Freeform lens predictions by a Neural Network and B-splines

Utilizing the Fraunhofer approximation to train a Neural Network unsupervised

by

Joost Imhof

to obtain the degree of Bachelors of Science
at the Delft University of Technology,

Student number: 4732111
Project duration: April, 2020 – October, 2020
Thesis committee:

Dr. A. Adam,	TU Delft, supervisor Applied Physics
Dr. M. Möller,	TU Delft, supervisor Applied Mathematics
Dr. L. van Iersel,	TU Delft, EEMCS
Dr. E. Greplová	TU Delft, AS

This thesis is confidential and cannot be made public until September 1, 2020.

TU Delft

Acknowledgement

Firstly I of course want to thank Alex Heemels for his consistent interest and helpful advice. It was great to have you always available for questions and to think about the next steps. In addition to Alex, Aurèle Adam and Matthias Möller first of all were very helpful in defining the project at the start. I am also very thankful that you made time to meet once every 2 weeks (or on demand when I needed a meeting), which really helped to stay on schedule, look back at previous results and define the next stage.

Luckily my mother, father, sister and brother were always there to listen to my excitement even though they probably understood one percent of every thing I said. Thank you guys!

Lastly I want to thank Leo van Iersel and Eliška Greplová for being part of the assessment committee.

Contents

Abstract

There exist a lot of methods to find the electromagnetic field behind a lens, so called the forward lens problem. In contrast very few methods can do the inverse, namely finding a lens that would produce a given image or light distribution for a known source. Here a solution to the forward problem, the Fraunhofer approximation, is used to find an approximate solution of the inverse problem using a neural network. Given an input image the network would predict the lens that can produce this image. In general this lens is not the classic convex/concave shape but is a freeform lens. The predicted image produced by the predicted lens can be computed by the Fraunhofer approximation. To train the network this predicted image is compared to the desired image. This unsupervised training method is similar to that of Physics Informed Neural Networks (PINN), which is a recent approach to solve PDE's.

The phase contour of the lens is represented by a B-spline curve. The control points of this spline are the output of the network. In this way very few output variables can be specified by the network to achieve a smooth detailed lens shape.

To give a proof of concept a one dimensional version of the Fraunhofer approximation is used. For this case the training already depends on many parameters. These are optimised for the lowest resulting loss.

The Fraunhoffer approximation limits the images that can be created. If the network, however, is trained on images that can definitely be created, it is able to almost exactly predict a lens that delivers the desired image.

The potential of the unsupervised training method in combination with a spline approximation should therefore be explored with other solutions of the forward problem. Namely a ray-tracing algorithm could be used, since this can create a wider variety of images. This would be computationally heavy compared to the Fraunhofer approximation.

Introduction

Many people will remember seeing and doing calculations with light passing through a convex lens in their secondary school physics class. It is widely known that such a lens will focus light on a single point. The electromagnetic field behind a more complex lens can be calculated in various ways; for example: via ray-tracing or a refraction integral. In contrast the inverse problem is immensely more difficult: what is a lens that can produce a given light distribution or image. For a complicated image this will in general not be a simple convex or concave lens but will be a so-called freeform lens. If such a lens could make any desired image, freeform lenses could have wide applications, such as inside various laser sensors, imaging systems or microscopes.[1] In the future freeform lenses may be exact enough to be able to print computer circuits. The great advantage of these lenses will be the weight of the systems, since often a couple of freeform lenses can be used instead of multiple classic lenses. Nowadays some freeform lenses are already used, for example inside the headlights of cars to focus the light down on the road or in streetlights to spread the light more evenly on the road.

Recently this inverse freeform problem has been explored by researchers. For example Y. Schwartzburg et al. have applied geometrical optics and optimal mass transport to approximate a lens that gives a desired image [Schwartzburg et al. (2014)].

In this thesis machine learning will be used to solve this problem. Ultimately a neural network is wanted that can output a lens shape that creates the desired image, with a known light source. A training technique used in Physics Informed Neural Networks (PINN) [Raissi et al. (2019)] is applied to the inverse lens problem. Using the training technique it should be possible to train the network unsupervised on arbitrary images, without prior knowledge of the lenses that created these images. To apply this training method a solution of the forward problem is needed, this is chosen to be the Fraunhofer approximation. Especially the 1 dimensional case is very easy to use and can be numerically computed very efficiently, which makes it perfect to use in neural networks. The phase contour of the lens will be represented by a B-spline, a method of parametrising a smooth curve. It enables us to describe a smooth lens with a few characteristic points, which are the output of the neural network.

This thesis will not solve the inverse problem for the most general case, it can serve as a proof of concept for the proposed method. This thesis tries to answer the following question:

In what way can a physical informed neural network and a unsupervised training method be used to solve the inverse lens problem using the Fraunhofer approximation, where the phase contour of the lens is described by a continuous spline function?

The proposed optical setup and approximation limits the images that can be created by a general freeform lens. Because of using the Fraunhofer approximation and the defined properties of the continuous spline, the inverse problem will in general not have a solution. Therefore the network will be trained on images that actually can be created by the optical setup. This limiting factor is caused by the proposed training method. This thesis will thus motivate further research using this training method.

When the training is confined to producible images, the network is able to decrease the loss to only 0.15% of the untrained average loss. This is very promising, since the network can almost exactly predict a right lens that creates the desired image. Consequently future research can use this unsupervised training in combination with a more flexible forward solution to solve the inverse problem. This can be a ray-tracing algorithm, such an algorithm will be computationally heavy if used in combination with a neural network.

In order to come to this conclusion first the underlying theory of the Fraunhofer approximation, neural networks and B-splines is explained in the next chapter. Afterwards the experimental setup of training the network is given. With this setup the network will be trained multiple times to optimise this process. The optimised network is then used to solve the inverse problem. At the end problems that arise with this method are discussed.

[1] See for example applications by the Dutch company TNO: https://www.tno.nl/en/tno-insights/articles/optics-2-0-the-future-of-freeform-optics/

1 Theory

1.1. Optics

1.1.1. Light Propagation

In a big portion of optics physicists are occupied in finding the resulting light distribution given a combination of light sources and an optical system. If for example we place a light bulb in front of a lens, what will the image be on a screen behind this lens? This problem is often presented in physics courses at secondary school. This kind of problem will be called the forward lens problem.

Geometrical Optics

The forward problem can be quite easily solved inside the framework of geometrical optics. Its origins goes as far back as ancient Greece and Egypt. Euclid (300 B.C.) made notions about the rectilinear propagation of light rays and the law of reflection [Vohnsen (2006)]. Ptolemy (170 A.D.) from Egypt stated an early version of the law of refraction, also known as Snell's law (sometimes called Descartes law). This law is named after Willebrod Snellius, who refined the law in 1621. These principles form the basis for geometrical optics. Here light follows a straight path inside a homogeneous medium and gets refracted at a boundary between two optical media. With geometrical optics we can easily trace every ray going trough a lens. It can for example be easily found that a convex circular lens focuses incoming parallel rays into one point. The output of more complex systems can be numerically computed with ray tracing programmes. However to compute the resulting image, millions of rays should be traced trough the system. This can be computationally expensive.

Wave Optics

In 1665 Grimaldi found that light could bend around a corner (this is called diffraction), just like sound waves can [Vohnsen (2006)]. This implicated that light may also have a wave aspect and does not only propagate rectilinear. Although Grimaldi himself denied this conclusion, Huygens strongly believed in the wave aspect of light. In 1690 he proposed the Huygens principle, explaining that waves propagate as a spherical wave front. Every point on this wave front, a wavelet, will itself then emit a secondary spherical wave. If these waves are added up we again find a spherical wave front. This is imaged in Figure 1.1(a). With this principle we can explain diffraction, since every point inside for example an opening, radiates a spherical wave. For large diffraction angles and large openings the light intensity will decrease. Here the waves will not form a nice wavefront. In other words, the spherical waves will not constructively interfere. This assumes monochromatic light, such that the waves all have the same frequency, amplitude and phase. [Ling et al. (2016)]

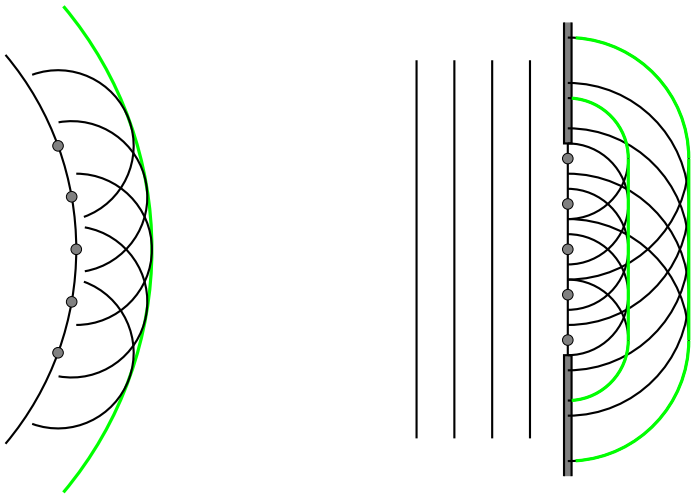
For a narrow opening, mostly called a slit, constructive interference and destructive interference occurs for certain angles from the surface normal. Destructive interference between two incoming waves occurs when the phase difference is exactly $\frac{1}{2}\pi$ or 90 degrees. Constructive interference occurs for a phase difference of 0 or π. Interference is schematically displayed in Fig. 1.2. A more extensive description of interference is given in [Ling et al. (2016)], here is also the angle criterion for constructive/destructive interference derived.

Fraunhofer Domain

If the plane, on which the light distribution is projected, is taken far (compared to the slit width) away, this far-field domain is often called the Fraunhofer domain. In which the Fraunhofer Approximation is valid. This approximation is proportional to the Fourier transformation (FT), an extremely widely used mathematical transformation. The Fourier transform can also be easily used in numerical analysis using the Discrete Fourier transform (DFT). Which can be computed very efficiently using the Fast Fourier Transform (FFT) algorithm. This efficiency of the Fraunhofer approximation is very useful in our application. The Fraunhofer approximation can be derived from the Rayleigh-Sommerfield Diffraction integral. The next sections are devoted to this derivation.

1.1.2. Scalar Diffraction Optics

In 1861 James Clerk Maxwell published an early form of the famous Maxwell equations. These equations form the basis of electromagnetism and demonstrate that fluctuating electric and magnetic fields are coupled and propagate with constant speed (c). This speed is exactly equal to the speed of light, implying that light consists



(a) (b)

Figure 1.1: (a): Huygens principle displayed, every point on a wave front emits a spherical wave. These spherical waves combine together in the green wavefront. The spherical waves are emitted from the grey dots. (b): Huygens principle applied to a plane wave propagating through a opening. After the opening the wave is bend around the edges, a.k.a diffraction occurs. The waves travel to the right.

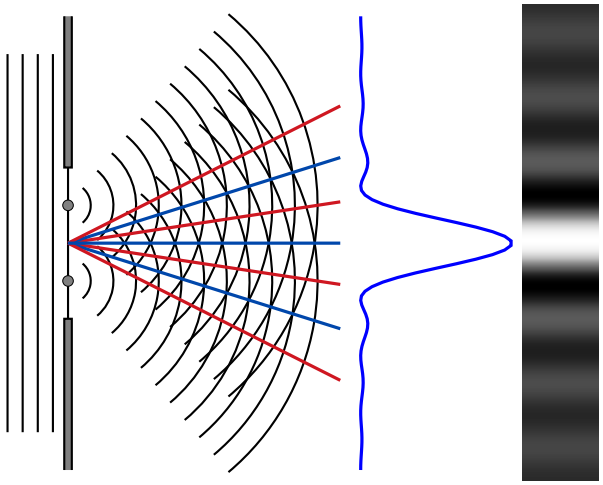


Figure 1.2: Interference caused by a single slit, simplified. From the left enters a plane wave, after the slit the two wavelets will cause an interference pattern. The blue and red lines indicate constructive and destructive interference respectively. You can see that where the wavefronts cross each other there is constructive interference, they exactly cancel each other in between. In blue is also given the resulting graph of the light intensity, this is a squared sinc function. To the right of this graph is given the intensity pattern. Note that there will exist an infinite amount of wavelets in the slit, these do not precisely constructively interfere, therefore the main peak is the largest.

of electromagnetic waves. In absence of free charge the Maxwell equations are given by:

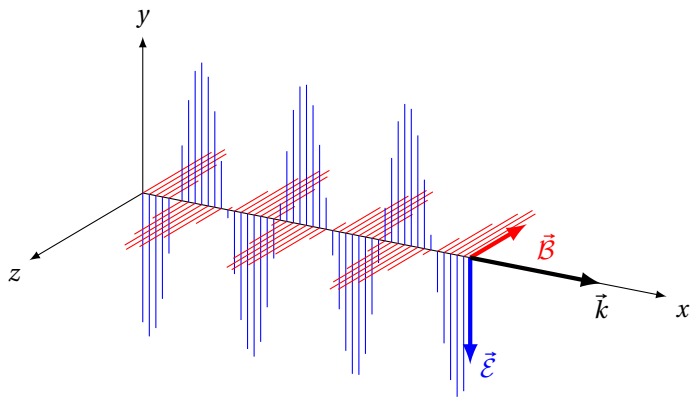


Figure 1.3: Light displayed as en electromagnetic wave. Decomposed in an electric field $\vec{\mathcal{E}}$ and a magnetic wave $\vec{\mathcal{B}}$. The direction of travel is also indicated with the wave vector $\vec{k}$, its length is inversely proportional to the wave length. Note that the 3 vectors are perpendicular.

$$\nabla \times \vec{\mathcal{E}} = -\mu \frac{\partial \vec{\mathcal{B}}}{\partial t} \tag{1.1a}$$

$$\nabla \times \vec{\mathcal{B}} = \epsilon \frac{\partial \vec{\mathcal{E}}}{\partial t} \tag{1.1b}$$

$$\nabla \cdot \epsilon \vec{\mathcal{E}} = 0 \tag{1.1c}$$

$$\nabla \cdot \mu \vec{\mathcal{B}} = 0. \tag{1.1d}$$

Here the electric and magnetic field are respectively $\vec{\mathcal{E}}$ and $\vec{\mathcal{B}}$, depended on the position vector $\vec{r}$ and the time t which both have been left out for convenience. ϵ and μ are the permittivity and permeability in the medium of propagation and may depend on $\vec{r}$. By using the nabla operator ∇, the curl and divergence of the fields can be represented by a short notation. Light can be described at every point in space $\vec{r}$ by 3 vectors; the direction of propagation, the electric field vector and the magnetic field vector, Fig. 1.3. From the Maxwell equations it follows that these are all perpendicular. The speed of light in vacuum is given by

$$c = \frac{1}{\sqrt{\mu_0 \epsilon_0}}, \tag{1.2}$$

where the subscripts indicate vacuum. By following the approach of [Lucke (2006)] and [Heurtley (1973)] the Rayleigh-Sommerfield integral can be derived as follows.

We assume the following reasonable properties of the medium: The medium has a constant permittivity (homogeneous), the permeability is equal to the permeability of vacuum (non-magnetic) and its properties are independent of the field polarisation (isotropic). From the Maxwell equations it then follows that the fields both obey a vector wave equation

$$\nabla^2 \vec{\mathcal{U}} = \frac{n^2}{c^2} \frac{\partial^2 \vec{\mathcal{U}}}{\partial t^2}. \tag{1.3}$$

With $\vec{\mathcal{U}}$ a vector field, in our case $\vec{\mathcal{E}}$ or $\vec{\mathcal{B}}$, and $n = \sqrt{\epsilon/\epsilon_0}$ the refractive index. This can be split up for every vector component into an identical scalar field equation. For the x component of the electric field $\mathcal{E}_X$ this scalar field equation is given by

$$\nabla^2 \mathcal{E}_X = \frac{n^2}{c^2} \frac{\partial^2 \mathcal{E}_X}{\partial t^2}. \tag{1.4}$$

This is similar for all the other components of the electric and magnetic field. It is therefore possible to describe the entire electromagnetic wave by one scalar field, we call this field $\mathcal{U}$. [Lucke (2006)]

For a monochromatic source an electromagnetic wave will only consist of one frequency ω. We can then describe the scalar field of a plane wave by a sinusoidal wave

$$\mathcal{U}(\vec{r},t) = A\cos(\vec{k}\cdot\vec{r}-\omega t+\phi) = A(\vec{r})\cos(-\omega t+\phi(\vec{r})) \tag{1.5}$$

Where $\vec{r}$ is the position vector, t is the time, A is the amplitude, $\vec{k}$ is the wave vector and ϕ corresponds to the phase, which is absorbed by the $\phi(\vec{r})$ in the second part. This field will be a solution of equation 1.4 when the length of $\vec{k}$ is equal to ω/c. For more general time harmonic waves the amplitude can depend on $\vec{r}$. Using complex notation we can write this as

$$\mathcal{U}(\vec{r},t) = \Re(A(\vec{r})e^{i\phi(\vec{r})}e^{-i\omega t}) := \Re(U(\vec{r})e^{-i\omega t}). \tag{1.6}$$

Where $\Re$ takes the real part of the complex number. Also the complex amplitude has been defined, sometimes called the complex field, $U(\vec{r})$. Adding a phase difference to this complex field is easy since it can just be multiplied by a complex exponential, e.g. $\exp(i\phi)$. Also since linear operations commute with taking the real part and multiplying by the time-dependent exponential, it is possible to use linear operation on the complex field instead of the real field for the same result. To obtain the actual field from the complex field again Equation 1.6 can be used. $\mathcal{U}(\vec{r},t)$ still satisfies the scalar wave Equation (1.4). If Equation 1.6 is substituted for $\mathcal{U}(\vec{r},t)$ in Equation 1.4, then a time-independent differential equation for the complex field is found:

$$(\nabla^2+\omega^2/c^2)U(\vec{r}) = 0. \tag{1.7}$$

This is known as the Helmholtz equation.

For spherical waves the field only depends on the distance to the centre of the wave $r = |\vec{r}|$. By rewriting Equation 1.4 to polar coordinates we can derive for a time harmonic spherical wave that

$$\mathcal{U}(r,t) = \frac{A}{r}\cos(kr-\omega t+\phi) \tag{1.8}$$

$$U(r) = \frac{A}{r}e^{ikr+\phi} \tag{1.9}$$

The amplitude falls of with $1/r$ which is typical for spherical expanding waves.

1.1.3. Rayleigh-Sommerfeld Integral

So far we have looked at the field at some point $\vec{r}$ for a plane and spherical wave. The spherical wave originates from a wavelet. Consider now a plane for which the amplitude and phase is known, or equivalently the complex field is known, call this field the pupil plane. What will then be the field at some point $\vec{r}$? We consider the situation as displayed in Figure 1.4. z is taken constant, the field is then to be calculated for a certain point T with coordinates (x,y).

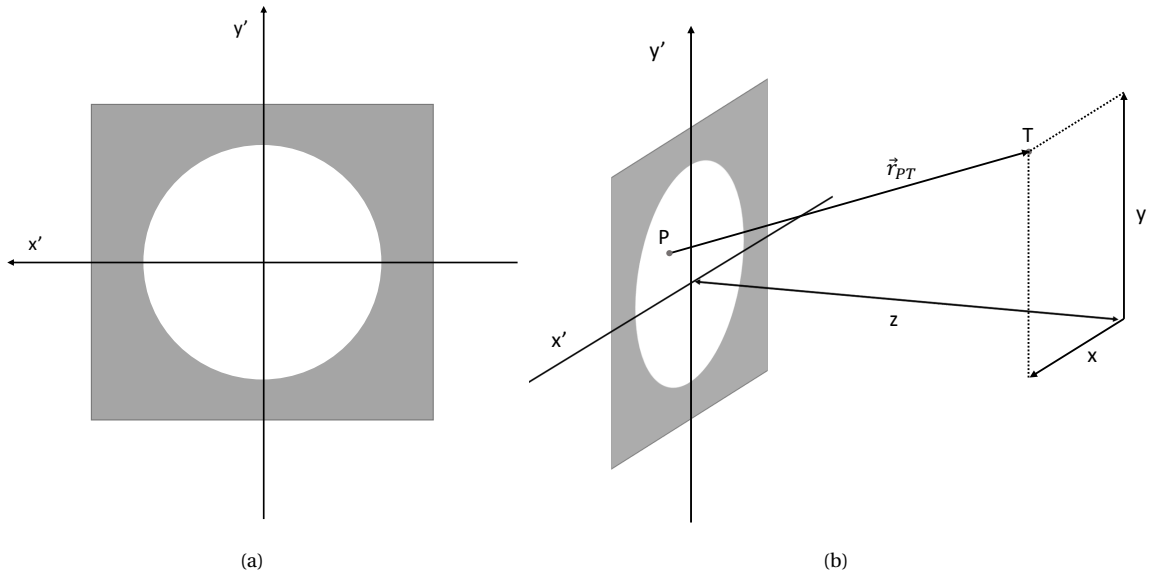
The Huygens principle states that every point in the pupil plane will be a wavelet and emit a spherical wave. The vector from such a point, P, to T is given by $\vec{r}_{PT} = (x-x', y-y', z)$. The field originating from this point is only dependent on the length of $\vec{r}_{PT}$, which is equal to $\sqrt{(x-x')^2+(y-y')^2+z^2} := r$. The field from one wavelet is then exactly given by equation 1.9. The amplitude A and phase ϕ depend on the field at the wavelets position, therefore take $U(x',y')$ to be the field in the pupil plane. The field at point T consists of the sum of all the spherical waves emitted by the wavelets in the pupil plane. Since there are an infinite amount of wavelets continuously distributed in the pupil plane, we must integrate over the entire pupil plane

$$U(x,y,z) = \iint_S U(x',y')\frac{e^{ik\sqrt{(x-x')^2+(y-y')^2+z^2}}}{\sqrt{(x-x')^2+(y-y')^2+z^2}}\,dx'\,dy' \tag{1.10a}$$

$$= \iint_S U(x',y')\frac{e^{ikr}}{r}\,dx'\,dy'. \tag{1.10b}$$

Here S specifies the surface of which the spherical waves originate. In this case the integral runs over the pupilplane. This can be interpreted as a convolution integral.

This integral is very similar to the Rayleigh-Sommerfeld Integral [Heurtley (1973)] which carries some



(a) (b)

Figure 1.4: (a): The pupil plane with a circular aperture (a pupil). The coordinates for points in this plane are displayed: x' and y'. (b): Propagation after the pupil plane: to be calculated is the field at point T at (x, y, z). The vector from a point in the pupil plane, P, to T is given by $\vec{r}_{PT}$. z is often taken constant such that x and y span a plane also, referred to as the target plane.

extra terms. This integral is given by

$$U(x,y,z) = \frac{1}{i\lambda} \iint_{pupil plane} U(x',y') \cos(\theta) \frac{e^{ik\sqrt{(x-x')^2+(y-y')^2+z^2}}}{\sqrt{(x-x')^2+(y-y')^2+z^2}} \, dx' \, dy' \tag{1.11}$$

$$= \frac{1}{i\lambda} \iint_{pupil plane} U(x',y') \frac{z}{r} \frac{e^{ikr}}{r} \, dx' \, dy'. \tag{1.12}$$

This introduces the wavelength λ of the wave, which is equal to $2\pi/k$, and the angle θ between the normal of the pupil plane and $\vec{r}_{PT}$, the cosine of this angle is exactly equal to z/r.

Rayleigh and Sommerfeld have added the factors $1/i\lambda$ and z/r to our original integral. This integral with its factors can be derived from the Helmholtz equation using Greens Functions, this is a somewhat extensive derivation, so it will not be given here but can be seen in for example [Goodman (2005)] or [Lucke (2006)]. The factor $1/i$ results in a $\frac{1}{2}\pi$ phase shift in the output field compared to the input wave, the amplitude of the field scales inversely with the wavelength due to the $1/\lambda$ factor and lastly the field amplitude is highest in the direction of travel of the incoming plane wave due to the $\cos(\theta)$ factor. These factors are mostly a mathematical artefact but are significant. The accuracy of this analytic expression compared to experimental results is widely investigate, it is also compared to other diffraction integrals like the Kirchoff Diffraction integral, see for example [Heurtley (1973)] or [Mendlovic et al. (1997)]. In the latter is also looked at the computational speed of this integral, it turns out that it is still computationally expensive.

1.1.4. Fresnel and Fraunhofer Approximation

Fresnel Diffraction Integral

The result of last section can be simplified, just like in [Lucke (2006)], by taking the distance to the screen z large compared to the pupil plane width, L (or somewhat equivalent the aperture radius R), and larger then every x and y considered. In this way only small angles θ will be considered and thus $\cos(\theta) = \frac{z}{r} \to 1$. Also $1/r \to 1/z$, Equation 1.12 results thus in

$$U(x,y,z) \approx \frac{1}{i\lambda z} \int_{-L}^{L} \int_{-L}^{L} U(x',y') e^{ikr} \, dx' \, dy'. \tag{1.13}$$

It is not feasible to replace r by z in the exponential since k can be of the order of 10^7 rad/m and a small phase difference can give a hugely different result. The binomial approximation can however be used, which can be derived by Taylor expanding $\sqrt{1+a}$ around zero, resulting in

$$\sqrt{1+a} = 1 + \frac{a}{2} - \frac{a^2}{8} + \mathcal{O}(a^3). \tag{1.14}$$

For a small we can use the first two terms only. By factoring out z we can for $z >> |x' - x|$ and $z >> |y' - y|$ approximate r by

$$\begin{aligned} r &= \sqrt{(x-x')^2 + (y-y')^2 + z^2} = z\sqrt{1 + (\frac{x-x'}{z})^2 + (\frac{y-y'}{z})^2} \\ &\approx z(1 + \frac{(x-x')^2 + (y-y')^2}{2z^2} = z + \frac{(x-x')^2 + (y-y')^2}{2z} \end{aligned} \tag{1.15}$$

Combined with Equation 1.13 this results in

$$\begin{aligned} U(x,y,z) &\approx \frac{e^{ikz}}{i\lambda z} \int_{-L}^{L}\int_{-L}^{L} U(x',y') e^{\frac{ik}{2z}((x-x')^2+(y-y')^2)}\, dx'\, dy' \\ &= \frac{e^{ikz} e^{\frac{ik(x^2+y^2)}{2z}}}{i\lambda z} \int_{-L}^{L}\int_{-L}^{L} U(x',y') e^{\frac{ik}{2z}(x'^2+y'^2)} e^{\frac{-ik}{z}(xx'+yy')}\, dx'\, dy'. \end{aligned} \tag{1.16}$$

By noting that $k = 2\pi/\lambda$ we can recognise the integral as the Fourier transform of $U(x',y') e^{\frac{ik}{2z}(x'^2+y'^2)}$ taken in the point $(x/\lambda z, y/\lambda z)$ thus

$$U(x,y,z) \approx \frac{e^{ikz} e^{\frac{ik(x^2+y^2)}{2z}}}{i\lambda z} \mathcal{F}[U(x',y') e^{\frac{ik}{2z}(x'^2+y'^2)}](\frac{x}{\lambda z}, \frac{y}{\lambda z}). \tag{1.17}$$

This integral is referred to as the Fresnel Diffraction Integral. The region for which this is integral is viable is referred to as the near-field. Exact constrains on z are derived in for example [Goodman (2005)]. A constraint is given by

$$z^3 \gg \frac{\pi}{4\lambda}((x-x')^2 + (y-y)^2). \tag{1.18}$$

For an aperture with diameter of around 1 cm and visible light, this constraint would give $z >> 25$ cm. This constraint can however be quite strict and better constraints can be derived. Notice that the convolution in Equation 1.16 changed to a multiplication in the x' and y' domain.

Fraunhofer Diffraction Integral

It is possible to constraint z even more: this is called the far-field or the Fraunhofer domain. The approximation of r in Equation 1.15 can be further approximated as follows

$$r \approx z + \frac{(x-x')^2 + (y-y')^2}{2z} \approx z + \frac{x^2 + y^2 - 2xx' - 2yy'}{2z} \tag{1.19}$$

Here the quadratic terms in x' and y' are left out, assuming that $z >> \frac{k(x'^2+y'^2)}{2}$. Note that this is a high lower bound for z since k is in general very large. With this approximation the factor $e^{\frac{ik}{2z}(x'^2+y'^2)}$ inside the Fourier Transform approaches unity. Therefore the field at (x,y,z) is surprisingly proportional to the Fourier transform:

$$U(x,y,z) \sim \mathcal{F}[U(x',y')](\frac{x}{\lambda z}, \frac{y}{\lambda z}). \tag{1.20}$$

In most problems we are not concerned with the pre-factors because most of the time we only look at the intensity, which is proportional to the fields amplitude squared. It can even be shown that the intensity is proportional to the amplitude of the complex field squared. In the end all the pre-factors scale the entire field or add a constant phase shift, which is not important for the amplitude. If we are concerned only with the shape of the intensity then all these pre-factors do not matter and are left out in most of the applications. In

the Fraunhofer domain, the image created by a plane wave entering a slit is, for example, the sinc function, defined by $\text{sinc}(x) = \frac{\sin(x)}{x}$. The sinc function is the Fourier transform of a square wave function.

As stated earlier, the Fourier Transform is a widely investigated mathematical tool and a whole subject on its own. The Fourier Transform also has a lot of useful and 'annoying' proprieties, these are also widely documented. For an entry level analysis of Fourier Transforms one can look at [Morrison (1994)]. Computational applications in optics can be found in [Voeltz (2011)] or [Schmidt (2010)].

1.1.5. Light Propagation through a lens

A lens can be added inside the aperture to create a different field in the pupil plane; the field directly behind the lens. This lens can be a classic spherical shape or some different (smooth) shape; called a freeform lens. At some point in the plane the lens can have a certain thickness Δz. It takes time for electromagnetic waves to travel trough a medium, specifically also trough this piece of the lens. The speed v with which it traverses a medium is variable for different media. The time it takes to cross this medium is $\Delta t = \frac{\Delta z}{v_{lens}}$. During this time it gets a phaseshift ϕ given by

$$\phi(\Delta Z) = 2\pi\Delta t f = 2\pi\Delta z \frac{f}{v_{lens}} = \frac{2\pi\Delta z}{\lambda_{lens}} = k_{lens}\Delta z = k_0 n_{lens}\Delta z \tag{1.21}$$

where we have defined the frequency f of the wave, the wavelength, λ_{lens}, and wavevector, k_{lens}, inside the medium and the wave vector k_0 in vacuum.

In the limit of a very thin lens the field behind the lens at (x',y') is only due to the field in front of the lens at (x',y'). The phase shift at this point will only be caused by the thickness of the lens at this point. This can be explained physically: Imagine that Rayleigh-Sommerfield propagation is used inside the lens. Assume that the field in front of the lens varies slowly compared to the thickness of the lens. Then the angle θ is large for points in front of the lens where the field varies greatly compared to the field at (x', y'). These points will not contribute in the integral of Equation 1.12, since $\cos(\theta)$ is small. In the limit of a very thin lens, the field in a point directly behind the lens is thus equal to that of the field in front of the lens with a phaseshift. This phaseshift is given by Equation 1.21. Using the complex amplitude notation we can add this phase with a complex exponential as follows

$$U_{behind}(x', y') = U_{infront}(x', y')e^{i\phi(\Delta z)} = U_{infront}(x', y')e^{ik_0 n_{lens}\Delta z} \tag{1.22}$$

The thickness Δz will depend on the position (x', y') in the lens and is often denoted as $W(x', y')$ and called the aberration function. In this case the lens is often called a phase shifting plate. The function that relates the field behind the optical system to the field directly in front is called the transmittance function of this system. For a lens, with thickness $W(x', y')$, inside an aperture the transmittance function, $P(x', y')$, is given by

$$P(x', y') = e^{ik_0 n_{lens} W(x', y')} \tag{1.23}$$

For a more in depth analysis of aberration diffraction one can look at [Goodman (2005)] or [Voeltz (2011)].

1.1.6. The Inverse Problem

In the last sections, a far-field solution of the forward lens problem is derived; how can we calculate the field given the light source and the optical system at some point behind the system. In the case of the Fraunhofer approximation the field, amplitude and phase, in a pupil plane is given. It is possible to calculate the resulting intensity pattern on some plane a distance z away, we call this plane the target plane. Is it however possible to do the reverse? Can we calculate the field in the pupil plane that results in a given intensity in the target plane? We call this the reverse problem. Can we even derive a lens or phase contour inside the aperture that creates this field in the pupil plane? Can this lens be smooth? Is the solution unique? To summarise we try to find a lens that creates a given target intensity for an specific incident field. A lens that is the solution of this problem will be in general a freeform lens.

Some optics groups have already looked at this problem. It can however be quite complicated because of a wide variety of optical setups. One can for example look at [Schwartzburg et al. (2014)] for a numerical approximation, here Optimal Mass Transport is used to find the right map from the starting light distribution to the target distribution. In this thesis the solution of the forward problem is used to find the corresponding lens shape. More specifically the Fraunhofer Approximation is used, since this approximation can be computed really fast using the Fast Fourier Transform. To fully incorporate this solution of the forward problem a Neural Network (NN) is used. The weights of this NN are adjusted with Back Propagation. The theory behind

Neural Networks and Back Propagation is given in the next section. A description of the simulation setup is given in the next chapter.

1.2. Neural Networks

Computers are taking over more and more work from humans. For over a century humans have looked at imitating humans and replacing daily tasks by machines. Since Warren McCulloch published a mathematical report in 1943 about the working of neurons inside our brain, humans have also been obsessed with mimicking the human brain. A brain is highly complex and has an enormous capability of learning. It takes toddlers 5 years to learn to have conversations and it takes another 5 years to learn to read and write. After more years this almost becomes instinctive and subconscious. Researchers and Mathematicians have tried to create networks that resemble the structure of the brain. These came to be known as Neural Networks (NN's). They try to achieve the same amount of learning capabilities as the human brain, or even better/faster. Data sets are fed into a machine on which the network can train and try to learn patterns. Neural Networks are one realisation of Machine Learning. Another one being Genetic Algorithms. These are all also a step towards Artificial Intelligence (AI). [Picton (1994)]

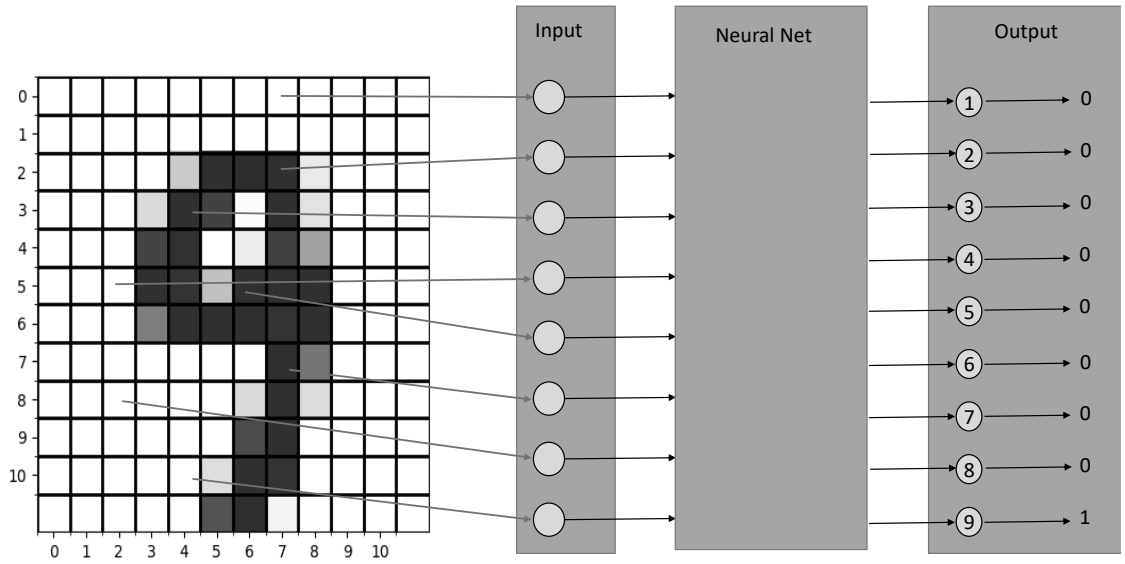


Figure 1.5: Input and output setup for a Neural Network that identifies handwritten digits. On the left is a 12 by 12 grid. Some of these pixels are given as input. In most applications all pixels will be used. The Neural Network will given 9 different output values between 0 and 1, indicating the digit. The close to one the more likely to be that corresponding digit. In the output the actual/target values are given, which the network uses to train itself.

1.2.1. In- and Output to the Network

A challenge in working with Neural Networks is finding a mathematical description of the problem with the right characteristic in- and output variables. Here this problem is displayed shortly.

Neural Networks can be well trained to recognise patterns. A Neural Network can for example determine which letter/digit is displayed (digital or handwritten), whether the picture is of a cat or a dog or whether a sentence with feedback is positive or negative. Consider the case of identifying handwritten digits (0-9). Let a picture of a digit of 12 by 12 pixels be given. The input can for example in this case be a 12 by 12 grid with a value representing the grey scale value in the corresponding pixel. The Neural Network could output 10 values ranging from 0 to 1, where every output corresponds to one digit. If the output corresponding to the digit 4 is 1 (or close to 1), it can be interpreted that the digit is probably a 4. A possible setup for the input and output is given in Figure 1.5.

Since the Neural Network will in most cases be a series of mathematical computations the input will most likely be some sort of numerical value; contained in a continuous interval, complex valued, integer

valued, binary valued (True/False) or representing a letter/word. However, most Neural Network packages are optimised for a normalised input; spread over the interval [0,1]. The Network will also output a numerical value, which can in theory be everywhere on the number line but most networks are again configured for normalised values. This output can then be interpreted as a probability, word, digit etc.

1.2.2. Single-Layer Networks

In the early phase of NN's, McCulloch and Pitts tried to mimic the biological neuron. The firing/activation of biological neurons is all or nothing, it fires or it does not. McCulloch and Pitts translated this to binary. A neuron fires if a large enough amount of other connected neurons fire, these activators need to pass a certain threshold. The neuron could be inhibited to fire by other neurons. An artificial neuron could be made to fire, output $y = 1$, if the sum of the activator inputs x_a surpasses a threshold T and the inhibitors x_{in} are all off (=0). This was the first artificial neuron [Graupe (1997)]. It can be mathematically expressed as

$$y = \begin{cases} 1 \text{ if } \sum x_a^i > T \text{ and } x_{in}^i = 0 \text{ for all i} \\ 0 \text{ otherwise} \end{cases} \tag{1.24}$$

Here the superscripts indicate the different inputs and there is summed over the different inputs. This McCulloch and Pitts neuron works great as a logic gate (e.g. AND), and can be combined with itself to mimic more complicated logic gates (e.g. XOR). The latter was not realised until later on.

However, it lacks the ability of learning and adapting, which is prominent in biological neurons, as stated by Hebb's rule. This ability to learn can be added by implementing adjustable weights w_i for every input, such that a weighted sum is computed. For the inhibiting inputs the weight is taken negative and are taken into the weighted sum. To this sum an offset b (sometimes called the bias) is added. This offset takes over the job of the threshold, which is set to 0. In this way we can take a Heaviside function, centred around 0, of this weighted sum and bias, to compute the output of the neuron. This kind of neuron is commonly called the perceptron and can be mathematically displayed as

$$\begin{aligned} z &= b + \sum w_i x_i \\ y &= f(z) \end{aligned} \tag{1.25}$$

When referring to a perceptron, the activation function $f(z)$ is often the Heaviside function, which returns zero for a negative input and one for a positive input. Perceptrons are even today the building blocks of networks and are sometimes called units in a network. Equation 1.25 is graphically displayed in Figure 1.6. For multiple binary output values perceptrons can be stacked on top of each other resulting in a layer. This setup will result in a finite set of output values (a 0 or a 1), which is useful in a classification problem as in the handwritten digits case. Such a single layer network could be able to classify digital digits, where every digit has only one representation on a small grid. Then every perceptron is trained for one digit. [Kurenkov (2018)][Graupe (1997)]

In 1960 Bernard Widrow and Tedd Hoff investigated outputting continuous functions [Kurenkov (2018)]. Classically the activation function for this neuron is the identity function, therefore the weighted sum will be the output. They called this kind of neuron an ADaptive LInear NEuron (ADALINE). To interpreted this output for a classification problem, the output could be thresholded. For the ADALINE case this is done with a sign function, a Heaviside function which outputs a -1 instead of a 0. The huge advantage of this neuron is the way of learning. Learning is done with the data from the weighted sum, before the squashing of the output by the sign function. In this way a learning algorithm sees how far of the prediction actually is. It can compute the difference, instead of only knowing if the neuron is right or wrong. Learning algorithms are later on this chapter explored. Note that classically this neuron is still a classifying neuron. Since the ADALINE is very similar it can also be graphically expressed as in Figure 1.6 and can be stacked on top of eachother to form a layer. [Graupe (1997)]

1.2.3. Multi-Layer Networks

Researchers soon argued that to solve more complicated problems, like mimicking a XOR logic gate which is not linearly separable, multiple layers of neurons needed to be used. An obvious choice is putting multiple ADALINE layers, including the sign function, in a sequence. This resulted in the Multi-layer ADALINE (MADALINE). A MADALINE has one or more hidden layers, for which the in- and output were not the in- and output of the network and therefore hidden. This could be used as a XOR gate. Achieving the right weights

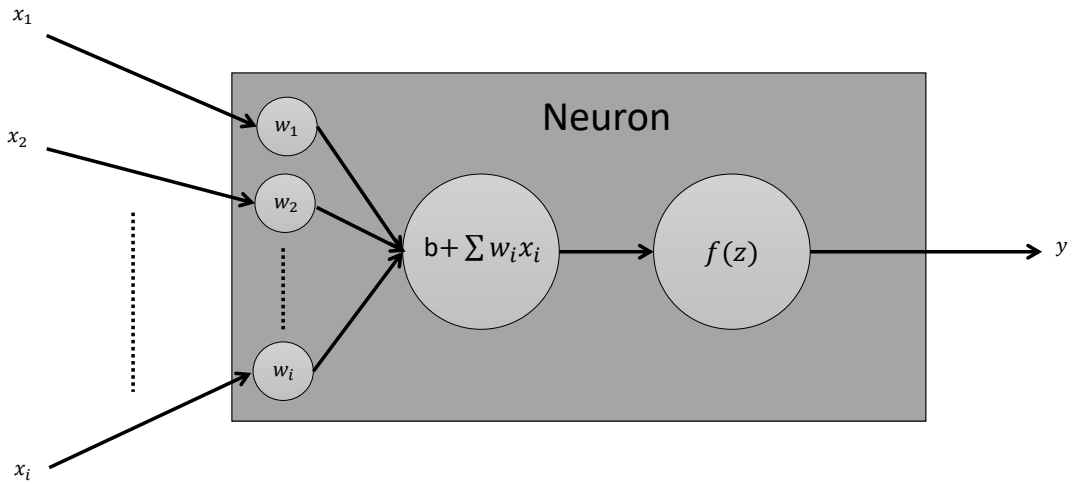


Figure 1.6: Diagram representing an artificial neuron. The inputs x_i are multiplied with the weights w_i and summed in the weighted sum $z = \sum w_i x_i$. The activation function $f(z)$ is taken of this sum after a bias b is added, which gives the output y. In the case of a perceptron the activation function is the Heaviside function. In the case of an ADALINE the activation function is the identity function. This diagram also shows a unit inside a network, like the MLP network, for such a unit the activation function is in general not linear but often continuous.

and biases was a hard problem, because the learning algorithm of the ADALINE only told how to compute the weights of the last output layer. This meant larger problems could not be solved yet. Also chaining together perceptrons did not have a working learning algorithm. This setback, together with negative reports about AI and the lack of fulfilment of the high expectations, resulted in huge budget cuts for AI research. This ushered in the first AI Winter, where close to no advancements were made. [Schuchmann (2019)]

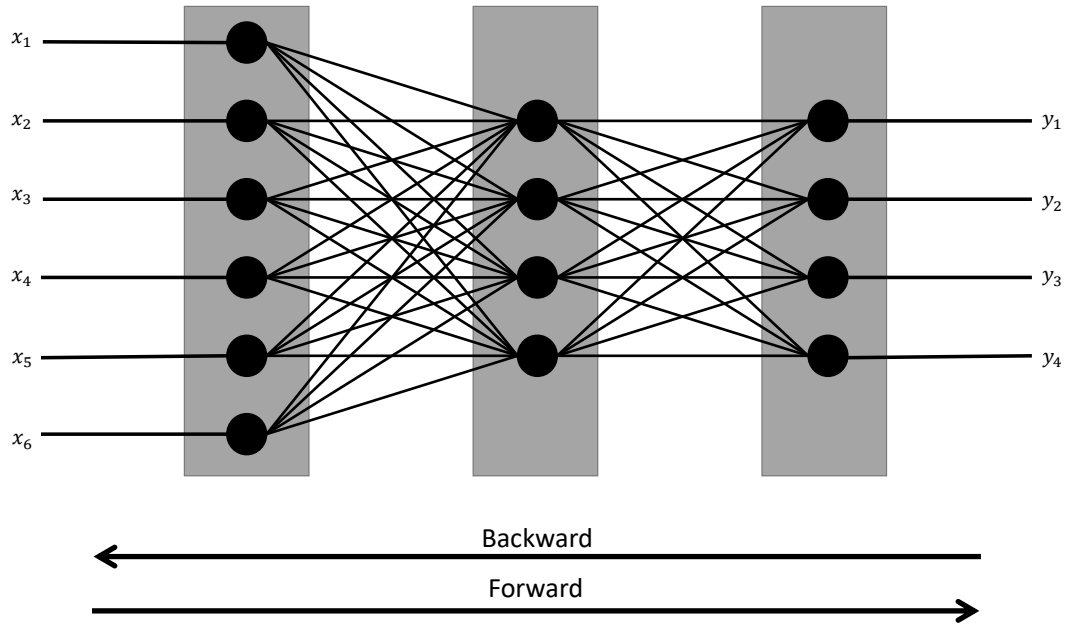


Figure 1.7: Diagram representing a feed forward MLP consisting of 3 layers, with inputs x_i and outputs y_i. The layers are linear and fully connected. The black dots represent a unit/perceptron with an arbitrary activation function. Data flows from input to output in the forward direction. Learning and updating the weights via backpropagation works via the backward direction.

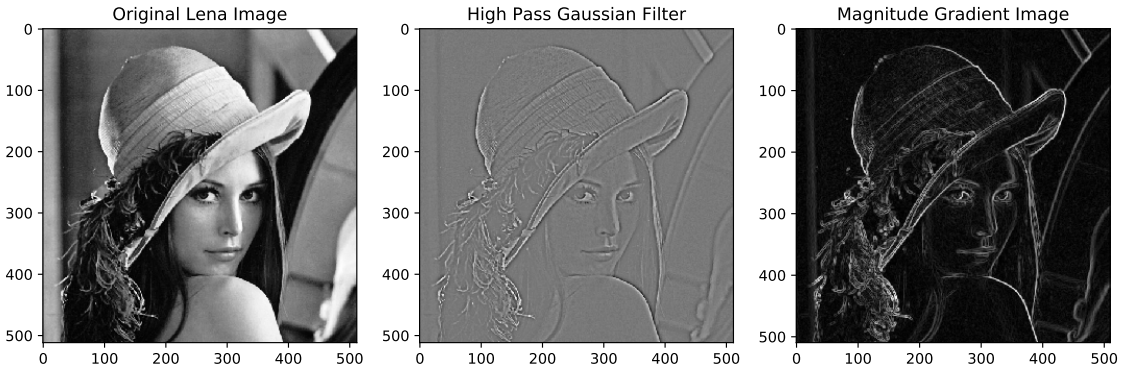


Figure 1.8: 3 images of Lena, a picture used in image processing, original image was placed in the Playboy magazine of November 1972. In the middle a high pass gaussian filter is used; here a gaussian low pass filter is taken of the image, this is substracted from the original. On the right the gradient of the image is taken, this is the root of the squared summation of the two directional gradients. Note that for the gaussian filter also negative values exist, therefore the value 0 corresponds to grey. In the gradient image every value is positive and therefore 0 value corresponds to black.

The end was marked by the discovery of backpropagation in 1986 by David Rumelhart. He proposed a Multi Layer Perceptron network (MLP), where the neurons inside the network were like perceptrons but with a differentiable activation function. With the chain rule it is then possible to compute the gradient with respect to all weights, also the ones in the hidden layers. The error could be backpropagated using gradient descent. This will be explained in section 1.2.5. Such a network consists of multiple layers, where every output is connected to the input of the next layer (fully connected). In this way the data is fed from a layer to the next one, always going from input to output/in the forward direction, this is therefore called feed-forward. A network that loops on itself is an example of a network that it not feed-forward. A MLP is schematically displayed in Figure 1.7.

This kind of layer is often called a linear layer due to the linearity of the weighted sum. If the emphasis is on the fully connectives then this kind of layer is called dense (densely connected).

The whole network is able to describe non linear relations by adding in non linear activation functions between the layers. Activation functions used mostly recently is the Recitfy Linear Unit (ReLU), this function returns zero for negative values and is linear for positive. For probabilty problems a sigmoid function is often used since this returns a value between zero and one.

Thanks to the discontinuity in the sign function MADALINE networks could not be trained via backpropagation. The training algorithm for MADALINE will not be explained here but can be read in [Graupe (1997)] or [Picton (1994)].

1.2.4. Convolution Layer

Early on researchers realised that these networks can recognise different patterns and features very well. The first hidden layer could for example find the basic shapes like lines, circles or a region with a plain color. The next layer can then "compare" these shapes and determine if it is what we are looking for. It was realised that this was so important that neural networks were often not trained on the raw data but data which was passed through some kind of feature extraction. Feature extraction can be achieved in a lot of different ways; image gradients, high- and low-pass filters or the Sobel operator are a few examples. These are all very similar and can all be somewhat generalised with a convolution kernel.[Kurenkov (2018)] Different filters are used in Figure 1.8. These filters clearly highlight lines and edges.

Here we will generalise image gradients to a convolution kernel, similar to [Graupe (1997)]. The gradient indicates the steepness of a function at a specific point, it points in the steepest direction. The gradient in the direction of an axis is equal to the partial derivative in this direction.

The discrete case will be explored here.

Say we have an image of NxN pixels, indexed by n (n'th row) and m (m'th column). The grey-scale value at a pixel is given by $F[n,m]$. To compute the discrete gradient at this pixel the difference quotient can be

computed. We centre this around the $[n,m]$ pixel, the gradient in the n direction is given by

$$\begin{aligned}\frac{\partial F[n,m]}{\partial n} &= \frac{-1*F[n-1,m]+0*F[n,m]+1*F[n+1,m]}{2} \\ &= \frac{1}{2}\begin{bmatrix}-1\\0\\1\end{bmatrix}\cdot\begin{bmatrix}F[n-1,m]\\F[n,m]\\F[n+1,m]\end{bmatrix}.\end{aligned} \tag{1.26}$$

In the second line the dot product of two (vertical) vectors is taken. The first vector can be called a kernel, although kernels are often taken square. The constant prefactor is often omitted. A square kernel results in an average of the gradient of neighbouring columns, as follows

$$\frac{\partial F[n,m]}{\partial n} \approx \begin{bmatrix}-1&-1&-1\\0&0&0\\1&1&1\end{bmatrix}\cdot\begin{bmatrix}F[n-1,m-1]&F[n-1,m]&F[n-1,m+1]\\F[n,m-1]&F[n,m]&F[n,m+1]\\F[n+1,m-1]&F[n+1,m]&F[n+1,m+1]\end{bmatrix}. \tag{1.27}$$

The matrix dot product is taken. This kernel is often referred to as the gradient kernel in the y (vertical) direction, G_y. Since the boundary pixels do not have neighbours we can only do this operation with a kernel centred on the 2nd to $(N-1)$'th pixel or equivalently with a kernel with its upper left corner on the 1st to $(N-2)$'th pixel. At most a resulting image of $(N-2)$x$(N-2)$ can be computed. Given a kernel G we can compute the resulting $(N-2)$x$(N-2)$ image, I, by convolving the kernel with the matrix representation of the image F. The convolution is mathematically $I = G \circledast F$, where the (n,m) element is given by

$$\begin{aligned}I[n,m] &= G\cdot\begin{bmatrix}F[n,m]&F[n,m+1]&F[n,m+2]\\F[n+1,m]&F[n+1,m+1]&F[n+1,m+2]\\F[n+2,m]&F[n+2,m+1]&F[n+2,m+2]\end{bmatrix}. \\ &= \sum_{i=0}^{2}\sum_{j=0}^{2} G[n+i,m+j]*F[n+i,m+j]\end{aligned} \tag{1.28}$$

Convolution is also a widely used mathematical tool. Here it can be interpreted as the amount of overlap between F and the kernel. More specifically, the value in the (n,m) pixel of I is a measure for the overlap between the kernel and the 3x3 square of F with its top left corner on (n,m). In the case of G_y the kernel elements sum to zero, this causes a constant value to return zero. For other kernels the elements can sum to 1, such that a constant value returns this same constant value.

To realise a different high-pass filter it is possible to negate low-frequencies in the Fourier domain. Although this can not directly be translated to a specific kernel, there exist quite a lot of different high pass filters. A (non-directional) high pass kernel and the Sobel kernel for the y direction are respectively given by:

$$\frac{1}{9}\begin{bmatrix}-1&-1&-1\\-1&8&-1\\-1&-1&-1\end{bmatrix} \text{ and } \frac{1}{6}\begin{bmatrix}-1&-2&-1\\0&0&0\\1&2&1\end{bmatrix} \tag{1.29}$$

Convolution layers have been used in Neural Networks for some decades now, it was even discussed in the paper of Humelhart in 1986 [Kurenkov (2018)]. In such layer some kernels are specified, these can be even larger then 3x3. The values of the these kernels are the learnable weights of this layer. For every kernel the convolution of the input is calculated, resulting in different "images" (for every kernel one). These different images are often called the channels of the layer.

If a convolution layer is compared with a linear layer, it is observed that these are very similar. They both calculate the weighted sum of some input values. However, where a linear layer has to learn the weights for every input, a convolution layer only has to learn the weights of the kernel, which can discover a specific feature independent of the location of this feature in the input image. As Kurenkov points out, [Kurenkov (2018)], this means that it is not needed to have "4 different neurons learn to detect 45 degree lines in each of the 4 corners of the input image, a single neuron could learn to detect 45 degree lines on subsets of the image and do that everywhere within it." This benefits the learning rate greatly, since less weights need to be learned and these weight have more impact since they do not rely on position.

Often the images are subsampled/pooled in between successive convolution layers. In this way the amount of pixels is decreased and faster calculations can be made. The input of MaxPooling layers is divided up in AxA squares, A being the pooling size. Of every square the Maximum value is taken as output, for Average

Pooling the average is taken. This takes an input image from NxN down to N/A)x(N/A). Often zero padding around the rim is added, if this does not result in an integer value. This operation is very similar to convolution, but for pooling the distance between successive kernels, the stride, is A such that there is no overlap between successive kernels. If a pooling layer is added in between successive convolution layers, a kernel in the last convolution layer encloses more pixels of the original image. In this way larger features can be detected. Pooling layers have no trainable weights.

1.2.5. Back Propagation

In order to optimise the output for the given input the weights need to be optimised for the problem at hand. This is often done by training on a training data set, where multiple inputs and the corresponding correct outputs are known. This is known as supervised training, since the correct outputs are given in advance. If data from this data set is passed through the network we can compute the difference between the output of the network and the desired output. This error is often expressed as a single value, the loss function L, and calculated with a mean squared error MSE. The squared sum "punishes" high outliers more than a mean absolute error MAE. The sum is taken over all the outputs of the network and all data points in the training set. For the j'th data point and i'th output take the desired output as d_{ji}, the network output as y_{ji}. The MSE loss function of the whole data set is defined as follows:

$$L = \frac{1}{PI} \sum_{j=1}^{P} \sum_{i=1}^{I} (d_{ji} - y_{ji})^2 \tag{1.30}$$

Where P and I are respectively the number of data points in the data set and the number of output values of the network. Often however the weights will be updated for every data point or a set of data points, a batch, since computing the whole MSE loss for the whole set can take quite some time [Picton (1994)] [Graupe (1997)]. Therefore we will omit the sum over the data points. The objective is now minimising this loss function so the error is as small as possible.

Lets start with the simple case of a single unit. We consider a possible bias as an input weight combo, with input equal to 1. The output y is given by $y = z = \sum w_i x_i$. For the loss function this results in

$$L = (d - \sum w_i x_i)^2 \tag{1.31}$$

This function is quadratic in all weights and therefore it has one minimum. This minimum can be found by equating the derivative with respect to every weight to zero. This set of equations can be solved for this case, as is done in [Graupe (1997)] and [Picton (1994)]. This solution is however not dependent on the output of the network and is hard to generalise for larger networks. As an alternative gradient descent is often used. Here the derivative, gradient in multiple dimensions, of the loss function is "followed down" towards the minimum, such that the next iteration always has a lower loss function. Gradient descent is graphically explained in Figure 1.9. Concrete, after each iteration the weight w_i is adjusted by adding Δw_i this adjustment is proportional to the derivative with respect to that weight:

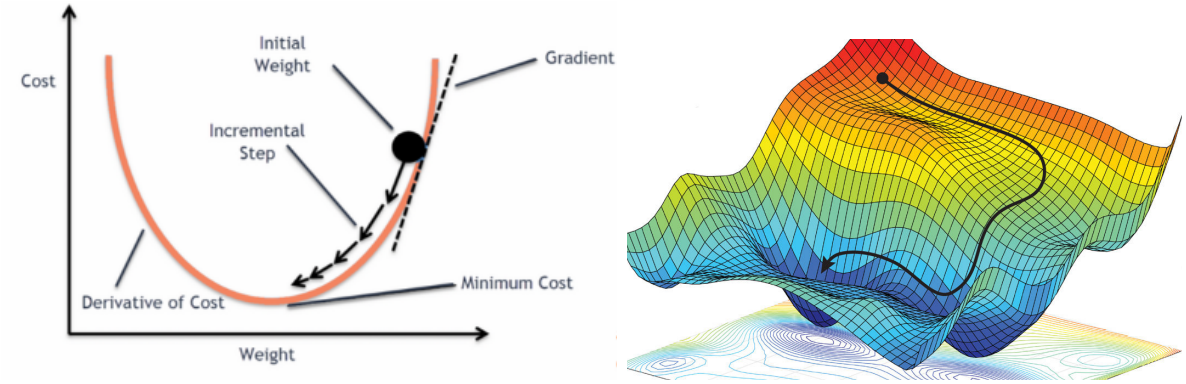
$$\Delta w_i = -\mu \frac{\partial L}{\partial w_i} \tag{1.32}$$

With μ a constant learning rate. A minus indicates that the weights needs to decrease if the slope is positive. Lets calculate this derivative for a single unit using the chain rule:

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial y} * \frac{\partial y}{\partial w_i} = -2(d - \sum w_i x_i) * x_i = 2 * e * x_i \tag{1.33}$$

Where the error e is defined as $(d - \sum w_i x_i)$. This is known as the delta rule. For a positive error, a.k.a. an underestimation of the output, the weights need to increase (positive x). During training of the network, every weight is updated according to a data point, by looping multiple times over the entire training set, the minimum will be approached.

Lets now consider an MLP network, with multiple outputs. Since there are multiple layers we index the variables different. For the k'th perceptron in the r'th layer, let the output be $y_k(r)$, the j'th input be $x_{kj}(r)$ with the corresponding weight $w_{kj}(r)$. Note that for the perceptrons in one layer the inputs are the same: $x_{aj}(r) = x_{bj}(r)$, we thus omit the k. Also $y_k(r) = x_j(r+1$') for $k = j$. Remember that the output $y_k(r)$ is equal



(a) (b)

Figure 1.9: The principle of gradient descent, the slope is followed down into the minimum. For 1 adjustable weight it is shown on the left, for 2 weights on the right. (Source: https://blog.clairvoyantsoft.com/the-ascent-of-gradient-descent-23356390836f and https://reconsider.news/2018/05/09/ai-researchers-allege-machine-learning-alchemy/).

to the activation function F of the weighted sum $z_k(r) = \sum_j w_{kj}(r)x_{kj}(r)$, that is: $y_k(r) = F(z_k(r))$. Let the derivative of F be f.

Consider the output layer, $r = R$, with the desired output values d_k. The weight adjustment $\Delta w_{kj}(R)$ is defined, equivalently to the single unit case, as

$$\Delta w_{kj}(R) = -\mu \frac{\partial L}{\partial w_{kj}(R)} = -\mu \frac{\partial}{\partial w_{kj}(R)} \frac{1}{K_R} \sum_{i=1}^{K_R} (d_i - y_i(R)^2. \tag{1.34}$$

The only term in this sum that depends on $w_{kj}(R)$ is the term with $i = k$. If the derivative with respect to this weight is taken the other terms are zero. We can thus write

$$\begin{aligned} \frac{\partial L}{\partial w_{kj}(R)} &= \frac{1}{K_R} 2(d_k - y_k(R)) * \frac{\partial y_k(R)}{\partial w_{kj}(R)} \\ &= \frac{1}{K_R} 2(d_k - y_k) * \frac{\partial y_k(R)}{\partial z_k(R)} * \frac{\partial z_k(R)}{\partial w_{kj}(R)} \\ &= \frac{1}{K_R} 2(d_k - y_k) * f * x_j(R) \end{aligned} \tag{1.35}$$

This can be calculated if a forward pass through the network has already been completed, since $x_j(R)$ and y_k are already computed.

Lets now consider the r'th layer in the network. Just as before the weights need to be optimised by gradient descent. Therefore the derivative with respect to the weight $w_{kj}(r)$ is computed as follows

$$\begin{aligned} \frac{\partial L}{\partial w_{kj}(r)} &= \frac{\partial L}{\partial y_k(r)} * \frac{\partial y_k(r)}{\partial w_{kj}(r)} \\ &= \frac{\partial L}{\partial y_k(r)} * \frac{\partial y_k(r)}{\partial z_k(r)} * \frac{\partial z_k(r)}{\partial w_{kj}(r)} \\ &= \frac{\partial L}{\partial y_k(r)} * f * x_j(r). \end{aligned} \tag{1.36}$$

Equivalently as before, but the first derivative can not be computed directly if r is not equal to R. It can, however, be noted that the error is only influenced by the weight $w_{kj}(r)$ via neurons upstream. Lets thus use

the chain rule on the next layer $r+1$

$$\begin{aligned}\frac{\partial L}{\partial y_k(r)} &= \sum_i \frac{\partial L}{\partial z_i(r+1)} * \frac{\partial z_i(r+1)}{\partial y_k(r)} \\ &= \sum_i \frac{\partial L}{\partial z_i(r+1)} * \frac{\partial z_i(r+1)}{\partial x_k(r+1)} \\ &= \sum_i \frac{\partial L}{\partial z_i(r+1)} * w_{ik}(r+1) \\ &= \sum_i \frac{\partial L}{\partial y_i(r+1)} * f * w_{ik}(r+1)\end{aligned} \tag{1.37}$$

It is now clear that the derivative of the loss function with respect to the weight $w_{kj}(r)$ can be expressed by the derivatives of the loss function with respect to the outputs of the next layer. To update all the weights in the network, the weights of the output layer should be calculated first. In the process $\frac{\partial L}{\partial y_i(R)} = \frac{1}{K_R} 2(d_i - y_i(R))$ is calculated for every perceptron in the output layer. With this the derivatives with respect to the weights in the $(R-1)$'th layer can be computed. Using this computation the weight in the $(R-2)$'th layer can be computed. The algorithm continues like this until the input layer is reached. [Picton (1994)] [Graupe (1997)]

Note that Equation 1.35 and 1.36 depend on the data via x_j, the weight adjustment will thus differ for every data point. In these equations the importance of the differentiable activation function can also be seen.

To adjust the weights the derivatives are multiplied by the learning rate. If this learning rate is high the steps for the weights will also be large and the network may converge faster to a solution. If the learning rate is too large the network may not converge. Too low and the loss function may get stuck in a local minimum. To counter this a variable learning rate is often used. Recently a lot of optimiser, which specify the variable learning rate, are developed. The last years the optimiser Adam and RMSprop have gained huge traction. For more information [Kingma and Ba (2014)] and [Zhang (2018)] can be read.

1.3. Representing the lens surface by a B-spline

The output of the neural network should describe the shape of a lens that will give the light distribution. The shape of the lens will be numerically described by the thickness in a point (or numerical bin). The neural network could output the thickness at every bin. This is comparable with the finite element method and PINN used in [Raissi et al. (2019)]. Specifying the thickness at every bin is pretty inefficient though if the lens is spread over a large amount of bins. Firstly the neural network will have to have a lot of outputs, which makes the learning hard. Secondly, varying the thickness at a point will not have a big influence on the shape of the lens. This variation will thus not change the resulting light distribution significantly. In addition the surface of a lens should also be smooth. The objective thus is to find a representation of the lens for which there can be specified a small amount of characteristic points. Similar to the hp-variational PINN used by [Kharazmi et al. (2020)], which uses projections on higher-order polynomials.

In optics polynomials are often used to described lens shapes, for example Zernike Polynomials. These polynomials can describe common light wave aberrations. See for example [Kaya et al. (2012)]. In this thesis, however, Basis Splines (B-Splines) will be utilised. In general splines can be used to interpolated between points and to form a smooth curve through these points. In this way a complicated smooth curve can be computed by only specifying a small number of control points. In general splines are defined piecewise by polynomials. If only polynomials are used to interpolate, the order should be equal to the amount of points. This can cause large oscillations for a decent amount of points. By defining a curve piecewise this can be avoided.

Suppose a set of 2 dimensional control points, $\vec{c}_1 ... \vec{c}_n$, is given. Following [Lyche and Mørken (2008)], these points can be simply interpolated linearly, with a straight line between the points. Such a linear curve through these points is often parametrized by a single variable t. This variable will be split up in intervals by $t_1 < t_2 < ... < t_n$, with the intervals given by $[t_0, t_1], [t_1, t_2]$.... These points are called knots and define the knot vector $T = \{t_1, t_2, ..., t_n\}$. The linear segment between $\vec{c}_i$ and $\vec{c}_{i+1}$ is given by

$$P_{1,i}(t|\vec{c}_i, \vec{c}_{i+1}, t_i, t_{i+1}) = \frac{t - t_i}{t_{i+1} - t_i}\vec{c}_i + \frac{t_{i+1} - t}{t_{i+1} - t_i}\vec{c}_{i+1} \text{ for } t \in [t_i, t_{i+1}] \tag{1.38}$$

The subscripted 1 indicates the first order, linear, approximation. This can interpreted as a weighted average sum between the two points $\vec{c}_i$ and $\vec{c}_{i+1}$, where the weights depend on how close t is to t_i and t_{i+1}. The entire

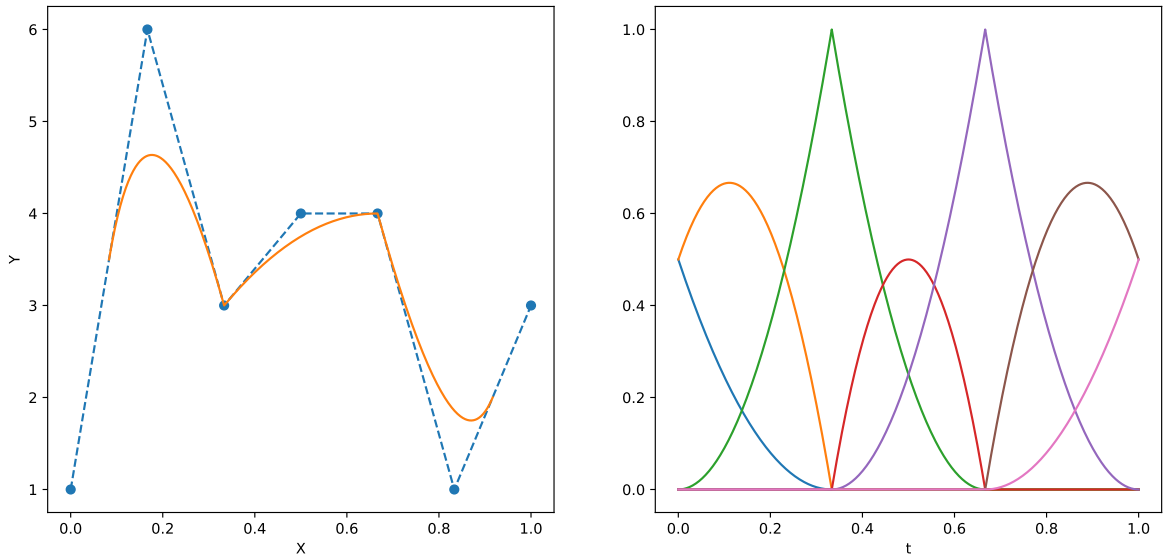


Figure 1.10: Left: The solid orange line represents a quadratic spline function, the dotted blue line represent the control polygon/linear spline. Right: the spline basis functions for the knot vector $\frac{1}{3}\{-2,-1,0,1,1,2,2,3,4,5\}$ parametrised by $t \in [0,1]$. Since some knots have multiplicity 2, the spline goes through the third and fifth control points, the derivative at these points is however not continuous.

linear curve $f(t)$ can be written as a sum of these $P_{1,i}(t)$ if specified zero outside of the interval $[t_i, t_{i+1}]$, mathematically:

$$\begin{aligned} f(t) &= \sum_{i=1}^{n-1} P_{1,i}(t) * B_{0,i}(t) \\ B_{0,i}(t) &= \begin{cases} 1 & \text{for} \quad t \in [t_i, t_{i+1}] \\ 0 & \text{else} \end{cases} \end{aligned} \tag{1.39}$$

By defining a new knot vector $K = \{t_0, t_1, t_2, ..., t_n, t_{n+1}\}$, with $t_0 = t_1$ and $t_{n+1} = t_n$, this can be written as

$$\begin{aligned} f(t) &= \sum_{i=1}^{n} \vec{c}_i * B_{1,i}(t) \\ B_{1,i}(t) &= \frac{t_i - t}{t_i - t_{i-1}} B_{0,i-1}(t) + \frac{t - t_i}{t_{i+1} - t_i} B_{0,i}(t) \end{aligned} \tag{1.40}$$

For this setup to work "$\frac{0}{0}$" is defined as zero.

$B_{1,i}$ is called the first order basis function. The first order spline is often called the control polygon of higher order splines. A higher order, k, spline can also be written as a linear combination of the k'th order basis function, exactly like Equation 1.40. These form a basis representation for splines. Splines represented in this way are called Basis-Splines (B-Splines). The derivation for higher order can be read in [Lyche and Mørken (2008)].

For a given knot vector, $T = \{t_1, t_2, ..., t_{n+k+1}\}$ the k'th order basis function is given by the Cox de Boor recursion formula

$$B_{d,i}(t) = \frac{t - t_i}{t_{i+d} - t_i} B_{k-1,i}(t) + \frac{t_{i+k+1} - t}{t_{i+d+1} - t_{i+1}} B_{k-1,i+1}(t). \tag{1.41}$$

$B_{0,i}(t)$ is specified in equation 1.39. [J Austin Cottrell (2009)]

For a knot vector of length $n + k + 1$ there will be n basis functions of order k, each corresponding to a single control point. Since there are more knots then control points the knots do not correspond one on one to a control point. By specifying a knot multiple times a wide variety of basis functions can be derived, see for an example Figure 1.10. By increasing the multiplicity of a knot the number of continuous derivatives is decreased.

Now suppose data points are given of a sampled function $F(x)$, with regularly spaced x-coordinates. It is possible to approximate the data points by a spline, as in Equation 1.39. The spline is however parametrised by parameter t, so if we want to know the y-coordinate of the spline given the x-coordinate, the corresponding t-coordinate should be known. This is often difficult since x is parametrised by t, not vice versa. Different spline function approximations, $f(x)$, can be derived though, these are a linear combination of the basis functions as a function of x: $B_{k,i}(x)$, like

$$f(x) = \sum_{i=1}^{n} c_i B_{k,i}(x). \tag{1.42}$$

The control coefficients c_j are scalars, instead of vectors. The problem is finding these coefficients and the right knot vector, that divides the x-coordinate up in intervals.

If it is required that the spline goes exactly trough the data points, an interpolation spline can be used. The control coefficients will however not be equal to the y-coordinate of the data points. In order to find the control coefficients a system of equations should be solved. See for example [Lyche and Mørken (2008)]

A different and more easy approach, the variation diminishing spline, can also be used. The spline will approximate the linear interpolation function, $f_l(x)$, of the data points. In contrast to the interpolation spline, the variation diminsishing spline will in general not go through the data points. The k order variation diminishing spline approximation, $f_V(x)$, of $f_l(x)$ given a knot vector $T = \{x_1, x_2, ..., x_{n+k+1}\}$ is given by

$$f_V(x) = \sum_{i=1}^{n} f_l(x_i^*) B_{k,i}(x). \tag{1.43}$$

Where the knot averages have been used: $x_i^* = \frac{1}{k} \sum_{j=i+1}^{i+k} x_j$. The control points, defining the control polygon of a spline function, have coordinates (c_i, t_i^*), for $i = 1, 2, ..., n$. [Lyche and Mørken (2008)] By choosing a smart knot vector, the knot averages can coincide with the x-coordinate of the data points. Such that the y-coordinates and data points coincide respectively with the control coefficients and the control points of the spline.

2 Experimental Procedure

In this chapter the setup of the experiments is explained. The goal of the experiments is solving the backwards freeform lens problem, by using neural networks. The Neural Network should be able to output the surface of a freeform lens that gives the input light distribution, from now on called the image. First, the general setup is explained. Afterwards, every part with the important parameters is explored in depth.

2.1. General setup

In- and Output

To create a neural network the in- and output should be defined. The input should represent the image, the output should represent a freeform lens. The surface will be described by a variation diminishing spline, the height of its control points are outputted by the neural network. Not a lot of control points are needed for a complicated spline, therefore not a lot of output variables are needed. The image will be represented by an array of light intensities in certain bins.

Training set

For the network to give a right lens corresponding to the inputted image, the network needs to be trained on a training set first. This training-set should be large and consist of all possible lenses and image varieties. For training, a loss-function should also be defined. This gives a numerical value for the similarity between the inputted image and the image as a result of the predicted lens.

Supervised training

If the original lens is known this loss function can also represent the similarity between the original lens and predicted lens. This way the problem is very similar to (the famous problem of) classifying handwritten digits. Here the solution for the training data, the actual digit (the lens), is already known in advance. The network can simply train on the data for which the solution is already known, this is often called supervised training. For the proposed generation of the training-data a lens will be known in advance. Supervised training is thus possible. This can easily be done with the setup described here and has also been done. The results are not presented in this thesis, but this has given very good results that motivated the alternative learning method in this thesis.

Generation of the training set

Ideally, we would like to train the network on a large amount of images for which the lenses, that will produce these images, are unknown. The optical setup of this thesis can not produce a wide variety of images, as will be seen in one of the first results presented in chapter 3. If the inputted image is not contained in the possible "image-space", even a perfect network can not give a lens that generates this image. Therefore the training-data needs to consist of images that can be produced by the optical system. Such a data set can be easily generated by creating random control point heights. With this the lens surface (spline) and the image (via the Fraunhofer approximation) can be computed. Again in this way, a lens corresponding to a generated image is known but will not be used to train the model.

Training method from PINN's

The used training method will be very similar to the training of Physics Informed Neural Networks (PINN's). These are introduced by [Raissi et al. (2019)], among others. In such networks physical knowledge is used to compute the loss function. Mostly PINN's solve (partial) differential equations (PDE). Where the input would be boundary conditions, the output will be the solution of the PDE. The loss will be computed with the PDE to identify how far the solution is from a physical possible solution that complies with the PDE. Having said that, in this thesis the PDE in the "PINN" will be swapped by the knowledge of the solution of the forward freeform solution, e.g. the Fraunhofer approximation. From the output of the network (the lens surface) the resulting image will be calculated with the Fraunhofer approximation. A loss function representing the differences between the predicted and desired image can be computed. The whole training method of the network is represented with a flow chart in Figure 2.1.

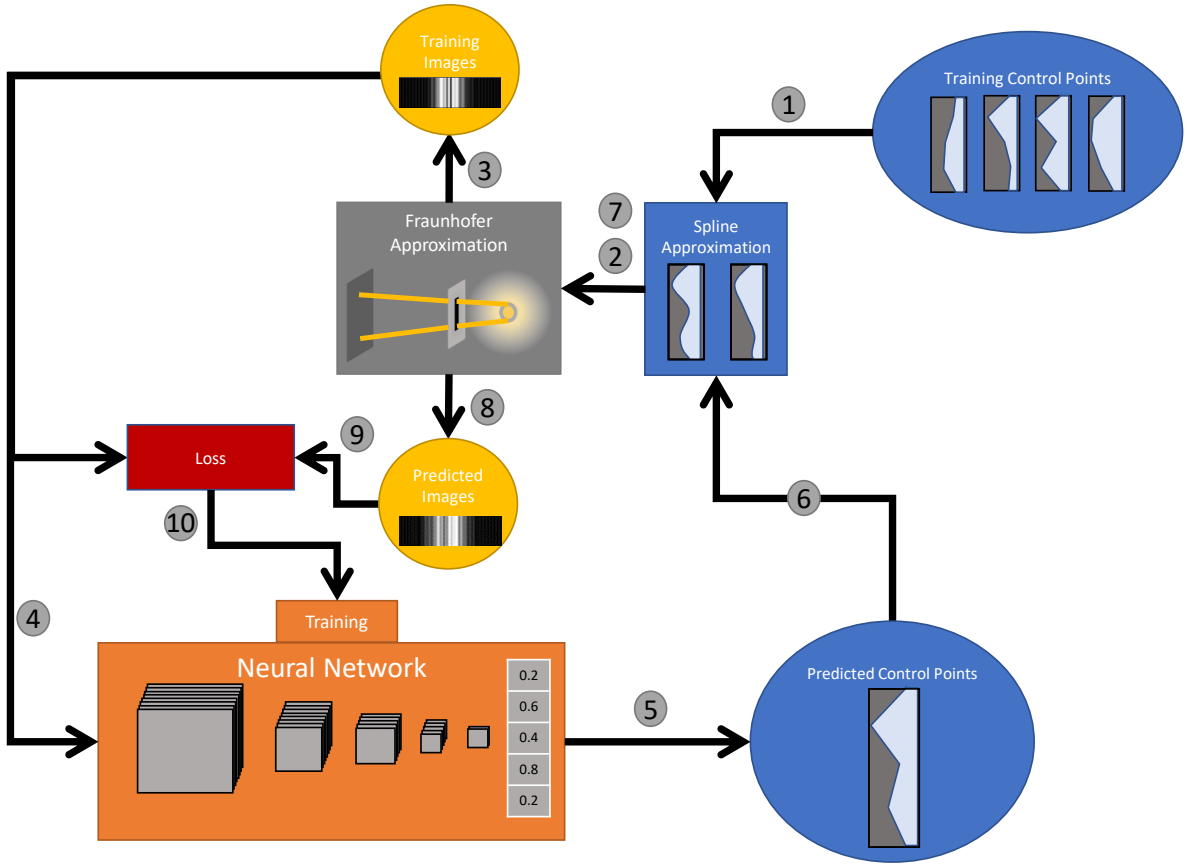


Figure 2.1: The setup for training the neural network. Before training random control points are generated (upper right). The resulting training images are calculated (1-3). The network makes a prediction of the control points given the training images (4-5). The predicted images are calculated with these control points (6-8). This image is compared to the training image via the loss function(9). The weights of the network are trained via back-propagation(10).

Code implementation
For all calculations Python (version 3.7.7) has been used. For neural networks two packages are commonly used: PyTorch and Tensorflow. With Tensorflow neural networks are faster to get running, however PyTorch will allow to set a loss function manually, therefore PyTorch (version 1.5.0) will be used for this thesis. PyTorch will keep a record of every calculation that has been done. When the gradient needs to be calculated it can then back trace it steps and calculate the appropriate gradients for back-propagation. It however needs to know the gradient of every single computation, these are only known for the computations of the PyTorch package itself. Although it represents NumPy very closely, nothing that is used for back-propagation can be coded with NumPy. This is of course also true for other packages like SciPy. The Fraunhofer and spline approximations have thus been coded manually.

2.2. Fraunhofer Approximation

An one dimensional lens surface, Fraunhofer approximation and image has been used, because the forward problem can in this way be calculated very fast and functions as a proof of concept of the proposed solution for the inverse lens problem. The 2 dimensional case is not explored but can be explored in future research. A plane wave parallel to the pupil plane, will enter the aperture.

Light propagation is in general a 3 dimensional phenomenon. With Equation 1.20 the field on a 2D plane (x, y) a distance Δz away can be calculated. We define the 1D Fraunhofer approximation as simply taking the Fourier transform of a 1 dimensional pupil-line, giving a 1D strip of the 2D light distribution. Mathematically y' and y are set to zero in Equation 1.20. Physically this can be interpreted as Fraunhofer propagating the transmittance function of a very thin slit, resulting in a 2 dimensional distribution, of which we take a very thin slice perpendicular to the slit. In the slit a lens can be placed that varies in thickness only in the direction of the slit length. The pupil-line will from now on also be called the pupil-plane if no ambiguity exists.

The intensity of the light a distance Δz away will thus be calculated with the Fraunhofer approximation. The intensity will be proportional to the squared modulus of the 1D Fourier transform of the pupil-plane. The proportionality constants are not of importance and will not be calculated for a faster computation.

The total 1D pupil-plane will consist of N (=1024) bins. Inside the pupil-plane an aperture with radius R centred in the middle is placed. The amplitude of the complex field in the aperture consists of an arbitrary constant intensity (taken to be 1), outside the aperture the amplitude will be 0. This corresponds to the plane wave entering the aperture. The phase variations will be due to the transmittance function of the lens inside the aperture. The transmittance function is given via Equation 1.21.

This setup assumes a monochromatic source very far away, such that a plane wave enters the aperture perpendicularly. Of the total pupil function the 1D discrete Fourier transform (DFT) is taken. For the Fraunhofer domain the aperture needs to be on the order of the product of the wavelength and Δz. These can be specified and will alter the length of the x-axis of the image-plane but do not alter the shape of the resulting image and can thus be chosen arbitrarily.

The resulting images will have a lot of oscillations, to average out these oscillations an apodization window function can be used. This is a function that is multiplied by the Fourier transform of an image. Here the Hann apodization filter will be used. This will filter out higher frequencies. Since the image is the Fourier transform of the pupil plane, we can multiply the pupil plane by such a window function. This is possible because of the duality property of the Fourier transform.

2.3. Spline Approximation

The lens inside the slit is described by a spline. This spline will be a variation diminishing spline of the linear polygon of the data points (output of the network), see Equation 1.43. These data points will be uniformly spaced inside the slit, the positions indicated by the x-coordinate for the interval [0,1]. The y-coordinate of the generated data points will be in between zero and one.

Say M data points are given. The x-coordinates will be $\{X_0, X_1, ..., X_{M-2}, X_{M-2}\} = \frac{1}{M-1}\{0, 1, ..., M-2, M-1\}$. To create a spline a knot-vector has to be specified first. The knot averages computed with this vector will be the x-coordinates that sample the linear approximation (Equation 1.43). It would be perfect if these x-coordinates correspond with the x-coordinates of the data points, such that they become the control points of the spline. In this case it will not even be needed to calculate the linear interpolation of the data points, every time a spline is calculated.

For this the knot-vector $T = \frac{1}{M-1}\{0,0,0,1,2,3,...,M-4,M-3,M-2,M-1,M-1,M-1\}$ is proposed. By the definition of the knot-averages the first knot average will be the average of $\frac{1}{M-1}\{0,0,1\}$, which is $\frac{1}{3(M-1)}$,

the second knot average is $\frac{1}{M-1}$, the third $\frac{2}{M-1}$ etc. In general the inner knot averages correspond to the x-coordinates of the data points, the first and the last do not however. This means that for these points the first and last section of the linear approximation still need to be calculated. Having said that, this can be avoided if these sections are zero everywhere. That is, if the first 2 and last 2 data points have a zero y-coordinate.

Therefore the proposed knot-vector is used with data points for which the inner $M-4$ points are given as output of the network and the outer 2 points on both side are always zero:

$$\{Y_1, Y_2, ..., Y_{M-1}, Y_M\} = \{0, 0, y_1, y_2 ... y_{M-5}, y_{M-4}, 0, 0\},$$

y_i is the i'th output variable of the network. This approach has two advantages: first, computations are tuned to a minimum per spline approximation and, secondly, the spline will be clamped to zero on the boundaries. Because of this clamping, the general shape of the lens will be convex.

If the length of the slit is known, the knot-vector and thus also the basis functions can be computed before the training of the network. These indeed do not rely on the control points. This is highly efficient since only the sum of Equation 1.42 has to be computed every iteration.

The resulting spline function will be multiplied by a maximum thickness $T = R * t$, with t a characteristic thickness ratio and R the radius of the spline. The lens thickness scales with the aperture radius, because larger lenses can, intuitively, be thicker and still satisfy the thin lens approximation used in Section 1.1.5. t can thus be interpreted as a characteristic ratio between the lens width and the maximum thickness. t will range from 1/800 to 1/20. In the end the lens thickness is the product of the spline function (between 0 and 1), the characteristic thickness ratio and the aperture radius R.

2.4. Neural Network

The neural network functions as the brain of the experiment. As explained in section 2.2 the input will be the N bins representing the light intensity, its output will be the inner control points of the spline. The network needs to be trained before it can recognise the intensity pattern. After training it can hopefully give a correct lens.

It will be trained on a data set consisting of samples, randomly generated. It will often be trained on this set multiple times, every iteration trough the data set is called an epoch.

Every epoch the data set will be divided up randomly in batches. For every batch the loss-function is calculated as the mean squared error of the predicted image and training image, Equation 1.30. Here P will be the batch-size. Although back-propagation is derived for a single sample, by including the sum over the batch it can also be derived for batches. Training with batches also decreases the training time, since for all the linear sums matrix multiplication can be used [Hoffer et al. (2017)].

The loss function can vary widely for different t and R, therefore an average loss for random samples is calculated. This loss is calculated by comparing every sample from the data set with a random other sample from the data set. We call it the initial loss. All losses calculated will be divided by the initial loss. For example, the network will thus give a loss of approximately 1 without training.

The network will consist of a maximum of 3 convolution layers and a minimum of one fully-connected linear layer, maximum of 3 linear layers. After every convolution layer a maxpooling layer is placed with kernel size 2. The layers will have up to 64 channels. Since adding more layers and/or channels will increase the training time significantly and a larger network will not necessarily give a better prediction, a larger network will not be investigated. After every convolution layer a ReLU (Rectify Linear Unit) activation function is used, for non-linearity. For all but the last linear layers also a ReLU is used. For the last layer the sigmoid activation function is used, in this way the output (y-coordinate of the control points) will be normalised between 0 and 1.

Lastly for training back-propagation is used. By keeping track of the calculations during the Fraunhofer approximation the error can be back-propagated with the derivative of these computations. To update the weights 2 optimisers are used. The network starts training with the RProp optimiser, this is a relatively fast optimiser but will have difficulties in reducing the loss value for maximum prescission. Therefore if a certain threshold is reached the Adam optimiser will be used. In contrast to the RProp optimiser, this optimiser will at the start not converge very quickly. It can however reach low loss values. [Zhang (2018)] Therefore RProp will be called the coarse optimiser and Adam the fine optimiser. Both optimisers will have a different learning rate (lr). Often the loss threshold will be reached at the end of the first epoch.

3 Results

3.1. Physics Simulations

In this first section a small amount of graphical results, concerning the forward Fraunhofer approximation in combination with splines, is presented.

Zernike Polynomials

The first result is a 2D spline approximation of Zernike polynomials, showing the potential of describing lenses by splines. Consider a 2D pupil plane with a circular aperture, containing a lens described by Zernike polynomials. Zernike polynomials are described in polar coordinates, characterised by 2 integer valued variables, n and m, respectively the radial and azimuthal degree. A Zernike polynomial is sampled by points that are radially and angularly equally spaced. These data points are interpolated by a 2D spline function,by the Scipy packages function bisplev. The actual Zernike polynomials and approximating splines with resulting Fraunhofer light distribution are shown in Figure 3.1.

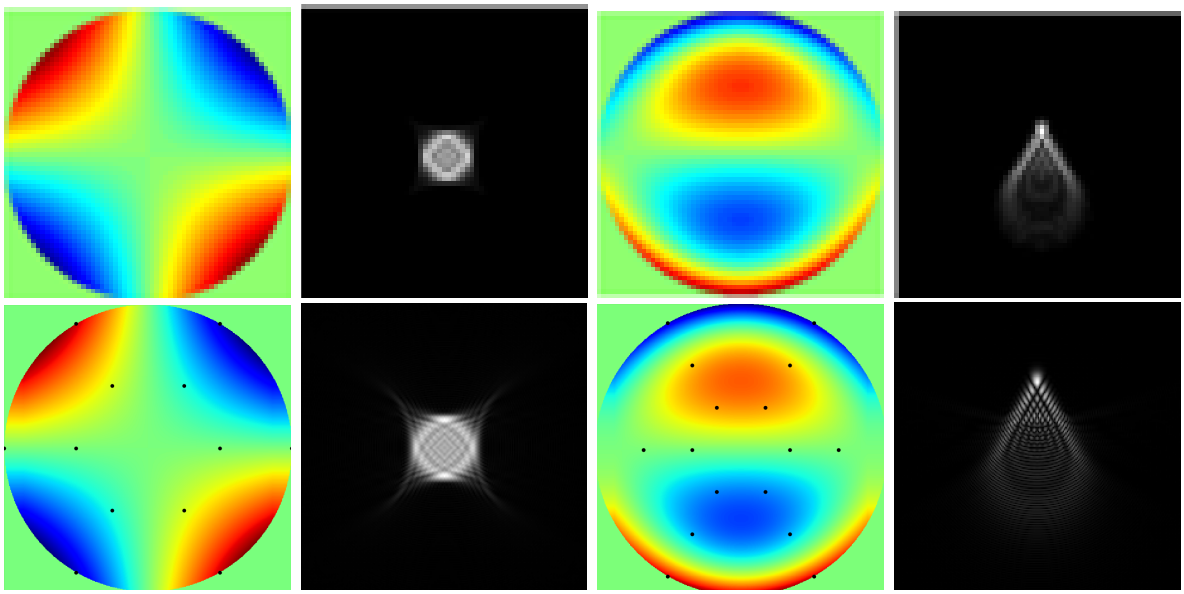


Figure 3.1: Zernike polynomials on a circular domain with the resulting image. Left side shows the Zernike polynomials with $n = 2$ and $m = -2$, called an astigmatism. The right side shows the polynomial with $n = 3$ and $m = -1$, called a coma. Top row: actual Zernike polynomials and the resulting image, source: https://www.intechopen.com/books/topics-in-adaptive-optics/the-need-for-adaptive-optics-in-the-human-eye. Bottom row: corresponding spline function approximations, with resulting Fraunhofer diffraction pattern. The black dots show the location of the sampled data points used for interpolation. For the astigmatism 2 points are sampled radially and 6 angularly, 12 points total, spline order is 2. For the coma 3 points are sampled radially and 6 angularly, 18 points total, spline order is 3.

The splines approximate the polynomials graphically very well. The resulting Fraunhofer diffraction pattern also matches when the functions are added into the aperture as a phase disturbance. It should however be noted that for more sampling in the azimuthal coordinate, the results can be worse. In such a case, the function may not be sampled symmetrically with respect to the symmetry of the Zernike polynomial. As the figure shows, it is possible to find a sampling setup which gives a good approximation with minimal sampling points. Lastly the Scipy function bisplev is and should be used with Euclidean coordinates. A spline function approximation in polar coordinates may give better results overall for a circular aperture.

Since these results do not involve back-propagation the SciPy package has been used. This gives an interpolation spline. From now on the spline approximation will be used in combination with back-propagation.

This means that a spline approximations is manually coded, this is a variation diminishing spline approximation.

Varying Lens Shapes
Next the results of the forward 1D Fraunhofer calculation are given. For a varying aperture radius, R, and the characteristic lens thickness, t, the spline function, the phase function (as a result of the thickness variations) and the resulting light distribution is given in Figure 3.2. Since these figures show a lot of oscillations, similar results are shown in Figure 3.3 but with a Hann windowing/apodization function applied to the pupil function. This filter will average out some oscillations. The Hann apodization filter is exactly as wide as the aperture width. It is multiplied by the pupil amplitude (not the phase, which is influenced by the thickness of the lens).

The splines (plot a) all display the same shape, but scaled in the y and x direction. These are mostly included for illustrative purpose. However it should be realised that small apertures contain a tiny amount of bins, 26 for the smallest aperture. The resulting spline is therefore not sampled with a high detail. This means that adding a lot more control points will make the spline approximation unnecessary for very small apertures.

The phase distribution in the pupil plane (plot b) shows more oscillations the thicker the lens. This is a result of a larger phase difference, since the phase plotted from $-\pi$ to π is discontinuous due to phase wrapping.

The light distribution (lower plots) shows more oscillations for a thicker lens. The phase varies more widely and thus also the complex exponential of the phase. For even thicker lenses no side lobes can be observed in the image and the oscillations will carry on all the way to the sides causing spectral leakage. If the thickness did not scale with the aperture width, the image would have the same shape for different aperture widths but scaled differently in the vertical direction.

For a thick and wide lens, the light intensity in a single bin can vary widely for a small change in the spline function, since the distribution oscillates so quickly. This thus may be an unstable setup. For a very thin lens and aperture, however, the distribution may vary not significantly if the spline is adjusted. For such a setup the influence of the spline can be very low. Ideally we want a lens not neither too thin and too thick.

Plots a and b do not change when the apodization function is applied in Figure 3.3. The light distribution is significantly smoothed out, because of the averaging effect of such a filter. This can clearly be seen for the tiny lenses (1c-5c) in the side lobes. Also the main lobes do not have the roughness of before. For the medium sized lenses the oscillations in the centre of the distribution have also become less. The peaks are in general more defined. For the large lenses the light distribution still oscillates very fast.

3.2. Neural Network Optimisation

Next the results achieved with the neural network will be displayed. Optimal parameters are searched for to minimise the total loss function. Firstly the physical parameters are optimised. Afterwards the structure and parameters of the neural network are optimised. At the end the network using these parameters is explored further.

3.2.1. Physical Parameters

To discover for which lens thickness t and pupil radius R the network works, the network will be trained for a range of these values. Since results can vary for every training session the average over 5 training sessions is given, for a set of parameters. From Figure 3.2 it can be seen that the network will not be able to train for R larger than $L/20$ since the resulting image will oscillate very quickly, where L is the width of the entire pupil plane. A small change in the output of the network may cause a large change in the total loss. However the aperture can not be taken very small, because the lens will not influence the image at all. Exactly the same can be said for the thickness of the lens.

The results for $R \in \{L/80, L/60, L/40, L/20\}$ and $t \in \{1/600, 1/450, 1/300, 150\}$ are displayed in Figure 3.4. Of course a wider parameter space can be explored, but this will take very long. With this setup a network is trained 80 (=4*4*5) times already.

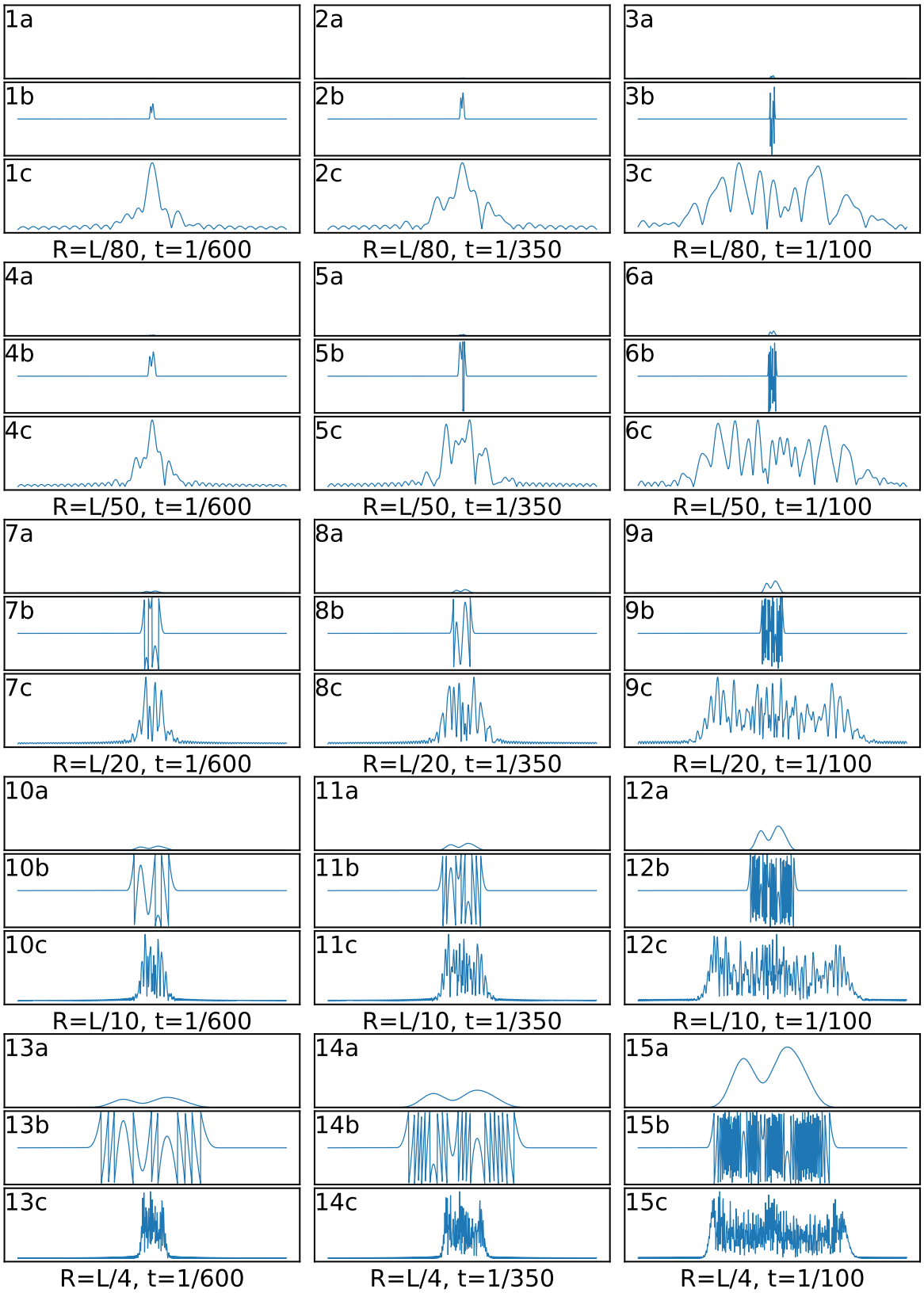


Figure 3.2: For 5 different aperture radius, R, and 3 different characteristic thickness', t, 3 plots are displayed. Plot a displays the lens thickness proportional by a spline with constant control points. The y-axis has the same scale for all the top plots, so the different lenses can be compared. Plot b shows the phase that is added to the field in the pupil plane, y-axis runs from $-\pi$ to π. Plot c shows the resulting light distribution calculated with the Fraunhofer. The y-axis is not constant for these plots, the scale for the upper left plot is around 10 times smaller compared to the lower right plot. This can also be seen by the lower high frequency fringes.

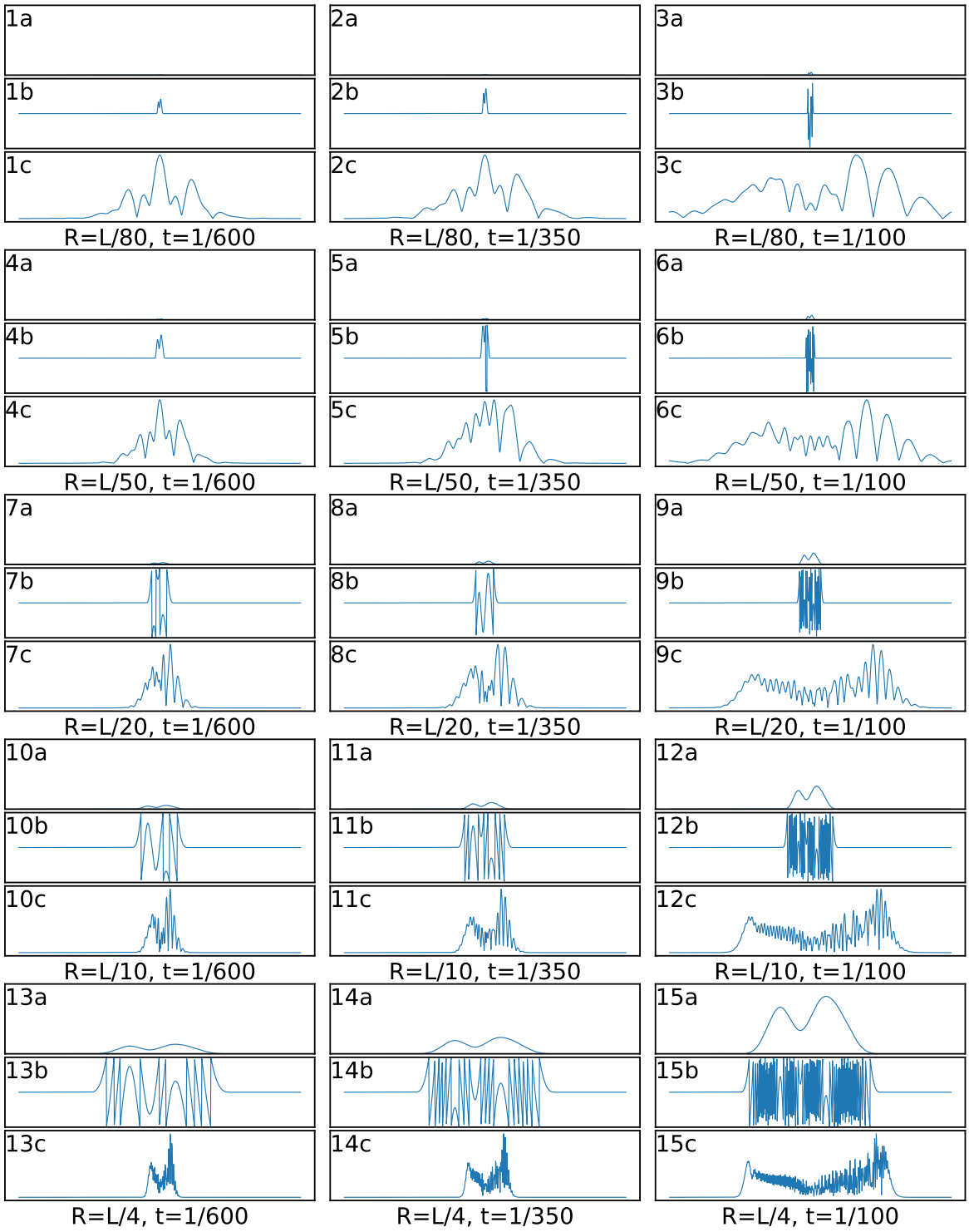


Figure 3.3: For 5 different aperture radius, R, and 3 different characteristic thickness', t, 3 plots are displayed. Plot a displays the lens thickness proportional by a spline with constant control points. The y-axis has the same scale for all the top plots, so the different lenses can be compared. Plot b shows the phase that is added to the field in the pupil plane, y-axis runs from $-\pi$ to π. Plot c shows the resulting light distribution calculated with the Fraunhofer. The y-axis is not constant for these plots, the scale for the upper left plot is around 10 times smaller compared to the lower right plot.Lastly the x-axis is the same for the top, midle and bottom plots, it displays 1024 bins.

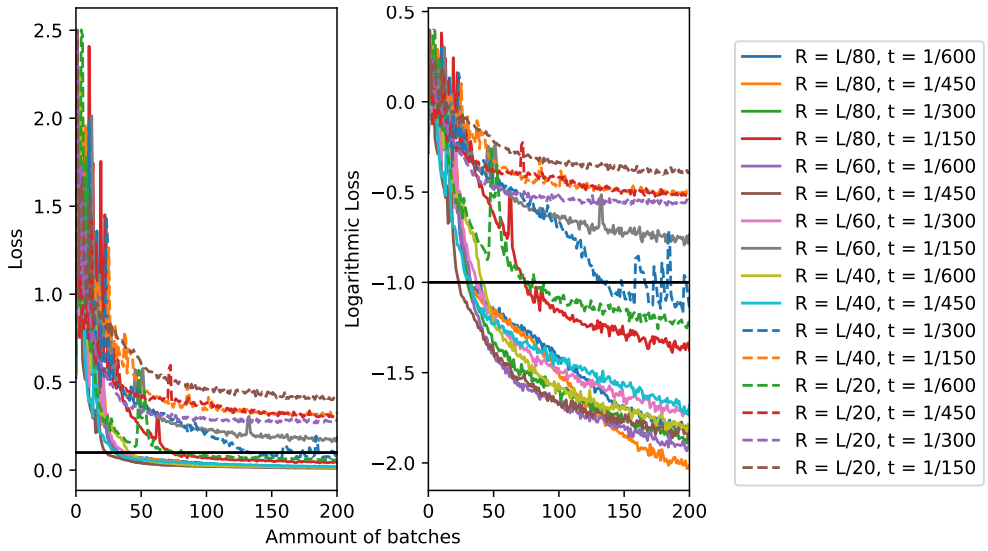


Figure 3.4: The average loss of 5 training sessions per set of parameters per batch as a function of the number of completed batches. R indicates the radius of the pupil and t the relative thickness. The left shows the loss, the right shows the logarithm (base 10) of the loss. The black horizontal line shows the threshold for the RProp and Adam optimiser. 2 convolution layers (32 and 16 channels), 1 hidden linear layer (64 channels), coarse lr=.03, fine lr=.001, optimiser threshold=0.1.

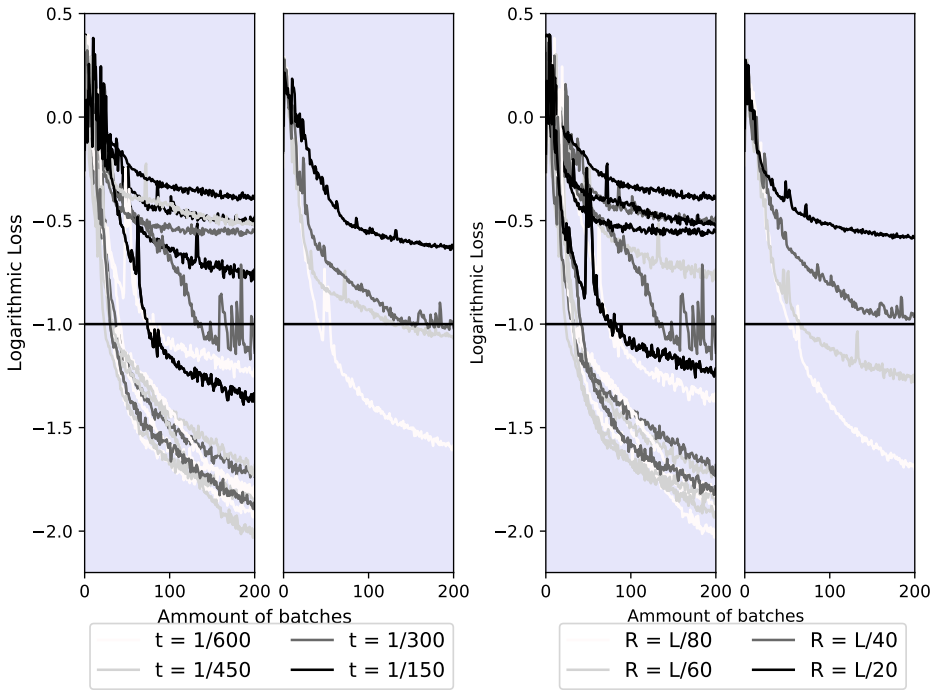


Figure 3.5: The same data as Figure 3.4 but visualised on a grey scale. The 2 left pictures show a darker grey for a thicker lens, the right 2 show a darker grey for a wider aperture. In both cases the right graph of the two shows the average for every single colour.

For completeness the loss is shown on a linear scale, but compared to the logarithmic scale this is not enlightening at all. Therefore the loss on a linear scale will only be displayed in the appendix A from now on. Although single cases and some trends can be identified, these plots do not give us a general picture. Therefore the same plot is given in grey scale in Figure 3.5, where the darkness indicates the value of the parameters.

Clearly all different networks start with a loss of approximately 1. Indicating that it gives a random output. Afterwards the loss goes down pretty fast as the network starts to learn. The lowest loss that is reached is 0.0097, or 0.97 percent of the random loss without training. This loss is achieved by a very small aperture and thin lens. In Figure 3.4 for some sets of parameters the neural network does not reach the optimiser threshold and is not trained with the Adam optimiser.

In Figure 3.5 it can clearly be seen that a thick and wide lens works very poorly, a very thin and small lens can achieve a very low loss. This is as expected because the resulting images for a large lens oscillate very quickly due to the Fraunhofer approximation. Although the average loss for the medium wide apertures is significantly worse, this average is actually raised by the high losses of the thick lenses. In the left plots of both sides some medium sized lenses give a fairly low loss. From Figure 3.2 it is seen that such lenses can also give a somewhat wide variety of images, which are not almost equal to the "simple" Sinc function. Therefore for the next results a medium sized lens is chosen: $t = 1/300$ and $R = L/40$.

3.2.2. Network Parameters

Network Layers

This next section will investigate which combination of linear and convolution layers work the best. Again a fairly small "layer-space" is explored, since large networks can take longer to train and evaluate. During the training the learning rates will be the same for every session. Since a large network has more weights to train the whole network may be altered more then a small network. Therefore the learning rates are taken lower ($lrCoarse = 0.02$ and $lrFine = 0.0005$) and the threshold higher(= 0.15) compared to the last section. Also the loss will not be averaged over 5 training sessions per set of parameters. The loss will be taken as the average of the 2 best of 5 training sessions per set of parameters. This is done, because the coarse learning rate is relatively high for some layer combinations such that some training sessions do not surpass the optimiser threshold. As before the loss of all variations is plotted linearly and logarithmic, this is included in the appendix, Figure A.1. Here only the grey-scale plots are given in Figure 3.6.

It can be seen that a single combination of parameters is in general not very bad. All setups may be able to solve the problem. There is however a difference in convergence speed. The setup that achieves the lowest loss on this time scale has one convolution layer with 32 channels and 2 linear layers with 64 channels in the hidden layer. However looking at the 5 training sessions of this setup (not displayed in the report), 2 of them did not even reach the optimiser threshold before the end but 2 gave a very low loss, which are represented in the figure. This may indicate that taking the 2 best sessions may not be fair.

Furthermore in general, 2 and 3 convolution layers perform better then none or 1. There is not a large difference between the ones with an increasing amount of channels and the decreasing ones. Of the other 4 different best possibilities, the setup, with 2 layers and a small amount of channels, performs second best. Since this one is also the smallest it can be trained the fastest (around 70 seconds) compared to the largest network (130 seconds). Therefore 2 convolution layers will be used from now on. It should be noted that a high amount of convolution layers often increases the complexity too much and the network can not train the huge number of parameters accurately.

Furthermore when looking at the number of linear channels, 2 layers with 64 channels in the hidden layer out performs the rest. That large networks get out performed by smaller ones is often the case for neural networks, since these will have more difficulty to update all the weights perfectly. The network with C=[8,16] and L=[0,64] also performs around the 4'th best, it is insignificantly beaten by two others and of course by the low out-liar discussed before. It takes around 80 seconds to complete the 4 epochs for this network.

Optimiser Parameters

Lastly the optimiser learning rates and threshold is optimised for the aperture and layer combination described above. As before the average loss of the 2 best performing training session is given (out of 5), to stimulate sets of parameters that can achieve very low loss but that can also sometimes get stuck with a very high loss. The results for the threshold set to 0.07 is given in Figure 3.7, the results of the threshold equal to 0.1 and 0.15 is given in the appendix, Figure A.2 and A.3.

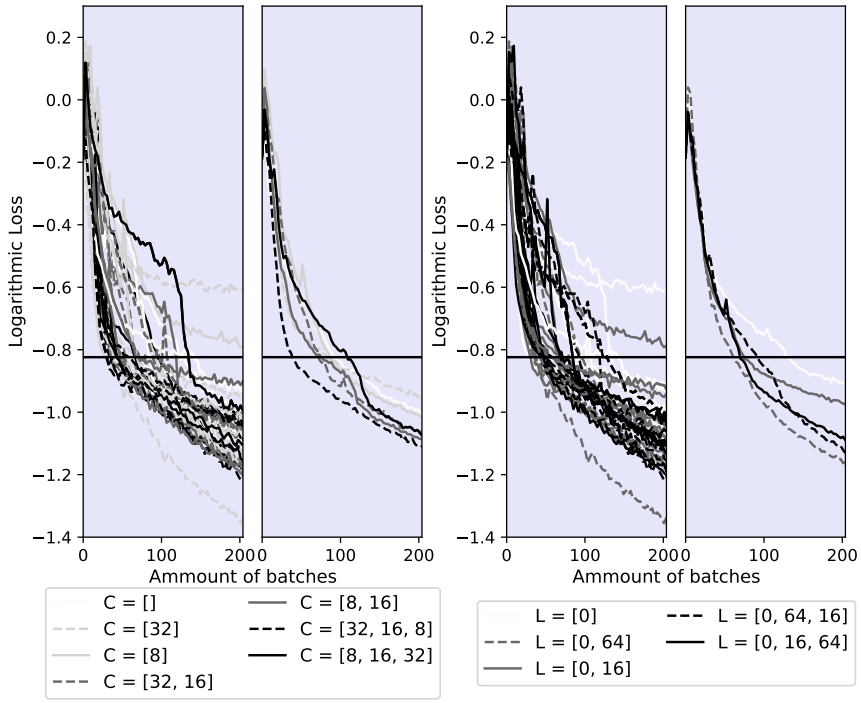


Figure 3.6: The same plot setup as Figure 3.5 but for different layers of the network. All plots show the logarithmic loss as a function of the realised batches. The 2 left pictures show a darker grey for more convolution layers, C indicates the channels of the layer ([] indicates no convolution layer). The right 2 show a darker grey for more linear layers, L also indicates the channels of the layers (0 indicates the output layer with a fixed amount of channels, equal to the amount of spline control points). In both halves the right graph of the two shows the average for every single colour.

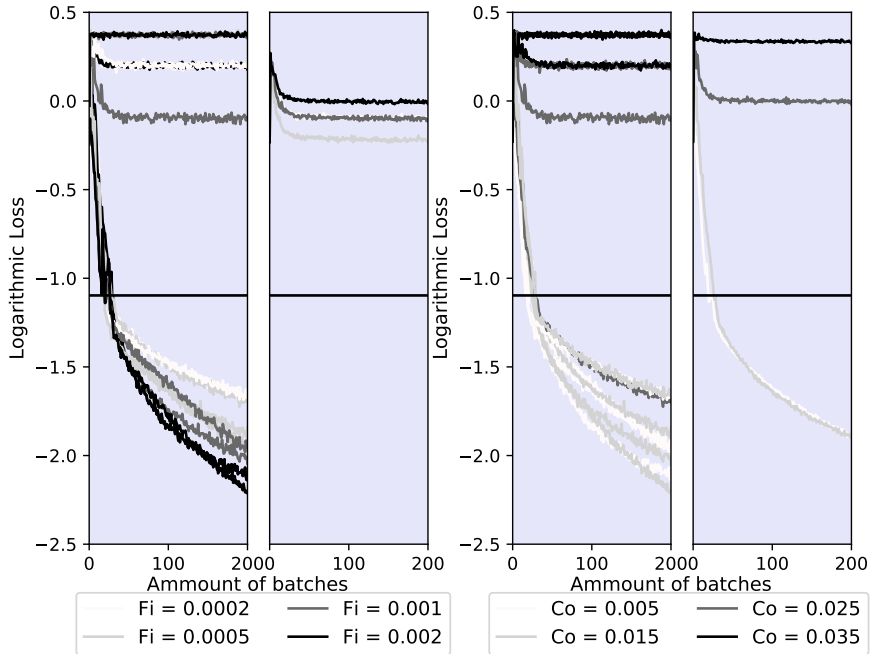


Figure 3.7: Again the same plot setup as Figure 3.5 but for different learning rates. All plots show the logarithmic loss as a function of the realised batches. The 2 left pictures show a darker grey for a higher fine learning rate, Fi. The right 2 show a darker grey for a higher coarse learning rate, Co. In both cases the right graph of the two shows the average for every single colour. Threshold is set to 0.07.

Obviously the two highest coarse learning rates are way to high and can not reach the threshold. Nevertheless the second highest learning rate is able to reach the threshold sometimes. The two lowest coarse learning rates do consistently hit the threshold. Most interestingly the lower coarse learning rate achieves the threshold faster than the higher learning rate. The latter however is able to achieve a lower loss after the threshold, although the fine learning rates beneath the threshold are the same.

From the most left plot it is also clear that the highest fine learning rate achieves the lowest loss. It should be noted that it may cap off higher when the network is trained longer. Lastly it is also interestingly to look at what happens at the threshold for high fine learning rates. It is oscillating a lot, jumping above and beneath the threshold. Because the Adam optimiser has not been used yet it does not know how to apply the back-propagation. The Adam optimiser needs some time to learn itself (it is an adaptive optimiser). When not trained it will increase the loss above the threshold, after which the coarse optimiser again decreases it below the threshold.

When also looking at the plots with the different threshold values given in the appendix the best set of parameters is the lowest of Figure 3.7. Which has a coarse learning rate of 0.015 and a fine rate of 0.002. This will be used in the next section. There is chosen to not train for an even lower optimiser threshold since the coarse optimiser will have more problems reaching this threshold. In addition, remember that only the best results are used in this figure, not the average of all sessions.

3.3. The resulting network explored

In this last section the optimal parameters from the last section will be used to train a network for more samples and epochs. This resulting loss will also be compared to the loss of the network on a independent validation set of 10.000 samples, as is usual for neural networks. At the end, the network will be trained by using an apodization filter on the image.

First the potential of the network with the optimal parameters is displayed. The network will be trained on a large amount of samples (= 60000) and more epochs (= 6). Although batch size can also influence the learning speed and the achievable loss [Hoffer et al. (2017)], it will not be adjusted here and remain equal to

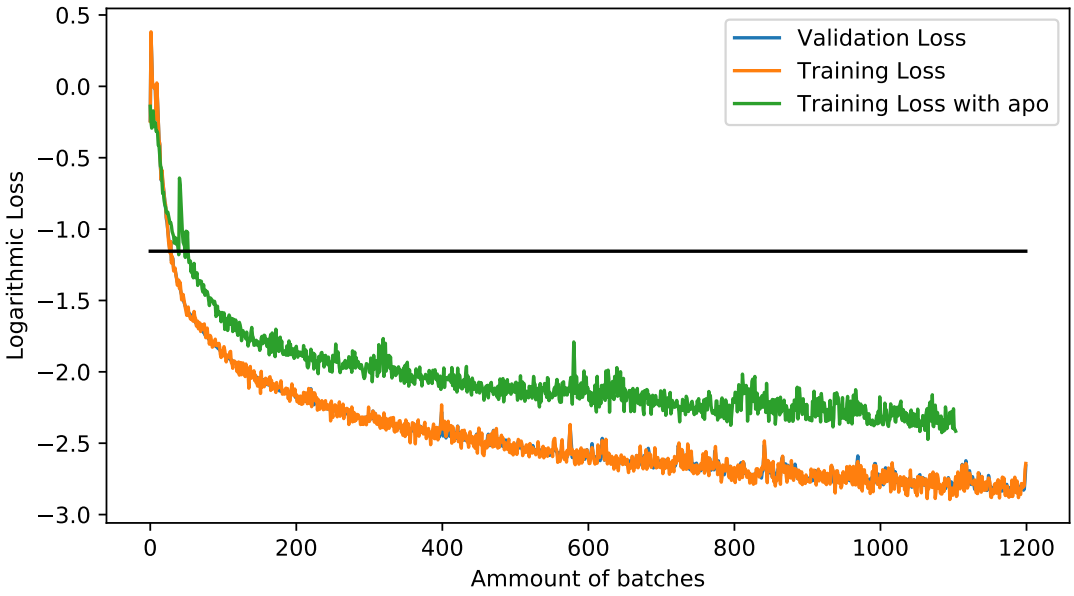


Figure 3.8: The logarithmic loss of the optimised network against the completed batches. The network is trained for 60000 samples for 6 epochs. The validation loss is calculated on a independently generated validation set. The training of the network took 72 minutes, calculating the validation loss every batch increased the training time. The logarithmic loss of the same network with apodization filter is also shown, when not calculating the validation loss, training time is greatly reduced to 13 minutes.

300 samples per batch. The plot of the logarithmic loss during training and of the validation loss is given in Figure 3.8.

The optimised network reached a loss of 0.0015, which is thus 0.15% of the untrained loss. Lens predictions and the resulting images are compared to the actual input image in Figure 3.9. This Figure gives an impression of what the result of a trained network can look like. From Figure 3.8 it is clear that the validation loss is equal to the training loss. It has a bit less oscillations probably because it is computed on the a much larger set of samples. Them being equal justifies using the training loss, which is already calculated for training, in the previous results.

The same network, using an apodization filter, is also optimised with respect to the learning rate. The apodization filter will average out the peaks and lulls. The optimisation is presented in the appendix, Figure A.4. The best learning rates are the same as with no apodization but the threshold is 0.15. These parameters are used to train a network for more samples and epochs. The result is also displayed in Figure 3.8. The width of the aperture, R, is taken larger (=L/40), such that the resulting image contains more detail. The apodization filter also averages out the detail in the intensity, as was seen in Figure 3.3. The prediction of the network is displayed in the appendix, Figure 3.9. The training with apodization filter is not able to reach the same loss as without a filter.

More Control Points

So far only 3 control points are used. The last result of this chapter will therefore represent the training of the neural network for different amount of output variables (=control points). The loss as a function of the completed batches is displayed in Figure 3.10. The threshold is set a bit higher because it will be harder for the network to train. For comparison the best performing training session from Figure 3.7 is added too. By adding a lot of control points the resulting image will oscillate more, therefore for 12 control points the neural network is also trained with an apodization function. Again the average of the 2 best out 5 training session is displayed. Some predictions of the network are displayed in the appendix again, Figure A.6.

Clearly the network has a harder time training for more control points. Which is expected since more output variables need to be optimised. If given longer the networks may achieve lower loss. Interestingly for 12 control points the use of an apodization filter decreases the total loss slightly. From the predictions in the appendix it can be seen that for this case the image is still oscillatory, even though an apodization filter is applied. It can also be seen that there are so few bins inside the aperture that the spline approximation is just a hair more detailed then the original linear approximation shown in the left plots. Of course the chosen learning rates and optimiser threshold will not be optimal for this setup. A similar optimisation as the case with 3 control points can be performed.

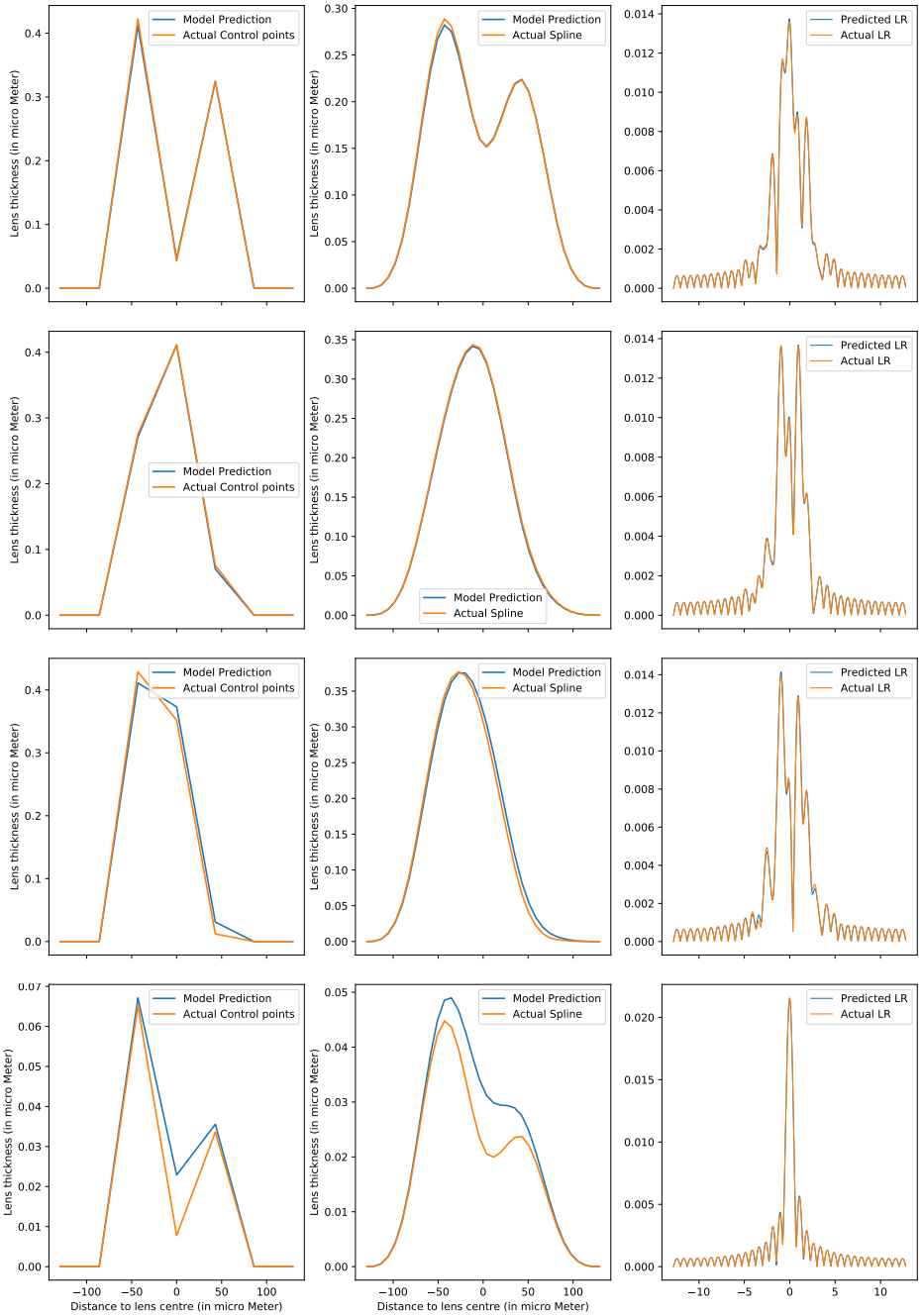


Figure 3.9: The predictions of the network compared to an actual lens and image by the network of Figure 3.8. Left: the height of the control points, note the the two zero points on the ends. Middle: the resulting spline approximation of the control points. Right: the light distribution (image) of the corresponding lenses. For the lens the y-axis is around 100 times smaller than the x-axis. For the image the axis are insignificant, these scale with parameters like the incoming intensity or the distance Δz. These parameters are not used in any other calculation and are not important for training the network. Lastly note that the lower plots have a very small y-axis.

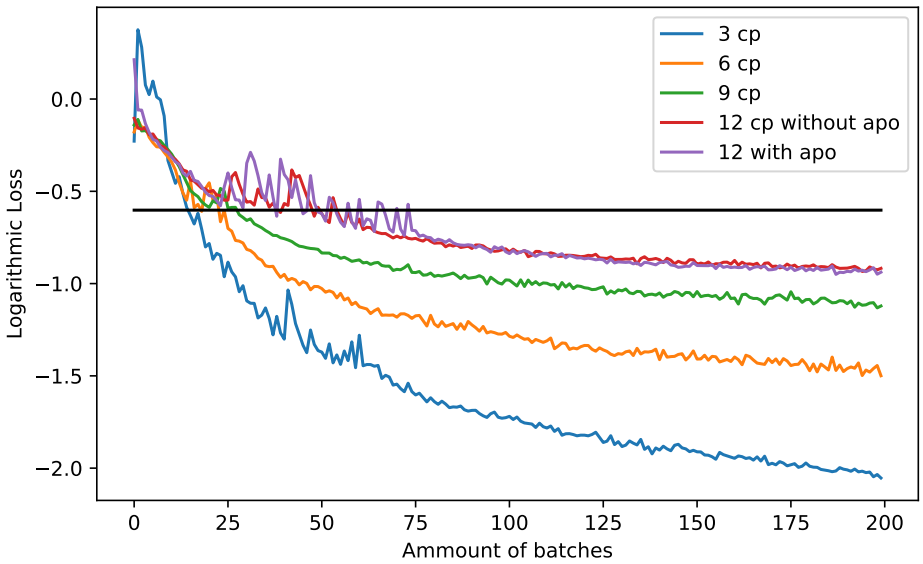


Figure 3.10: The logarithmic loss of networks with a different number of control points (cp) against the completed batches. The network is trained for 15000 samples for 4 epochs (same as for example Figure 3.7). The threshold for 3 control points is 0.07, all other networks are trained wit an threshold of 0.15.

4 Discussion

The representation of the surface of a lens by a spline proofed to be quite useful. Partly because they are used often in Neural Networks. In the setup of this thesis they can represent smooth lenses easily while maintaining a low amount of characteristic variables that can be outputted by the Neural Network. The number of control points can be varied to achieve a more detailed lens. From Figure 3.2 it can be derived that the lens needs to be thin and small compared to the overall pupil-line to get a decent image (that does not oscillate a lot). A large lens will cause spectral leakage in the resulting image. This puts a bound on how much bins are used inside the aperture to approximate the spline. Having a relatively high amount of control points for the lens inside this aperture may therefore not induce more detail. This bound is mostly due to the nature of the Fraunhofer approximation, which is limited by the properties of the Fourier transform. Increasing the total numerical bins of the pupil-line can solve this issue slightly but will increase the computation time of every calculation. Adding an apodization filter to average out the oscillations has also been tried. The filter allowed the aperture to be larger but also removed some detail from the resulting images. In the end the network would attain an equal or higher loss when trained with an apodization filter.

The Fraunhofer approximation in combination with the inherent smoothness of the spline also limits which images can be created (the image space). Since a small aperture is needed to avoid spectral leakage the resulting image will be a close resemblance of the Sinc function. Because of the smoothness of the spline it is not possible to create for example a constant light intensity.

It can, consequently, be established that the Fraunhofer approximation is limiting in two ways for the neural network. Firstly it limits the aperture width to avoid spectral leakage. Which in turn limits the detail achievable by the spline. Secondly, combined with the smooth spline it limits the different images that can be created. Since every image has an inverse Fourier transform (inverse Fraunhofer), the problem is not finding the inverse. Because the phase of this inverse will be wrapped between 0 and 2π, unwrapping this phase into a continuous phase shift is very hard. Therefore also finding a smooth lens that creates this phase shift is a complex problem.

Since for arbitrary images there does not exist a smooth lens, within the predefined specifications, in general, the network will not be able to find a lens. The training samples (images) can, however, be generated by a random spline and the Fraunhofer approximation. In this way the image can certainly be created by the smooth spline. The network may consequently be able to find the inverse solution. In this thesis the defining parameters of the network have been optimised to train on such a training set. The optimal network was able to reduce the loss to 0.15% of the loss of an untrained network. This is very promising. A network is able to almost exactly give a lens that could produce the desired image. Interestingly the predicted lens corresponded most of the time with the lens used to generate the image. This indicates that the inverse lens problem may have a unique smooth solution up to an additive constant.

It should be noted that the optimal parameters derived in the results may not be overall the best, since for example the optimal learning rates can vary for different network layer setups. To investigate the ultimate optimal parameters the whole parameter space should be explored simultaneously. This can even include the amount of samples, epochs and batch size. However, this would take exponentially longer for every different parameter (value) added. That said such an optimisation should only be done once and for certain applications it is not a problem if this takes a month.

Most of the results have been derived for 3 control points. When more control points are used, similar results can be obtained. For more control points the lowest loss achievable will not be as low as for 3 control points. This loss could be more optimised then was done in this thesis, by again optimising for the learning rate and other parameters. Having a huge number of control points for the spline is however useless since there are only a few numerical bins inside the aperture, which is a consequence of using the Fraunhofer approximation. In addition the amount of control points can also be seen as parameter for which the loss can be optimised, because, in general, it is unknown how many control points produce an optimal image.

For future research it might be very interesting to explore adding multiple levels of splines onto each other. The first level could for example describe the rough outline of the lens, with a large distance between the minima and the maxima but with only a small number of control points, such that the large variations happen slowly. A second layer can be added on top, which has more control points but with less distance between the

minima and maxima. The second layer would then have less of an impact on the general shape of the lens but have a larger impact on the details. Even more layers with more control points could be added. All control points can be the output of the neural network. With this approach a smooth lens but with a detailed surface can be described, such that a wider image space may be explored. In addition a different representation of splines can be used.

In this thesis only the 1 dimensional case is explored. The 2 dimensional case can potentially be equivalent. A large problem would be to make a good definition of the locations of the control points. This can be on a rectangular grid or using polar coordinates. The 2 dimensional case can also be very limited by the variety of images it can create. It is unclear if the 2 dimensional Fraunhofer and 2 dimensional spline representation can fix the problems of the 1 dimensional case. However, it may clear up underlying problems more easily.

For future research it is better to look at for example the Fresnel approximation. This is also a diffraction integral and therefore may result in oscillations in the output image as well. Even better would be to have a ray-tracing program function as the solution of the forward lens problem. Opposite to the refraction integrals this does assume non-monochromatic light. In contrast, such a program can be very computationally heavy, especially when it is used almost half a million times to train a neural network. That said, it is fine if training takes very long, since to solve the inverse problem this has to be done once but finding the optimal training parameters may take a very long time. A ray-tracing program should be coded such that it can be used by the back-propagation of PyTorch, which means that it should be coded manually for this application.

The non supervised training of the network is similar to the training of PINN's. PINN's often solve partial differential equations, in which the gradients of the outputs need to be calculated for the loss function. These gradients can immediately be used for back propagation. The Fraunhofer approximation does not have this computational advantage, but does have a clear inverse (the inverse discrete Fourier transform). This makes back-propagation possible, although less efficient then the original PINN's. When using ray-tracing programs back-propagation may be hard since the ray-tracing as a whole does not have a clear inverse. However every single computation inside the ray-tracing algorithm has a clear inverse, so back-propagation may be possible. For example; it should be possible to find the mathematical inverse of every ray.

Ultimately this unsupervised learning can be very powerful. Namely a training set of arbitrary images can be generated. The network does not need to know the lenses that produced these in advance, which is in general not known. The network can still train this way since it is physically known which image the predicted lens will produce. It is thus able to calculate how far off the prediction is. This unsupervised learning similar to PINN's is therefore worth investigating for different solutions of the forward lens problem.

Lastly it is worth to mention that a wider variety of images can be produced if the incoming optical wave is altered. Only a point source precisely in front of the lens an infinite distance away has been used to create the wave so far. By having a point source of axis the image can be shifted. By adding multiple point sources different images can be created. Solving the inverse problem for varying sources and optical setups may be possible with a neural network.

5 Conclusion

The experimental setup, with a smooth lens and the Fraunhofer approximation to create an image, is very limiting in the images that can be created (the image space). This is mostly due to the Fraunhofer approximation, since to avoid spectral leakage an aperture should be used. In combination with an undetailed smooth spline this will always create images which are very similar to the sinc function. Because of this limited image space, the neural network can only be trained on images of a training set that are generated using this setup. In this way there will always be a lens that will create this image.

The neural network can be trained unsupervised on only a set of images, thanks to the physical knowledge of the forward lens problem. The lens that created the image would need to be known for supervised training. If the network is trained on an image training set that is in the image space of the experimental setup, the network can very well give a predicted lens that can produce these images. A network trained on 60000 randomly generated images is able to reduce the loss to 0.15% of that of random solutions. For this special case this is thus very good. To reach this low loss the parameters of the experiment have been optimised for the lowest loss, while maintaining a somewhat useful output image.

Unfortunately the full power of the proposed unsupervised training method is in this way not used. Theoretically it could train on very random images and, if enough are provided, the network could be able to give a lens that creates these images.

To represent the outputted lenses by a B-spline was quite useful. It allows to have a smooth lens while maintaining a low number of output variables for the neural network.

Since the unsupervised training worked very well in this special case, it may hopefully achieve similar results in more general cases. More precisely, a similar experiment can be set up with a ray-tracing algorithm as the solution to the forward lens problem. This physical algorithm is able to achieve a hugely wider image space. Using a neural network it may be possible to find a lens that creates an arbitrary image and thus gives an approximate solution to the inverse lens problem. A problem that could occur when using ray-tracing is that back-propagation might not work, since it does not have a clear inverse.

A Additional graphs for the results

Here will be displayed some extra plots of the results discussed which are either too large or too much to also display in the results. No further comments on this graphs will be given since they are very similar to the plots displayed already.

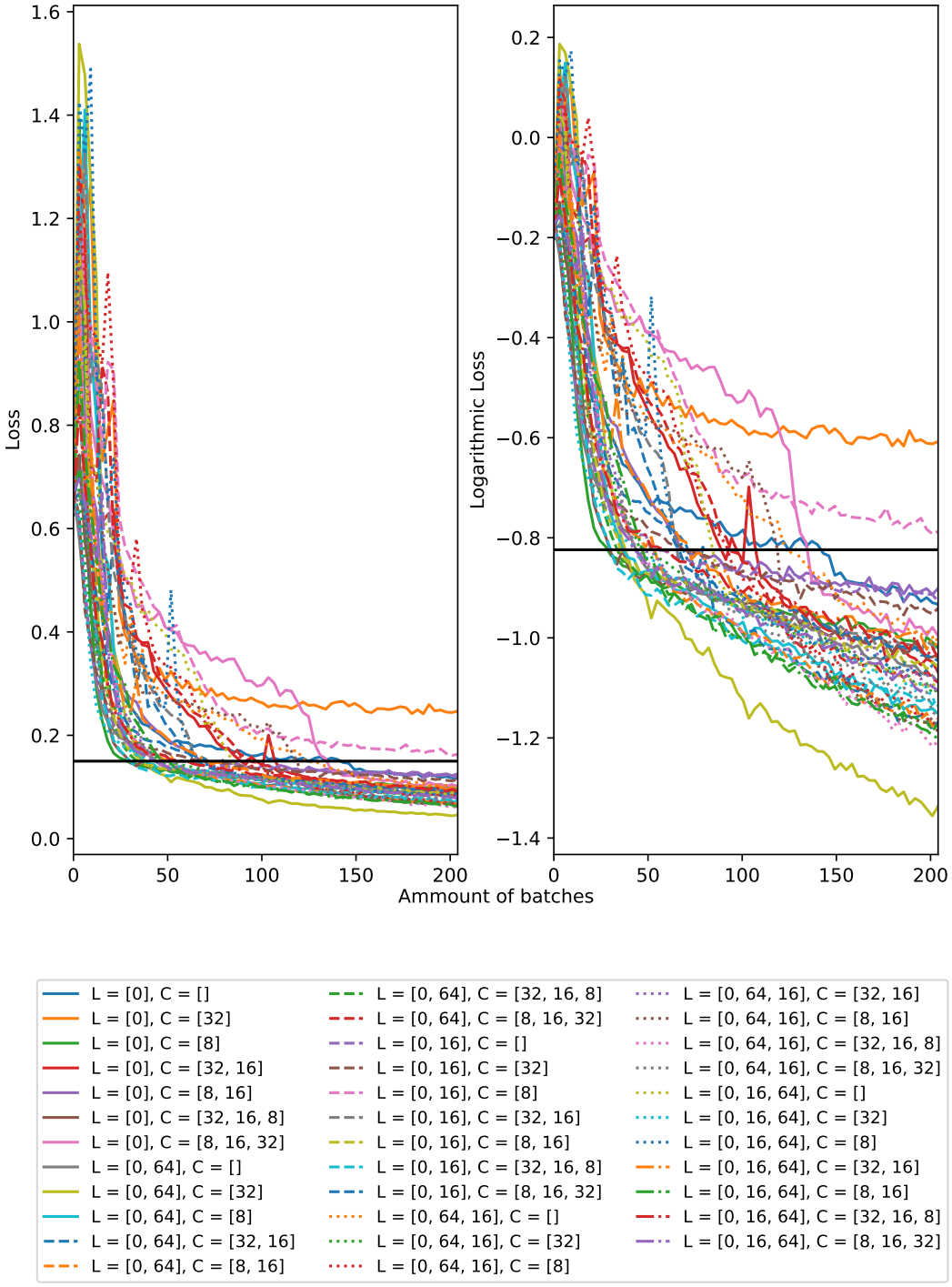


Figure A.1: Corresponding to Figure 3.6. The average loss of the 2 best training sessions per set of parameters per batch as a function of the number of completed batches. The left shows the loss, the right shows the logarithm (base 10) of the loss. C indicates the channels of the convolution layers ([] indicates no convolution layer). L indicates the channels of the linear layers (0 indicates the output layer with a fixed amount of channels, equal to the amount of spline control points). The black horizontal line shows the threshold for the RProp and Adam optimiser. $t = 1/300$, $R = L/40$, $lrCoarse = 0.02$, $lrFine = 0.0005$ and threshold = 0.15.

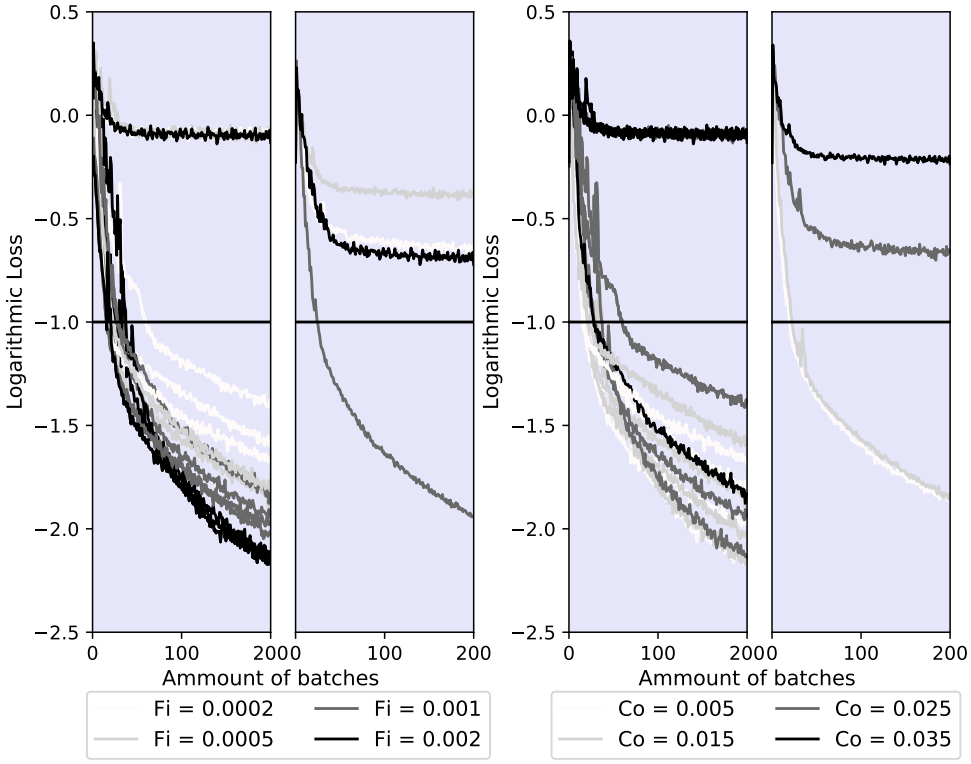


Figure A.2: The exact same as Figure 3.7 but the threshold set to 0.1.

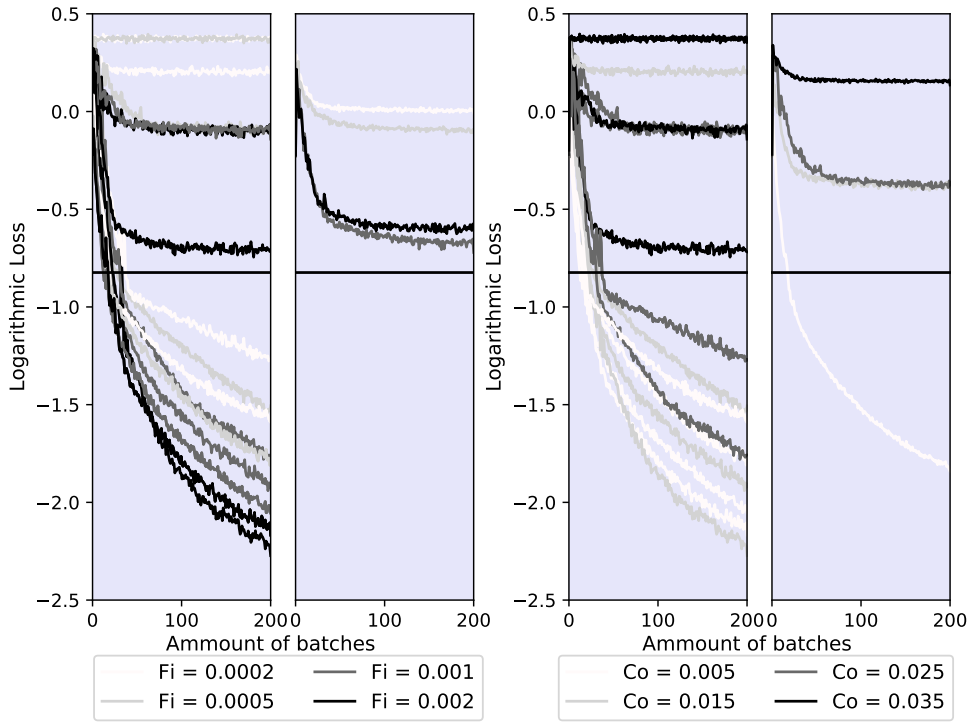


Figure A.3: The exact same as Figure 3.7 but the threshold set to 0.15.

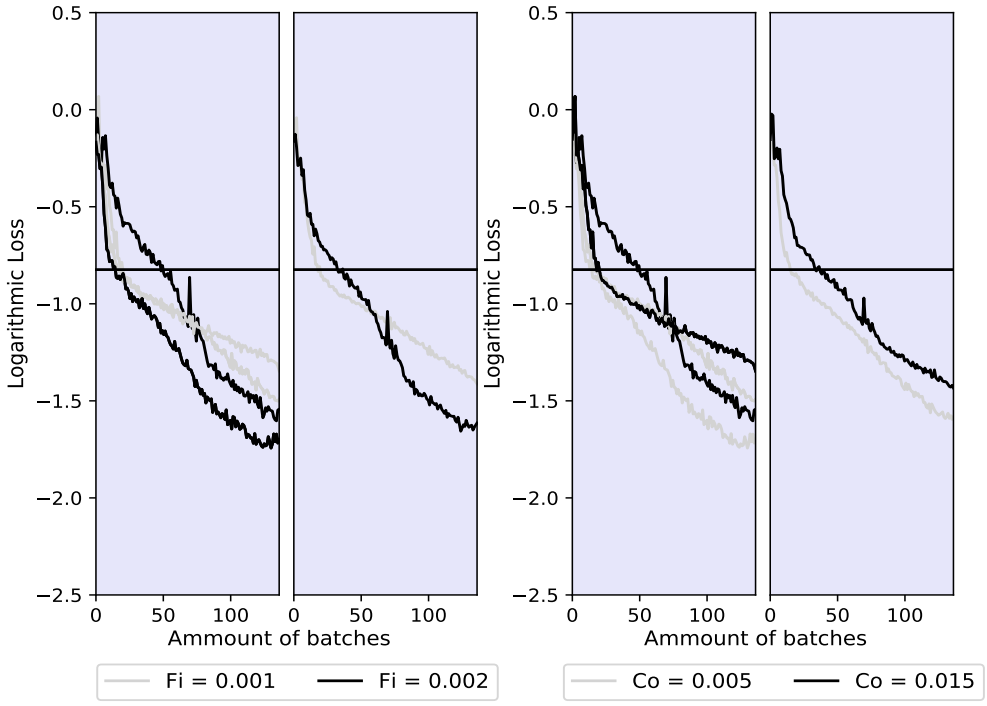


Figure A.4: The exact same as Figure 3.7 but with an apodization filter used and the threshold is set to 0.15, this gave a slightly lower loss then the threshold set to 0.07 (result not displayed). Only the 2 best learning rates from the earlier results are used. $R = L/40$

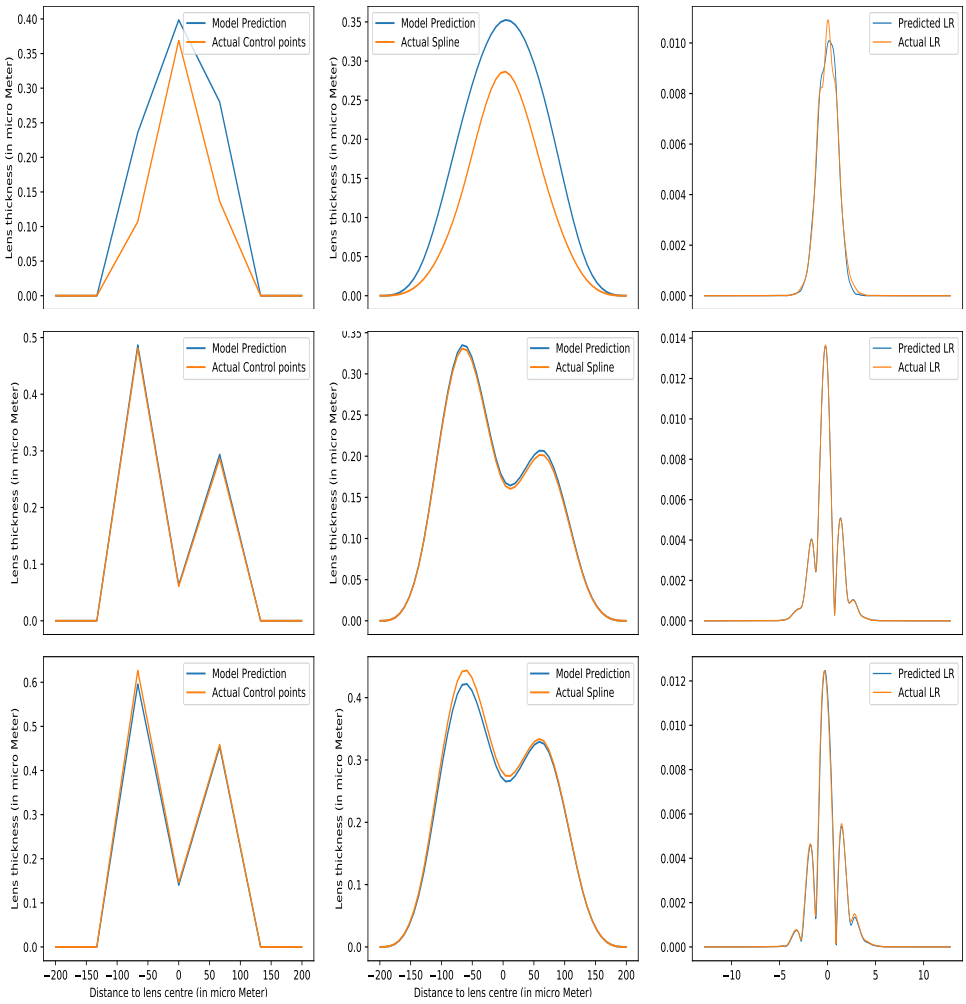


Figure A.5: Exactly the same setup as Figure 3.9. An apodization filter is however used to average out the resulting image, this is also used in training. The network has a slightly harder time to train for this setup.

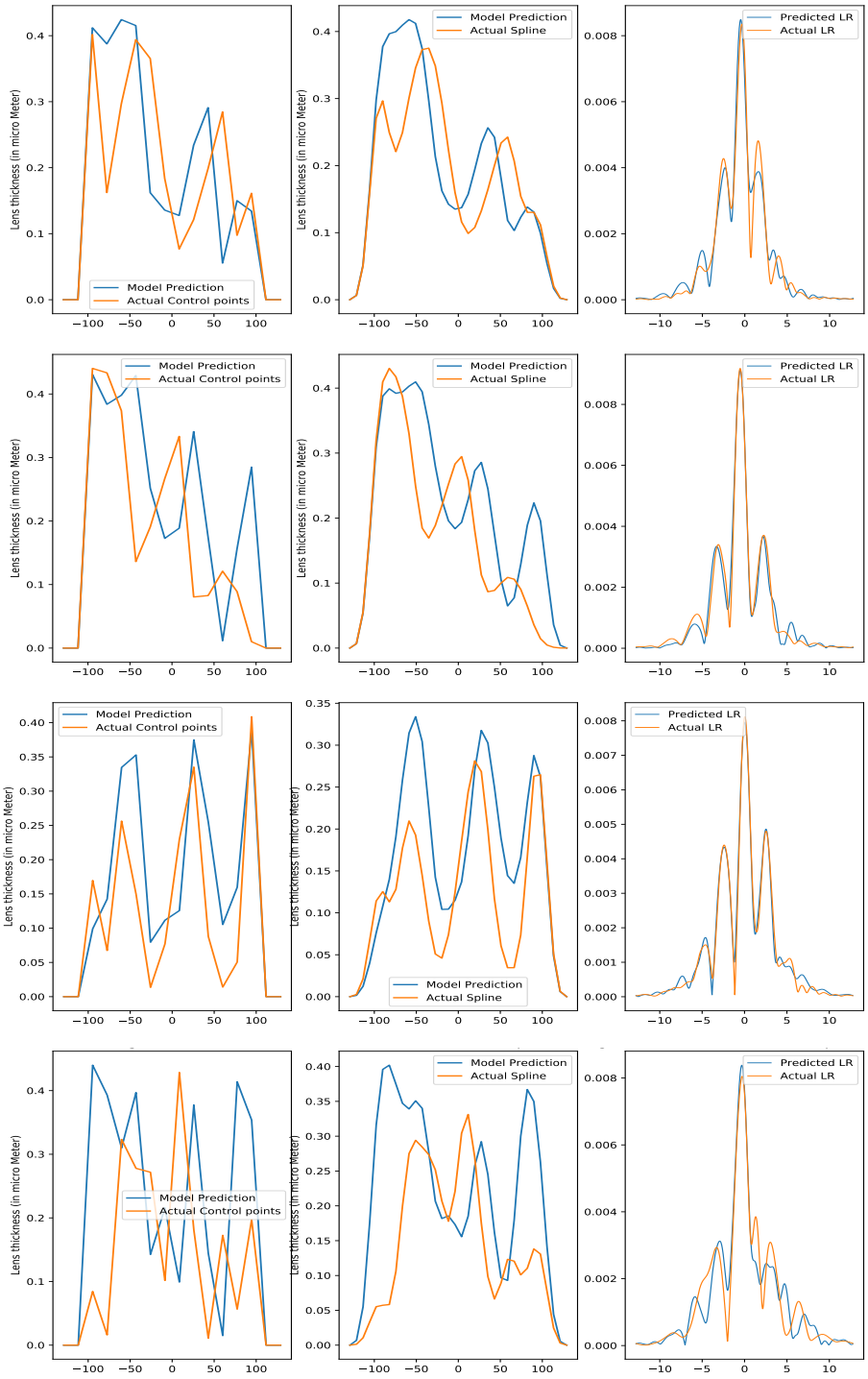


Figure A.6: Exactly the same setup as Figure 3.9. 12 control points are used, however, with an apodization filter. The network has a even harder time to train for this situation. Note that this is after 15000 samples and 4 epoch, compared to 60000 samples and 6 epochs in Figure 3.9.

B Python code

All of the Python code used in this thesis can be viewed on GitLab via: https://gitlab.tudelft.nl/ajladam/free-form-optics.

Bibliography

Joseph W Goodman. Introduction to Fourier optics. 1, 2005.

Daniel Graupe. *Principles of Artificial Neural Networks*. World Scientific Publishing Co., Inc., 1997. ISBN 9810225164.

John C. Heurtley. Scalar rayleigh–Sommerfeld and Kirchhoff diffraction integrals: A comparison of exact evaluations for axial points∗. *J. Opt. Soc. Am.*, 63(8):1003–1008, Aug 1973.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: Closing the generalization gap in large batch training of neural networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1729–1739, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Yuri Bazilevs J Austin Cottrell, Thomas J R Hughes. Isogeometric analysis : toward integration of cad and fea, 2009.

Ilhan Kaya, Kevin P. Thompson, and Jannick P. Rolland. Comparative assessment of freeform polynomials as optical surface descriptions. *Opt. Express*, 20(20):22683–22691, Sep 2012. doi: 10.1364/OE.20.022683.

Ehsan Kharazmi, Zhongqiang Zhang, and George Em Karniadakis. hp-vpinns: Variational physics-informed neural networks with domain decomposition, 2020.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2014.

Andrey Kurenkov. A brief history of neural nets and deep learning, 2018. URL `http://www.andreykurenkov.com/writing/ai/a-brief-history-of-neural-nets-and-deep-learning/`.

Samuel J. Ling, Jeff Sanny, and William Moebs. University physics volume 3, 2016.

Robert L Lucke. Rayleigh–Sommerfeld diffraction and Poisson's spot. *European Journal of Physics*, 27(2): 193–204, jan 2006.

Tom Lyche and Knut Mørken. *Spline Methods*. University of Oslo, 2008.

David Mendlovic, Zeev Zalevsky, and Naim Konforti. Computation considerations and fast algorithms for calculating the diffraction integral. *Journal of Modern Optics*, 44(2):407–414, 1997. doi: 10.1080/09500349708241880.

Norman Morrison. *Introduction to Fourier Analysis*. New York : Wiley, 1994. ISBN 978-0-471-01737-0.

Phil Picton. *Introduction to neural networks*. Macmillan Basingstoke [England], 1994. ISBN 0333618327.

M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, February 2019. doi: 10.1016/j.jcp.2018.10.045.

J.D. Schmidt. *Numerical simulation of optical wave propagation: With examples in MATLAB*. 2010. doi: 10.1117/3.866274.

Sebastian Schuchmann. History of the first AI winter, 2019. URL `https://towardsdatascience.com/history-of-the-first-ai-winter-6f8c2186f80b`.

Yuliy Schwartzburg, Romain Testuz, Andrea Tagliasacchi, and Mark Pauly. High-contrast computational caustic design. *ACM Trans. Graph.*, 33(4):74:1–74:11, July 2014. ISSN 0730-0301. Proc. SIGGRAPH 2014.

David Voeltz. *Computational Fourier Optics*. Spie, 2011. ISBN 978-0-8194-8204-4.

Brian Vohnsen. A short history of optics. *Physica Scripta*, 2004:75, 07 2006. doi: 10.1238/Physica.Topical.109a00075.

Z. Zhang. Improved adam optimizer for deep neural networks. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pages 1–2, 2018.