UNIVERSITY OF ZAGREB
FACULTY ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS nu. 1382

Image Based Phylogenetic Classification

Vinko Kodžoman

Zagreb, svibanj 2017.

Umjesto ove stranice umetnite izvornik Vašeg rada.

Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`*.*

Thank you...

CONTENTS

1. Introduction

Since the dawn of time, people have tried to explain their surroundings. Life is all around us in many forms, and as such people have tried to categorize it by keen observation, both through its visual and genetic features. Today, it is organised into a taxonomic hierarchy of eight major taxonomic ranks. The number of known species on Earth is in the millions and climbing every year. Great numbers of species make it difficult to classify them only based on images and often requires domain knowledge. Therefore, an algorithm with the capability to classify species on the filed or from an image using only the image itself could provide great benefits for field researches.

Machine learning allows computers the ability to learn without being explicitly programmed (Samuel). It, together with an increase in available quality data (CIFAR, Imagenet, Kaggle) has yielded great results in the area of deep learning - a class of machine learning algorithms. Deep learning algorithm's accuracy scales with the amount of data used by the algorithm, that together with the improvements in hardware - mainly general purpose graphic units (GPUs) - has yielded significant performance gains in the last couple of years. One of the most rapidly advancing filed of deep learning is image recognition (Krizhevsky et al.; Simonyan i Zisserman; Szegedy et al.; He et al., a) with new neural network architectures being developed almost at a yearly basis. The performance of deep neural networks on image recognition has achieved results previously thought impossible.

In this thesis I propose a solution for a scalable classification of species from images, based on convolution neural networks and recent modern deep learning techniques.

2. Research context

To fully understand the depth of the image recognition using deep learning, we need a better understand of the underlying algorithms and methods in machine learning, as well as fundamental terms and concepts. In the next section, an introduction of basic terms is given, followed by a detailed explanation of fundamental machine learning algorithms.

2.1. Definitions and notation

2.1.1. Image representation

Matrix is a rectangular array of numbers. It is used because some numbers are naturally represented as matrices. Matrix A with m rows and n columns often written as $m \times n$ has $m * n$ elements and is denoted as $A_{m,n}$. Elements are denoted as $a_{i,j}$ where i and j correspond to the row and column number respectively, as shown in 2.1.

$$A_{m,n} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix} \tag{2.1}$$

Each image is represented as a 3 dimensional matrix. One pixel in the image represent a single element in the matrix and as images have multiple channels (RGB) each channel is a 2 dimensional matrix. Image I denoted as $I_{k,m,n}$ where $k \in [0, 2]$ represent the channel - red, green or blue - and $m, n \in [0, 255]$ represent the pixel intensities in a particular channel as 2 dimensional matrices. Figure 2.1 shows a representation of an image as a 3 dimensional matrix where each pixel is denote as $I_{k,m,n}$.
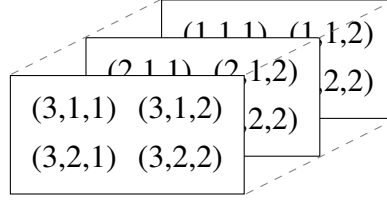
Figure 2.1: RGB image with 4 pixels represented as a 3 dimensional matrix

2.1.2. Gradient

A gradient is a generalization of the derivative in multi-variable space and as such it is represented as a vector. Like the derivative, it represents the slope of the tangent of the graph of the function. Therefore, it points in the direction of the greatest rate of increase of the function. Gradients are widely used in optimization theory as they allow the parameters to shift in a direction which will minimize or maximize a given function. In machine learning the function we want to minimize will be the loss function, which we will define in further chapters in more detail. Gradient of f is denoted as ∇f, where every component of ∇f is a partial derivative of f, denoted as $\frac{\partial f}{\partial x}\vec{e}$. Notice that gradient components are vectors denoted as $\vec{e}$. Every vector is written with an horizontal arrow above the letter - $\vec{a}$. The gradient for a n dimensional space is defined in 2.2.

$$\nabla f = \frac{\partial f}{\partial x_1}\vec{e_1} + \ldots + \frac{\partial f}{\partial x_n}\vec{e_n} \tag{2.2}$$

2.1.3. Activation functions

Machine learning models use nonlinear functions to gain more capacity - expressiveness . The most popular nonlinear functions are $sigmoid, tanh, relu$. All nonlinear functions have to have easy to compute gradients, as they are computed on parameters in order to reduce loss as explained above.

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

$$tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2}} \tag{2.4}$$

$$relu(x) = max(0, x) \tag{2.5}$$

The order of nonlinear functions is given in order of their discoveries. Today relu is used the most, since it solves the problem of vanishing gradients for very deep neural networks, this does not apply to all network types. Recurrent neural networks (RNN)

(a) $simgoid$ **(b)** $tanh$ **(c)** $relu$
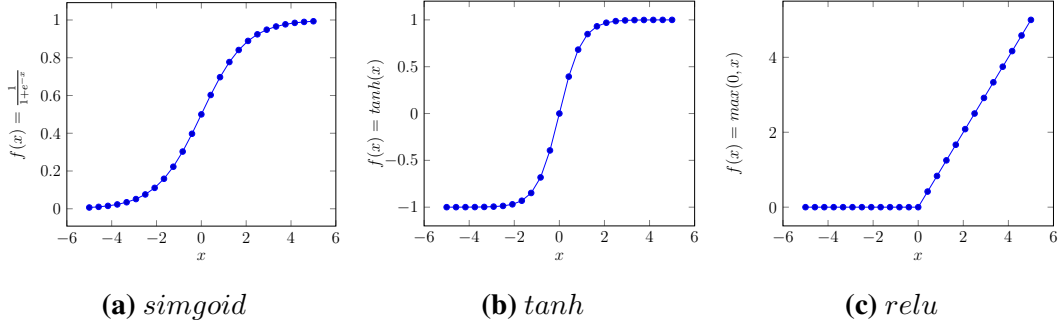
Figure 2.2: Nonliear activation functions

are a class of neural networks that often use $tanh$ as it is better suited for the particular recurrent architecture.

2.1.4. Metrics

In order to compare different models a set of metrics is employed. Accuracy which gives the accuracy of a model, it is often used on balance datasets (2.6). The problem with unbalanced datasets can be easily explained with a short example. Image having 2 classes $K = \{dog, cat\}$ and there are a total of 100 images in the dataset, of which only 2 are dogs. The model if optimized for accuracy might say the whole dataset is cats which will yield an accuracy of 98%. To solve the previous problem, additional metrics were introduced for the task of classification; precision (2.7), recall (2.8) and F1 score (2.9). Precision - positive predictive value - is defined as a fraction of retrieved instances that are relevant. Recall - sensitivity - is a fraction of relevant instances that are retrieved. In order to represent the performance of a model as a single variable F1 score was introduced, it represent a harmonic mean of recall and precision.

$$Accuracy = \frac{tp + tn}{tp + tn + fp + fn} \tag{2.6}$$

$$Precision = \frac{tp}{tp + fp} \tag{2.7}$$

$$Recall = \frac{tp}{tp + fn} \tag{2.8}$$

$$F1score = 2 * \frac{precision * recall}{precision + recall} \tag{2.9}$$

Classification results are often represented as a confusion matrix, also known as an error matrix. It is a performance visualisation of a classification model - classifier.

4

Table 2.1: Confusion matrix

	prediciton positive	**prediction** negative
actual positive	True Positive (TP)	False Positive (FP)
actual negative	False Negative (FN)	True Negative (TN)

To build the classification matrix, conditions of the experiment must be labelled as positive and negative. Using the cats and dogs example from before and marking the cats and a positive and dogs as a negative class. Doing so creates a $2x2$ matrix of actual and predicted values as shown in table 2.1.

2.1.5. Data

The input data of the machine learning algorithm is labelled as D, and it consists of X and y_t, where X is one input data point (an image in our case) and y_t is the true label of the picture - species' name. Written formally the whole input dataset is represented as $D = \{X^i, y^i\}_{i=1}^N$, where i is the i-th data point and N in the number of data points. Prediction of the algorithm is labelled as y_p.

The input data set is usually split into two datasets called the *training* a *test* dataset. The training dataset is used to optimize them models parameters while the test data is used to evaluate the model's performance. Sometimes the training dataset is split further into training and *validation* where the validation dataset is used to tune the models *hyperparameters*. Hyperparameters are parameters that do not belong to the model but non the less effect the model's performance. Depth of the neural network is a hyperparameter and will be discussed in later chapters in more detail.

2.2. Machine learning

As said in the Introduction chapter, machine learning allows computers the ability to learn. Giving data to a machine learning algorithm - model - allows it to find patterns within the dataset. The function that maps the input X to y_p is called a *hypotesis* and is denoted as h (2.10). The hypotesis $h(X; \vec{\theta})$ is parametrized with $\vec{\theta}$ - model's parameters.

$$h(X; \vec{\theta}) : X \rightarrow y \tag{2.10}$$

The model is defined as a set of hypothesises H, $h \in H$. Machine learning is the

search of the best hypothesis h from the hypothesis space H - typical optimization problem. The algorithm tries to minimize the empirical error function $E(h|D)$ - loss function. The error indicates the accuracy of the hypothesis and is called empirical because it is computed on D. Therefore, every machine learning algorithm is defined with the model (2.11), error function (2.12) and the optimization method (2.13).

$$H = \{h(X; \vec{\theta})\}_\theta \tag{2.11}$$

$$E(h|D) = \frac{1}{N} \sum_{i=1}^{N} I\{h(X^i) \neq y^i\} \tag{2.12}$$

$$\theta^* = argmin_\theta E(\theta|D) \tag{2.13}$$

Machine learning algorithms are divided into groups depending on the task, the groups are classification and regression. Each can be represented as a result of $h(X; \vec{\theta})$. Classification hypothesis takes the input X and returns a class k, example of this method would be image classification. Regression hypothesis takes the input X and returns a number, for example predicting house prices.

$$Regression \equiv h(X; \vec{\theta}) : X \to y, y \in \mathbb{R} \tag{2.14}$$

$$Classificatio \equiv h(X; \vec{\theta}) : X \to y, y \in K = \{k_0, ..., k_n\} \tag{2.15}$$

In this chapter we mentioned that machine learning algorithms are trained by minimizing the empirical loss, and in the subsection 2.1.5 we mentioned the idea of having a training and test set. The goal of the machine learning model is to have good generalization - the ability to perform well on never before seen data X. The training set X_{train} is used by the optimization algorithm to tune the model's parameters, while the test set X_{test} is used to evaluate the generalization performance of the model. If the optimization algorithm does not know when to stop it can learn the whole training set and be able to predict every value, this is called overfitting and it is to be avoided. When the model overfits, it learns to predict based on the data points in X_{train} instead of the learning the underlying patterns in X_{train}, which in turn leads to poor generalization performance. This is commonly seen in models with high capacity. On the other hand, if the model has low capacity, it will under-perform - work below its potential. That is referred to as underfitting. To goal is to find the optimal spot of the model's performance - between underfitting and overfitting. This behaver can be seen in figure 2.3.
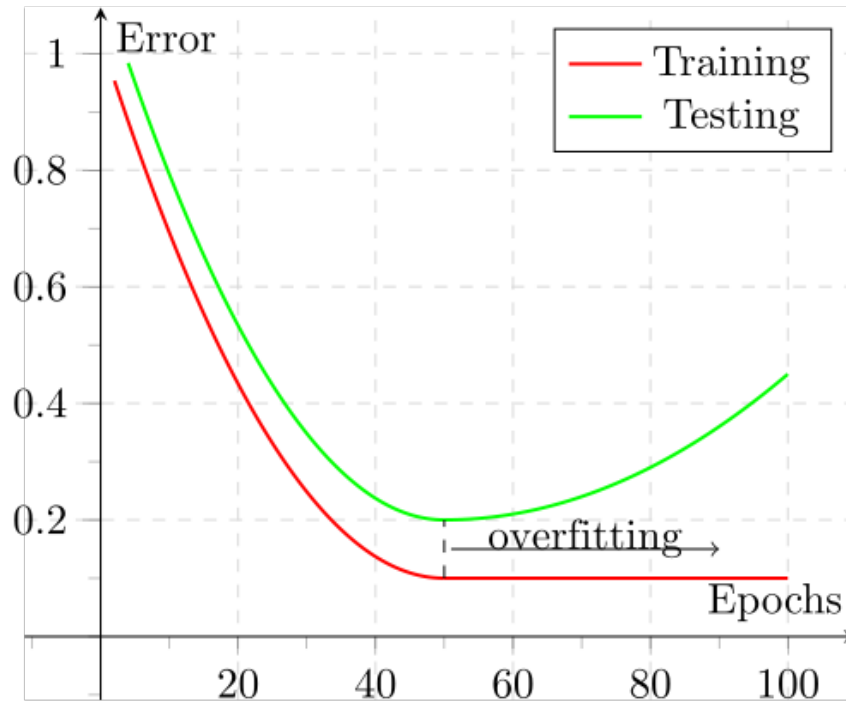
Figure 2.3: Performance of a model with time using training and test set

The x-axis shows the number of training epochs - each epoch increases the capacity of the model. We see that the training error E_{train} is falling asymptotically towards zero while the the generalization performance start to fall certain a point (50 epochs). The ideal situation for the model is to stop the training when the model starts to overfit, which is not that easy as we will see in the future chapters.

One way to combat overfitting is by early stopping during training. Another, more popular way is to use regularization. Regularization is a technique for increasing the generalization performance of the model. There is a broad set of regularization techniques used in machine learning models. Some of the most popular for artificial neural networks are called dropout (Srivastava et al.) and data augmentation, both of which will be explained in more detail in future chapters (subsections 2.3.2 and 2.3.3). Regularization techniques usually add additional components to the loss function in order to prevent the model from learning to map X_{train} to y_{train} directly.

### 2.2.1. Supervised and unsupervised learning

For the model to be train it needs data D which is defined as $\{X^i, y^i\}_{i=1}^N$. Unfortunately, y is not always available. In such cases, the model can still use only X to infer and that is called unsupervised learning. When the model uses both the data

points X and labels y it is called supervised learning. Classification and regression are both cases of supervised learning. Cluster analysis or more commonly called clustering is a type of unsupervised learning. In clustering the model tries to group a set of objects in such a way that objects in the same group are more similar based on a cretin metric. The groups are called clusters. Some algorithms need the number of clusters to be predefined, while others discern it by themselves. As the topic of this thesis is classification, in following chapters supervised learning methods will be explained in more detail.
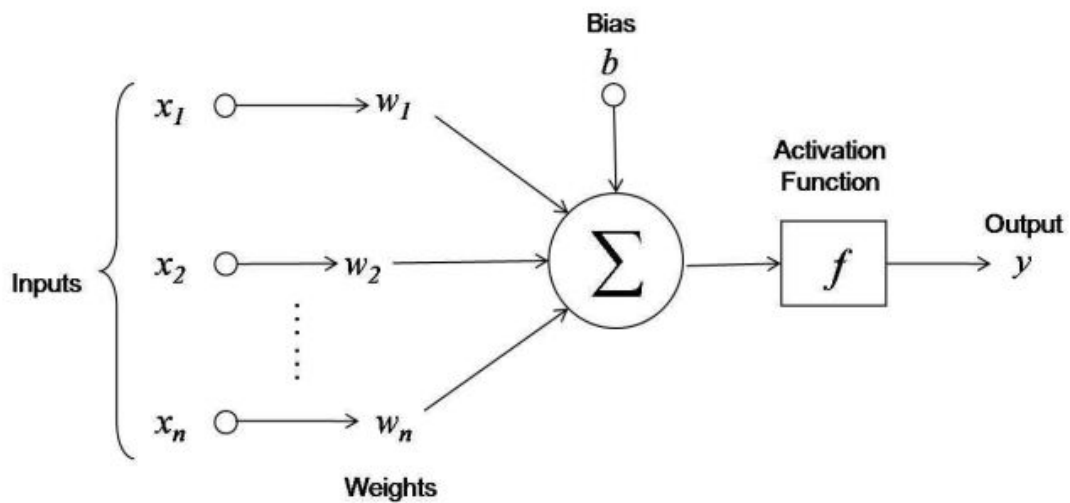
2.2.2. Artificial neural network (ANN)

One of the most interesting models in machine learning, with a diverse set of variations are artificial neural network (ANN). The computational model based on mathematics for neural networks was first introduced in 1943 by Warren McCulloch and Walter Pitts (McCulloch i Pitts) called threshold logic. Later on in 1951 a influential paper was published by S. C. Kleene on linking neural networks to finite state automata (Kleene). Artificial neural networks are based on a large collection of small computational unites called neurons (figure 2.5). The architecture was inspired by axons (2.4) from our biological brains. The idea is that even though a single neuron dose not have the capacity to express complicated levels of abstractions - neurons together can. Neurons are connect in order to transfer signals. If a neuron receives a strong enough signal it becomes activated and propagates the received signal toward his output. Just propagating the signal would not be of benefits, as the single would not transform while passing trough the network, therefore non linear activation functions, as explained in subsection 2.1.3, are added to each output of a neuron.

One of the most popular ANN architectures are feed-forward networks (FFN), sometimes also referred to as fully-connected or dense. They are called feed-forward because the data X enters at on side of the network (the input layer) and exits at the other end (output layer). Neurons are grouped into layers, and layers make an artificial neural network. The first layer is called an input layer, the last layer is called the output layer and everything in between are called hidden layers. The network is called fully-connected because each neuron in the hidden and output layers are connected with ever neuron of the previous layer as seen in figure 2.6. Each neuron consists of connections, bias and activation function. Each connection has a weight that singles the importance of the connection to the neuron. To get the net output of the neuron we multiple the input vector $\vec{x}$ with the weight vector $\vec{w}$. As the bias (commonly denoted
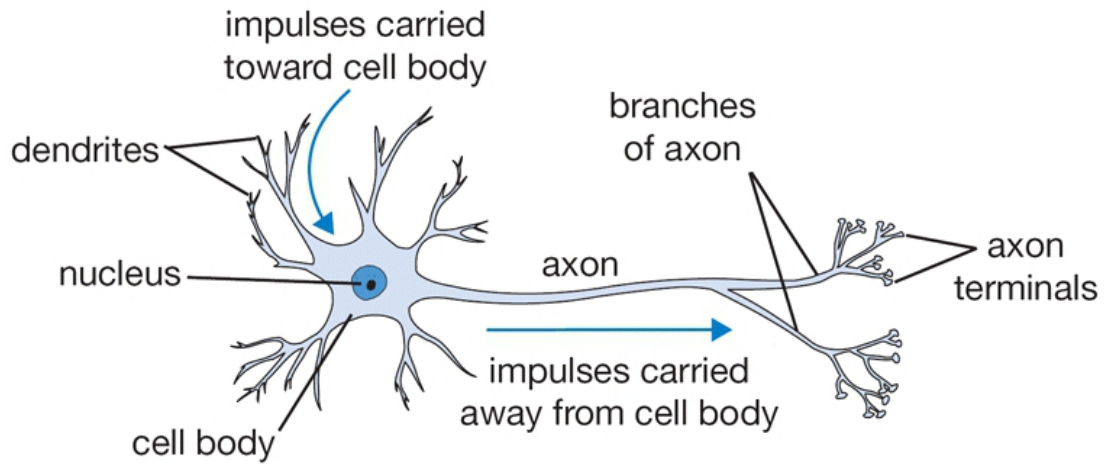
Figure 2.4: Neuron cell in a biological brain.

Figure 2.5: Artificial neuron cell.
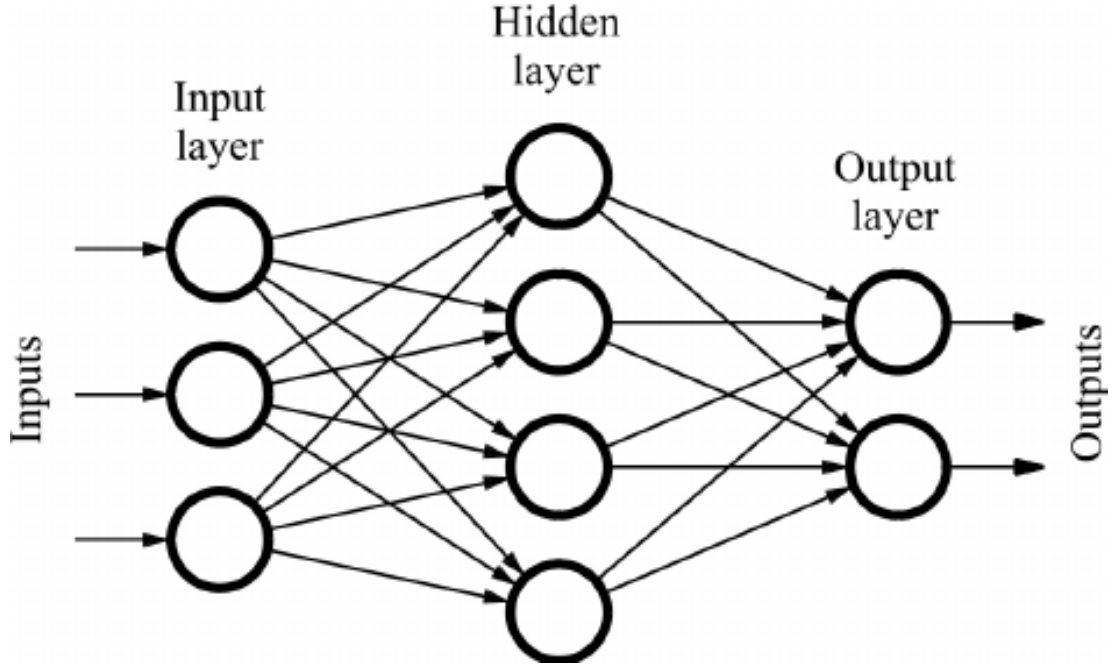
Figure 2.6: Feed-forward neural network

with w_0 or b), does not have an input of its own, we assigned x_0 to 1 so we can to a vector multiplication (2.16). To get the final output y of the neuron, the activation function f is applied to the net output, as shown in 2.17). computational model of the artificial neuron can bee seen in the figure 2.5.

$$net = 1 * w_0 + w_1 * x_1 + ... + w_n * x_n = \vec{x} * \vec{w} \tag{2.16}$$

$$y = f(net) = f(\vec{x} * \vec{w}) \tag{2.17}$$

Artificial neuron networks have the ability to adjust their capacity to the task. Adjusting the number of neurons in a single layer, the number of hidden layers or the type of activation function used in each neuron affects the capacity of the ANN. Such flexibility also leads to a broad number of architectures, and there is no clear way to determine which one suits best. It is often left to the user to test a broad variety of architectures. Looking at the figure 2.6 we can see that the number of neurons in the input layer matches the input size, this is true also for the output layer. Therefore, when designing an ANN the size of input and output layers are determined by the task and the number of hidden layers by the user.

2.3. Deep learning

Deep learning is a class - subset - of machine learning algorithms. Even though, deep learning may be connect to using "deep" neural networks - neural networks with many layers - it was created as a subset of machine learning that promised the automation of feature extraction task by using unsupervised learning. Unsupervised feature extraction and feature representation is done through a cascade of many nonlinear layers, each of which contains artificial neurons.

Visual recognition is one of the fields that is most affected by improvements in deep learning. It's difficult and time consuming to build hand crafted features for each visual recognition problem. Deep learning allows the user to use raw images without any preprocessing - making it applicable with ease to almost any task.

Deep learning algorithms consist of enormous numbers of artificial neurons (hundreds of millions), therefore training them is computational expensive. Recent breakthroughs in deep learning were driven by advances in hardware - graphics processing unit (GPUs). GPUs are made of a lot of processor unites, making them ideal for highly concurrent tasks. Neural networks are easily trained concurrently because of their composite structure and a special training algorithm called stochastic backpropagation (discussed in detail in subsection 2.3.4), commonly referred to as backprop.

2.3.1. Convolutional Neural Network (CNN / ConvNet)

Convolutional neural network is a type of feed-forward network that was inspired by the organization of animal's visual cortex. They are used in recommender systems, natural language processing and image recognition. The architecture was inspired by the works of Huble and Wiesel in 1950s and 60s (Hubel i Wiesel), their work showed that animal's visual cortex contains neurons that individually responds to small regions in the visual field. Their experiments showed that neighbourhood neurons had similar receptive fields.

The necognitron was introduced in 1980s by Kunihiko Fukushima (Fukushima i Miyake; Fukushima) and served as an inspiration for today's modern convolutional neural network architecture. The necognitron is a hierarchical multilayered artificial neuron network. It was used for handwritten character recognition, which belongs in the domain of image recognition.

Today's modern convolutional neural network architectures, such as: AlexNet, VGG, Inception and ResNet (Krizhevsky et al.; Simonyan i Zisserman; Szegedy et al.;
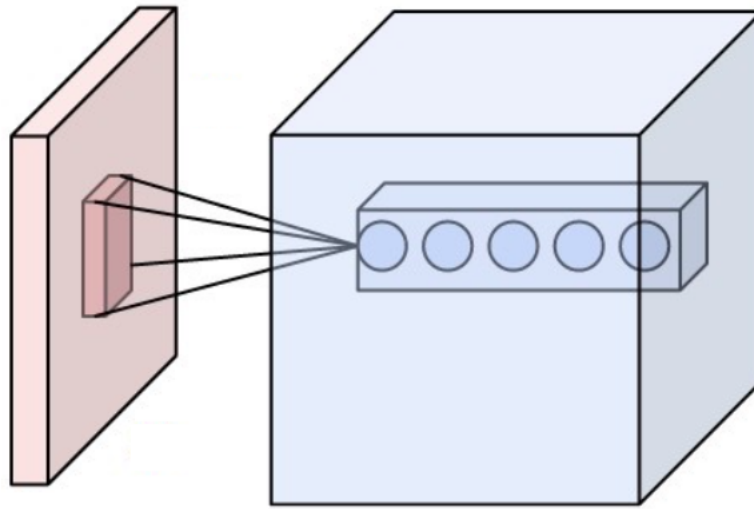
Figure 2.7: 3D structure of a convolutional filter (on the left painted in pink is the input and the blue rectangle with circle is a filter)

He et al., a) are made mostly of convolution and pooling layers. These architecture have an enormous number of parameters and are trained on powerful GPUs or clusters of servers over a period of days, sometimes even weeks.

The convolution layer is a building block of modern CNNs. The layer's parameters consist of a set of kernels (filters). Each filter has a small receptive field, but extends through the whole depth of the input. During the forward pass of the input through the network, each filter is convolved across the width and heigh of the input volume. Each time the filter moves to neighbourhood pixels it creates an output pixel (shown in figures 2.7 and 2.8). Each convolutional layer consists of multiple filters, each specialized (because of training) to detect different features in the input. Filters will detect diagonal edges, top edges, corners, etc. Each filter will use the features of the previous layer to create more abstract features. For example, a filter in the first layer might capture edges, while filters in deeper layers may use the edge filters to create corner and circle detectors (shown in detail in figure 2.9) (Simonyan i Zisserman).

The second core layer of almost every modern CNN is the pooling layer. It, in contrast to the convolution layer, does not have any parameters. Pooling layer takes the input and reduces its dimension. CNNs use max pooling, which maps the input into a reduced output by taking only the max values in the mapping (showed in figure 2.10). Use of pooling layer is a heated topic in today's CNN architectures, unfortunately hardware can't keep up with the depth of the modern architectures without the use of pooling layers.
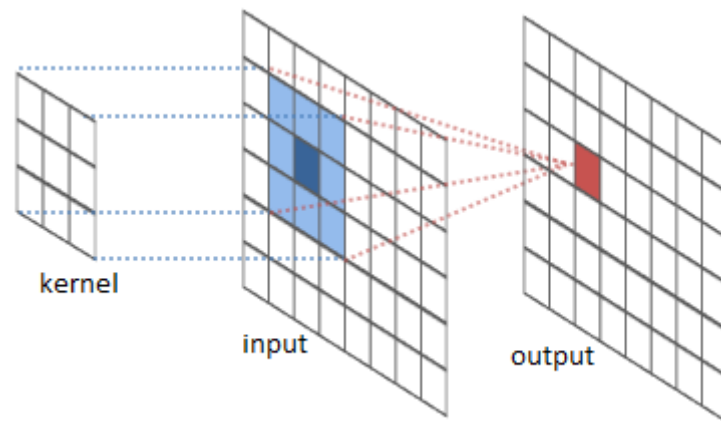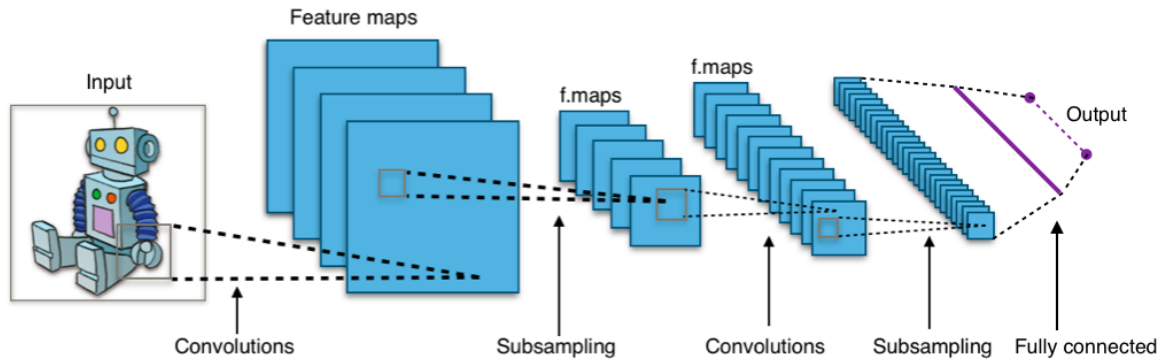
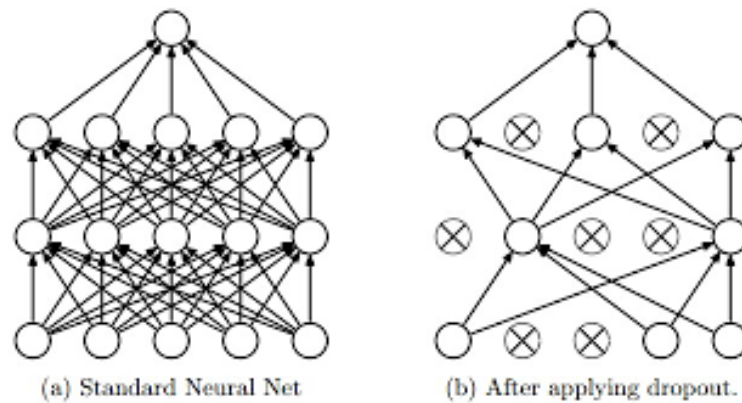**Figure 2.8:** A 3x3 kernel (filter) convolving and input to generate an output
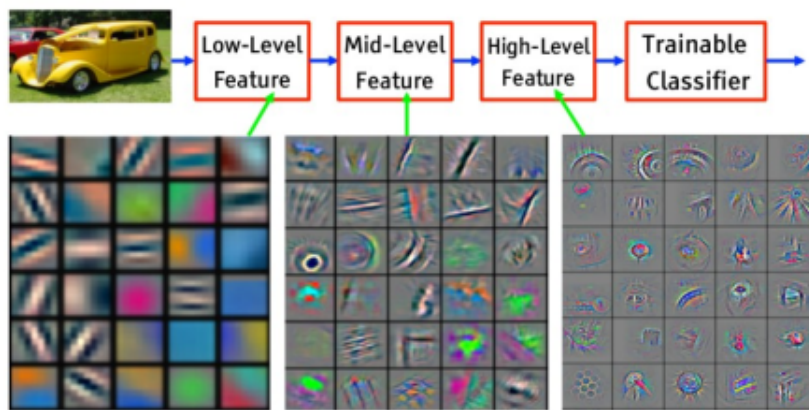
Figure 2.9: Representation of filters through the CNN
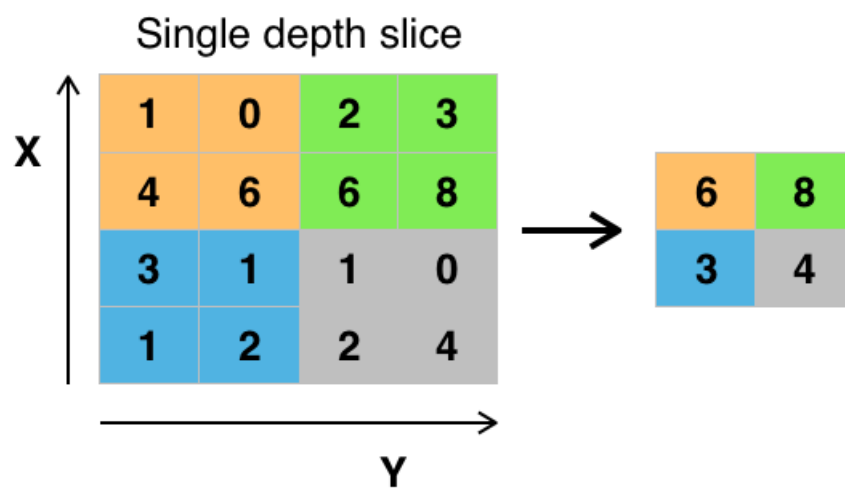
Figure 2.10: Max pooling a 4x4 input to 2x2

Figure 2.11: Typical modern CNN

(a) Standard Neural Net   (b) After applying dropout.

Figure 2.12: An example of the dropout technique

The typical structure in a CNN network will be a couple of convolution layers followed by a max pool layer. The most common dimension of convolution filters is 3x3, made popular by the huge success of VGG (Simonyan i Zisserman). The typical CNN is shown in figure 2.11.

2.3.2. Dropout

On of the most important techniques in machine learning is regularization (subsection 2.2). The modern approach to regularization is called dropout and was introduced in 2014 (Srivastava et al.). Dropout "drops" (removes) a percent of neurons on every forward pass (figure 2.12). The network cannot count that the same neurons will always be active and, as a result it increases the robustest of the model together with its generalization performance. As in every forward pass, the network has a different set of neurons, this technique can be seen as using multiple variations of the same network on the same problem.
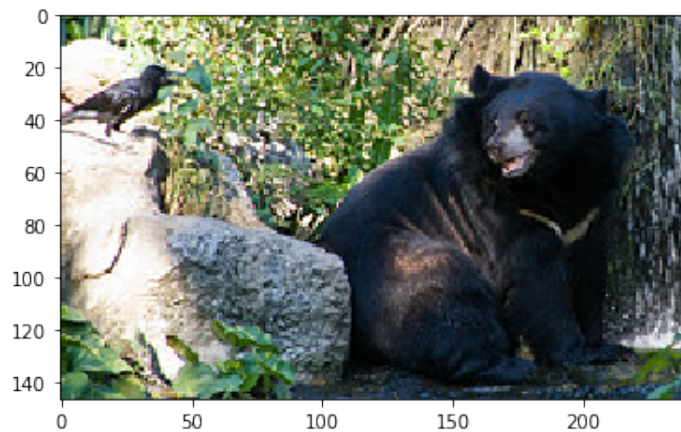
Figure 2.13: An example of the original image 224x224x3 in X_{train}

2.3.3. Data Augmentation

Another popular regularization technique is called data augmentation. The model usually trains in tens or even hundreds of epochs - a signal epoch is when the training algorithm uses every data point in X_{train} (one pass over X_{train}). Reusing the same data points can lead to overfitting. To prevent that we can augment X_{train} on every epoch. This approach is commonly called data jittering. Usually the data is augmented with rotation, mirroring, cropping, blurring (all these effects can be seen in figures 2.13 and 2.14). Both, dropout and data augmentation is used in almost every state-of-the-art CNN model today. There are some exceptions due to a new technique called batch normalization (Ioffe i Szegedy).

2.3.4. Backpropagation

To be able to train the convolution neural network on huge datasets, we need an algorithm that allows us to train the model with a subset of X_{train}. Fitting the whole X_{train} in the GPU memory is not possible for datasets of serious size. Also, the algorithm should be highly concurrent in order to use the structure of neural network to its advantage. The backward propagation of errors or more commonly called backpropagation (backprop) used together with gradient descent is the most popular method for training most neural networks.
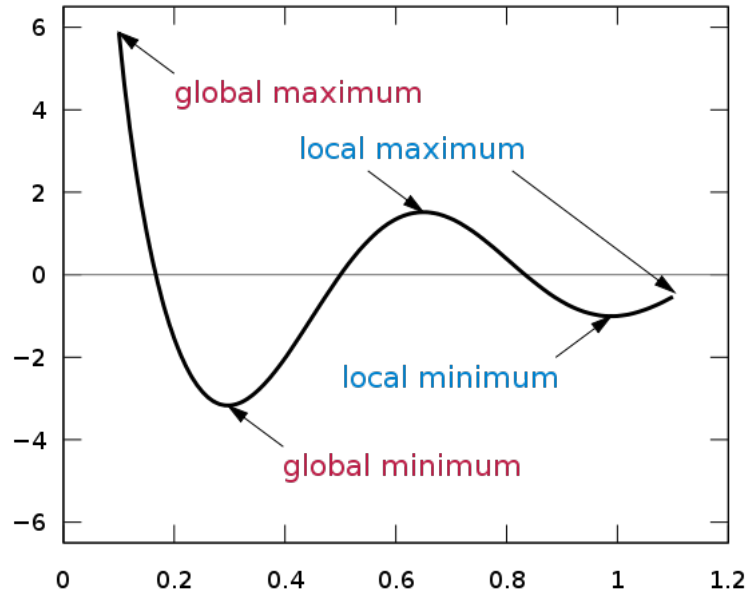
Gradient Descent

Gradient descent is an iterative optimization algorithm. It uses the gradient 2.1.2 of a function find a point on a domain that minimises or maximises the function's value.

Figure 2.14: Example of data augmented images based on the original image 2.13

Figure 2.15: An example of local and global maxima and minima in a function

Gradient descent does not guarantee to find the global maxima or minima (figure 2.16). There have been several variations of the gradient descent algorithm that deal with the algorithm being stuck in local maxima, the most famous and most widely used at this moment is Adam (Kingma i Ba).

$$x_{n+1} = x_n - \eta * \nabla f(x_n) \tag{2.18}$$

Gradient descent uses the gradient to move in a way that minimises or maximises the given function (2.18). The amount that is used to move the data point - called the step (η) - is crucial for the convergence of the algorithm. If the step is too big the algorithm might diverge, while if the step is too small the algorithm might take too long to converge. Gradient descent, like many other optimization algorithms needs to have its parameters properly tuned.

Stochastic Gradient Descent

The main disadvantage of the gradient descent (2.18) is the need to calculate the gradient on all the dataset for the update state. The gradient has to be calculate on the whole X_{train}. A huge improvement on the basic gradient descent is a the stochastic gradient descent. It updates the gradient by sampling the dataset in a minibatch, usually the size 64, 128 or 256 data points (depending on GPU memory limitations). After every epoch the the whole training dataset X_{train} is shuffled so that different
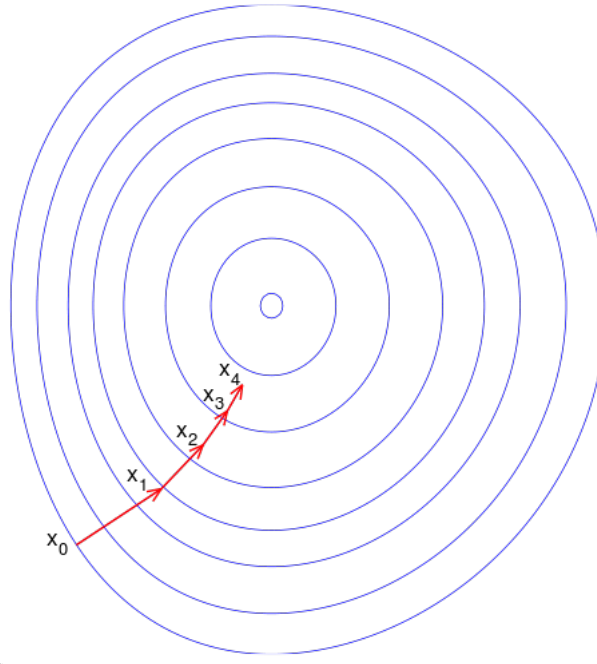
Figure 2.16: An example of gradient descent converging to global minima in a 2D space (2 variable function)

minibatches will be sampled. The rule of thumb is if you can increase the size of minibatch - do it. The more data points are in the minibatch, the more precise is the gradient approximation, also the model trains faster.

Optimization Target

The goal of the gradient descent algorithm is to minimise or maximise a function. In deep learning, when training the model, we want to minimise the loss function $E(h|D)$ by computing its gradient. In classification (2.15) the goal is to predict for the predicted class y_p to match the true class y_t. Neural networks for classification usually have a softmax layer as their last layer. Softmax (2.19) is a normalization exponential function, which transforms neural network's outputs to probabilities for every class. The sum of the softmax values for all classes is equal to 1.

$$softmax(\vec{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e_k^z} \quad \text{for} \quad j = 1, ..., K \tag{2.19}$$

For the task of classification, the softmax function is the final layer - the output. Using this information allows us to compute the loss function. The loss of a soft-

max function is called cross-entropy loss and is computed with negative log likelihood (2.20).

$$L(y_t, y_p) = -\sum_{d=1}^{D} \sum_{k=1}^{K} y_t^i * log(y_p^i) \tag{2.20}$$

To use the cross-entropy loss in the backpropagation we have to calculate its gradient. The gradient for cross-entropy loss is shown in 2.21).

$$\nabla L = \vec{y_t} - \vec{y_p} \tag{2.21}$$

2.3.5. Vanishing Gradient

On of the problems in using backpropagation is the propagation of loss gradients in the neural networks. As the neural network is composed of smaller unites called neurons, its output is also a function composed of linear combinations and non-linear activation functions of previous layers. The backpropagation algorithm calculates the gradient losses from the output to the front layers of the neural network - hence the name backpropagation. The deeper the network, the more information is lost duo to gradient decay - low gradient values. The vanished gradient means that the network is not efficiently training and leads to poor performance. One way to combat that is to use special activation functions (Clevert et al.; Xu et al.; He et al., b) or special architectures (He et al., a).

2.3.6. Batch Normalization

Normalizing input data before it enters the machine learning algorithm usually increases the performance of the algorithm. It is easier for the model to learn that all of its inputs will be in a certain range. Input normalization in neural networks will not yield significant performance improvements because the input for images are normalized by nature in the range of 1 to 255 RGB (2.1.5). The second problem is the modularity of neural networks. Each layer feeds its outputs into the inputs of the next layer - making any input normalization futile. To combat this a new solution in a form of normalization layer has been introduced in 2015 called batch normalization (Ioffe i Szegedy). It normalizes the data for the whole batch that is used in the forward pass of the backpropagation algorithm (2.3.4). As the authors have showed, using batch normalization improves generalization, in some cases removes the need for dropout

(2.3.2) and makes the neural network more robust to initial hyperparameter initialization.

3. TaxNet

To make a scalable classifier that works with specific hardware requirements I had to make a classifier that yields the best performance and can be trained on a GPU with 2GB of VRAM in a reasonable time frame. In this chapter I introduce my solution - the TaxNet.

3.1. Implementation

4. Dataset

4.0.1.   ImageNet

5. Results

Graphs graphs graphs...

6. Conclusion

Zaključak.

BIBLIOGRAPHY

Djork-Arné Clevert, Thomas Unterthiner, i Sepp Hochreiter. Fast and accurate deep
network learning by exponential linear units (elus). URL `https://arxiv.org/abs/1511.07289`.

Kunihiko Fukushima. A hierarchical neural network model for selective attention.
U Rolf Eckmiller i Christoph v.d. Malsburg, urednici, *Neural Computers*, stranice
81–90. Springer Berlin Heidelberg. ISBN 978-3-540-50892-2 978-3-642-83740-1.
URL `http://link.springer.com/10.1007/978-3-642-83740-1_10`. DOI: 10.1007/978-3-642-83740-1_10.

Kunihiko Fukushima i Sei Miyake. Neocognitron: A self-organizing neural net-
work model for a mechanism of visual pattern recognition. U *Competition and
cooperation in neural nets*, stranice 267–285. Springer. URL `http://link.springer.com/chapter/10.1007/978-3-642-46466-9_18`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, i Jian Sun. Deep residual learning
for image recognition. U *Proceedings of the IEEE Conference on Com-
puter Vision and Pattern Recognition*, stranice 770–778, a. URL `http://www.cv-foundation.org/openaccess/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, i Jian Sun. Delving deep into rectifiers:
Surpassing human-level performance on imagenet classification. U *Proceedings of
the IEEE international conference on computer vision*, stranice 1026–1034, b. URL
`http://www.cv-foundation.org/openaccess/content_iccv_2015/html/He_Delving_Deep_into_ICCV_2015_paper.html`.

D. H. Hubel i T. N. Wiesel. Receptive fields and functional architecture of monkey
striate cortex. 195(1):215–243. ISSN 0022-3751. URL `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1557912/`.

Sergey Ioffe i Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. URL `https://arxiv.org/abs/1502.03167`.

Diederik Kingma i Jimmy Ba. Adam: A method for stochastic optimization. URL `https://arxiv.org/abs/1412.6980`.

Stephen Cole Kleene. Representation of events in nerve nets and finite automata. URL `http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA596138`.

Alex Krizhevsky, Ilya Sutskever, i Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. U *Advances in neural information processing systems*, stranice 1097–1105. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-network`

Warren S. McCulloch i Walter Pitts. A logical calculus of the ideas immanent in nervous activity. 5(4):115–133. URL `http://link.springer.com/article/10.1007/bf02478259`.

A. L. Samuel. Some studies in machine learning using the game of checkers. 3(3): 210–229. ISSN 0018-8646. doi: 10.1147/rd.33.0210.

Karen Simonyan i Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. URL `https://arxiv.org/abs/1409.1556`.

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, i Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. 15(1):1929–1958. URL `http://www.jmlr.org/papers/volume15/srivastava14a.old/source/srivastava14a.pdf`.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, i Andrew Rabinovich. Going deeper with convolutions. U *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, stranice 1–9. URL `http://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Szegedy_Going_Deeper_With_2015_CVPR_paper.html`.

Bing Xu, Naiyan Wang, Tianqi Chen, i Mu Li. Empirical evaluation of rectified activations in convolutional network. URL `https://arxiv.org/abs/1505.00853`.

Image Based Phylogenetic Classification

Sažetak

Sažetak na hrvatskom jeziku.

Title

Abstract

Abstract.