Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

Debugging deep neural networks

Or: Are final grades predictable from resume photos?

Project Thesis

Lino von Burg

December 22, 2017

Supervisors: Dr. T. Stadelmann, Prof. Dr. O. Stern

Institute of Applied Information Technology, ZHAW School of Engineering

Abstract

This project thesis argues that interpretability of Artificial Neural Networks (ANN) will be key to its adoption, especially in high-stakes applications where human lives or livelihoods are at risk. It then provides a bird's eye view at the fundamentals of ANNs and one of its specialisations, Convolutional Neural Networks (CNN). To dispell the notion of ANNs as complete black boxes, a review of current visualisation methods for CNNs is done. Finally a pre-processing pipeline for an image classification and visualisation system to answer the question "Are final grades predictable from resume photos?"

Contents

Chapter 1

Introduction

1.1 Motivation

Artificial Intelligence (AI) is finding its way into more and more everyday processes, although so far it is mostly hidden from sight. During the last decade, huge advances were made in the field. Machine translation is now, at least for some languages, nearly reaching human performance. Voice recognition and speech to text engines have improved to the point that most modern smartphones contain a digital, voice activated and controlled assistant. Even though technology developed in the field of AI has already been built into many different consumer devices, most of the AI applications that are easily visible and, crucially, recognisable as such in everyday life are perceived as little more than toys so far, but that will likely change in the very near future. The most prominent example of this are arguably autonomous vehicles, which can already be seen on the roads of several countries worldwide. In regards to consequences of errors, autonomous vehicles are on a completely different level than assistants like Apple's Siri. Other high-stakes applications of AI include automated stock trading, surgery, power grid management and even weaponry. All of these applications have the potential to do considerable harm. Since even small errors can lead to catastrophic outcomes, the engineers and scientists building them need to ensure their robustness.

Even models used to solve comparatively simple image recognition tasks involve hundreds of thousands of changing parameters. Systems involved in the aforementioned high-stakes applications are many orders of magnitude more complex than that. This means that it is practically impossible for humans to conceptualise in detail what is happening inside of those models.

Without understanding exactly what is happening, we can only rely on empirical validation to show that they work correctly. This approach will always be limited by the available training and test data. Even if a model

reaches a certain precision in performing its task on a test set, the same precision cannot be guaranteed to apply to real world data. It also means that we are restricted to trial and error for any improvement attempts.

It is therefore crucial for the future of AI to design methods to increase the interpretability of such models.

1.2 Contributions

The current literature on visualisation of Artificial Neural Networks (ANN), with a focus on visualisation of image classification tasks, will be reviewed and the identified approaches will be categorised.

A preprocessing pipeline will be created to prepare a system to answer the question "Are final grades predictable from resume photos?" Additionally, challenges to this task and possible countermeasures will be discussed.

1.3 Organisation Of Thesis

The second chapter will first give a brief introduction to Artificial Neural Networks (ANN) and then explain how Convolutional Neural Networks (CNN) applied to image classification work. The third chapter will explore the current state of the art in visualising the inner workings of CNNs. The literature on this topic will be reviewed and methods sorted into groups. The fourth chapter will describe a preprocessing pipeline for a system to predict grades of students from their portrait photos and discuss several challenges to this task, proposing possible countermeasures. The fourth chapter will summarise the previous findings and give an outlook of possible future work to build upon them.

Chapter 2

Fundamentals

ANNs are computational models inspired by biological neural networks as in human brains [1]. They are currently a main focus of research in the discipline of Machine Learning. The next section (**??**NN) is to a large degree paraphrased from section 18.7 of [1] since it provides an explanation that is simple enough to understand without leaving out too many details.

The section about CNNs is inspired by [2].

2.1 Overview Artificial Neural Networks

2.1.1 Neuron

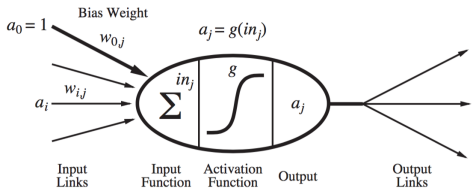

Figure 2.1: Diagram of artifical neuron or node in a NN with labelled components. Reprinted from [1]

Similar to neurons in brains, the basic unit of an ANN is an artifical neuron, also called *node* or *unit*. A neuron consists of several parts as shown in Fig. 2.1. It receives a set of inputs, which are modified by a set of weights. The weighted inputs are then passed into an activation function which computes the output.

ANNs are composed of layers of neurons where the output of neurons in one layer are connected to the inputs of neurons in the next layer via a directed link. Layers where every neuron of one layer has a link to every

neuron in the next layer are called *fully connected*. These links propagate the activation of a neuron to connected neurons in the next layer. Each of these links has an associated weight.

This way, data input to the network flows through each layer, finally emerging from the last layer. This structure, where data only flows from input to output, is called a feed-forward network.

It is also possible to make connections from higher layers back to lower layers. Networks with such backward connections are called Recurrent Neural Networks (RNN). Since this thesis is only concerned with feed-forward networks, RNNs will not be expanded upon.

Figure 2.2: (a) A network with two input and two output neurons. (b) A network with two input neurons, two hidden neurons and two output neurons. Reprinted from [1]

2.1.2 Single-layer Networks

A single-layer network consists of a layer of input neurons connected to a layer of neurons, with the output of the network corresponding directly to the output of the neurons (Fig. 2.2, (a)). The input layer performs no computation and just passes input it receives on to the next layer unchanged. In practice, this means each neuron in the output layer is a separate network, since its weights (which are essential to the learning process) only affect its own output.

2.1.3 Multi-layer Networks

A multi-layer network is a layer of input neurons to which more than one layer of neurons are connected (Fig. 2.2, (b)). Any layer between the input and the last layer of neurons (output layer) is called a hidden layer, since they are not exposed to the outside.

Training of a multi-layer network is achieved by backpropagation of Errors from the output layer. Before training, all weights in the network are initialized with random values. For every input fed to the network, the output is compared with the desired output. The error is then propagated back through the network and weights are adjusted accordingly. This is repeated a fixed number of times or until the output error reaches a predetermined threshold.

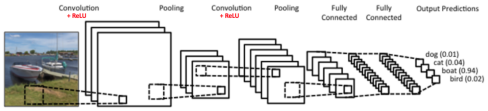

Figure 2.3: A simple CNN. Reprinted from [2]

2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are ANNs with a specific architecture. In recent years they have emerged as one of the most effective architectures for pattern recognition/feature extraction tasks. This thesis focuses on CNNs in the context of computer vision, specifically image classification, so the CNN will be explained in this context. The input of a CNN for image processing has 3 dimensions: height, width and depth (corresponding to the colour channels of the image). CNN are generally composed of four main operations:

- Convolution
- Non-Linearity (ReLU)
- Pooling
- Classification (Fully Connected Layers)

Convolution, ReLU and pooling are usually combined to form a higher level building block. This block of layers can be repeated multiple times before a number of fully connected layers is attached, with a final output layer with the same number of outputs as expected classes.

2.2.1 Convolution

The convolution operation is a matrix operation, computing element wise multiplication between two matrices and then adding the results to form a single integer. A convolution layer extracts features from the input by sliding a square matrix (called a filter) over the input image, performing the convolution operation for each of its pixels.

To allow the outermost pixels of the image to be convolved, the image can be padded with zeros on all sides. The result of applying a filter to the whole input is a new matrix with the results of each convolution. This is called a feature map. Usually a convolution layer consists of more than one filter, each producing a feature map. The output of a convolution layer has the same height and width as the input (if zero padding is used) and its depth corresponds to the number of filters used.

2.2.2 ReLU

To introduce non-linearity into the CNN, a Rectified Linear Unit (ReLU) operation is performed after every convolutional layer. This basically replaces every negative pixel value in the feature maps by zero. The ReLU operation is necessary because real-world data is usually non-linear and convolution is a linear operation. Instead of ReLU, other non-linear functions like tanh or sigmoid can be used, but ReLU has shown the best performance for most situations.

2.2.3 Pooling

Spatial Pooling aggregates the values of several input pixels to a single output pixel, thereby reducing the size of each feature map. A feature map of size 4x4 can be reduced to an output of size 2x2. For this, a window of size 2x2 is moved over the input, considering every pixel only once. The pixels inside the window are aggregated, usually by selecting the maximum value. This is called Max Pooling.

The pooling step:

- reduces the size of the input representation, making it more manageable
- reduces the number of parameters and thereby computations, controlling overfitting and increasing performance
- increases resistance to small transformations, distortions and translations
- renders the input representation almost scale invariant, meaning the object to detect can be located anywhere in the input image

These three layer types differ from traditional multi-layer networks, as not every neuron of a layer is connected to each neuron of the next layer.

2.2.4 Fully Connected Layers

Convolution, ReLU and Pooling layers together perform feature extraction, learning the most effective features for a given dataset. Every successive block of convolution, ReLU and pooling learns features on a higher level than the last, from edges up to complete objects. The highest level features (the output of the last convolution block) are then passed to a group of fully connected layers, where every neuron of each layer is connected to every neuron in the next layer. The last fully connected layer finally corresponds directly to the class predictions, with each neurons output representing the probability of one class.

2.2.5 Training

As with other ANNs, CNNs are trained using backpropagation, learning both what features to extract and which classes the features or combinations of features correspond to. Since feature extraction is mostly independent of the classification task, it is possible to freeze its parameters and only train a classifier that uses the already learned features. This can significantly reduce the required size of the dataset needed for training, as long as the images the feature extraction was trained with are similar to then new dataset.

Chapter 3

Literature Survey – Visualisation and Understanding of NN

There are several distinct ways to visualise what is happening inside of a neural network. The most obvious way is to graphically represent the structure of the network. This can be used independently of the task, which also means that it is difficult to gain insight into the processes involved in specific tasks. The next section provides a short introduction to network visualisation with TensorFlow. After that, this thesis will focus on methods for visualising image recognition and classification tasks, specifically for feed forward CNNs. [3] lists different analysis techniques. Two of those are concerned with visualisation: feature map reconstructions and input-feature based model explanations. These correspond to what [4] calls feature visualisation and attribution, respectively. Theses terms will be used in the remainder of this thesis.

3.1 Network Visualisation

Depending on the technology used, there are several tools available for this task. For TensorFlow, the main tool available is TensorBoard.

A short overview of TensorBoard functionality is included, for in-depth explanation and code samples TensorFlow provides an excellent tutorial in its documentation [5] and on the github repository of TensorBoard [6].

This shows a representation of the network as a graph of connected tensors (generally representing layers of the network as matrices of neurons that form the layer) and their parameters. It also provides other tools to help understand and debug TensorFlow programs, including dashboards for scalar values, plots of metrics and images that pass through the network. There are also various add ons to enhance and add functionality. TensorBoard works by reading the events file of TensorFlow runs. To create these event files,

the TensorFlow program needs to define data to be exposed to TensorBoard by adding summary operations to the TensorFlow code. A FileWriter then dumps these values and optionally the graph of a session to a event file. [6]

3.1.1 Graph Visualisation

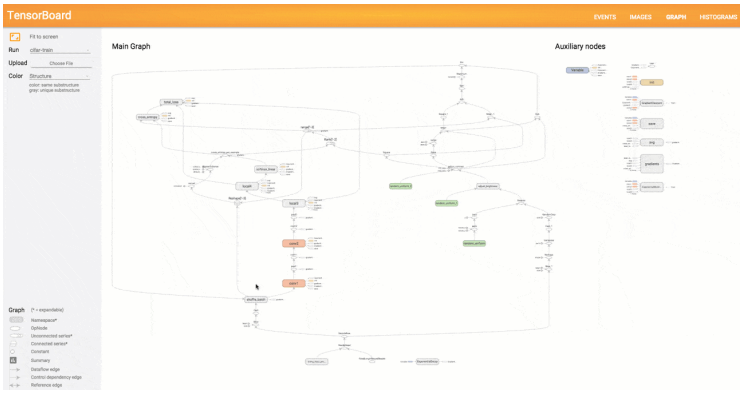

Figure 3.1: TensorBoard graph visualisation. Reprinted from [5]

TensorBoard graph visualisation is a powerful tool to inspect the computation graph of any TensorFlow program (see Fig. 3.1.1. Since TensorFlow graphs typically have thousands of nodes, it is advisable to group the nodes together. TensorFlow provides name scopes which TensorBoard uses to define a hierarchy of nodes of which only the top level is shown by default. To see more details about top level nodes, they can be expanded, revealing all contained nodes of the next level.

3.1.2 Scalar Dashboard

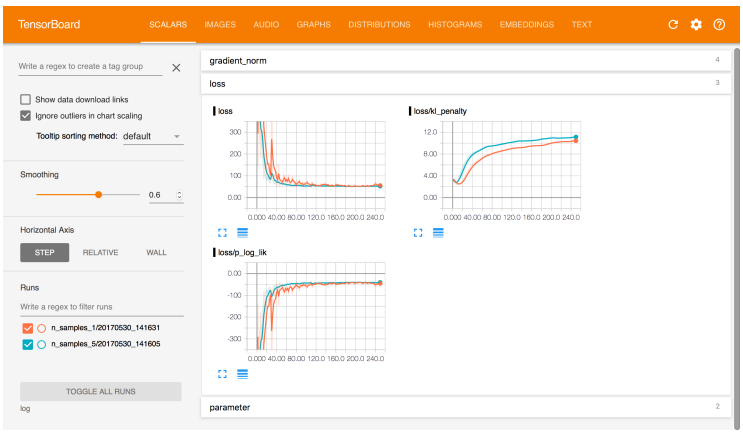

Figure 3.2: TensorBoard scalar dashboard. Reprinted from [7]

In the same way, summary operations can be defined to track scalar values like the model's loss or learning rate. These values are then plotted as graphs to show their change over time and runs.

3.2 Feature Visualisation

The feature extraction part of a CNN is composed of various layers, these layers each contain multiple feature detectors or filters. These filters are learned by the network, not engineered. To understand how those filters look and what kind or part of an input image activates a given filter, it is immensely helpful if they are visualised in some way. Understanding features learned by CNNs can be approached from two perspectives: visualisation of the response of the CNN to a specific input image and visualisation of the notion of a unit in the CNN[8]. [9] defines three different type of visualisations: inversion, activation maximisation and caricaturisation. Inversion visualises the response to a specific input, activation maximisation and caricaturisation visualise the "meaning" of a component and combinations of components respectively. This section explores several different approaches for each of these types.

The representation of the response to a specific image is created through different backpropagation based approaches. Projecting the filter back to the input pixel space from a feature map for a given image yields visual representations of the filters that created the map. While it is possible to project feature activations from the first convolutional layer back to the input pixel space, the non-invertible nature of pooling operations requires a different approach for later layers.

The notion a network has of a class or unit is visualised by finding an input image that maximises activation. Another method is to optimise an input image (or choose the optimal of multiple input images) for maximum activation of different CNN components (neurons, channels, layers, class logits/probabilities) and combinations of such components.

3.2.1 Inversion

Deconvolutional Networks (deconvnet)

[10] introduces a method to "map [feature activities in intermediate layers] back to the input pixel space" by attaching a deconvnet to each layer of the CNN Figure 3.3. This is one of the earliest approaches, it will be explained more in-depth than the rest. Most later approaches introduce improvements but are conceptually similar, so this will provide a good starting point.

First, an input image is fed into the CNN and features computed through the layers. To examine a given activation, all other activations in the layer

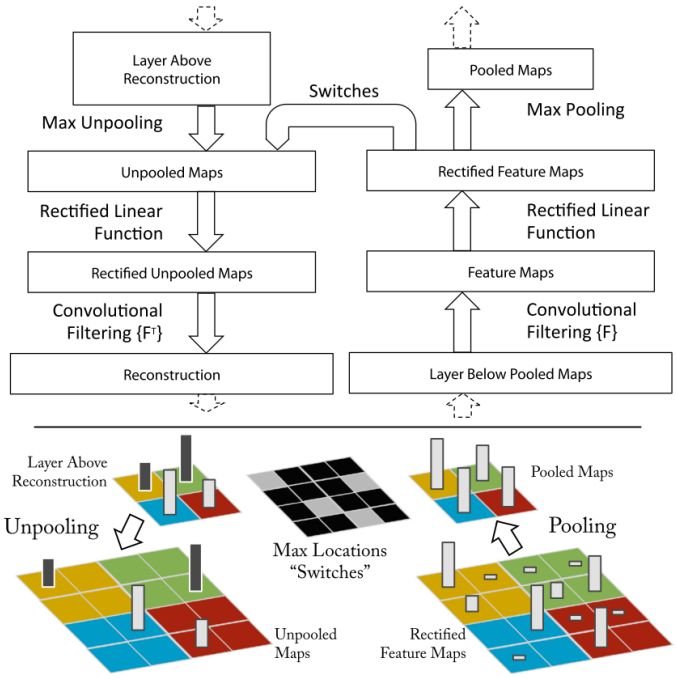

Figure 3.3: Top: A deconvnet layer (left) attached to a CNN layer (right). Bottom: An illustration of the unpooling operation in the deconvnet, using switches which record the location of the local max in each pooling region (colored zones) during pooling in the convnet. The black/white bars are negative/positive activations within the feature map. Reprinted from [10]

are set to zero and the feature maps are passed to the attached deconvnet layer. A deconvnet is essentially a CNN with its components reversed: the input is unpooled, rectified and filtered. This is repeated for every layer until input pixel space is reached.

Because the pooling operation is non-invertible, the unpooling operation needs a way to place the reconstructions of the layer above at appropriate locations to preserve the structure of the activation. A set of switch variables is used to store the location of the maxima in each pooling region during max pooling operations. This allows the deconvnet to create an approximate inverse of the pooling operation.

The unpooled maps are then rectified by passing them through a rectified linear function (ReLU) to ensure they are positive.

The filters of the deconvnet are transposed versions of the learned filters of the CNN. When applied to the rectified maps, this produces an approximate inverse of the original convolution.

Since the switch variables used in the unpooling step are specific to a given input image, this process produces a reconstruction resembling the input

Figure 3.4: Visualisation of features in a fully trained model alongside corresponding image patches. Images shown are not part of the model but patterns from the validation set causing high activation. Reprinted from [10]

image where structures are weighted according to their contribution to the activation.

Gradient Based

[11] introduce a gradient based approach, generalising on the approach of [10]. This technique can be applied to any layer of the network, not only convolutional layers. The two approaches are compared in detail in [12].

All-Convolutional Network and Guided Backpropagation

[13] introduces a CNN that replaces max pooling layers with convolutional layers, removing the main impediment for backprojection of representations learned by higher layers of a CNN. This allows for deconvolution without a previous forward pass, since no switch variables are needed. In higher layers, the deconvolution approach does not produce sharp, recognizable images. To remedy this, [13] proposes a combination of [10] and [11], called guided backpropagation, that significantly increases resulting image quality.

Guided Gradient-weighted Class Activation Mapping (Grad-CAM)

[14] introduces Grad-CAM, a generalisation to Class Activation Mapping (CAM) [15]. CAM/Grad-CAM is a localisation/attribution technique section 3.3. This technique is combined with guided backpropagation to remove parts of the activation map that do not contribute to the final classification.

DeepLIFT

[16] introduces DeepLIFT, an approach that compares the difference of a given output and a reference output with the difference of the inputs and reference inputs. Choosing a suitable reference to compare against is critical and requires domain-specific knowledge.

Forward-Backward Scheme

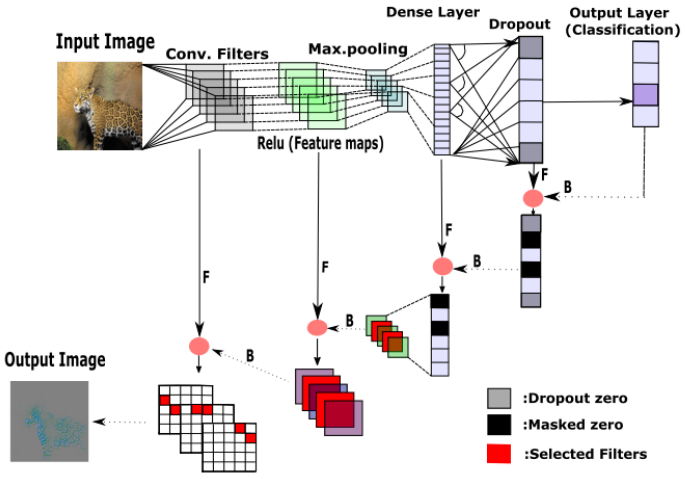

Figure 3.5: Visualisation of features in a fully trained model. Reprinted from [17]

[17] introduces another alternative to the deconvnet back-propagation of [10], replacing it with a forward-backward scheme. It leverages both the backward information flow during backpropagation from output layers to input and the forward information flow during feature extraction. This approach is computationally more efficient than [10] and does not have the potential numerical stability and robustness issues of gradient based schemes like [13] and [14]. It also removes the need for a reference image as in [16].

Optimisation

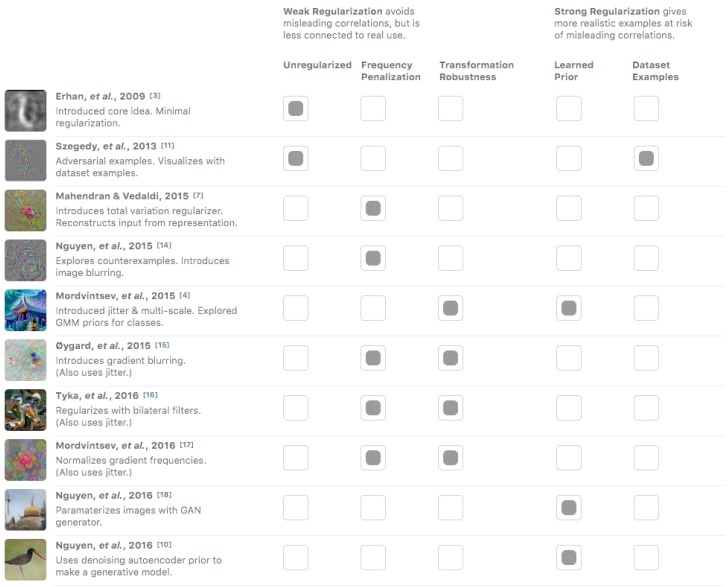

Figure 3.6: Overview of activation maximisation approaches grouped by regularisation family. Reprinted from [4]

[4] provides an excellent overview of the current state of optimisation approaches (Fig. **??**). They explain that synthesising an example input via optimisation is better at explaining what the CNN is looking for than identifying examples from the dataset, because "it separates the things causing behavior from things that merely correlate with the causes" (Fig. 3.8) They note that optimisation provides high flexibility, since it can synthesise an example for any component or combination of components of the CNN. They also note that there are significant challenges to this approach: achieving diversity of examples, understanding the interaction of neurons and preventing high-frequency artefacts.

Diversity can be achieved by adding a "diversity term" that pushes multiple examples to diverge. This leads to diverse feature visualisations that, when inspected alongside dataset examples, allow to check prediction about what

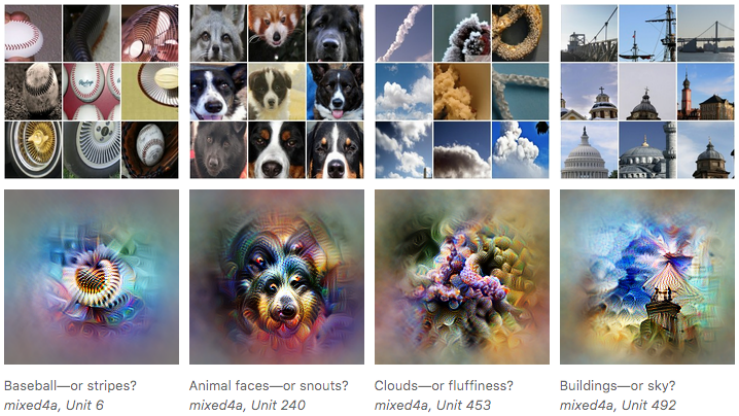

Figure 3.7: Top row: Dataset examples show what neurons respond to in practice. Bottom row: Optimisation isolates the causes of behavior from mere correlations. Reprinted from [4]

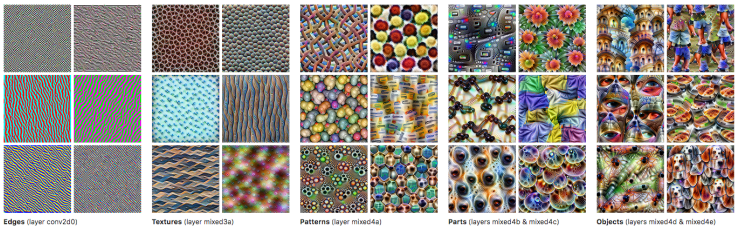

Figure 3.8: Feature visualisations of different layers in GoogLeNet[18] trained on the ImageNet dataset. Reprinted from [4]

kind of input will activate a component.

Figure 3.9: By jointly optimising two neurons we can get a sense of how they interact. Reprinted from [4]

Jointly optimising for the activation of two or more components reveals how they interact. This is especially interesting on the neuron level, since in

nature they work together to represent images.

Unfortunately, simply optimising an image to activate neurons produces images with a high level of noise and high-frequency patterns. This has been one of the top challenges in this and other feature visualisation approaches and most notable papers in this area contain some regularisation technique as a main point (Fig. 3.6.

A selection of different approaches from 3.6 is briefly explained below, starting with the core idea of activation maximisation.

Activation Maximisation

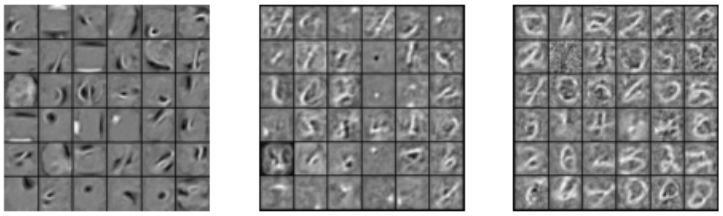

Figure 3.10: Activation maximisation applied on MNIST. Visualisation of 36 units from the first (1st column), second (2nd column) and third (3rd column) hidden layers of a Deep Belief Network (DBN) generated by activation maximisation. Reprinted from [17]

[19] introduces a method they call activation maximisation. The goal is finding an input sample that maximises the activation of a target hidden unit. The simplest approach to this would be to identify the input samples (from the training or test set) which leads to the highest activation in the unit. Unfortunately, this does not provide much insight into the common properties of the top samples. By looking at the method as an optimisation problem instead, a sample can be generated by optimising a randomly initialised input for maximal activation.

Optimisation of Objective Function

[20] introduces a method that reconstructs images using only information from the CNN and a generic natural image prior, starting from random noise. The reconstruction is generated by minimising an objective function that compares the reconstruction to the representation in the CNN. This technique can target any layer of a CNN and [20] concludes "that a progressively more invariant and abstract notion of the image content is formed in the network."

TBD

Maybe add a two or three more here?

3.2.2 Caricaturisation

Caricaturisation is a variant of Optimisation where instead of a random input, an initial image is modified by optimisation to enhance any pattern that triggers activation[9].

3.3 Attribution

Attribution has the goal to identify and highlight the parts of an input that contribute most to the final classification. It is closely related to object localisation in scenes.

Two types of visualisation are employed for attribution in computer vision [3]:

- **Correct class heat maps:** A heat map is overlaid or opacity reduced on the input image according to the encoded probability of the correct class. This shows which superpixels are important.
- **Most-likely class image** Each superpixel in the original image is coloured according to its most likely class.

Occlusion Sensitivity Analysis

[10] describes a method to identify the contribution of superpixels or regions of the input image by successively occluding part of the image by a gray square and running the classifier on the occluded image. Each occluded part is then coloured according to the class probabilities of the occluded image.

Class Activation Maps (CAM)

[15] introduces CAM, a method using global average pooling to identify the discriminative image regions used by the CNN for classification.

Chapter 4

Towards an Experimental Testbed for Debugging NN

4.1 Data

In the bachelors degree programme in Computer Science (BA in CS) at Zurich University of Applied Sciences (ZHAW), the first two semesters (three semesters for part-time students) form the assessment level. An average grade of 4.0 or higher in all modules is the requirement for admission to the main study programme.

A dataset of portraits of both current and former students the BA in CS at ZHAW and their respective average grade for the assessment level was provided by the school. For privacy reasons, no names or other identifying information was included. Additionally this ensured that no other information could be used by the network for classification.

4.1.1 Challenges

The dataset contains data for 430 students, which is an extremely low number for training a CNN. Standard training datasets for specific types of images like Labelled Faces in the Wild (LFW) [**?**] contain over ten thousand images, others like ImageNet [**?**] even contain Millions. Two approaches are proposed to alleviate the impact of the small number of images. First, Data Augmentation can been integrated into the learning process by randomly transforming the images, thus producing several versions of each image. Second, a network model that has already been trained on a large dataset of faces and has thus learned relevant features can be used as a basis. On top of that model, a number of fully connected layers can be trained to classify images using the features from the pre-trained model.

Another challenge is the class imbalance in the dataset when using pass/fail

as classes. Since most students have so far passed the assessment with only a few failing to do so (about 400 passing and 30 failing), a CNN could achieve over 90% accuracy by simply classifying all images in a test dataset that has the same distribution of classes as *pass*. This can be alleviated by moving the "pivot point" from 4.0 to 4.75, which is the median of grades over all students. A second approach to fix the class imbalance issue would be to adjust the loss function to account for the imbalance and punish false negatives in addition to rewarding true positives. Several other methods to address imbalance are surveyed in [**?**].

4.2 Preprocessing

The data was provided as an Excel file containing the image, grade and a numerical identifier.

4.2.1 Extraction from Excel file

To simplify working with the images, a Python script was developed to extract the images from the Excel file and store them in a directory. Additionally, a CSV file with the file name and grade for a student in each row was created for assigning each student to a class for training and validation.

4.2.2 Cropping Faces

For use in training a CNN, all input images need to have the same size. Most network architectures also expect the input to have a square shape. A Python script was written to automate this task. To identify the face and centre the crop region on it, OpenCV was used. For each image, the script first locates the face and crops away its surroundings. The face is then saved to a separate folder, resizing it to 160x160 pixels in the process.

4.2.3 Sorting Faces by Class Membership

To assign images to a class a script was developed to sort them into directories corresponding to correct class labels. This was done to prepare the data to be read by Keras [**?**], which was chosen as the framework to build the CNN with. Keras provides methods to read training and test data from this type of stucture and automatically extract and assign the correct class label for each image.

4.3 Proof of Concept

A first small proof of concept CNN (POC) was built using Keras to demonstrate that the data is suitable to be used for training after these preprocess-

ing steps. Input images were grouped into the two classes *pass* (grade ¿= 4.0) and *fail* (grade ¡ 4.0) and the POC was trained with this data.

Chapter 5

Summary and Future Work

After establishing that interpretability is key for reliability, especially in critical applications, an overview of the current state of the art methods for enhancing the interpretability of CNNs was provided. Even considering that the overview focussed on methods compatible with CNNs and specifically the visualisation of image classification tasks, many of the explored approaches are applicable to other domains and architectures. The wealth of methods and approaches shows that the view of ANNs as a inscrutable black boxes is at best outdated. There still is much room for deeper understanding of the inner workings of ANNs, but we already have a lot of tools to support us in this challenge.

A preprocessing pipeline has been created to support future experimental exploration of the question "Are final grades predictable from resume photos?" Additionally, several challenges to this task have been identified, for most of which countermeasures have been proposed.

The next step would be combining these results to build a system for training, evaluating and visualise image classification tasks. For this, a pretrained model needs to be chosen and, if necessary, adapted to ensure compatibility with different visualisation techniques.

Bibliography

[1] S. Russell and P. Norvig, "Artificial Intelligence A Modern Approach, 2nd edn," 2003.

[2] U. Karn, "An Intuitive Explanation of Convolutional Neural Networks." [Online]. Available: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[3] M. Thoma, "Analysis and Optimization of Convolutional Neural Network Architectures," jul 2017. [Online]. Available: http://arxiv.org/abs/1707.09725

[4] C. Olah, A. Mordvintsev, and L. Schubert, "Feature Visualization," *Distill*, vol. 2, no. 11, p. e7, nov 2017. [Online]. Available: https://distill.pub/2017/feature-visualization

[5] "TensorBoard: Visualizing Learning." [Online]. Available: https://www.tensorflow.org/get{_}started/summaries{_}and{_}tensorboard

[6] "Tensorboard Github." [Online]. Available: https://github.com/tensorflow/tensorboard

[7] "Edward - Tensorboard." [Online]. Available: http://edwardlib.org/tutorials/tensorboard

[8] L. M. Zintgraf, T. S. Cohen, and M. Welling, "A New Method to Visualize Deep Neural Networks," mar 2016. [Online]. Available: http://arxiv.org/abs/1603.02518

[9] A. Mahendran and A. Vedaldi, "Visualizing Deep Convolutional Neural Networks Using Natural Pre-Images," dec 2015. [Online]. Available: http://arxiv.org/abs/1512.02017http://dx.doi.org/10.1007/s11263-016-0911-8

[10] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks." Springer, Cham, 2014, pp. 818–833. [Online]. Available: http://link.springer.com/10.1007/978-3-319-10590-1{_}53

[11] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps," dec 2013. [Online]. Available: http://arxiv.org/abs/1312.6034

[12] A. Mahendran and A. Vedaldi, "Salient Deconvolutional Networks." Springer, Cham, oct 2016, pp. 120–135. [Online]. Available: http://link.springer.com/10.1007/978-3-319-46466-4{_}8

[13] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for Simplicity: The All Convolutional Net," dec 2014. [Online]. Available: http://arxiv.org/abs/1412.6806

[14] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization," oct 2016. [Online]. Available: http://arxiv.org/abs/1610.02391

[15] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning Deep Features for Discriminative Localization," dec 2015. [Online]. Available: http://arxiv.org/abs/1512.04150

[16] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning Important Features Through Propagating Activation Differences," apr 2017. [Online]. Available: http://arxiv.org/abs/1704.02685

[17] A. Balu, T. V. Nguyen, A. Kokate, C. Hegde, and S. Sarkar, "A Forward-Backward Approach for Visualizing Information Flow in Deep Networks," nov 2017. [Online]. Available: http://arxiv.org/abs/1711.06221

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," sep 2014. [Online]. Available: http://arxiv.org/abs/1409.4842

[19] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, "Visualizing higher-layer features of a deep network," *Bernoulli*, no. 1341, pp. 1–13, 2009. [Online]. Available: http://igva2012.wikispaces.asu.edu/file/view/Erhan+2009+Visualizing+higher+layer+features+of+a+deep+network.pdf

[20] A. Mahendran and A. Vedaldi, "Understanding Deep Image Representations by Inverting Them," nov 2014. [Online]. Available: http://arxiv.org/abs/1412.0035

[21] "Labelled Faces in the Wild."

[22] "ImageNet."

[23] M. Buda, A. Maki, and M. A. Mazurowski, "A systematic study of the class imbalance problem in convolutional neural networks," oct 2017. [Online]. Available: http://arxiv.org/abs/1710.05381

[24] "Keras." [Online]. Available: https://keras.io/

List of Figures

Appendix A

Code Listings and Dataset

Scripts for the preprocessing pipeline and the training data are provided on an external storage medium and will be handed in with the print version of this thesis.

Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

Erklärung betreffend das selbständige Verfassen einer Projektarbeit an der School of Engineering

Mit der Abgabe dieser Projektarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Projektarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum: Unterschriften:

…… ……

……..

……

Das Original dieses Formulars ist bei der ZHAW-Version aller abgegebenen Projektarbeiten zu Beginn der Dokumentation nach dem Titelblatt mit Original-Unterschriften und -Datum (keine Kopie) einzufügen.