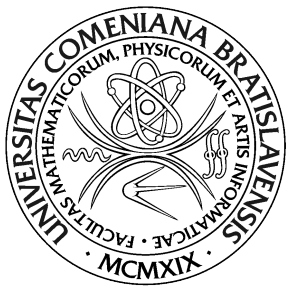
**UNIVERZITA KOMENSKÉHO V BRATISLAVE**
**FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY**




# PROCESSING OF IMAGE METEOROLOGICAL DATA BY DEEP LEARNING

Diplomová práca

2021 Bc. Filip Pavlove

**UNIVERZITA KOMENSKÉHO V BRATISLAVE**
**FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY**


# PROCESSING OF IMAGE METEOROLOGICAL DATA BY DEEP LEARNING

Diplomová práca

| | |
|---|---|
| Študijný program: | Aplikovaná informatika |
| Študijný odbor: | Aplikovaná informatika |
| Školiace pracovisko: | Katedra aplikovanej informatiky |
| Školiteľ: | RnDr. Andrej Lúčny, PhD. |

Bratislava, 2021 Bc. Filip Pavlove

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

93301259

# ZADANIE ZÁVEREČNEJ PRÁCE

| | |
|---|---|
| **Meno a priezvisko študenta:** | Bc. Filip Pavlove |
| **Študijný program:** | aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma) |
| **Študijný odbor:** | informatika |
| **Typ záverečnej práce:** | diplomová |
| **Jazyk záverečnej práce:** | anglický |
| **Sekundárny jazyk:** | slovenský |

**Názov:** Processing of Image Meteorological Data by Deep Learning
*Spracovanie obrazových meteorologických dát pomocou hlbokého učenia*

**Anotácia:** Práca má výskumný charakter. Vstupom sú merané a z časti anotované obrazové meteorologické dáta, predovšetkým kamerové snímky z automatických meteorologických staníc. Tieto treba spracovať: normalizovať ich, očistiť a doplniť k nim anotácie podľa kódov meteorologického pozorovania, geografických údajov a ručne zadaných hodnôt. S nimi potom treba navrhnúť a vyskúšať rôzne metódy spracovania obrazu na hľadané fyzikálne veličiny, predovšetkým horizontálnu a prevládajúcu dohľadnosť. Ťažisko týchto metód bude spočívať v použití hlbokého učenia. S touto, relatívne modernou technikou sa treba dôkladne oboznámiť, rovnako ako s podobnými prácami v danej aplikačnej oblasti.

**Cieľ:** Cieľom práce je navrhnúť viacero metód určovania fyzikálnych veličín z obrazových meteorologických dát, vyhodnotiť ich úspešnosť a porovnať ich.

**Literatúra:** Chollet, F.: Deep learning v jazyku Python, Grada, 2019
Learning OpenCV 3, Computer Vision in C++ with the OpenCV Library By Gary Bradski

**Poznámka:** Platforma: OpenCV, Keras alebo Pytorch, Python

**Kľúčové slová:** hlboké učenie, počítačové videnie, meteorologia, spracovanie obrazu

| | |
|---|---|
| **Vedúci:** | RNDr. Andrej Lúčny, PhD. |
| **Katedra:** | FMFI.KAI - Katedra aplikovanej informatiky |
| **Vedúci katedry:** | prof. Ing. Igor Farkaš, Dr. |
| **Dátum zadania:** | 07.02.2020 |

**Dátum schválenia:** 13.02.2020 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

**Poďakovanie:**

# Abstrakt

Práca má výskumný charakter. Vstupom sú merané a z časti anotované obrazové meteorologické dáta, predovšetkým kamerové snímky z automatických meteorologických staníc. Tieto treba spracovať: normalizovať ich, očistiť a doplniť k nim anotácie podľa kódov meteorologického pozorovania, geografických údajov a ručne zadaných hodnôt. S nimi potom treba navrhnúť a vyskúšať rôzne metódy spracovania obrazu na hľadané fyzikálne veličiny, predovšetkým horizontálnu a prevládajúcu dohľadnosť. Ťažisko týchto metód bude spočívať v použití hlbokého učenia. S touto, relatívne modernou technikou sa treba dôkladne oboznámiť, rovnako ako s podobnými prácami v danej aplikačnej oblasti.

**Kľúčové slová:** hlboké učenie, počítačové videnie, meteorologia, spracovanie obrazu

# Abstract

The work has a research character. The input is measured and partially annotated image meteorological data, especially camera images from automatic meteorological stations. These have to be processed: normalize, cleanse and annotate according to meteorological observation codes, geographic data and manually entered values. With them then it is necessary to design and test different methods of image processing into the desired physical quantities, especially horizontal and predominant visibility. The focus of these methods will be on the use of deep learning. This relatively modern technique needs to be thoroughly familiar with, as well as similar work in the application area.

**Keywords:** deep learning, computer vision, meteorology, image processing

# Contents

# List of Figures

# List of Tables

# 1. Introduction

# 2. Related work

# 3. Theoretical background

## 3.1 Machine learning

Machine learning as a branch of artificial intelligence is a study focused on development of algorithms capable of performing tasks without being explicitly programmed. Over the past decades machine learning based algorithms showed great deal of success in the domains such as medicine, computer vision, computer graphics, bioinformatics, finance, and many more.

Generally, machine learning methods are either supervised, unsupervised or reinforcement. The goal of supervised learning methods is to predict desired output based on input given to the trained model. Some of the widely used supervised algorithms are K-Nearest Neighbours, Support Vector Machines, Decision Trees or some of the Neural Networks. The unsupervised methods such as Principal Component Analysis, and many more, unlike supervised, lack the desired output, and are mainly used for finding the patterns in the given data set. Reinforcement learning aims on achievement of goal performed by the agent in its environment. To obtain desired behaviour, agent performs series of actions with respect to reward function.

Due to the increasing trend of computational power, mainly of GPU, TPU (Tensor Processing Unit), and the availability of large data sets, field of the deep learning is nowadays more relevant then ever. Deep Learning models refers to the set of neural networks with many hidden layers, hence the name "deep".

### 3.1.1 Supervised learning

Supervised learning models are learned by the example. Examples make up the data set, and every example $(x^{(i)}, y^{(i)})$ is the pair of input $(x^{(i)})$, and desired output $(y^{(i)})$, sometimes referred to as label or target. Superscript $(i)$, refers to the index of sample in the data set. Input may be either vector of features, matrix (eg. grey image) or tensor (eg. RGB image). Depending on the learning problem, target dimensions may also differ. We will denote set of possible inputs as $\mathcal{X}$, and set possible targets as $\mathcal{Y}$. To solve supervised task, learning algorithm trains model, often denoted in the literature as function $h(x)$, $h : \mathcal{X} \mapsto \mathcal{Y}$ (hypothesis), which attempts to approximate real problem as closely as possible.

Two of the most common tasks in the domain of supervised learning are regression, and classification tasks. The objective of the classification tasks is to specify in which category some input belongs to. The output of a machine learning system tackling classification problem is usually a function $f : \mathbb{R}^n \mapsto \{1, ..., k\}$, where $k$ denotes the number of encoded categories. An example of the classification task is the prediction of the visibility into categories like good, moderate or bad, based on the input images. The output of the regression task is a function $f : \mathbb{R}^n \mapsto \mathbb{R}$. An example of the regression task is the prediction of visibility represented by continuous number in same units like metres or kilometers.

The goal of the training process is not only to perform with good results on the data presented to the learning algorithm, but also to generalize. By generalization, we refer to the model's ability to react to new data. Theory behind the Bias–variance tradeoff aims for formalization of such concept, but we will not get into these details (for more see Figure 3.1). Instead, we will explain some of the basic terminology, and practices of the training process.

When the trained model is not generalizing properly, the problem is usually due to overfitting or underfitting. Overfitting happens when the picked learning algorithm is too complex, and the training examples are being 'memorized'. On the other hand, underfitting occurs when the model is too simple. Both of these problems results in poor performance on the testing set. One of the ways to address these issues is by regularization or splitting the dataset into training,

testing, and validation set. The ratio between training, and testing, set is usually 80/20. The remaining 80% of the training set is then split between training and validataion set with the same ratio. It is important to note, that such a split is just a rule of thumb. The validation set serves us for estimation of the performance during or after the training. If the error on the validation set is way too high, we may be overfitting or underfitting the model. We can react either by choosing less or more complex model, or by tuning of the hyperparameters. In machine learning, a hyperparameter is a setting of the model chosen before the training. On the other hand parameters of the model are learned during the training.

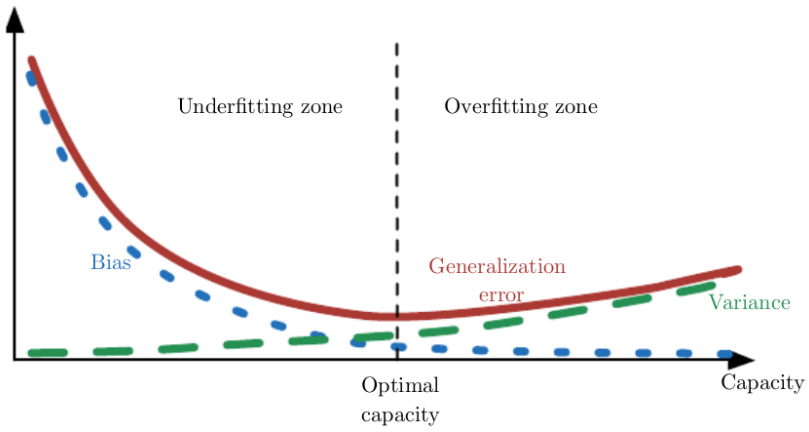



Figure 3.1: "As capacity (complexity) increases (x-axis), bias (dotted) tends to decrease and variance (dashed) tends to increase, yielding another U-shaped curve for generalization error (bold curve)." [5]. One of the most formal ways to measure models capacity is by VC dimensions [15].

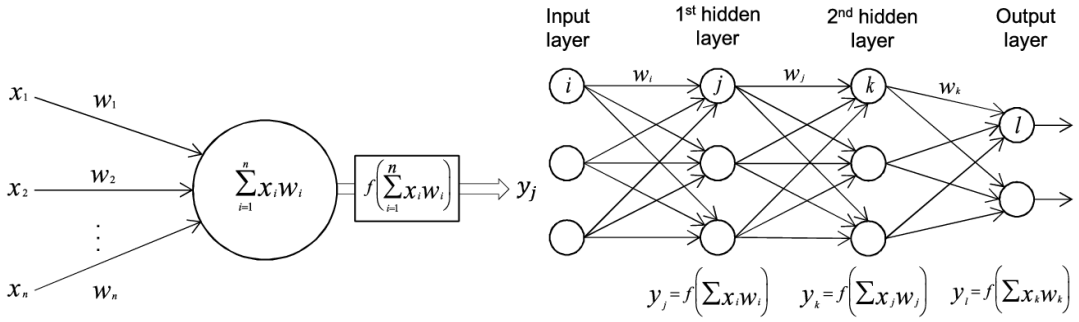
## 3.2 Neural networks

The birth of neural networks might be contributed to the year 1943, when McCulloch and Pitts [9], described neurons with threshold logic. With no learning required these networks assembled in two layers are capable of simulating any function $f : \{0,1\}^n \mapsto \{0,1\}$, with $n$ inputs. Since then many advances have

been made, but we will not cover those, since history is not of our concern in this thesis.

## 3.3 Multilayer Perceptron

Multilayer perceptron (MLP), often referred to as deep feedforward neural network is type of artificial neural network, quintessential to the field of deep learning. It can be shown, that 2-layer MLPs posses an universal approximation property, making MLP capable of approximating any continuous nonlinear function with arbitrary accuracy [(Horniket al., 1989; Cybenko, 1989)]. To unravel the jargon of neural networks, we might start with building blocks of MLPs. Typical MLP consists of input layer, at least one hidden layer, and output layer. Hidden layers are build up of arbitrary number of units which are interconnected between sub-sequential layers. The number of units refers to the dimensionality of the layer, or width. The number of layers on the other hand, refers to the depth of the network. The name "deep" arose from this terminology. In the fully connected layers, value of each unit, (sometimes called neuron) is calculated as weighted sum of the input connections from previous layer. Computed weighted sum is afterwards fed into activation function. Figure 3.2 shows simple architecture of neural network with tweo hidden layers (Figure 3.2b), and illustrates of computation of the single unit (Figure 3.2a).




(a) (b)

Figure 3.2: Figures taken from [16].

As an activation function might serve any differential function. With exception of identity, activation functions are essential part of the MLP, and contribute to its capacity of modeling nonlinear functions. There is not a single rule, stating which function should serve as an activation in the hidden layers. It is highly experimental, and one might choose between functions such as sigmoid, hyperbolic tangent, or rectified linear unit (ReLU) given in equation 3.1, 3.2, and, 3.3, respectively. Despite that, most of the modern neural networks use ReLU [0] activation at the hidden layers. The mathematical equations of the stated activation functions are given as follows:

$$g(x) = \frac{1}{1 + e^{-x}} \quad (3.1) \qquad g(x) = \tanh(x) \quad (3.2) \qquad g(x) = \max(0, x) \quad (3.3)$$

After the input is fed forwards through the sub-sequential layers, the computation approaches final stage at the output layer. Dimensinality as well as activation function of the output layer is highly constrained to the task of trained network. Classification tasks such as digit recognition could be modeled by the simple neural network with 10 outputs with the sigmoid, or softmax (see equation 3.4) activation function. The advantage of softmax over the sigmoid is that it turns outputs to probabilities that sum up to one. Regression tasks, concerned with the predictions of some unbounded continuous variable, could be modeled by network with single output unit with identity function.

$$g(x) = \frac{e^{x_i}}{\sum_j e^{x_j}} \quad \text{where,} \qquad (3.4)$$

$x_i$ represents $i$-th input from previous layer, and

$j$ in the sum of denominator goes through the all inputs of previous layer.

## 3.4 Training

So far, we have talked about the building blocks of the typical feed-forward neural network. We also mentioned classification, and regression tasks in the relation to the inference at the output layer. With the understanding of the covered

terminology, we will move on to the training stage of the networks. The objective of the neural networks training, and generally of training any other machine learning method, is to find the best set of parameters. To be more specific, the goal of the neural network training, is to minimize some loss function $L$ with respect to the weights $W$. The formalized minimizing problem may be written as

$$W^* = \arg\min_W \mathcal{L}(f(X;W), Y) \tag{3.5}$$

where $f(X;W)$ is the set of network outputs given the weights $W$, and $Y$ is the set of desired outputs. Methods used for estimation of weights $W$ are usually gradient based. Widely used algorithm in the context of the neural networks is called backpropagation.

### 3.4.1 Loss Functions

Loss functions are fundamental part of neural network. The computed value of Loss function tells us how good our network performs on the given data. We might be computing either with the whole training set or with the smaller sized batches (called mini-batches). Consider set $\{x^{(i)}, y^{(i)}\}_{i=1}^{N}$. Loss function for the given set of data is computed as follows

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_i(f(x^{(i)}, W), y^{(i)}). \tag{3.6}$$

,where $N$ is the size of the data set, and $f(x^{(i)}, W)$ is the output of the network given the input $x^{(i)}$ and the set of parameters $W$.

The central part of the presented equation is choice of the loss $\mathcal{L}_i$, for the single input. The choice is highly constrained to the task of the model, and the nature of the data. Some loss functions might combine multiple losses, which are used in the final joint loss function. Usually object detection models such as YOLOv3 [11] or Faster R-CNN [12] use these joint loss functions, to estimate correct detection and classification of the objects in the image. We will not cover

those in the greater detail, instead of that we will describe and illustrate some of the most generally known losses used in the majority of applications.

Regression tasks often use loss functions, which capture difference between predictions and outputs of the network. Well known functions with this property are L1, and L2 losses defined as

$$L1 = \frac{1}{N}\sum_{i=1}^{N} \|f(x^{(i)}, W) - y^{(i)}\|_1 \quad (3.7) \qquad L2 = \frac{1}{N}\sum_{i=1}^{N} \|f(x^{(i)}, W) - y^{(i)}\|_2 \quad (3.8)$$

$$= \frac{1}{N}\sum_{i=1}^{N} \left|f(x^{(i)}, W) - y^{(i)}\right| \qquad = \frac{1}{N}\sum_{i=1}^{N} (f(x^{(i)}, W) - y^{(i)})^2$$

The L1, and L2 losses are often refereed to as mean absolute error (MAE), and mean squared error (MSE), respectively. In practice, the L2 shown in Figure 0 is usually preferred over the L1.

During the classification tasks, we may represent the output vector as the probabilities of the $C$ possible classes. The objective of the loss function would be to maximize the probability of the correct class. The categorical crossentropy (CCE) loss function (see Figure 0) with the softmax activation function address exactly that. CCE is given by the equation

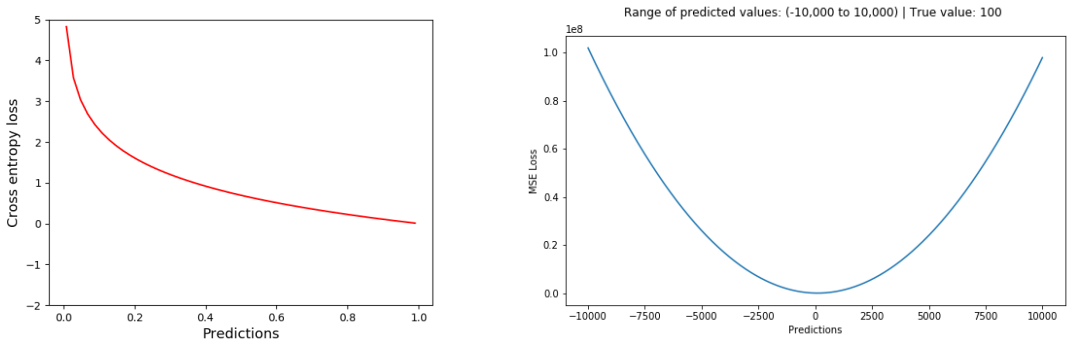
$$\text{CCE} = -\sum_{j}^{C} y_j^{(i)} \log(f(x^{(i)}, W)_j) \quad (3.9)$$

$$= -\log\left(f(x^{(i)}, W)_k\right) \quad (3.10)$$

where, $f(x^{(i)}, W)_j$ represents the calculated probability of $j$-th class given the input $x^{(i)}$. Since for all $j$ only one in the desired output $y_j^{(i)}$ is 1 with the rest set to 0, we could rewrite the equation to the form 3.10. With the value $k$ linked to the only $j$ set to the 1.

### 3.4.2 Gradient descent

It is generally known that training a neural network is NP-complete [4] problem in the sense of finding the set of parameters convergent to the global minimum




(a) (b)

Figure 3.3: Figure of losses.

of the loss function. Thus, the analytical solution of such a complex system as neural network, gets out of the window.

One of the easiest ways to train the neural networks is to use gradient based methods. The gradient descent (GD) is one of them. The objective of GD, is to update the set of parameters in an iterative manner, to minimize a loss function. The result does not guarantee convergence to the global minimum, however local minimum is sufficient. At each step of the GD the weights of the network are updated in the direction of the steepest descent. The direction of the steepest descent is computed as an opposite direction to the gradient of the loss function at the current step. Formally, the update of the weights in the simple gradient descent is computed as follows

$$W = W - \alpha \frac{1}{N} \sum_{i=1}^{N} \nabla_W \mathcal{L}_i(f(x^{(i)}, W), y^{(i)}) \tag{3.11}$$

Where $\nabla_W \mathcal{L}_i$ represents gradient of loss function with respect to the weights $W$ of the network. Hyper-parameter $\alpha$ is set before the training, and indicates the step size of the gradient. The update is performed until the loss is sufficiently low.

Traditionally the gradient in the GD is computed over all training samples. This approach is in the most cases computationally expensive due to large datasets. Slightly modified algorithm Stochastic gradient descent (SGD) address

this limitation, by computing the gradients of the loss function over the subset of the training data. These subsets of the training set are called mini-batches. The figure 3.5 illustrates the iterative approach of gradient descent in the hypothetical landscape of loss function.

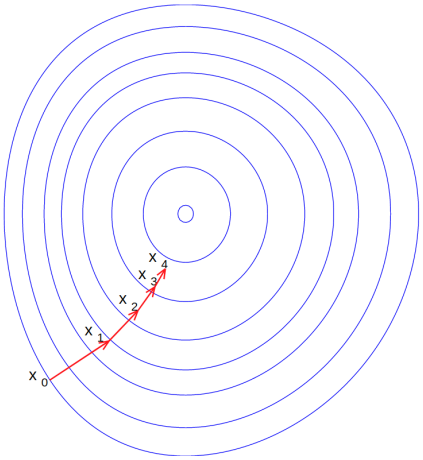



Figure 3.4: As the algorithm starts, the loss is pretty high at the point $x_0$. Through the iteration of 4 updates, we almost converged to the local minimum $x_4$. The illustration of GD is taken from [2].

### Backpropagation

In the previous section concerned with gradient descent, we have not mentioned the computation of gradient. When we use a neural network, an output is computed by flowing forward the information from input $x$ through the network. As the value of loss function $\mathcal{L}$ is computed, backpropagation [8] algorithm allows the information from the loss, to propagate backward through the network in order to compute the gradient. In the short, the backpropagation treat the feed forward neural network as a complex composition of many functions. The gradient is then computed with the chain rule. For example, suppose the real valued functions $y = f(x)$ and $x = g(t)$, then chain rule states that

$$\frac{dy}{dt} = \frac{dy}{dx}\frac{dx}{dt} \tag{3.12}$$

Of course, neural networks work with higher dimensional functions, and the chain rule could be rewritten in the terms of partial derivatives.

Most of the modern libraries like PyTorch represents neural networks as a computational graph, which computes the gradients of the networks with high efficiency.

### Optimizers

Most of the modern neural networks are trained by more efficient variations of traditional gradient descent, which is not sufficient in certain situations. For example, stochastic gradient descent computes zero gradient in the saddle points of the loss function. Consequently, the training gets stuck. To overcome problem like this with addition of more stable and faster training, methods like momentum [10] have been developed. Lets simplify SGD equation 3.11 as

$$x_{t+1} = x_t - \alpha \nabla f(x_t) \tag{3.13}$$

The stochastic gradient descent with momentum is computed as

$$v_{t+1} = \rho v_t \nabla f(x_t) \tag{3.14}$$

$$x_{t+1} = x_t - \alpha v_{t+1} \tag{3.15}$$

The effect of momentum may be understood as accumulation of speed from previous steps. Hyper-parameter $\rho$ represents friction and is usually set to $\rho = 0.9$. Another variant of momentum is Nesterov momentum which is computed as

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + v_t) \tag{3.16}$$

$$x_{t+1} = x_t + v_{t+1} \tag{3.17}$$

The Figure shows effect of both momentum variants to the actual training step in the loss landscape. Optimization techniques, are still in the process of ongoing research. Nowadays, the state of the art networks usually use even more complicated variants of optimizers, like AdaGrad, RMSProp [14], or Adam [7].

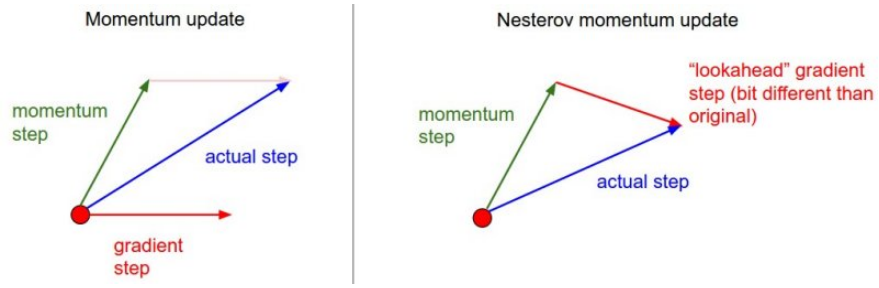



Figure 3.5: Effects of momentum and Nesterov momentum to the training step. Source [1]

### 3.4.3 Regularization

One of the biggest problems of training neural network or any machine learning algorithm is, how to make it perform with good accuracy not just on the training data, but also on new data. In other worlds, how to avoid overfitting. The strategies tackling this problem are called regularization techniques.

#### L1 and L2 regularization

One of the strategies, is to constrain the weights of the network by adding a regularization term to the loss function. The regularized loss function may be then written as follows

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_i(f(x^{(i)}, W), y^{(i)}) + \lambda R(W) \tag{3.18}$$

The $R(W)$ refers to the penalty. The penalty is weighted by hyper-parameter $\lambda$. Higher values of $\lambda$ correspond to large penalty, and values approaching 0 correspond to almost no penalty. The L2, and L1 regularization are the most common types of regularization techniques. L2 regularization is also commonly known as weight decay. The L1, and L2 regularization equations, are given by the is given by 3.19, and 3.20, respectively.

$$R(W) = \|W\|_1 = \sum_{\forall w \in W} |w| \tag{3.19}$$

$$R(W) = \|W\|_2 = \sum_{\forall w \in W} w^2 \tag{3.20}$$

#### Dropout

Dropout [13] is a computationally effective method of regularization. The key idea is to randomly drop out units with probability $p$ during the training. Consequently, dropout trains exponential number of different thinned networks. The effect could be thought of as an ensemble of all the subnetworks. The probability $p$ is set before training as a hyper-parameter. For each mini-batch, we randomly sample a binary mask with prob $p$ to apply to the different units. Multiplying the value of unit by zero has the effect of dropping them out.

#### Data Augmentation

One of the ways to make an machine learning algorithm generalize better, is to train it on the bigger dataset. However, limitations in practice may prevent us from acquiring more data. To get over this limitation data augmentation techniques have been developed. Augmentation is used to increase the amount of data by creating fake data from existing data.

Lets take an example of an image classification task. Affine transformations like translation (moving an image), scaling (increasing or decreasing the size of an image), and rotation (turning an image by some degree) are used in different combinations to generate new data. There are no limitations to the ways of transforming the original image, while it is still recognizable as an original class. We could also add noise, crop a region, or increase the contrast.

### 3.4.4 Batch normalization

Batch normalization is a method of adaptive reparametrization introduced by Ioffe and Szegefy in 2015 [6]. Training of very deep neural networks is very difficult task. It is often accompanied by instabilities, like vanishing or exploding gradient. The resulting neural networks after batch normalization, are generally more robust, and easier to train. The batch normalization is implemented by standardization of the prior layer activations. To be more concrete, let $x$ be an input to the next layer. Dimension of $x$ is $N \times D$. Size of mini-batch is $N$. The

normalized output $y$ is computed as

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j} \tag{3.21}$$

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2 \tag{3.22}$$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}} \tag{3.23}$$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j \tag{3.24}$$

The terms $\sigma$, and $\mu$, represents standard deviation, and mean, respectively. The variables $\gamma$ and $\beta$ are learn-able parameters that allow the normalized output to have any mean and standard deviation. At test time, $\sigma$ and $\mu$, are replaced by averages collected during training.

## 3.5 Convolutional Neural network

The convolutional neural networks (CNN), are in a class of neural networks with at least one convolutional layer. ConvNets are specialized in the processing of grid-like structures like images. There are many other application utilizing the CNNs. Some of them are time-series predictions (1D grid) or processing of 3D data. In this section, we will describe what convolution is and operations that are used in combination with it. We will also describe convolutional layers, and computations accompanied with them.

### 3.5.1 Convolution

The convolutional operation of functions $x(t)$ and $w(t)$, can be thought of as an sliding average. Function $x(t)$ being the function we want to average over and function $w(t)$ being the function we want to average with. This interaction is

mathematically defined by integral as,

$$s(t) = (x * w)(t) \tag{3.25}$$

$$= \int x(a)w(t-a)da. \tag{3.26}$$

If we represent function $x(t)$, as an discrete series of measurements (like cells of image grid), the already presented integral takes form of sum as

$$s(t) = (x * w)(t) = \sum x(a)w(t-a). \tag{3.27}$$

Now, lets take an example of two dimensional discrete image $I$ and kernel function $K$. The convolution of an image $I$, with kernel $K$ is defined as,

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \tag{3.28}$$

### Convolutional layers

The layers in convolutional neural networks are organized in three dimensions. Every convolutional layer has at least one filter with its, height, depth, and width. Similarly, the inputs of the layers are organized in 3D manner. As an exmaple of an input, we can think of an image with dimensionality $(W \times H \times 3)$. Last dimension three, representing channels of RGB model. The typical interaction of input with convolutional filters is illustrated in the figure 3.6.

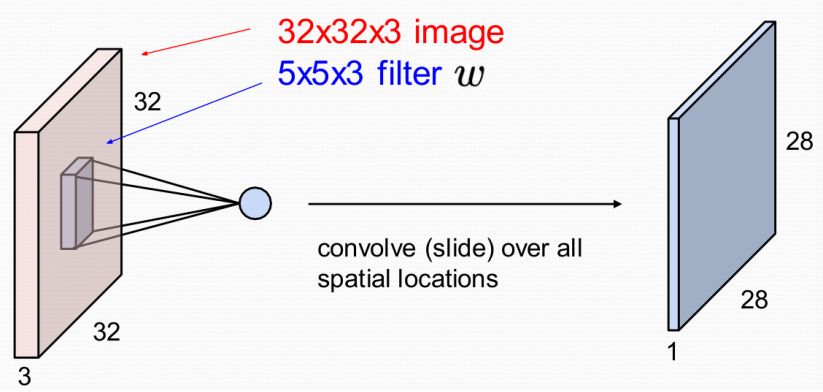



Figure 3.6: Interaction of convolutional filter with the input. Source [3].

The input of volume $(32 \times 32 \times 3)$ is convolved by filter of $(5 \times 5 \times 3)$. After the computation the volume of output is $(28 \times 28 \times 1)$. The outputs of convolutions are sometimes called activation maps. Kernel consists of $5 \cdot 5 \cdot 3 = 75$ weights.

Lets take another example of layer with 4 $(5 \times 5 \times 3)$ filters. The input dimensionality remains. The resulting volume of output computed by convolutional layer with described configuration will be $(28 \times 28 \times 4)$.

The stride refers to the hyper-parameter of kernel, that controls the step size of receptive field over the input. Consider stride of 2, input 3.29 of volume$(4 \times 4 \times 1)$, and $2 \times 2 \times 1$ kernel 3.30. The resulting output 3.31 will be of volume $(2 \times 2 \times 1)$.

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix} \tag{3.29}$$

$$\begin{pmatrix} w & x \\ y & z \end{pmatrix} \tag{3.30}$$

$$\begin{pmatrix} aw + bx + ey + fz & cw + dx + gy + hz \\ iw + jx + my + nz & kw + lx + oy + pz \end{pmatrix} \tag{3.31}$$

Even the strides of size one, are reducing the spatial dimension of input. To avoid the shrinkage, we may use zero-padding which extends the borders of the original input with zeros. However, it might be desirable to reduce the spatial dimension of input. Besides the usage of bigger strides, one might use pooling layers.

### Pooling layers

A typical CNN consists of series of convolutional layers, with non-linear activations, and pooling layers. The pooling layer operates upon each feature map separately. The result is a new set of the same number of pooled feature maps. Hyper-parameters of pooling layers are the size of pooling operation. The second hyper-parameter is stride, that functions similarly to stride in convolutional layers. The most common pooling is Max Pooling and Average Pooling. As an example, lets have an feature map 3.32. After the Max Pooling of size $(2 \times 2)$,

and stride 2, the transformed feature map gets the form of 3.33.

$$\begin{pmatrix} 1 & 5 & 2 & 1 \\ 4 & 0 & 3 & 1 \\ 6 & 1 & 9 & 0 \\ 2 & 8 & 1 & 0 \end{pmatrix} \tag{3.32}$$

$$\begin{pmatrix} 5 & 3 \\ 8 & 9 \end{pmatrix} \tag{3.33}$$

# 4. Proposed methods

# 5. Software design

# 6. Research

## 6.1 Methods

### 6.1.1 Approach 1 - Classifier

The first approach consists of an ensemble of eight convolutional neural networks (CNNs), each corresponding to a particular direction. For example, Table **??** row *180* gives us information about the size of the outputs layer. In the instance of the network concerned with direction *180*, the size of the output is 15. Overall there were tested three models for different scales of the input image. Therefore, $3 \cdot 8 = 24$ CNNs.

#### Architecture

Differences between networks with different scales were marginal. Architectures of networks with the same scale for different directions were the same. Let us consider one of twenty-four architectures in Figure 6.1. Scaled image is fed into the network through a series of three sets of Convolution/ Batch-Normalization /Pooling layers with ReLU activation function. The output of the latent space is afterward fed to two fully connected dense layers with sigmoid at the output layer.

#### Hyper-parameters

Hyper-parameters of the networks are presented in Table 6.1.

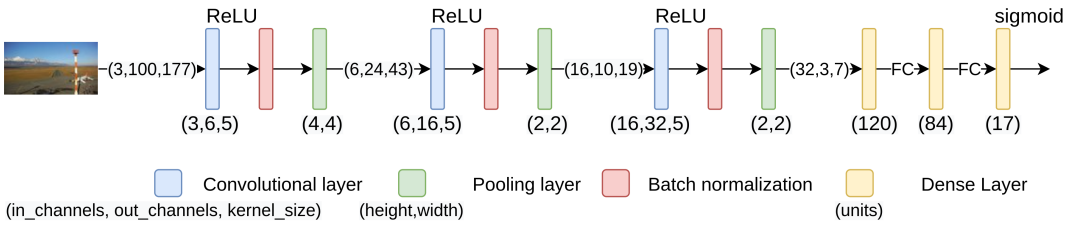



Figure 6.1: The architecture of networks trained on dataset scaled to the height of 100px.

| Scale | No of epochs | Learning rate | Batch size | Optimizer | Loss function |
|---|---|---|---|---|---|
| *100px* | 110 | 0.0001 | 32 | Adam | Mean squared error |
| *250px* | 200 | 0.00005 | | | |
| *600px* | 250 | 0.000015 | | | |

Table 6.1: Settings of hyper-parameters per scaling.

### 6.1.2 Approach 2 - Regressor

The architecture of CNN-regressor is alike to the one of the CNN-classifier. Just instead of multiple outputs with sigmoid activation function, single output with ReLU has been used. Consequently, the most significant advantage of this architecture is, that instead of multiple networks for every direction, only one is needed. However, the output of such a network does not tell much and after a few initial test with poor performance, this approach has been neglected and would not be mentioned further in text.

## 6.2 Evaluation

In the preceding section, we mentioned the importance of dataset split. Another advantage of the validation set is its importance in the training process. To avoid overfitting, it is essential to track the learning curves of the network and stop the training, after accuracy on the validation set is starting to decrease. This technique is often referred to as *early stopping*. The learning curves of different networks are shown in the Appendix.

The predictive abilities of the trained models have been tested against three metrics of choice. APA, APS, and MAE representing accuracy per *arrow*, accuracy per *situation*, and mean absolute error, respectively. Accuracy per *arrow* shows the percentage accuracy of classification per individual arrows. Likewise, accuracy per *situation*. As the name of the metric already suggests, MAE, represents mean absolute error (difference) of computed prevailing visibility, against ground truth visibility. MAE is computed in meters. Each one of these metrics can be computed as follows,

$$\begin{aligned}
\text{APA} &= \frac{\sum_i^N \text{sim}\left(y^{(i)}, t^{(i)}\right)}{\sum_i^N \dim\left(y^{(i)}\right)} \cdot 100, \\
\text{APS} &= \frac{\sum_i^N \text{eq}\left(y^{(i)}, t^{(i)}\right)}{N} \cdot 100, \\
\text{MAE} &= \frac{\sum_i^N \left|\text{dist}\left(y^{(i)}\right) - \text{dist}\left(t^{(i)}\right)\right|}{N},
\end{aligned}$$

where

$$N = \text{length of set}$$
$$y^{(i)} = \text{output} \quad \ldots \quad \text{(prediction of a model)}$$
$$t^{(i)} = \text{target} \quad \ldots \quad \text{(ground truth)}$$
$$\text{sim}(x, y) = \sum_j |x_j - y_j| \quad \ldots \quad \text{(similarity of vectors)}$$
$$\dim(x) = \text{dimensionality of vector}$$
$$\text{eq}(x, y) = \begin{cases} 1, & \text{if} \quad \text{sim}(x, y) = 0 \\ 0, & \text{otherwise} \end{cases} \quad \ldots \quad \text{(element-wise equivalence of vectors)}$$
$$\text{dist}(x) = \text{tabular numeric value representing distance assigned to vector.}$$

Calculated results per different scales are presented in Table 6.2.

| Scale | % APA | % APS | MAE |
|---|---|---|---|
| *100 px* | 94.42 | 46.81 | 2999 |
| *250 px* | 94.56 | 51.91 | 2228 |
| *600 px* | 95.03 | 54.46 | 2231 |

Table 6.2: APA - Accuracy per arrow. APS - Accuracy per situation. MAE - Mean absolute error in metres.

# 7. Results

# 8. Conclusion

# Bibliography


[1] Cs231n convolutional neural networks for visual recognition. https://cs231n.github.io/neural-networks-3/. (Accessed on 01/15/2021).

[2] Gradient descent - wikipedia. https://en.wikipedia.org/wiki/Gradient_descent. (Accessed on 01/14/2021).

[3] Neural networks for computer science. http://www.sccg.sk/~ftacnik/NSPV-2020-21.htm. (Accessed on 01/15/2021).

[4] Avrim L Blum and Ronald L Rivest. Training a 3-node neural network is np-complete. *Neural Networks*, 5(1):117–127, 1992.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[6] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[7] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[8] James L McClelland, David E Rumelhart, PDP Research Group, et al. Parallel distributed processing. *Explorations in the Microstructure of Cognition*, 2:216–271, 1986.

[9] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[10] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[11] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[12] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[13] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[14] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[15] Vladimir N Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity*, pages 11–30. Springer, 2015.

[16] Sandra Vieira, Walter HL Pinaya, and Andrea Mechelli. Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications. *Neuroscience & Biobehavioral Reviews*, 74:58–75, 2017.