WAVEFRONT SENSORLESS ADAPTIVE OPTICS USING MACHINE LEARNING AND NEURAL NETWORKS

Callum Francis
CALTA
STFC
20/07/2023

Contents

1 Abstract

This is a summary of work done during my placement year at STFC on wavefront sensorless adaptive optics. Machine learning algorithms in a feedback loop with a deformable mirror accomplished reliable corrections in around 8 minutes. For instantaneous corrections, different neural network structures were tested; the most successful of these, QualNet, improved the Strehl ratios of simulated far-fields from 0.0138 to 0.1861 post correction, with some achieving ratios of $0.3-0.4$. Applying the network to real corrections on a deformable mirror or spatial light modulator requires knowledge of the influence functions, but is something the network appears capable of. These results could be improved upon by training on larger data sets, changing the structure of QualNet, and training the model on real far-field images.

2 Modelling

Wavefronts become deformed when different regions travel at varying speeds, for example, this could be caused by imperfections in a lens with changing refractive indices. When focussing the beam, these aberrations cause the rays in different parts of the wavefront to be focussed in different locations, negatively affecting image quality. The goal of adaptive optics (AO) is normally to correct these aberrations, restoring the flat wavefront and improving image quality. AO systems can also be used to manipulate the wavefront shape to give specific intensity distributions at the focus - such as a line focus.

An adaptive optic is a device that can be used to correct wavefront aberrations, such as liquid crystal spatial light modulators or deformable mirrors. The optic that was used in this application was a bimorph deformable mirror, consisting of piezoelectric disks glued to a thin sheet of glass that has a reflective coating - applying voltages to these actuators caused the glass to bend.

Figure 1 gives a general overview of how a deformable mirror can be used to correct a wavefront. An aberrated wavefront, ϕ_{abb}, is incident on a deformable mirror; the voltages of the actuators in the mirror have been set to give a shape that matches that of the incoming wavefront. Upon reflection, the resultant wavefront shape is given as $\phi_{res} = \phi_{abb} + \phi_{mirr}$, where ϕ_{mirr} is the mirror shape. Normally, ϕ_{res} is determined using a wavefront sensor which feeds back to the deformable mirror in a closed loop, until ϕ_{res} is the desired shape. However, wavefront sensors are expensive and may not always be available. Wavefront sensorless adaptive optics aims to remove the need for a sensor, requiring just a normal camera.

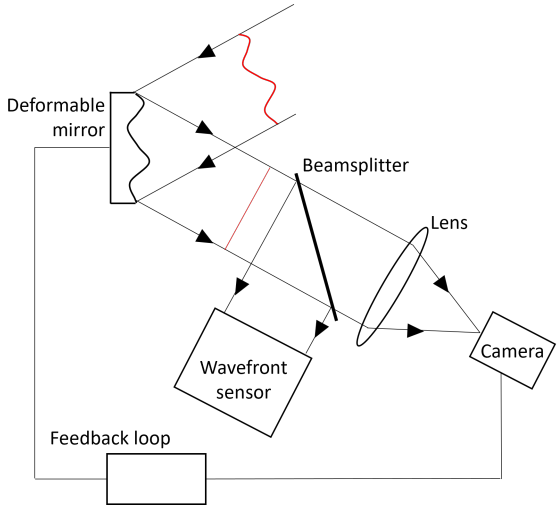


Figure 1: Deformable mirror correction. The shape of the deformable mirror has been set to be the same shape as the incoming wavefront. The resultant wavefront is then flat. A beamsplitter diverts a proportion of the beam onto a wavefront sensor. Normally, this sensor would be in a feedback loop with the mirror to determine its shape. In sensorless AO, the mirror is instead connected to a camera at the far-field.

The pupil function can be expressed as,

$$P(x,y) = A(x,y)e^{i\phi(x,y)} \tag{1}$$

where A is the amplitude distribution and ϕ is the phase distribution that we are interested in. As the camera only measures intensity, simply looking at the beam with a camera would measure only the amplitude distribution. Instead, the beam must be focused on the camera. Positioning the mirror at a distance of the focal length before the lens means that the camera measures the intensity distribution of

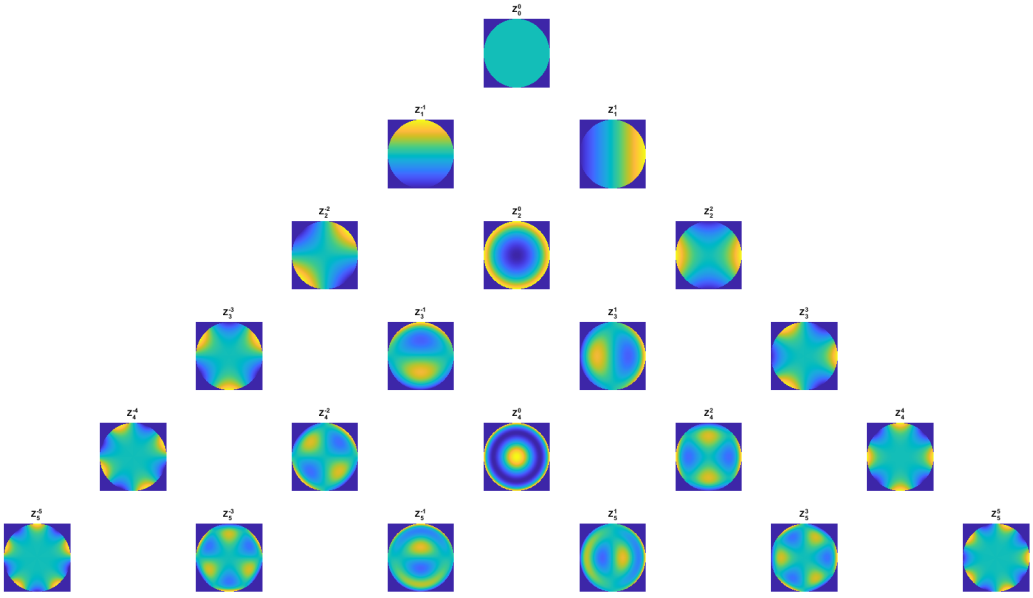


Figure 2: The first 21 Zernike polynomial shapes

the exact Fourier transform of $P[1]$. A perfectly flat wavefront would give a single, perfect focus spot on the camera. This makes it possible to define an image quality metric to evaluate the wavefront shape, and feed back to the deformable mirror. The actuator positions can then be updated according to a machine learning algorithm until the best possible focus spot is achieved.

It was much more convenient to test algorithms on a model of a deformable mirror where wavefronts could be defined as needed and noise effects could be disabled. When the simulation script, `WSAOsim`, is run, it prompts ask the user to select the mirror model and algorithm to use.

The aberrated wavefront can be randomly generated from a sum of Zernike polynomials, Z, which are orthogonal over the unit disk, as follows,

$$\phi_{abb}(x,y) = \sum_i \alpha_i Z_i \tag{2}$$

where α is the coefficients. Figure 2 displays the shapes of the first 21 polynomials. Zernike polynomials themselves are defined as

$$Z_n^m(\rho,,\theta) = \begin{cases} R_n^{|m|}(\rho)cos(m\theta) & m \geq 0 \\ -R_n^{|m|}(\rho)sin(m\theta) & m < 0 \end{cases} \tag{3a}$$

$$R_n^{|m|}(\rho) = \sum_{s=0}^{\frac{n-|m|}{2}} \frac{(-1)^s(n-s)!}{s!\left[\frac{n+|m|}{2}-s\right]!\left[\frac{n-|m|}{2}-s\right]!}\rho^{n-2s} \tag{3b}$$

$$\tag{3c}$$

The simulation uses the first 32 Zernike polynomials to generate a wavefront, ignoring the first 3 terms (piston, tip and tilt), as these have only effect the position of the far field but not the shape of it. The function `generate_coeffs` also weights the coefficients to be smaller at higher orders, as it is more realistic.

The mechanism of the deformable mirror model is explained in [2]. Overall, it is made up of a matrix of influence functions (one for each actuator) that is multiplied by voltages and summed to give a total shape. The shape is added to the input wavefront to give the resultant, output wavefront shape, ϕ_{res}.

The far-field intensity distribution image was calculated from the output by taking the Fourier transform

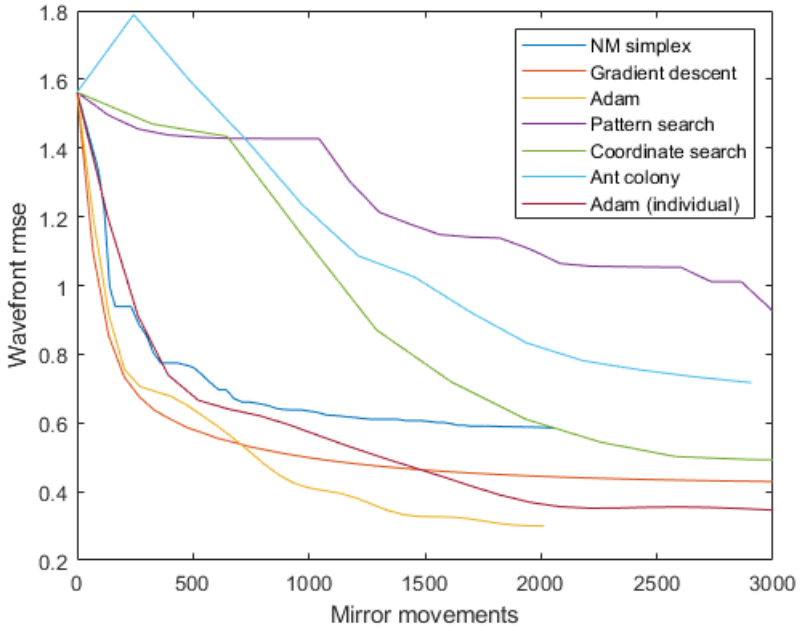


Figure 3: Root mean squared wavefront error against number of mirror movements required to achieve that error for machine learning algorithms, tested on a model of a deformable mirror, for the same initial wavefront.

of the pupil function as follows,

$$I(u,v) = \frac{A^2}{\lambda^2 f^2}\left| \iint_{-\infty}^{\infty} A(x,y) e^{\frac{2\pi i}{\lambda}\phi(x,y)} e^{\frac{2\pi i}{\lambda f}(xu+yv)} dudv \right|^2 \tag{4}$$

where f is the focal length the amplitude, $A(x,y)$ was assumed to be uniform across the beam. The constant coefficient is largely unimportant and the intensity was scaled so that the maximum value was always equal to 1 before evaluating.

The cost function in the simulation has an option to easily change the metric that is used. In general, the power in the bucket metric was successful, defined as

$$J(u,v) = \frac{\int I(u,v)^2}{\left[\int I(u,v)\right]^2} * n_p \tag{5}$$

where n_p is the number of pixels in the image. As an alternative, another metric could be to just sum all of the normalised pixel values,

$$J(u,v) = \sum \frac{I(u,v)}{max(I(u,v))} \tag{6}$$

2.1 Algorithms

A number of machine learning simulations were tested on a model of a deformable mirror for a generated wavefront shape. The wavefront shape was the same in each case. Figure 3 shows the loss function against number of mirror movements required for each optimisation - lower loss is better. Mirror movements was chosen instead of time taken as this should be the limiting factor in the speed of optimisation when using a real mirror. Most algorithms here were written by students working on this project before me and explanations can be found in the report by Petros Oratiou[2]. I have explained another in this report and gone into more detail about the most successful one, ADAM.

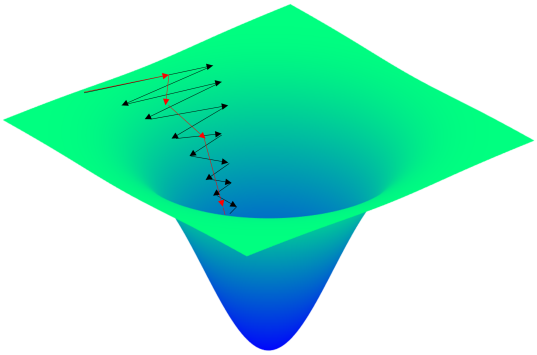


Figure 4: A surface plot of a typical cost function - the lowest cost corresponds to the optimal solution. The arrows show the examples of the optimisation paths that can occur during gradient descent (black) and gradient descent with momentum or RMSprop (red). These two algorithms have techniques to prevent the oscillatory patterns that slow down learning

From Figure 3, we can see that the ADAM algorithm was able to correct the image to the highest degree in a sufficiently small number of movements. This was constantly the case for all wavefronts tested on the model. The mechanism of the ADAM algorithm is explained below.

2.1.1 ADAM

First, it is important to understand exponentially weighted moving averages. For a set of data, θ, this is found using,

$$v_t = \beta v_{t-1} + (1-\beta)\theta_t \tag{7}$$

where v is the weighted average, t is the index of the data and β is a constant. The value v_t is approximately averaging over the last $\frac{1}{1-\beta}$ data points. However, if $v_0 = 0$ and all data points $\theta_t > 0$, for example, v_1 will be calculated to be lower than it should be. It is therefore common to introduce a bias correction term to negate this,

$$v_t^{corr} = \frac{v_t}{1-\beta^t}. \tag{8}$$

ADAM stands for adaptive moment estimation; it is a combination of two other algorithms: gradient descent with momentum (GDM) and root mean squared propagation (RMSProp)[3]. Both of these algorithms are designed to mitigate the oscillation effect displayed by the black arrows in Figure 2, that can occur during standard gradient descent. Gradient descent algorithms update a parameter, p, based on the gradient of a cost function J,

$$p_t = p_{t-1} - \alpha \frac{dJ}{dp}. \tag{9}$$

Adding a momentum term can speed up optimisation as the components of the updates in opposing directions cancel each other out. This is demonstrated by the red arrows in Figure 4. The equation for GDM is very similar to Equation 1, for exponentially weighted averages,

$$V_t = \beta_1 V_{t-1} + (1-\beta_1)\frac{dJ}{dp_t} \tag{10}$$

where β_1 is the constant, momentum term. The momentum term can also stop the algorithm optimising at a local, instead of global, minimum. With this term, the parameter would then change according to,

$$p_t = p_{t-1} - \alpha V_t \tag{11}$$

where α is the learning rate. RMSProp is slightly different in that it finds a moving average of the squared

gradients for each weight [4],

$$S_t = \beta_2 S_{t-1} + (1 - \beta_2) \left(\frac{dJ}{dp_t} \right)^2 \tag{12}$$

where β_2 is the momentum constant. The parameter, p would then be updated using,

$$p_t = p_{t-1} - \alpha \left(\frac{dJ}{dp_t} \right) \left(\frac{1}{\sqrt{S_t + \epsilon}} \right) \tag{13}$$

where ϵ is a small number to prevent dividing by zero.

Within ADAM, a small change is made to the voltage of a single actuator and both V_t and S_t, are determined by Equations 10 and 12 respectively; then bias correction is applied to give

$$V_t^{corr} = \frac{V_t}{1 - \beta_1} \tag{14}$$

$$S_t^{corr} = \frac{S_t}{1 - \beta_2} \tag{15}$$

Finally the new parameter is updated as,

$$p_t = p_{t-1} - \alpha \frac{V_t^{corr}}{\sqrt{S_t^{corr} + \epsilon}} \tag{16}$$

The common values for α, β_1, β_2, and ϵ are 0.001, 0.9, 0.99, and 10^{-8} correspondingly, although the learning rate should be tuned to suit the problem.

2.1.2 Ant Colony

The ant colony algorithm is based on the paths taken by ants to find food[5]. At first, each ant takes a random path, leaving pheromones for other ants to follow; subsequent ants will follow the most successful ant, maintaining the pheromones on this path while those on other paths decay away. If an ant diverges slightly from the most successful path and finds a better one, this becomes the new preferred path - over time the ants will find the most direct route to the food (i.e. the route with the lowest cost function). When applied to optimising the shape of the deformable mirror, the actuator voltage range is first divided into N parts. Each voltage is originally given an equal probability of being chosen. The 'ants' take a path through the actuators, selecting one voltage for each. The resulting shapes for every ant are evaluated and the voltage probabilities along the best path are increased dependent on how favourable it is, while the others depreciate. Any voltages above or below certain threshold values are restrained to the limits. The next set of ants are then more likely to follow a path similar to the previous most successful ant. The voltage probabilities are additionally normalised across each actuator and the range of allowed voltages gradually focuses in around the most successful values, to give more accurate results.

While the ant colony algorithm should eventually find a good solution, it can initially find a path worse than the starting point: simply because it is reliant on the probability of an ant finding a decent path over time. Ordinarily, this algorithm would perform slower than ADAM.

2.1.3 Gerchberg-Saxton

The Gerchberg-Saxton algorithm is very different from those above in that it seeks to deduce the wavefront shape from a single image of the focus and the intensity distribution of the beam[6]. Equation 17 illustrates the iteration loop. Starting from a random guess of the wavefront shape, the far field of the guess is generated, the modulus of the far field is replaced with the real far field image. A new guess is then determined from the inverse Fourier transform, and replacing the modulus with the intensity distribution of the beam.

$$g_k(x,y) = |g_k(x,y)|e^{i\theta_k(x,y)} \tag{17a}$$
$$G_k(u,v) = |G_k(u,v)|e^{i\phi_k(u,v)} = \mathcal{F}\left[g_k(x,y)\right] \tag{17b}$$
$$G'_k(u,v) = |F(u,v)|e^{i\phi_k(u,v)} \tag{17c}$$
$$g'_k(x,y) = \mathcal{F}^{-1}\left[G'_k(u,v)e^{i\phi_k(u,v)}\right] = |g'_k(x,y)|e^{i\theta'_k(x,y)} \tag{17d}$$
$$g_{k+1}(x,y) = |f(x,y)|e^{i\theta'_k(x,y)} \tag{17e}$$

where $g(x,y)$ is the guess pupil function, $\theta(x,y)$ is the phase component (wavefront shape) of the guess, $G(xu,v)$ is the guessed far field distribution, $\phi(u,v)$ is the complex component of the far field, $F(u,v)$ is the real far field distribution, $f(x,y)$ is the near field intensity distribution and the subscript, k, is the iteration number.

Eventually, the guess wavefront was expected to be the same shape as the real wavefront. Figure 5 nonetheless depicts an outcome that was not initially anticipated. The algorithm did find a solution, but it was not the same as the input wavefront: it was the input wavefront reflected in all three axis, x,y,z. Both of these wavefront shapes give the same far field distribution but different Zernike coefficients. Another weakness of the algorithm is that the phase component, $\theta(x,y)$, will be returned in radians, where $-\pi < \theta(x,y) \leq \pi$; any regions of the wavefront beyond these bounds are wrapped, and must therefore be unwrapped to return the corrected shape. If the phase distribution was smooth, and the discontinuities due to wrapping obvious, this would not be an issue. Unfortunately, the algorithm often returns fairly 'bumpy' distributions, making it harder to spot where wrapping has occurred. For this reason, this algorithm was not thought to be reliable enough for wavefront sensorless adaptive optics in this case.

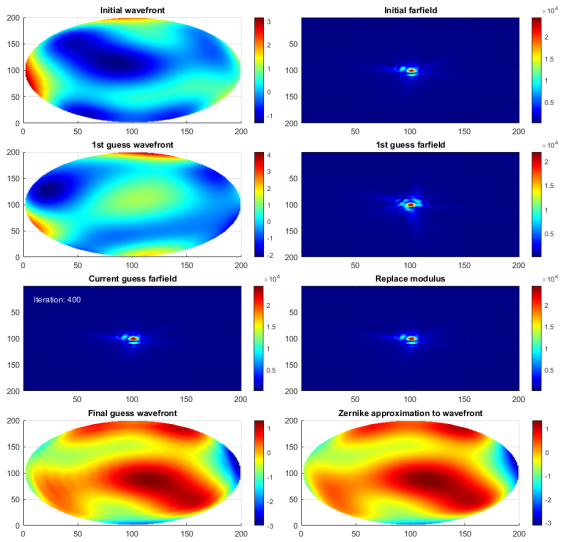


Figure 5: The Gerchberg-Saxton phase retrieval algorithm. From an inital guess of the wavefront, the algorithm produces guesses that give far-field images closer and closer to that formed by the initial wavefront. Each far-field image has 2 wavefront solutions, meaning the algorithm may not return the same wavefront as the input, even if it converges on a solution.

2.2 Model-based approach

Model-based aim to do with the potentially slow iterative corrections of other algorithms, and they generally require far fewer far field images per correction[7]. The method relies on prior knowledge of the variation of the cost metric and the influence functions applied to the mirror. The influence functions could, for example, be the actuator voltage influences[8]. Often, the gradient function can be approximated near the maximum by a parabola [9]. By simulating a flat wavefront and moving actuators in turn, the gradient functions of each actuator in the simulated mirror was calculated. Then, after sending a more realistic, randomly aberrated wavefront, a small change was applied to individual actuators to determine their position on the gradient functions. The voltages were then updated to immediately move to the maximum

points.

This was ultimately unsuccessful as a result of the overlap between the influence functions - the gradient functions that had been measured became inaccurate when neighbouring actuators were moved. A system of improved influence modes could be derived experimentally to resolve this[10, 11]. Additionally, more sophisticated model-based methods have been shown to give improved performance[8, 12], but was not attempted here.

3 Real mirror results

The original implementation of the algorithms measured small changes in each actuator separately, then updated all of them at the same time. Due to coupling of the actuators, algorithms performed significantly better when actuators were optimised individually, before moving to the next. Algorithms duplicated and altered slightly to do this, for example, `adam_individual`. In addition, creating a GUI in Matlab app designer made it much easier to apply the correction algorithms, presented in Figure 6. Before running the optimisation with `Start`, the parameters of the beam, pixel spacing on the camera, and number of mirror voltage channels can be inputted. Of these, only the number of mirror channels and image resolution obtained by the camera affect optimisation, the others are used only to generate the scaled optimal far-field, necessary for calculating the Strehl ratio. The optimisation algorithm and mirror model were selected from the drop down menus. This mirror model selection only affects the surface plot displayed, used to give a live plot of what the real mirror surface should look like. This does not affect optimisation. In the case shown in Figure 6, the influence functions were only estimated with reasonable values. Consequently, the shape of this surface plot should used as only an approximation of the general mirror shape. A filter wheel was placed in the path of the beam before the camera, containing neutral density (ND) filters of increasing optical density. When any pixel on the camera reached its maximum value, the filter wheel moved to increase the optical density, decreasing the overall intensity at the camera. The reverse process was similar if the maximum intensity on the camera fell too low. The live filter wheel (FW) position could be controlled from the GUI, with the optical density at that position displayed. Upon pressing start, the GUI connects to the filter wheel and deformable mirror automatically. Green lights portray the connection statuses of these, and whether the optimisation corrections have started. The learning rate of the algorithm can be changed before and during optimisation; this value decreases after each iteration according to the specified decay rate (which can also be set at any time), resulting in smaller corrections as the algorithm approaches the optimum position. In addition, there is a switch to choose between the set of values that the algorithm optimises: either the voltages of the actuators or the Zernike coefficients (of polynomials 3-15 using normal indexing) that describe the shape of the mirror. Selecting the Zernike coefficients option requires the coefficients returned by the algorithm to be converted into voltages to send to the mirror - meaning that, ideally, the influence functions of the actuators should be known accurately. Finally, `Recentre`, re-centres the camera on the intensity peak after the current iteration is finished. `Stop` stops the program once the iteration completes and disconnects from both the filter wheel and deformable mirror. As Figure 6 demonstrates, the mirror was able to successfully correct the wavefront and improve the image quality, but the process is far from quick, with this example taking around 8 minutes.

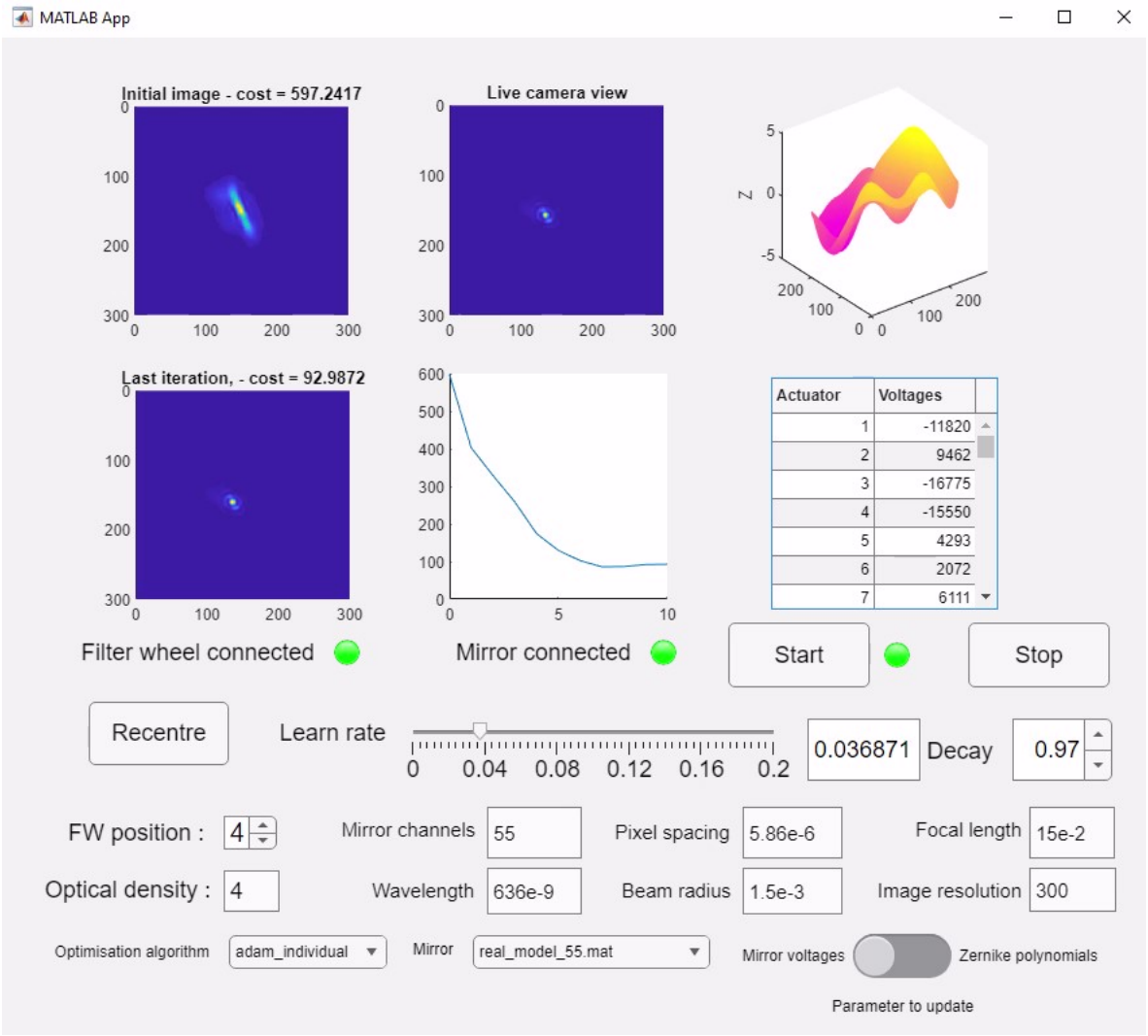


Figure 6: GUI created for more easily applying corrections to a real deformable mirror.

3.1 Strehl ratio

The Strehl ratio is defined as the ratio of the maximum intensity of a far-field distribution compared to the maximum intensity of a diffraction limited far-field. To obtain the maximum theoretical intensity, a perfect far-field was modelled by taking the Fourier transform of a flat wavefront. The radius of of a diffraction limited Gaussian beam was calculated from,

$$w_f = \frac{\frac{\lambda f}{\pi w}}{\left[1 + \left(\frac{\lambda f}{\pi w^2}\right)^2\right]^{\frac{1}{2}}} \tag{18}$$

where w_f is the spot radius at the focal plane and w is the spot radius at the lens. The ideal far-field image could then be scaled according to the known pixel spacing of the camera. The area underneath the intensity distribution of all images was normalised, allowing comparison of real images, with the model.

4 Neural Networks

Neural networks are capable of learning complex non-linear problems. Networks are generally made out of 'neurons' or nodes that take inputs, multiplies these by a weight matrix, W, adds a bias vector, b, and applies an activation function, $g()$. A network could consist of just a single node that may, for example, take as inputs some parameters about a car and generate a price. More complex problems can be learned by grouping many of these nodes into layers.

Figure 7 portrays a basic, 2 layer neural network - input layers are not usually counted. The superscript, $[l]$, is used to denote the layer and subscripts denote the specific node in that layer. There are 4 input values, x_1,x_2,x_3 and x_4, 1 hidden layer and 1 output layer, outputting a single value, $\hat{y}$ (note that this could also be called $a^{[2]}$, the output of the final layer). For simplicity, this network be referred to as network A. The inputs to each node are the outputs from those before it, as follows,

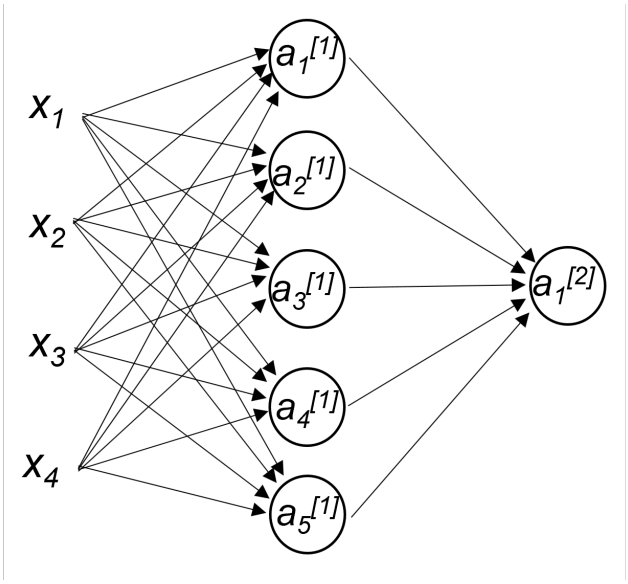


Figure 7: Example of NN made out of nodes. There are 4 inputs, 1 hidden layer, and 1 output value. The superscript in square brackets displays the layer number, and the subscript denotes the node within that layer.

$$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]} \tag{19}$$

$$a^{[l]} = g^{[l]}(z^{[l]}) \tag{20}$$

where $W[l]$ has dimensions (no. of nodes in layer [l] x no. of nodes in layer [l-1]) and $b^{[i]}$ has dimensions (no. of nodes in layer [l] x 1). For example, in layer 1 of Figure 7,

$$z^{[1]} = \begin{bmatrix} w_{1,1}^{[1]} w_{1,2}^{[1]} w_{1,3}^{[1]} w_{1,4}^{[1]} \\ w_{2,1}^{[1]} w_{2,2}^{[1]} w_{2,3}^{[1]} w_{2,4}^{[1]} \\ w_{3,1}^{[1]} w_{3,2}^{[1]} w_{3,3}^{[1]} w_{3,4}^{[1]} \\ w_{4,1}^{[1]} w_{4,2}^{[1]} w_{4,3}^{[1]} w_{4,4}^{[1]} \\ w_{5,1}^{[1]} w_{5,2}^{[1]} w_{5,3}^{[1]} w_{5,4}^{[1]} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \\ b_5^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \\ z_5^{[1]} \end{bmatrix} \tag{21}$$

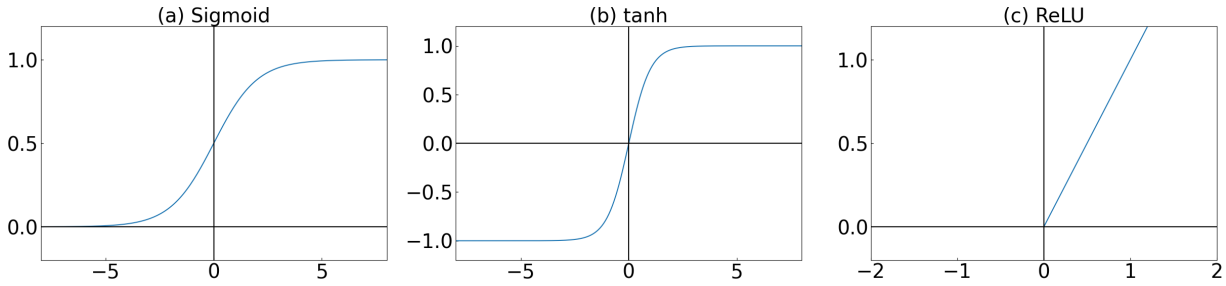


Figure 8: The 3 most common activation functions.

If there are m input examples, $X = \left[x^{(1)}, x^{(2)}, x^{(3)}, ..., x^{(m)}\right]$, where each $x^{(i)} = \begin{bmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{bmatrix}$, the specific input example is expressed as another superscript in round brackets - $^{(i)}$. Therefore, Equations 19 and 20 become

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]} \tag{22}$$

$$A^{[l]} = g^{[l]}(Z^{[l]}) \tag{23}$$

where $A^{[l]} = \left[a^{[l](1)}, a^{[l](2)}, ..., a^{[l](m)}\right]$ and $Z^{[l]} = \left[z^{[l](1)}, z^{[l](2)}, ..., z^{[l](m)}\right]$.

Activation functions are the cause of the non-linearity of neural networks. Without them, having hidden layers in a network would be meaningless - it would be equivalent to a single layer. Any non-linear function could be used as an activation function but common examples include sigmoid, tanh and ReLU (rectified linear unit), shown in Figure 8, parts a, b, and c respectively. Generally, sigmoid is used for classification tasks and ReLU is used widely between layers in CNNs.

4.1 Hyperparameters

Hyperparameters are parameters that are set manually affect the training of the parameters, W and b. There are many hyperparameters that need to be tuned to improve network performance; Table 1 contains a list of some of these.

Table 1: Hyperparameters to optimise to improve learning speed and proficiency

Learning rate	The speed at which the parameters are updated. Higher learning rates train faster but may not converge. Apply a decay rate to decrease the learning rate over time.
Mini-batch size	Number of training examples given to network before updating parameters. Generally sizes of 2^n will lead to faster training
Number of epochs	Number of passes through the dataset during training
Number of hidden layers and units	More layers can improve accuracy but also may lead to overfitting - need to use regularisation techniques

4.2 Backpropagation

The parameters, W and b, are randomly initialised at first and updated using an optimisation algorithm, such as ADAM. To do this, a loss function needs to be defined to evaluate the model. In supervised learning (where the dataset is labelled), the loss function relates error in the output, $\hat{Y}$, compared to the expected output, Y. The cost function J of the parameters is the average of the loss function for every input example. For example for the network shown in Figure 7, the cost function might be,

$$J(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}) = \frac{1}{m}\sum_{i=1}^{m}(A^{[2]} - Y)^2 \tag{24}$$

In order to update the values of W and b, we need to find the rate of change of the cost function with respect to each parameter. This is possible using the chain rule. Again using network A as an example, the gradient of the cost with respect to $W^{[2]}$ would be

$$\frac{\partial J}{\partial W^{[2]}} = \frac{\partial J}{\partial A^{[2]}}\frac{\partial A^{[2]}}{\partial Z^{[2]}}\frac{\partial Z^{[2]}}{\partial W^{[2]}} = \frac{2}{m}\sum_{i=1}^{m}(A^{[2]} - Y)\cdot g'(Z^{[2]}) \times A^{[1]T} \tag{25}$$

and similarly,

$$\frac{\partial J}{\partial W^{[1]}} = \frac{\partial J}{\partial A^{[2]}}\frac{\partial A^{[2]}}{\partial Z^{[2]}}\frac{\partial Z^{[2]}}{\partial A^{[1]}}\frac{\partial A^{[1]}}{\partial Z^{[1]}}\frac{\partial Z^{[1]}}{\partial W^{[1]}} = W^{[2]T} \times \left(\frac{2}{m}\sum_{i=1}^{m}(A^{[2]} - Y)\cdot g'(Z^{[2]})\right) \cdot g'(Z^{[1]}) \times X^T \tag{26}$$

In this way the gradients can be propagated from the final layer of the network back to the first. Luckily, software, such as Tensorflow, can implement backpropagation by itself.

4.3 Convolution layers

When training neural networks on images, which could be made up of many thousands or even millions of pixels each in high resolution images, it is clear that using layers made of the nodes mentioned previously would quickly require a huge number of parameters, and be very computationally expensive to train. Convolution layers offer a solution to this problem. The convolution acts like a filter: it is a matrix of weights called a kernel that sweeps over an image, performing an element-wise multiplication with the part of the input it is currently over, and then summing the results into a single output value. The process is demonstrated in Figure 9 using a 6x6 image and a 3x3 convolution filter with a stride of 1. The weights in this filter make it suited for detecting vertical edges in images - large magnitude output pixels correspond to vertical edges in the input image. Each convolution layer will learn to identify features during training, from low level features such as edge detection in lower layers, up to more complex features - for example eyes or ears if the network was trained on images of animals - in higher layers. To prevent the size of the output matrix being smaller than that of the input, padding can be added around the input.

An single image can have multiple channels, for example a colour image is formed out of three channels (red, green and blue pixel values). This would require a three dimensional convolution. The number of channels in the filter must be equal to the number of channels in the image. Each filter channel convolves with the corresponding channel in the image, and the outputs are summed over every channel, returning an output in only two dimensions. However, a convolution layer can also contain multiple filters of equal size. The outputs from each of these are not combined, but are returned separately. Consequently, an image of size (n, n, n_c) convolved with n'_c filters of size (f, f, n_c), with no padding and a stride of 1, returns an output with dimensions $(n-f+1, n-f+1, n'_c)$ Commonly, as an image progresses through a convolution network, the width and height dimensions decrease (as a product of not using padding) and the number of channels increases.

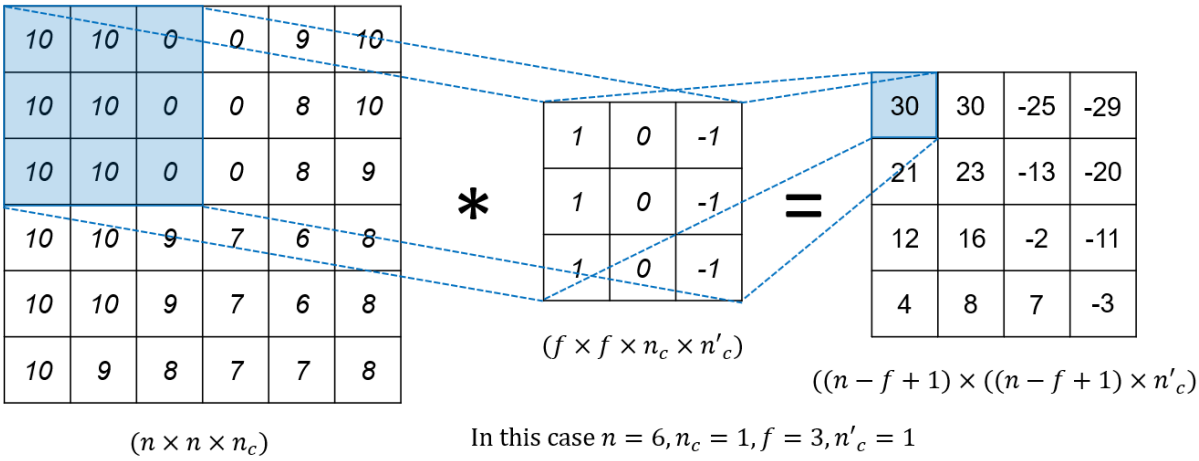


Figure 9: A convolution layer with a (6×6) input and a (3×3) with a stride of 1 and no padding.

4.4 Regularisation

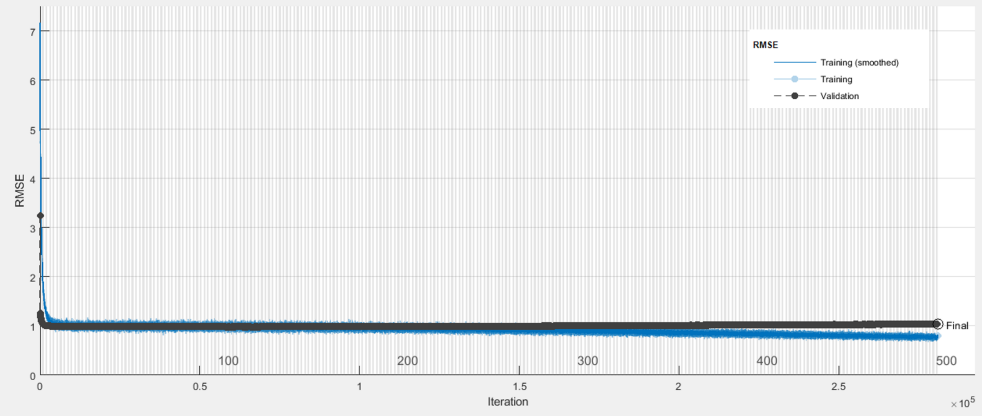
Regularisation techniques aim to reduce overfitting - when the network learns the features of the training data and does not generalise, performing badly on unseen data. These techniques are particularly important in larger networks. Some regularisation techniques are outlined in Table 2

Table 2: Regularisation techniques

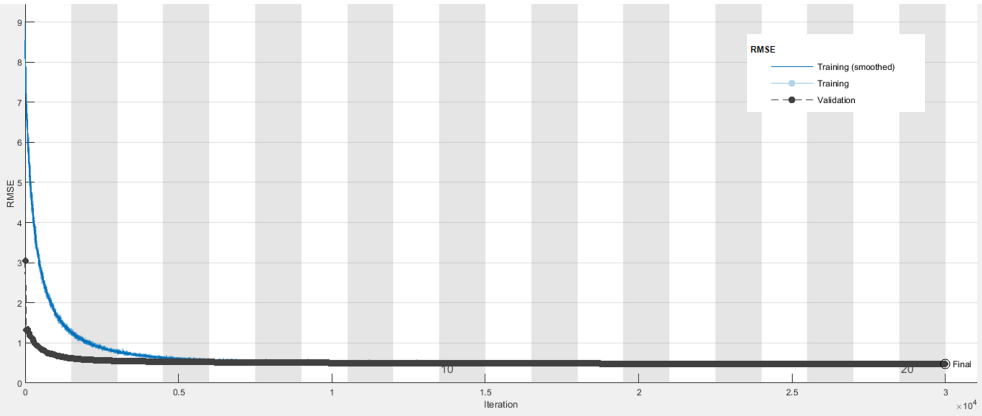
Batch Normalisation	This layer normalises the activations of each channel by subtracting the mini-batch mean and dividing by the mini-batch standard deviation. Then, the inputs are shifted by learnable offset β and scaled by learnable factor γ. Consequently, training is sped up and sensitivity to initialisation is reduced.
Dropout Layer	Dropout randomly sets inputs to 0 with a given probability, p, and then scales remaining inputs by $\frac{1}{1-p}$. This stops the network from becoming reliant on just one or a few nodes.
L_1 and L_2 regularisation	Addition of a factor to the loss function of $\|\|W^{[l]}\|\|_2^2 = \sum_i \sum_j (w_{ij}^{[l]})^2$, for L_2, and $\|\|W^{[l]}\|\|_1 = \sum_i \sum_j \|w_{ij}^{[l]}\|$ for L_1 regularisation. These factors have the effect of preferring less complicated functions.

4.5 Training existing architectures on far-field images

The first thought was to try and train a convolutional neural network to predict the Zernike coefficients from far-field images. This was done simply by training existing architectures on a dataset of far-field images, generated by creating a wavefront from randomly generated Zernike coefficients. The coefficients were weighted to be decrease at higher orders, as this was more realistic to a real wavefront. The pupil function was then found with Equation 1 and taking the Fourier transform gave a far-field image. Each far-field image was labelled with its corresponding first twelve Zernike coefficients (excluding the first three terms). The far-field images themselves were generated from 32 coefficients, but it would be harder for the network to guess this many; twelve was considered to be a good a approximation of the wavefronts. The approach was first attempted on the existing Alexnet[13] architecture, a CNN that has already proven successful at image classification. Various methods of pre-processing the input images were attempted, including normalising the images and square rooting, to try and exaggerate lower level features away from the central focus. These methods hoped to make it easier for the network to learn differences between



(a) Alexnet training progress on 36,000 images. The error was not improving so the network was trained for many epochs. It has started overfitting as the training loss decreased below that of the validation loss.



(b) Alexnet training progress on 96,000 images. A lower error was achieved but it is still far too high for this application

Figure 10: Alexnet results

far-fields. In general, this approach was unsuccessful. The original implementation of Alexnet on a dataset of 36,000 images, Figure 10a, failed to achieve RMSE values lower than about 1 on the validation set. It additionally started overfitting to the training dataset, shown by the RMSE of the training dataset (blue line) continuing to decrease but the validation dataset RMSE starts to increase. To try and prevent this, the network was retrained 96,000 images, Figure 10b. Here, the validation RMSE improved to 0.47966, which is still poor given the coefficients were on the order $\tilde{0}.1$, but there were no signs of overfitting.

Given the high bias in the results from Alexnet, a much deeper network was tested: Xception [14], a network with 71, layers but a small enough number of parameters, that training would not take too long, at 23 million. The results of which are shown in Figure 11. While the bias was reduced for a dataset with only 36,000 images, it was still too high, at 0.3778.

4.6 Creating custom architectures

4.6.1 SimilarityNet

Returning to the outputs of the Gerchberg-Saxton algorithm, the network architectures above do not consider the fact that there are two solutions for each far-field. A possible reason for the high bias could be that the network outputs the coefficients for one solution, but the expected outputs are those of the other solution. This would result in a high loss value even when the network has worked correctly. To try and rectify this, new architectures were required.

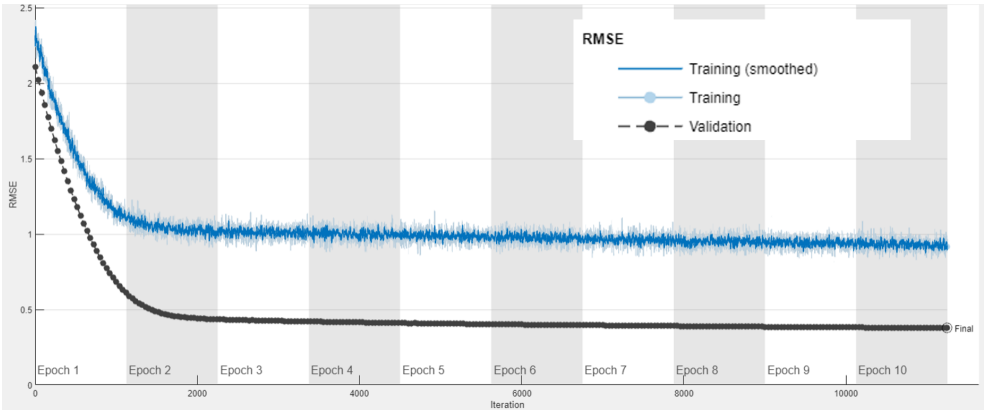


Figure 11: Xception training progress on 36,000 images. Again the error was too high. The validation error was lower than the training error because regularisation, such as dropout, was applied to the training dataset but not the validation dataset.

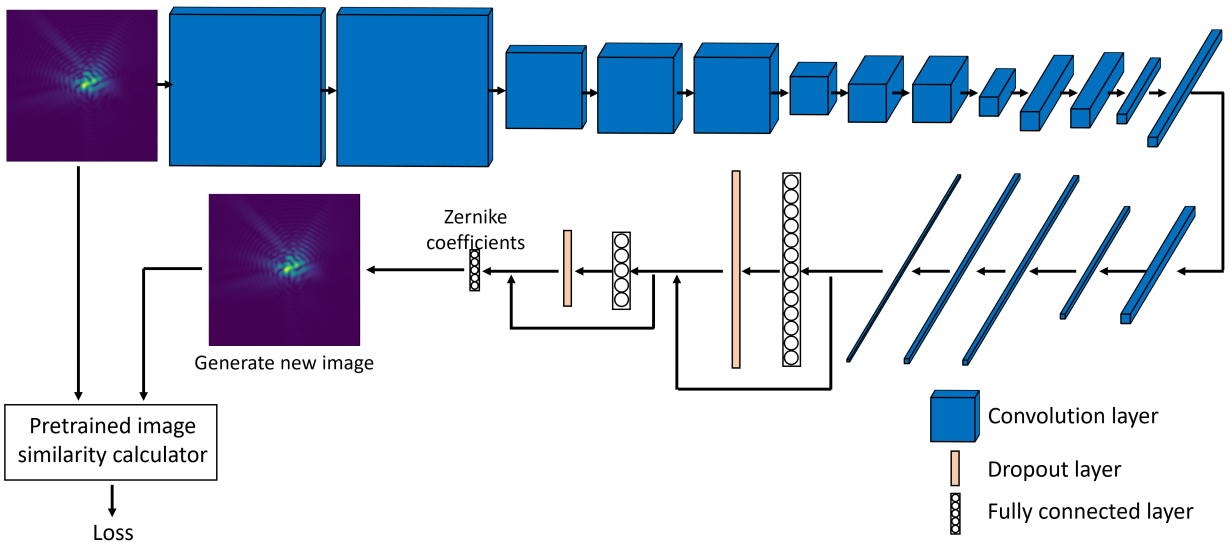


Figure 12: SimNet architecture. Zernike coefficients are predicted from an input image. The far-field that these coefficients would produce is found and compared to the input image. This similarity is the loss function used during training.

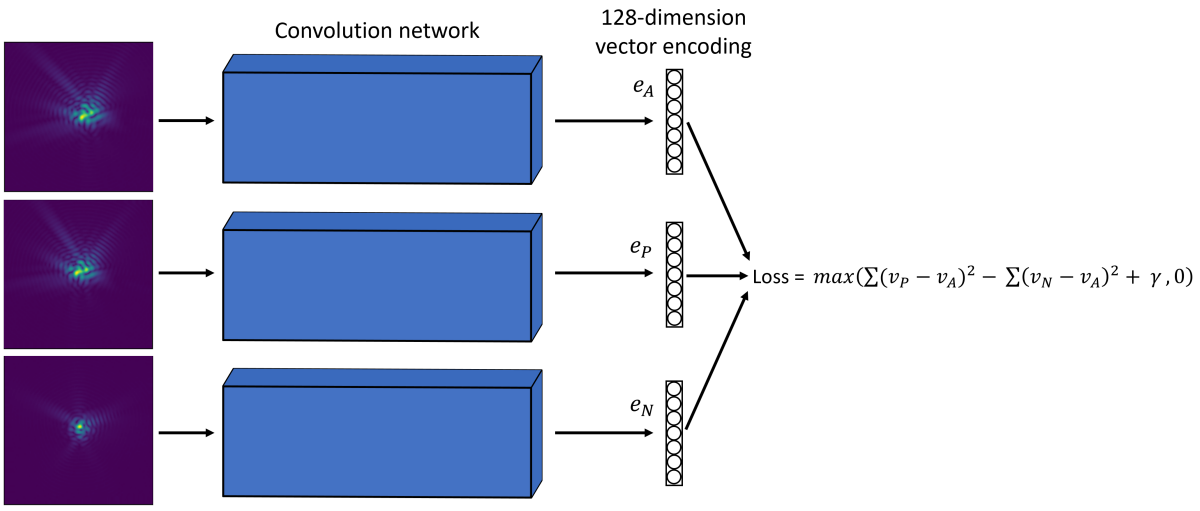


Figure 13: Similarity calculator. Trained as a siamese network, there are 3 instances of the same, identical network. The dataset used to train this was in batches, with an anchor, positive, and negative image. The network outputs a 128-dimension vector encoding of each image and the loss function tries to make the positive and anchor encodings as similar as possible, and the negative and anchor encodings as different as possible.

The initial plan was as outlined in Figures 12 and 13. The portion of the network that reduced the input image down to Zernike coefficients was similar to before, but following this, the coefficients were used to generate a new far-field shape, which was compared to the input by a pre-trained similarity calculator. The image similarity was then used to update the trainable layers; the structure was referred to as SimNet, based on the measurement of image similarity.

SimNet consisted of 18 convolution layers, with batch normalisation and ReLU activations. Pooling layers were not included because of the nature of far-field images: there are small concentrations of high intensity and large areas of very low intensity. Therefore, it was thought that pooling would neglect many important features. By the end of the convolution layers, the output has dimensions $(4 \times 4 \times 512)$, at which point it is flattened and passes through 2 fully connected layers, again with RelU activations. After each of these, there is a dropout layer with a drop probability of 0.5. Additionally, there are skip connections both from the flattened outputs to the second fully connected layer, and from this layer to the final layer, to pass information deeper into the network. The final layer itself is a fully connected layer with 12 outputs (the 12 coefficients) and no activation function.

The similarity calculator was itself a CNN that reduced input images down to 128-dimension vector encodings (or embeddings) of the images. While similar to how images are reduced to Zernike coefficients, this network did not suffer from the same two solution weakness as before, because of the way in which it was trained as a Siamese network. The structure consisted of three identical network instances that each took an input far-field: an anchor, a positive image (that was the same far-field as the anchor), and a negative image (simply a different far-field image). The Zernike coefficients for the positive image were altered very slightly from those for the anchor, so that the images were not identical, as this would be too easy and the network would not properly learn the relevant features. Small random rotations and zooms were also applied to make the images more realistic. The loss function was defined as,

$$J(e_A, e_P, e_N) = max\left(\sum (e_A - e_P)^2 - \sum (e_A - e_N)^2 + \gamma, 0\right) \tag{27}$$

where γ is a constant and e_A, e_P, and e_N are the vector encodings of the anchor, positive, and negative images respectively. The network updates to make the difference between e_A and e_P as small as possible,

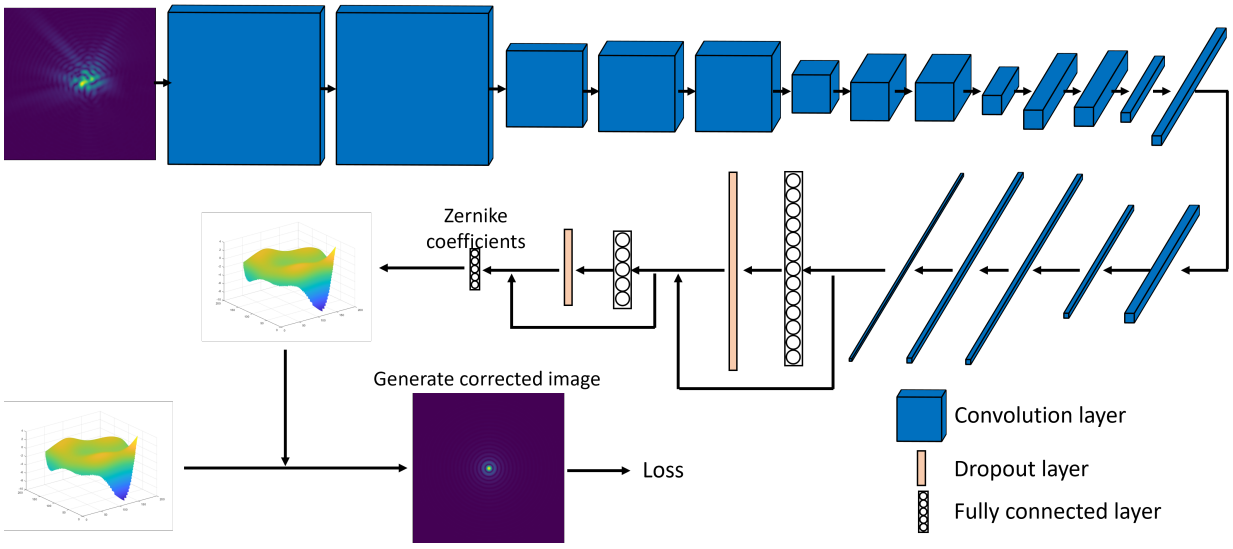


Figure 14: QualNet architecture. Again, Zernike coefficients are predicted from a far-field image. The wavefront shape they define is found and so is the corresponding inverse that gives the same far-field. Each shape is combined with the known input wavefront; the image quality of the far-field of the flattest of these is then calculated and used as the loss function.

while maximising the discrepancy between e_A and e_N. The constant, γ, is the margin that defines how large the distinction is between the summations. Once trained, image similarity can be found using the cosine similarity between the encodings, where a value of 1 would denote identical images, 0 would be completely unrelated images, and -1 would be given for opposite images. The full formula for the loss function used by SimNet was,

$$J(e_I, e_G) = 1 - \frac{e_I \cdot e_G}{||e_I||||e_G||} \tag{28}$$

where e_I and e_G are the encodings of the input and generated image correspondingly.

Siamese nets have already proven their ability to learn similarities in images [15]. The cosine similarity between the positves and anchors was generally ≈ 0.999, which is ideal. However, the similarity calculator struggled to learn the distinctions between the negative and anchor images, giving a cosine similarity between them of ≈ 0.997. Far-field images are always going to be very similar, and learning the differences with this method would likely require a much larger network and more data, making it extremely computationally expensive. Therefore this was considered too inefficient and was not attempted. Given the flaws of the similarity calculator, SimNet was unable to yield useful results.

4.6.2 QualNet

A new network design removed the need for the similarity calculator. This architecture will be referred to as QualNet, as it is based on utilising the image quality metric. the full structure is shown in Figure 14. Again a far-field image was converted to estimated Zernike coefficients, in the same way as in SimNet; then, the wavefront shape described by these coefficients is produced. A duplicate of this shape is inverted in every dimension, the equivalent possible solution. Combining each of these shapes with the original, input shape gives the potential resultant shapes after correction. Only the resultant shape with lowest root mean squared error (RMSE) was included in the loss calculation; this RMSE value could itself be used as the loss function, but considering that the far-field was generated to view the correction results, the image quality of the far-field was used instead - Equation 6.

The network was, at first, trained on $36,000$ images of square rooted far-fields. After 4 epochs (complete passes of the dataset), the network was relatively successful at quickly improving the far-field as seen in

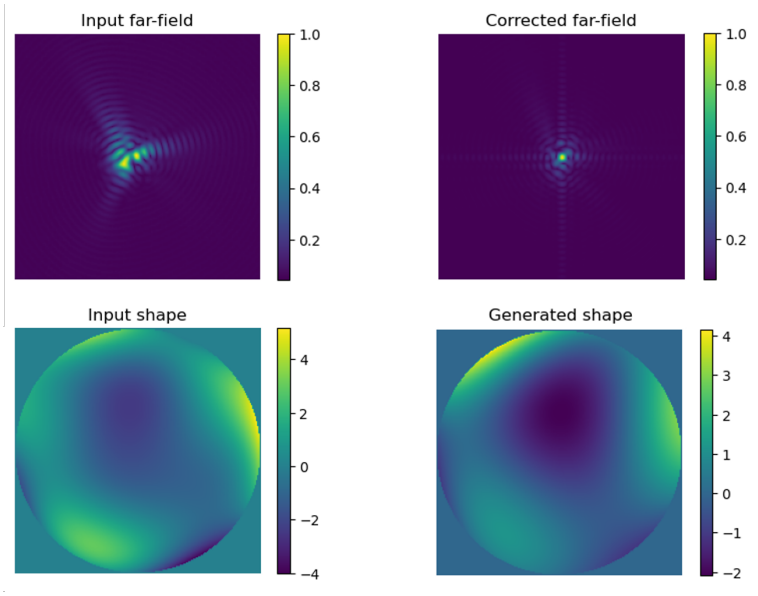


Figure 15: QualNet square rooted far-field correction example

Figure 15. Real, square-rooted far-field images would not be possible to obtain from 8-bit images returned from a camera, as pixel values are rounded to the closest integer, losing information in the low intensity regions. Nonetheless, using 12 or 16-bit images should provide the accuracy at low level pixel values to properly distinguish between features and allow for this pre-processing. On the other hand, the network should ideally work on raw images returned by the camera. Therefore, the network was retrained on unprocessed, simulated far-field images. The training again consisted of $36,000$ images but, after 4 epochs, the model appeared to be starting to overfit. Subsequently the dataset was swapped out for another $36,000$ images, made with the same method, before training for another 4 epochs, the results of which are in Figure 16. While the corrected far-fields are greatly improved compared to the inputs, the network is struggling without the assistance that square-rooting provided. While the resultant wavefront in Figure 16 is not perfect, it is greatly improved, with the average Strehl ratio of the inputs increasing from 0.0138 to 0.1861 in the corrected far-fields, with some achieving ratios of $0.3 - 0.4$. This improvement should continue to increase as the network is trained on more data, although changes to the architecture and implementing more regularisation techniques will likely be required. Additionally, the network could be applied to find an immediate rough solution before a machine learning algorithm, such as those in Section 2, finishes off the smaller details.

Applying a real correction with QualNet was attempted in Figure 17. The focus has visibly improved but not by a particularly large degree. However, there are several factors that could be limiting the performance, particularly the fact that the Zernike coefficients returned by the network were converted to voltages for the actuators on the mirror. This requires knowledge of the influence functions of the actuators, something that was not possible in this case: the influence functions were instead reasonable estimations. Given that, despite this, the far-field has visibly improved, the possible applications of the network appear quite promising. Examining the simulated and real far-field images also shows some differences. The minimum values in the 8-bit simulated images seems to be 11, whereas that for the real images is a more logical value of either 1 or 0. It is not clear if there is an error in how the simulated images are generated that is causing this, but the fact that the real data that the model needs to correct for is different to the data it was trained on would only harm the performance. A logical next step would be to continue training the model on real images from the camera. A second deformable mirror would probably be needed to introduce changing abberations for the network to identify; these could then be directly applied to the other mirror and the loss function could be the image quality of the corrected far-field. The exact influence

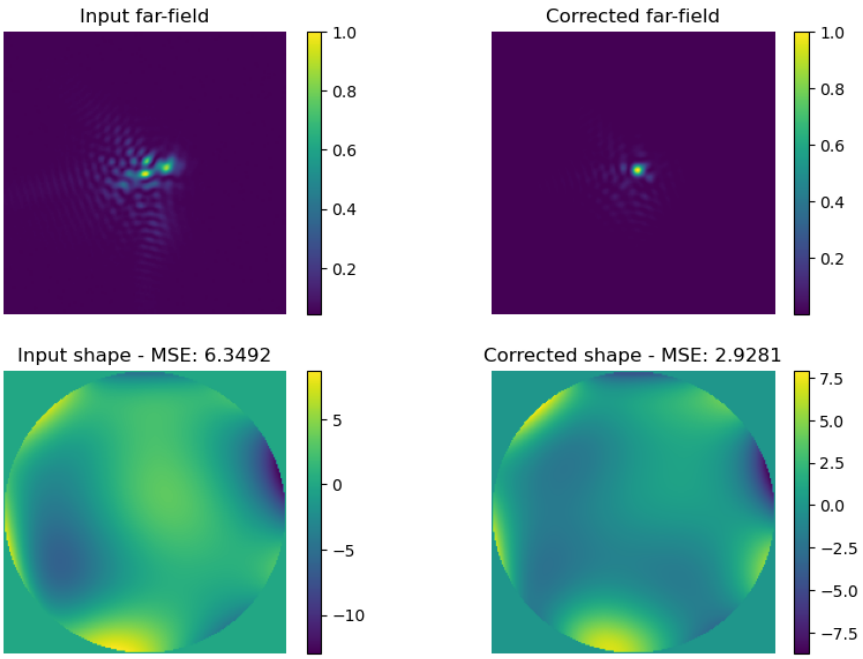


Figure 16: QualNet correction of unprocessed far-field

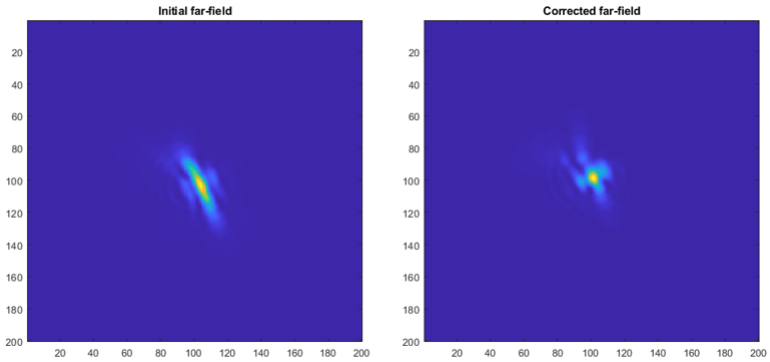


Figure 17: Real mirror correction with QualNet

functions of the mirror actuators would also not need to be known as the network would learn to account for these in its outputs. Although, if the network learns to account for the influences on one mirror, it may affect performance if used on a different mirror.

For a few examples of the test set, the corrected wavefront shapes actually had slightly higher mean squared errors than the input shapes. Despite this, the corrected far-fields were still generally noticeably improved, as the shapes were more flat overall in terms of their smoothness. It would be interesting to see how using the mean squared error of the corrected shape as the loss function during training would affect the performance, instead of using the image quality of the far-field.

References

[1] J. Goodman, *Introduction to Fourier Optics*, 2nd ed. McGraw-Hill, 1996.

[2] P. Oratiou, "Wavefront sensor-less adaptive optics," 2020, unpublished.

[3] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[4] G. Hinton, "Coursera neural networks for machine learning lecture 6," 2018.

[5] M. Dorigo, M. Birattari, and T. Stützle, "Ant colony optimization," *Computational Intelligence Magazine, IEEE*, vol. 1, pp. 28–39, 12 2006.

[6] R. Gerchberg, "A practical algorithm for the determination of phase from image and diffraction plane pictures," *Optik*, 1972.

[7] J. Antonello, M. Verhaegen, R. Fraanje, H. C. G. Tim van Werkhoven, and C. U. Keller, "Semidefinite programming for model-based sensorless adaptive optics," 2012.

[8] O. S. H. Yang and M. Verhaegen, "Model-based wavefront sensorless adaptive optics system for large aberrations and extended objects," *Opt. Express*, vol. 23, pp. 24 587–24 601, 2015.

[9] R. L. Karen Mary Hampson, Jacopo Antonello and M. Booth, *Sensorless Adaptive Optics*, 10 2020. [Online]. Available: https://doi.org/10.5281/zenodo.4066425

[10] B. Wang and M. J. Booth, "Optimum deformable mirror modes for sensorless adaptive optics," *Optics Communications*, vol. Volume 282, Issue 23, 12 2009.

[11] A. F. Delphine Débarre and E. Beaurepaire, "Assessing correction accuracy in image-based adaptive optics," 2012.

[12] H. Linhai and C. Rao, "Wavefront sensorless adaptive optics: a general model-based approach," *Opt. Express*, vol. 19, pp. 371–379, 2011.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," 2012.

[14] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," 2016.

[15] H. Essam and S. L. Valdarrama, "Image similarity estimation using a siamese network with a triplet loss," https://keras.io/examples/vision/siamese_network/, 2021.