modification for use with projections. "Layers" marked with an asterisk are not layers, but are rather the steps used in the model.

3 Results

3.1 Difference Model, Top Down

Before training the Difference Model, it was important to check if the model was capable of succeeding. When devising machine learning models, the first consideration is usually the outcome. 'What do I want my model to do?'. In this case, it is to find the orientation angle of an input image, or rather, its projection. After this, the model is designed. Before it can be trained, one must check the model design is capable of producing the desired outcome. There must exist a minimisation of the loss function, or an optimisation of the hyperparameters, for the model to find in order to succeed. If the model finds a most successful method of reducing its loss, it will strive to do this regardless of the needs of the designer which can often lead to errors caused by the model's design.

The first stage of training the models was to separate them into components, and check that the components functioned as expected. For the Difference Model, this meant testing the CNN component on the ground truth, "cheat training". This was to make sure that, given the correct answer, it could find a way to train. This involved training the CNN as its own model, with the truth labels being the ground truth, on a small angle range of $0° - 60°$. This was done to prevent the model from encountering the problem of wrapping around boundary zones in angle space. This was successful. After trying to expand the angle range to $90°$, the model began to break down within 10 degrees of the end. This problem was expected, although a solution had not been found. At this point, the model did not use the sine and cosine as outputs, rather just the angle itself. This change was made in response to seeing the model consistently fail within the $10°$ stereo angle of the boundary, $\frac{360}{4}°$. Then the model was trained using the full angle range. This was a success and the model was able to perform reasonably accurate predictions on unseen data (after training).

The next step was to check the additional components in the Difference Model. First, the CNN was pretrained on the ground truth, as above, and then the Difference Model was trained (on the stereo angles). The Difference Model includes the CNN but has additional steps in its process. This revealed another issue, caused by using sine and cosine. When the stereo angle crossed the boundary at $90°$, the difference was no longer calculated correctly. This was solved by the addition of the final step of the model, the addition of $\frac{2\pi}{4}$ if the stereo angle crossed the boundary between periodic zones of angle space. This prevented the frequent guess of an $-80°$ stereo angle. Which, after only a small number of epochs (on the scale of 10-20), would result in the stereo angle prediction going to zero, ruining the pretraining. This is for the reasons explained in section 2.4. After this change the model worked.

Once it was clear the model worked, the next challenge was to train the Difference Model without using the ground truth, which is much more difficult. The model cannot rely upon being shown where in its parameter space to begin guessing so will struggle to find the correct angles. Initial training attempts were unsuccessful, the model did not find any easy route to minimise its loss, and would frequently predict the angles to all be the same, giving a stereo angle prediction of zero. I believe this is because the stereo angle, 4*0.17≈0.68 radians, is very close to zero, which is a much easier answer to find. The next step was to train the model using an expanding angle range. The previous cheat training had shown the model was able to train more quickly (and therefore easily) with smaller angle ranges. In order to test this idea, the model was trained using a subsection of the dataset, which was expanded in stages to the full angle range. This method was successful, but still relied on using the ground truth, albeit in a less direct manner. The results of this training are in figure 3.1 and figure 3.2. As can be seen, the model is able to identify a mapping for each input angle to a corresponding output angle. The final trick for success was a method of identifying shapes in a similar angular range, without checking using ground truth labels.

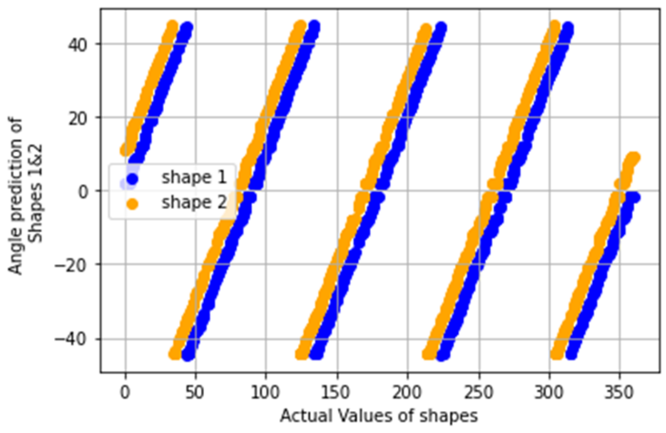


Figure 3.1: CNN predictions after using the ground truth to train on an expanding angular range. The angle prediction is found using the formula $\theta_p = \frac{1}{4}arctan2(\frac{S}{C})$, where S and C are the outputs of the CNN.

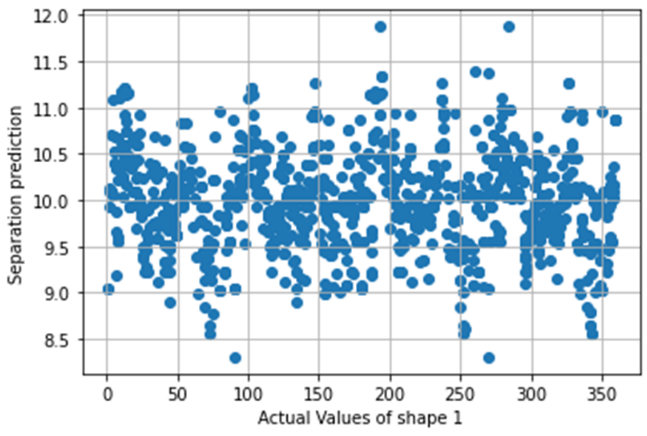


Figure 3.2: Difference Model predictions using the ground truth to train on an expanding angular range. The raw values have been divided by 4.

After selecting a random image, the standard deviation of the dataset from this image can be calculated as in figure 3.3. This allowed for choosing images with standard deviation below a certain threshold. As can be seen in figure 3.3, this corresponds to choosing images that all exist in a small angular range. By choosing values below appropriate thresholds, training was conducted over an increasing percentage of the angle range. This was done for thresholds of approximately: 2%, 10%, 25%, 33% and 100%, each for 5 epochs, with a batch size of 64 and 4096 images. Then the model was trained for an additional 5 epochs on the full dataset. The predictions done after this training are found in figure 3.4.

Figure 3.4 shows the predictions of the orientation angle extracted from the CNN in the Difference Model. The fitting function (red) has used a modulus as the predictions are periodic, with this particular trained model happening to equate $\theta_p = 0°$ with $\theta_t = 0°$. As will be seen in section 4.2, this does not always happen. This is due to the model not needing to understand the same concept of the original position, which for the viewer is usually the square with its edges parallel to the axes of the image. This can cause an arbitrary offset which can easily be removed by finding the value of θ_p at $\theta_t = 0°$, which, once again, will be shown later.

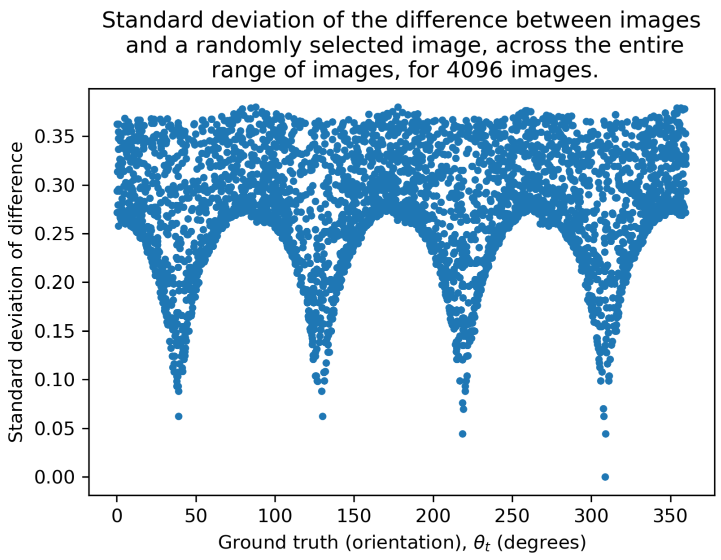


Figure 3.3: The standard deviation from a random, unknown primary image, for the entire dataset of primary images, compared to their ground truths.

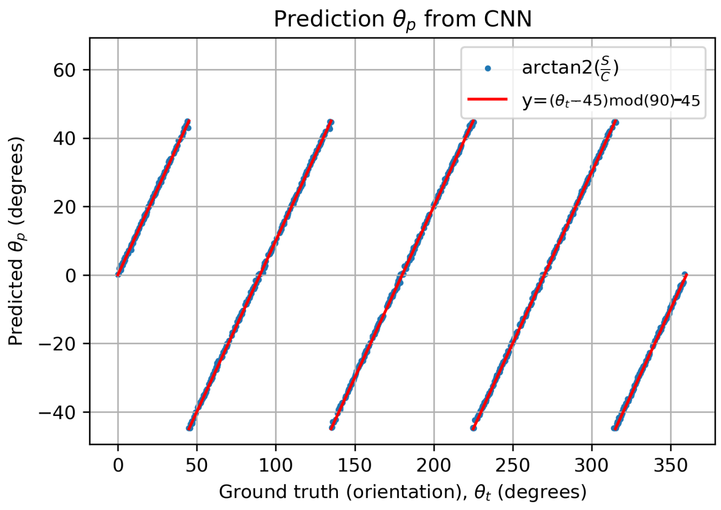


Figure 3.4: Predicted angle, θ_p, compared to known ground truth angle, θ_t, for an unseen dataset.

3.2 U-Net Model, Top Down

The two models were designed and trained in parallel, while the Difference Model was successful the U-Net was not. Trial versions of the model seemed unable to predict the images correctly. In the final form it reached, figure 2.3, the model was still not able to produce acceptable results. I believe this is due to the skip connections. They seemed to allow the model to minimise loss without using the bottleneck of the angle prediction, as it was able to create images that were on the right track with angle predictions that were completely wrong. It also took much longer to train this model than the Difference Model, which may be due to the creation of larger sized data inciting computation speed constraints. The model also had significantly more parameters which would have slowed it down too. While this model was failing to produce results, the first model was succeeding. Consequently, a plan for the next stage of the project was developed. At this point it became clear that this U-Net model would not be able to work with the projected data. If the data was taken from shapes with any plane/line symmetries, then the angle predicted is not unique anymore, with the stereo angle addition causing the model to move in different directions in angle space for different primary angles, as can be seen in figure 3.5. The right image shows the symmetric nature of the angle space for shapes with spatial symmetry, as with figure 3.3. This would prevent a given image being associated with a specific angle. A useful solution would be to somehow write this symmetry into the model design. This would not work for the U-Net, for reasons made clear by the left image of figure 3.5. After adding $10°$ to the angle found from the projection of $-90°$, we want the model to create the image for $-80°$, however this is also the output wanted from the input projection at $-110°$. The model is not able to train its CNN to find the angle if the total process is not unique for every input. This means that both the primary and stereo partner projections would be required to establish a location in angle space, and the model would need to generate the shape of a third image to function as intended. Ultimately, the model design did not work. At this point, the U-Net model was declared to be no longer useful and development ceased.

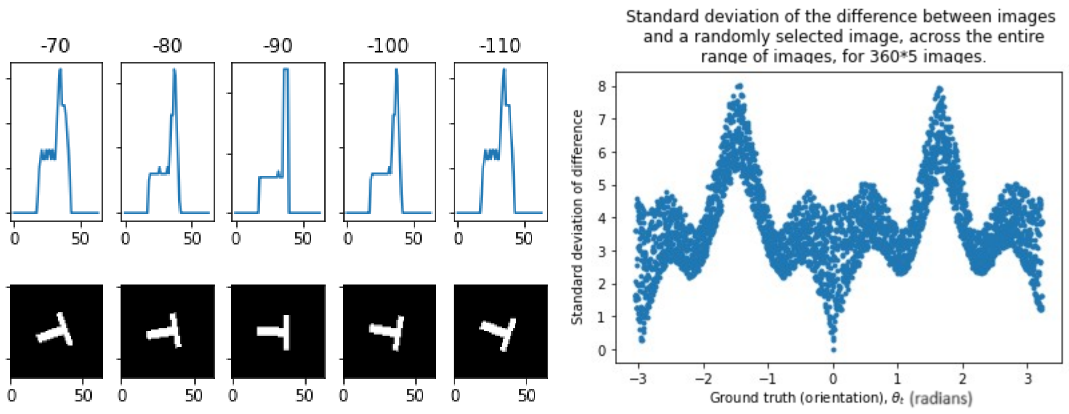


Figure 3.5: LEFT: Projections of ’T’ shape around $90°$. Note - $-70°$ and $-80°$ are equivalent to $-110°$ and $-100°$, respectively.
RIGHT: The standard deviation from a random, unknown primary projection, for the entire dataset of primary projections, compared to their ground truths.

3.3 Difference Model, Projected S Shape

The projected shapes proved more challenging for the Difference Model, when tested on an expanding angle range with the ground truth angles used as the labels for data training, it was only moderately successful. The minimum loss achieved was an order of magnitude higher than training done on 2d, top down view, shapes. Unfortunately, the method of training used in section 3.1 could not be used for the projections. This is clear from figure 3.6, there are no local minima which allow a significant portion of the dataset to be accessed by cutting the dataset at a y-axis value below this minima without entering a new angular region. In other words, the minima are too closely packed to use the global minimum for training with increased values of the standard deviation as done before. Using the ground truth, the standard deviation was checked for every angle that could be randomly chosen. This ruled out the method completely as the maxima between the orange and red lines is never less than the minima at the yellow line. If it had been, then the global minimum could have been used to slowly expand the angle range by moving to said maxima. Unfortunately, it is not possible to smoothly expand the angle range using a randomly selected image without creating two disconnected angle ranges that the model is training on simultaneously. This figure contains the red to orange maximum that is closest to the yellow minimum, meaning all other random choices would be worse yet still. If the model trains on two angle ranges simultaneously, the gradients of the predicted values will be correct in both ranges but will not join, causing an unrecoverable error on almost all training cycles. It would not be useful to keep trying until a success was found as this problem would occur again and again for every new type of shape introduced to the model.

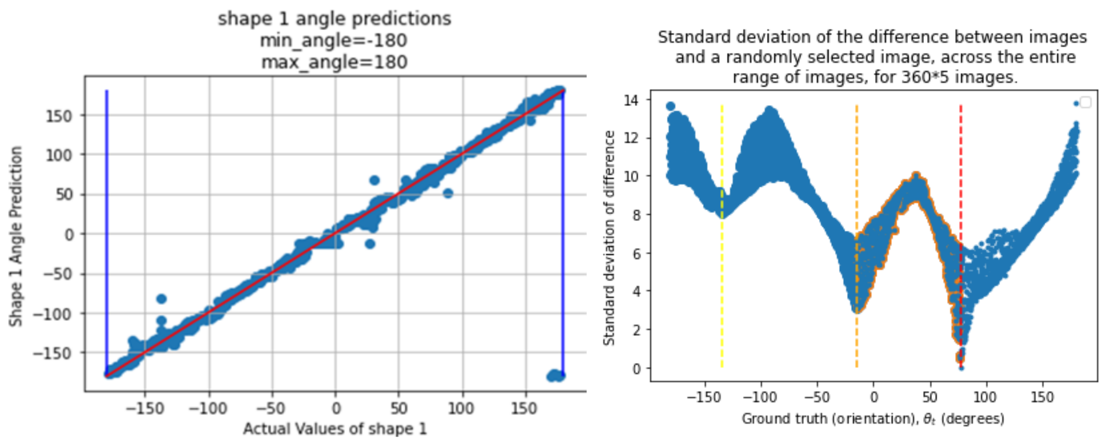


Figure 3.6: LEFT: Predicted angle, θ_p, compared to ground truth angle, θ_t after training on ground truth in expanding angle range.
RIGHT: The standard deviation from a chosen primary projection, for the entire dataset of primary projections, compared to their ground truths. The primary projection was chosen to have the least difference in similarity between the orange maxima and the yellow minima. Coloured vertical lines show position of local minima.

4 Interpretation

4.1 Difference Model, Top Down

The Difference Model is designed to output two values. There is no guarantee these will be what is wanted. That is enforced by their usage in calculating loss. By forcing an inverse tangent of the two values, it creates a requirement for them to be sine and cosine of the angle being detected. It does not however, enforce them to be normalised, only they must be normalised with respect to each other. This can be seen when examining their raw data, and by its normalisation. Figure 4.1 shows the normalised outputs of both sets of data, and also fitting functions generated by SciPy curve_fit. This uses a non-linear least squares function fitting with the chosen functions found in the images. As can be seen, they both very clearly show fits of sin(4x) and cos(4x) with only very marginal error. The only significant deviation of note is the phase of these two functions. They both deviate by a slight values, c and g. This was expected, and not problematic.

The Difference Model enforces relative normalisation of its outputs, via inverse tangent. It does not prevent angle calculation from being out of phase, as the phase of a periodic function is somewhat arbitrary in this context. Calling the sample's zero position a square with its edges parallel to the axes of the image is no different to calling it a diamond with internal diagonals both parallel to the axes. Whatever the labelling, the difference in angle between two images $10°$ apart will be $10°$. This offset can be found and removed from the data when calculating the results. Figure 4.2 shows an image of the difference in predicted and actual values of the same, unseen, dataset as in figure 4.1. Note the occasional values far from zero at roughly $\pm 80°$, these are caused by the slight phase differences $c = 0.016 \pm 0.002$ and $g = 0.024 \pm 0.001$. These incorrect values are unimportant and anomalous. The values arise from rounding errors in the modulus applied to the data. They are at values of $x_0 + 90k$, where k is an integer. The legend shows that a modulo has been applied to the predictions, θ_p, this is to account for the periodic zones in angle space and their equal output of a $0° - 90°$ range, figure 3.4. Below this, figure 4.3 shows the data, excluding the large values. The positive mean shows the predictor is not entirely accurate, with the values appearing to have some arbitrary offset from the ground truth. There are multiple factors that could have caused this. First and foremost, the images themselves are not continuously changing for a continuously changing ground truth. A 64*64 pixel screen only has so many ways to combined its pixels and show the square, especially since the pixels are either active or inactive (1 or 0). The ground truth however, is effectively continuous. It uses randomly generated values from the NumPy package so contains decimal values. The mean and standard deviation do show the average value falls reasonably close to 0, and certainly close enough for the accuracy humans operate at. There are few, if any, people who can determine a

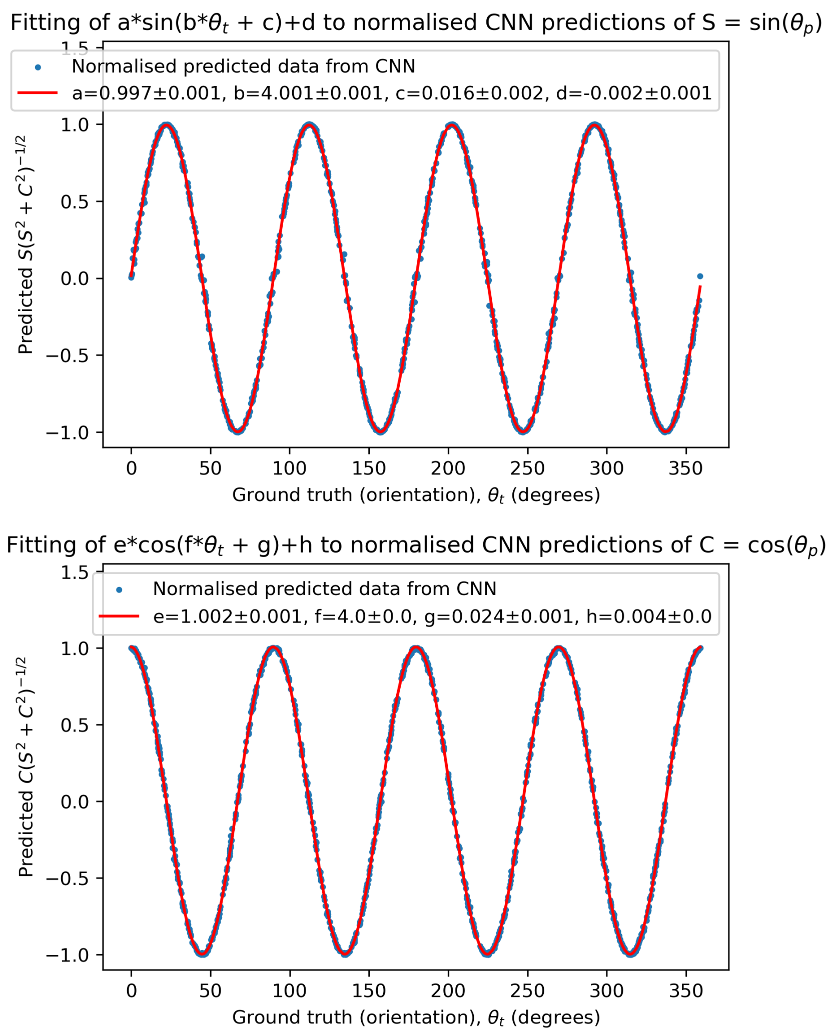


Figure 4.1: Normalised outputs of CNN predictions on unseen data with sine and cosine fitting functions.

$0.3°$ difference in angle. The phase differences in sine and cosine, c $\&$ g also play a role in this offset.

It is interesting to see that the noise in $\theta_p - \theta_t$ is not uniform. The noise appears to loosely follow a negative sinusoidal form, with the lowest average noise being close to multiples of $45°$. This is almost certainly caused by the trigonometric functions being used. It is likely caused by the calculation of θ_p via inverse tangent. Around these multiples of $45°$, sine is close to zero and cosine is close to one, which results in the tangent being close to zero. In between these values, the tangent is obviously closer to plus or minus infinity as cosine goes towards zero, which would be a likely source of this trend in the error.

4.2 Testing of stereo angles.

After success with the ten degree stereo angle, the difference model was modified to run with various stereo angles. The lowest stereo angle that was able to produce a trained model was $3°$. The highest was $35°$. Both of these values took the same amount of training as the original, section 3.1. However, the values below $5°$ and above $15°$ were significantly less likely to train. When the model is trained, it is not always successful. Some attempts result in failure; the model struggles to identify patterns in the early epochs of its training and cannot pass these on to the later stages on a larger angular range. This can be seen by the loss, at the end of an epoch the loss is shown by Keras[8], if it does not reach an appropriately low value at the end of a batch of 5 epochs, it will not be able to move on. The expanding of the angular range causes the loss to increase by one or often two orders of magnitude. If, by the end of the five epochs, the loss is not one the order of 10^{-2} or less, the model will not be able to recover once the angle range is expanded. This results in the model guessing a stereo angle

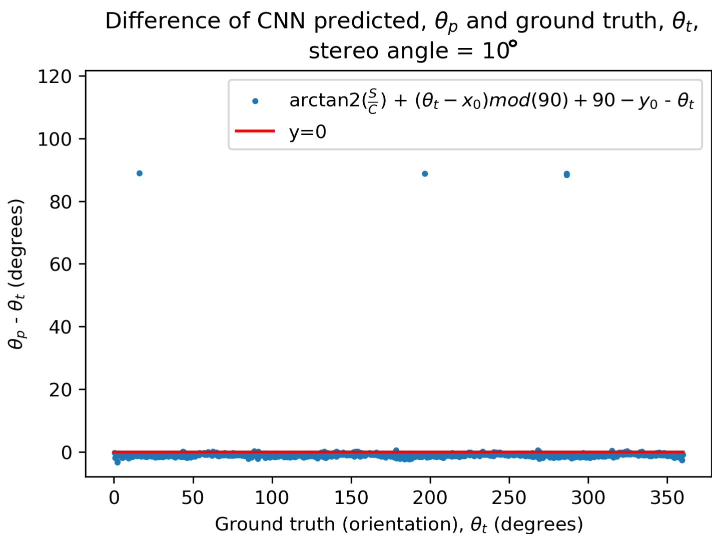


Figure 4.2: Predictions of orientation by the Difference Model on an unseen dataset, as in figure 4.1. Stereo angle = $10°$. Values x_0 and y_0 are simply the values of θ_p for $\theta_t = 0°$ and θ_t for $\theta_p = 45°$, which add to $45°$ and can be found by examining the dataset.

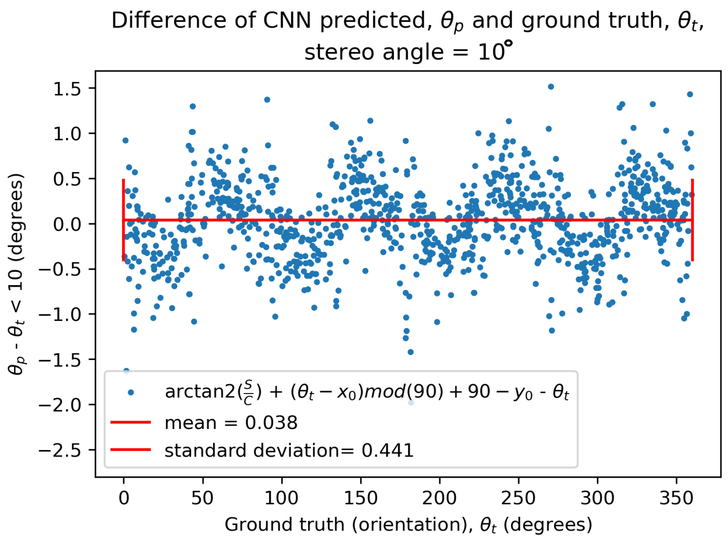
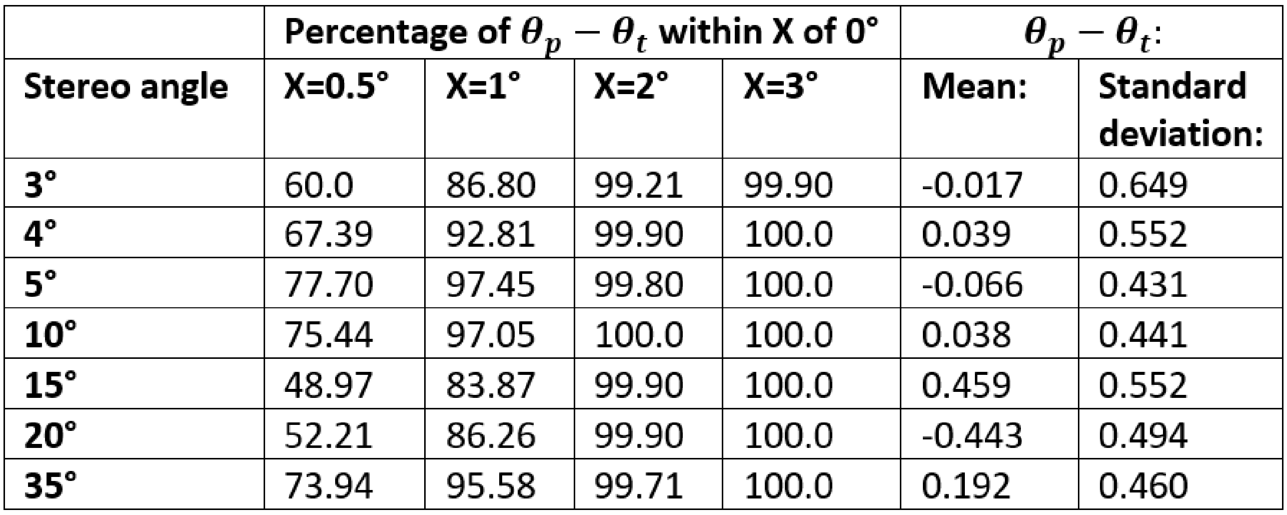


Figure 4.3: Predictions of orientation by the Difference Model on an unseen dataset as in figure 4.1. Large values beyond $10°$ in the y-axis have been omitted. Stereo angle = $10°$. Values x_0 and y_0 are simply the values of θ_p for $\theta_t = 0°$ and θ_t for $\theta_p = 45°$, which add to $45°$ and can be found by examining the dataset.

of $0°$ and labelling all values of θ_p to be within a couple degrees of each other. The stereo value of ten degrees was the most likely to train, with values higher or lower more often being trained on to achieve nothing. I am unsure if this is a result of the model being designed around a ten degree stereo angle, or mere coincidence. The lower limit on possible stereo angle size is due to the image quality. If the images were generated at a higher, say 128*128 pixel, resolution, the squares would be distinguishable at lower angles. As they are, this cannot be done, a square at $0.5°$ and a square at $1.5°$ are the same image. The upper limit on stereo angle size does not have as obvious of a cause. At first, I believed this to be caused by the periodicity of the angles. I expected the values to be periodic. In order to test this, new values were tried. Based on the success of $10°$, the next step was to try $-10°$, $80°$ and $100°$. The latter value being hopefully equivalent to $10°$ and the first two being equivalent. None of these values had any success. The loss was incredibly high, surprisingly, an entire order of magnitude higher. This knowledge provided a new idea. The upper limit on the stereo angle may be caused by the model needing most values to not cross boundaries in the angle space. By this I mean, the model needs the two images to usually fit onto the same zone (line) in figure 3.4. Otherwise, the stereo angle does not enable the model to establish the gradient of those lines, which is its method of finding the angles.

Each time the model was trained, it ended up with a different (arbitrary) offset between θ_p and θ_t. The figures 4.5 and the table 4.4 are examples of these training runs. The mean value calculated varied quite a lot, probably due to the various offsets in the predicted angles. The standard deviation, however, was far more consistent with changes of around 20% of the values shown.



	Percentage of $\theta_p - \theta_t$ within X of 0°				$\theta_p - \theta_t$:	
Stereo angle	**X=0.5°**	**X=1°**	**X=2°**	**X=3°**	**Mean:**	**Standard deviation:**
3°	60.0	86.80	99.21	99.90	-0.017	0.649
4°	67.39	92.81	99.90	100.0	0.039	0.552
5°	77.70	97.45	99.80	100.0	-0.066	0.431
10°	75.44	97.05	100.0	100.0	0.038	0.441
15°	48.97	83.87	99.90	100.0	0.459	0.552
20°	52.21	86.26	99.90	100.0	-0.443	0.494
35°	73.94	95.58	99.71	100.0	0.192	0.460

Figure 4.4: Data from figure 4.5 collated.

In figure 4.5, the difference in prediction and known truth values can be seen for a variety of stereo angles. For some, it is clear the data follows a similar sinusoidal trend to that of the $10°$ stereo angle. For $5°$ and $35°$ it is less clear if this is the case. Despite this, the data is still very similar, I assume the same trends are present but are being disguised by greater levels of noise caused by the difference in angular resolution between labels and images. From the table, we can see the final result of a model being trained to be 100% accurate at orientation angle prediction within $2°$ of the truth value, and 99.7% within $1°$. The model is able to predict the angle, within $3°$, 99.9% of the time. I have not performed a test on any colleagues, but this level of accuracy seems comparable to human capabilities.

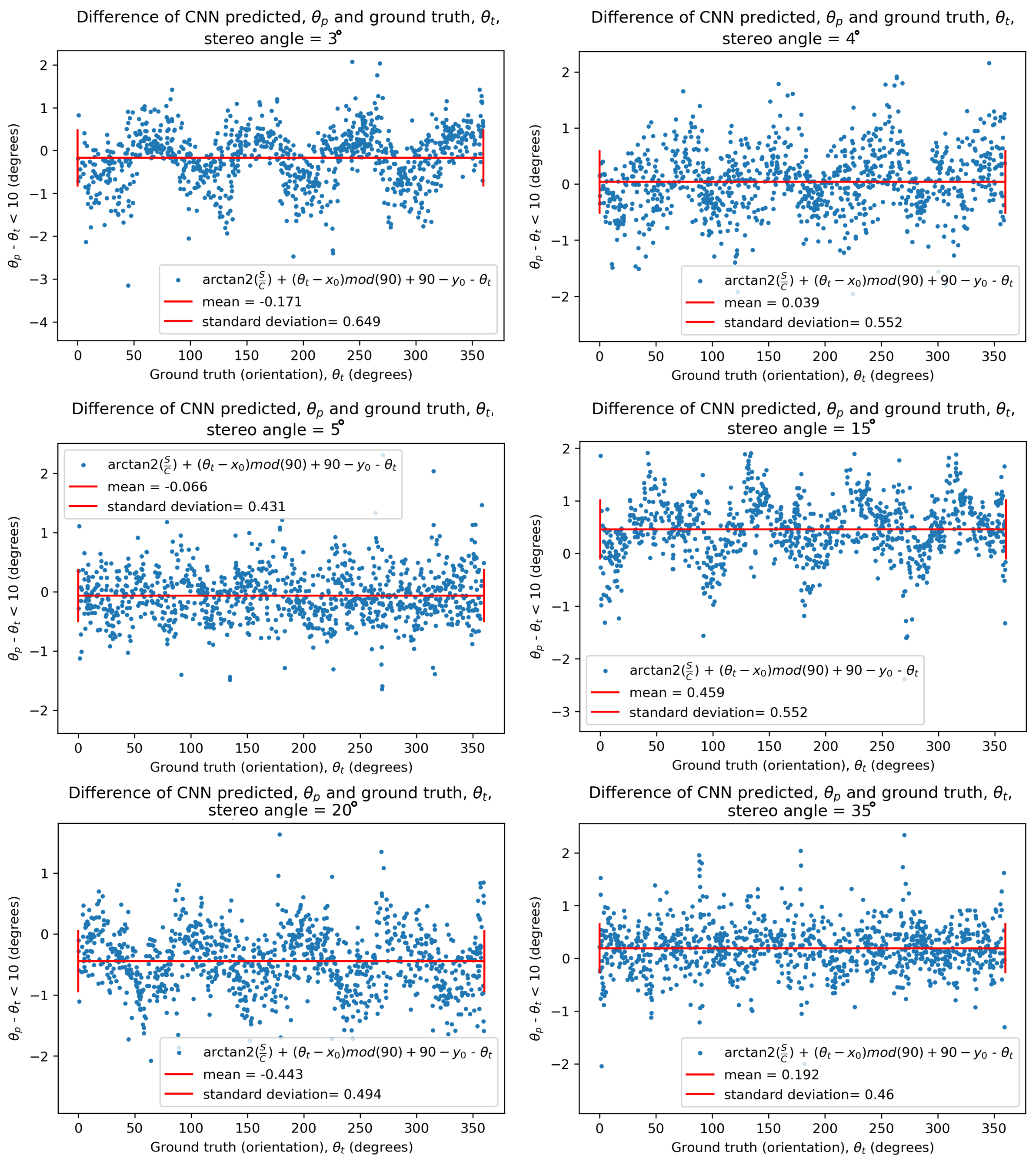


Figure 4.5: Predictions of orientation by the Difference Model on random unseen datasets for stereo angles = $3°, 4°, 5°, 10°, 15°, 20°, 35°$. Large values beyond $10°$ in the y-axis have been omitted. Values x_0 and y_0 are simply the values of θ_p for $\theta_t = 0°$ and θ_t for $\theta_p = 45°$, which add to $45°$ and can be found by examining the dataset.

5 Conclusion

On the one hand, the U-Net model was unsuccessful. It was not able to learn how to predict orientation angles, the skip connections may have been a design flaw. Potentially, the model might be more successful without them, although it would then be an autoencoder, rather than a U-Net. On the other hand, the Difference Model is clearly a viable method of training a machine learning agent to identify orientations of shapes. With a $10°$ stereo angle, the model is able to guess within two degrees of the orientation angle 100% of the time, in a time period that is significantly less than a human would take. The model takes only a few seconds to make these guesses on thousands of images. The model could be a significant aid to lab personnel. Stereo angles close to ten degrees are most successful when training, with three and thirty five forming lower and upper bounds on the range of acceptable stereo angles. The available stereo angles are highly suitable for use in HAADF-STEM, where commonly used values are in the $5°$ to $10°$ range[6]. With an average error of $0.038°$ and a standard deviation of $0.441°$, the model is significantly more accurate than human vision. It would not be possible to gain perfectly accurate results via this method of testing because the model is comparing discrete images to psuedo-continuous truth values. The randomly generated truth values, used to generate the images and label them, have a much finer resolution than the 64*64 pixel grid they are displayed with.

Unfortunately, it has not been possible to train a model that uses projections of images without access to the ground truth. This is due to the lack of a method for finding an expanding angle range to train on. For the non-projected images, the standard deviation of the images proved a useful metric, although this was found using the ground truth. For the 'S' shape, none of the standard deviation, the maximum value position, the maximum value or the width provided a way to extract the expanding angle range without using the ground truth. The standard deviation for the 'T' shape is utilisable, however, the dihedral nature of this shape poses its own difficulty. While only shown as a potential application, the accuracy of the sample reconstruction, figure 2.8 was disappointingly low. The limiting factor was the noise in the sinogram, which could have been reduced by using a different method to produce the projections. If the projections had been made from higher resolution images and then downsampled to reduce said projections back to their required size, the projections would have contained less noise. There would be sigificantly less pixel jumping if the original images had been 512*512 before projection, or if anti-aliasing had been incorporated into the process.

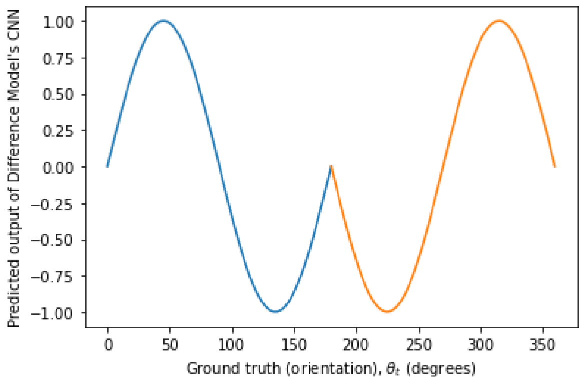


Figure 5.1: Potential modification of CNN output for the Difference Model to account for projections of shapes with mirror plane symmetry. This would be the expected output of a shape with two-fold symmetry.

A possible way to move the project forward would be to adapt the Difference Model to account for mirror plane symmetries in the input images. In other words, the model currently cannot perform on projections where the sinogram of the projections has identical outputs from different input values, or indeed is symmetric itself. I believe the way to do this would be to use the programmatically acquired symmetry value, n, to reverse the value of the output function n times throughout the angle range. Effectively multiplying the sine and cosine values by a square wave that oscillates between one and minus one every $\frac{360}{n}$ degrees. For the 'T' shape, this would cause the output of figure 4.1 to be $sin(2\theta)$ up to $180°$ and $-sin(2\theta)$ up to $360°$. See figure 5.1. When the

'T' is rotated $90°$ clockwise, the value would be calculated as $180°$, and a $10°$ rotation in either direction would produce the same output (projection), as expected.

Regardless of the way it is done, the dihedral projection problem must be solved before the model can be applied to data that accurately simulates that found in the lab. The project has not found a way to interpret projections of shapes with multiple degrees of freedom. This would also be needed for it to move forward, but currently no plan exists for how to do so. The CNN will need to output values from which can be extracted the two, or three, angular coordinates of the projection. This will require a much more clever way of mapping the whole $(0, 360), (0, 360), (0, 360)$ degree space into a single, unique region which represents all possible orientations the shape can acquire, and a function with unique outputs for them all. This is difficult as there are many 3D transforms that map to the same final orientation[14]. A simple example being $90°$ rotations on any of the Euler axes for a cube being identical.

6 References

All websites checked as of 17:45 on Friday 5th April, 2024

References

[1] Marwa Al and Mohammed Al-Hayani. The use of filtered back projection algorithm for reconstruction of tomographic image, 01 2014.

[2] Jason Brownlee. https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/.

[3] Rahul Chauhan, Kamal Ghanshala, and R. Joshi. Convolutional neural network (cnn) for image detection and recognition, 12 2018.

[4] HAADF-STEM. www.jeol.com/words/emterms/20121023.031059.php#gsc.tab=0.

[5] Juliza Jamaludin, Mohd Zawahir, Fazlul Yunus, Nor Muzakkir Nor Ayob, Ridzuan Aw, Suzzana Muhammad, Naizatul Mohd Fadzil, Zulkarnay Zakaria, and Mohd Hafiz Fazalul Rahiman. A review of tomography system. *Jurnal Teknologi*, 64, 10 2013.

[6] JEOL. https://www.jeol.com/words/emterms/20121023.051758.php#gsc.tab=0.

[7] Willi Kalender. X-ray computed tomography. *Physics in medicine and biology*, 51:R29–43, 08 2006.

[8] Keras. https://keras.io/api/layers/.

[9] Ibrahim Khan, Khalid Saeed, and Idrees Khan. Nanoparticles: Properties, applications and toxicities. *Arabian Journal of Chemistry*, 12:908–931, 11 2019.

[10] Yafen Li, Wen Li, Jing Xiong, Jun Xia, and Yaoqin Xie. Comparison of supervised and unsupervised deep learning methods for medical image synthesis between computed tomography and magnetic resonance images. *BioMed Research International*, 2020:1–9, 11 2020.

[11] Qiong Liu and Ying Wu. Supervised learning, 01 2012.

[12] Katherine Macarthur, Lewys Jones, Sergio Lozano-Perez, Dogan Ozkaya, and P Nellist. Structural quantification of nanoparticles by haadf stem. *Journal of Physics: Conference Series*, 522:012061, 06 2014.

[13] Shinya Maenosono and Derrick Mott. Structural analysis of nanoparticles using scanning transmission electron microscopy, 07 2013.

[14] Aurélien Martinet, Cyril Soler, Nicolas Holzschuch, and François Sillion. Accurate detection of symmetries in 3d shapes. *ACM Trans. Graph*, 25, 04 2006.

[15] J.E. Morris. Nanoparticle properties, 12 2008.

[16] Max T. Otten. High-angle annular dark-field imaging on a tem/stem system. *Journal of Electron Microscopy Technique*, 02 1991.

[17] Alexander J. Pattison. Applying automation and machine learning to scanning transmission electron microscopy, 2020.

[18] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *LNCS*, 9351:234–241, 10 2015.

[19] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 10 2015.

[20] Saman Siadati. What is unsupervised learning, 08 2018.

[21] Tensorflow. Keras: The high-level api for tensorflow, https://www.tensorflow.org/guide/keras.

[22] Juan Terven, Diana-Margarita Cordova-Esparza, Alfonzo Ramirez-Pedraza, and Edgar Chavez-Urbiola. Loss functions and metrics in deep learning. a review, 07 2023.

[23] Sonia Tiquia-Arashiro and Debora Rodrigues. Application of nanoparticles, 09 2016.

[24] Qi Wang, Yue Ma, Kun Zhao, and Yingjie Tian. A comprehensive survey of loss functions in machine learning. *Annals of Data Science*, 9, 04 2022.

[25] various Wikipedia. https://en.wikipedia.org/wiki/annular_dark-field_imaging.