Master Thesis

Radboud University

Modelling User Context using Deep Neural Networks

Author:
Dennis Verheijden
s4455770

ICIS Supervisor:
Arjen de Vries
Second Reader:
Faegheh Hasibi

August, 2019

Disclaimer

Abstract

In the current age, more photographs are taken than ever. To handle this, we could use automatic photoset summarization. However, a good summarization is highly dependend on the user. To this end, it is desirable to include the context of the user when judging the relevance of photographs.

This thesis provides a framework for modelling user context using neural networks. This framework consists of three components.

The first component is modelling the context of a user. Using neural networks, characteristics of a user were learned. From these networks, an abstract representation of a user is extracted, called the user vector.

The second component includes finding groups of users that share similar user vectors. This was done by clustering the user vectors.

The last component is incorporating user context into a Learning to Rank model.

A visual analysis of the filters, gradients and activations of the neural networks revealed that mainly facial and environmental features in the images are used for predicting characteristics of users.

Contents

Chapter 1

Introduction

Since the rise of social media platforms like Facebook, Snapchat and Instagram, the internet is filling with user-uploaded photographs. Every day, multiple petabytes of new multimedia is uploaded through these social platforms (Wiener, 2014). In 2017, the number of digital photographs that were taken in that year was estimated at a total of 1.2 trillion (Richter, 2017). It may come as no surprise that the most-used camera is the one that you carry around every day: your smartphone. This has made capturing and sharing your favorite moments easier than ever.

This is one of the main reasons why the number of photographs that are taken is increasing each year. A solution to this abundance of digitally-captured moments is photoset summarization. However, the process of manually creating such a summarization may be a time consuming practice, even when done digitally. This can be circumvented by using a platform which automatically creates a summarization. This is done by automatically selecting the *best* photographs, leaving out photographs that are blurry or duplicates.

Automatically generating photoset summarizations is a complex process. First, photographs are judged as to whether they are suitable for the summarization. These photographs are selected based on the value that they add to the whole, referred to as the quality score. This score consists of two components: A sentimental component, which is influenced by the contents of the image, for example: if the image contains people, landmarks or landscapes; a qualitative component, which is influenced by meta attributes of the image, for example: the blurriness of the image, whether the image is a duplicate or how beautiful the image is aesthetically-speaking.

The goal of photoset summarization is to generate a subset of the complete photoset which maximizes the joint quality score. The joint quality score is not merely the sum of the quality scores of individual photographs, as the subset should minimize redundancy and maximize novelty. In the context of photoset summarization, the selected photos must be a good representation of the whole photoset, while avoiding selecting multiple images that share a sentimental experience, e.g. selecting images of the same object.

Without considering the context of the photoset, i.e. the person creating the summarization, the produced selections are suitable for any person. This is a valid approach for the general population of people who wish to summarize a photoset, as this would be considered a good summarization. However, if the selection of the photographs is tailored to the preferences of the person creating it, the specific information needs of that person may be satisfied, which creates a more positive experience. Personalization is necessary as individual people may have different preferences, which leads to different information needs. This relies on the assumption that people attribute different quality scores to the same image, due to their personal preferences. For example, it might be the case that certain groups of people may prefer images of lush green environments, whereas others may prefer more urban environments.

This way, we may reframe the summarization problem to take the context of the creator into account: Given a set of photographs *and* the preferences of a user, generate a subset that maximizes the joint *personal* quality score.

The solution to the summarization problem may be given in two steps. The first step includes assigning a quality score to photographs and the second step includes selecting a subset of the given photoset. This thesis will focus on the first step, where the goal is to incorporate the preferences of a user for the approximation of the quality score of a photograph.

For the approximation of a personal quality score, a profile is constructed which defines the person creating the summarization. Constructing such a profile is not a trivial task, as users may not have interacted previously with the summarization system. To this end, the creation of this profile is constrained by only using photographs from the given photoset. For being able to generalize the preferences of unseen users, the constructed profiles are clustered to create groups of similar profiles. These groups are then used to learn a personalized quality score for a given photograph.

This thesis focuses on the question: "How can we learn the context of a user and incorporate this for the approximation of the quality score of a photograph?". This question can be divided into smaller problems. First, a method is required for learning a representation of the user context. Second, users should be grouped based on their representation. Lastly, user context should be incorporated into a model for predicting the quality score of a photograph. These smaller problems are examined in their corresponding chapters. Each chapter provides the required methodology and approaches for solving the problem. The approaches in these chapters are experimentally tested and the results will be provided in the corresponding sections.

This thesis is organized as follows: The next chapter, Chapter 2, provides background information about used methods and related research. Chapter 3 provides a framework for modelling user context. Chapter 4 examines different clustering approaches for finding groups of users with similar profiles. Chapter 5 provides a method for applying user context for the approximation of a quality score of a photograph.

In Chapter 6, an analysis is given. The goal of this analysis is to provide a possible explanation of the results found in previous chapters. This is done by opening the black-box that arises from the nature of neural networks. In Chapter 7, the full approach is discussed. Here, the results are interpreted and discussed, including the potential limitations and material for future work. Finally, in Chapter 8, the conclusion of this thesis is presented.

Chapter 2

Background

This thesis focusses on approximating a personal quality score for a given photograph. This requires a representation[1] of the user in the form of a user profile.

As mentioned in Chapter 1, learning such a representation has the constraint that only photographs from the photoset may be used. To this end, a deep learning approach is used as they are very capable of extracting high level features of their inputs. This learned representation will then be used for finding groups of users with similar representations. Finally, these groups will be used to learn the quality score of a photograph.

This chapter will provide background material on methods that are applied in this thesis. First an introduction is given on neural networks, including variants which are used within this thesis.

This is followed by related work. This section will include studies based on photoset summarization and deep learning approaches for extracting features from photosets. Afterwards, the data pipeline used within this thesis is presented. In this section, the retrieval, cleaning and pre-processing of the data is discussed. This section is followed by several dimensionality reduction techniques that may be used to generate compact user representations. Finally, a framework is discussed for learning an indirect ranking using Learning to Rank. This framework will be central for learning a quality score of a photograph.

2.1 Neural Networks

Artificial neural networks have become the driving force for achieving new state-of-the-art performances in various domains, for example in Computer Vision (Hu, Shen, & Sun, 2017; Howard et al., 2017; Alom et al., 2019) and Natural Language Processing (Edunov, Ott, Auli, & Grangier, 2018; Radford et al., 2019). In this thesis, neural networks are central for learning high level features from photosets and quality scores for photographs. As such, some general knowledge of neural networks is required.

[1] This thesis may contain terms like 'representation', which may be unclear to some readers. To this end, a list of terms and their descriptions is provided in the Glossary. The first occurrence of a term contained in the glossary is styled with a red font.

This section will provide a basic introduction to neural networks and explain the concepts that are relevant for this thesis. The following mathematical notations will be used:

- x non-bold lower-case letters denote elements. If the element is indexed, the subscript denotes the position of the element in the vector.
- $\mathbf{x}$ bold lower-case letters denote vectors.
- W capital letters denote matrices. If the matrix is indexed, the subscript denotes the position.

2.1.1 The Basics of Neural Networks

Neural networks are a class of connectionist machine learning algorithms. A neural network consists of processing units and connections between them. These will be referred to as the nodes and weights of a neural network. This mechanism is vaguely inspired by the human brain (van Gerven & Bohte, 2018). Here, the structure of the neural network resembles the structure of the brain with neurons and synapses respectively.

A neural network may be described as a system that takes input features and outputs a high-level representation (Bishop, 2006). This high-level representation is computed by feeding the input through a series of layers. In each layer of the network, a (non-)linear transformation is applied to the input $\mathbf{x}$. This transformation is computed using trainable weights $\mathbf{W}$ and bias terms $\mathbf{b}$. The output of a layer is then activated using an activation function ϕ.

Throughout this thesis, four different types of layers will be distinguished:

- *input layer*: the first layer of the network which receives external inputs.
- *hidden layer*: an intermediate layer of the network which computes abstract features.
- *feature layer*: the last hidden layer. The output of this layer holds all information for making predictions.
- *output layer*: the last layer of the network. This layer leverages the information of the feature layer to compute the predictions of the network.

To explain the mathematical concepts of a neural network, an example will be given using a Multilayer Perceptron network (MLP) with one hidden layer. A graphical representation of this example may be found in Figure 2.1. This type of network is more commonly referred to as a fully-connected network. The provided example is adapted from Bishop (2006).

Each layer in the MLP network contains M nodes, referred to as perceptrons. A layer of perceptrons are referred to as a dense layer. A neural network containing only dense layers are called Dense Neural Networks (DNNs).

The output unit activation, i.e. the output before activation, of the j-th perceptron is given by:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + b_j^{(1)} \tag{2.1}$$

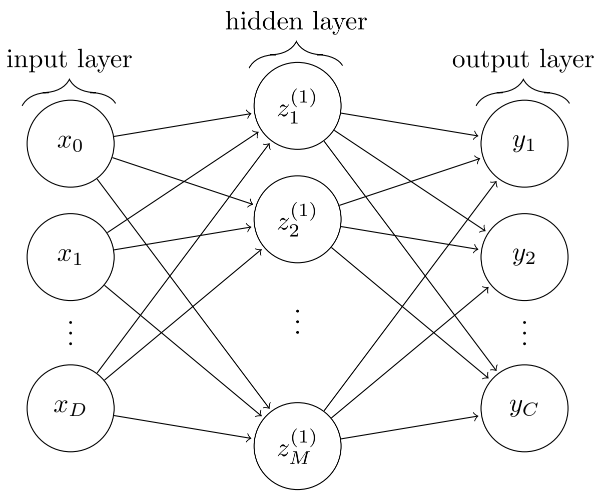


Figure 2.1: Multilayer Perceptron Neural Network with one hidden layer

where x_i referes to the i-th element of the input $\mathbf{x}$. The superscript in $w^{(1)}$ and $b^{(1)}$ indicates that the corresponding parameters are in the first layer of the network.

The output of the layer is then activated using a differentiable (non-)linear activation function ϕ:

$$z_j = \phi(a_j) \tag{2.2}$$

where z_j corresponds to the activated output of perceptron j. The output of intermediate layers are referred to as hidden units.

Since this network contains only one hidden layer, this layer is also the feature layer. Thus, the next layer is the output layer. The output unit activations is then given as:

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + b_k^{(2)} \tag{2.3}$$

where $k = 1, \dots, K$, and K is the number of outputs of the network. Finally, the output unit activations are transformed using an activation function to compute the output of the model $\hat{\mathbf{y}}$. For intermediate activation functions, hyperbolic tangent (tanh, Equation 2.4) activation or the Rectified Linear Unit (ReLU, Equation 2.5) (Nair & Hinton, 2010) is generally used. However, since tanh activation suffers from the vanishing gradient problem (Nwankpa, Ijomah, Gachagan, & Marshall, 2018), the latter is commonly preferred.

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

$$\text{relu}(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.5}$$

The choice of the activation function for the output layer, is dependent on the type of task.

For regression tasks, there are multiple choices for the activation function. If the output is allowed to be greater than 1, the Identity function $\mathbf{y} = \mathbf{a}$ may be used. If the output should be within the range $[0, 1]$, the sigmoid function may be used. This function is given by:

$$\mathbf{y} = \sigma(\mathbf{a}) = \frac{1}{1 + e^{-a}} \tag{2.6}$$

For classifying samples belonging to C classes, the output of the model should be a distribution P over the classes C. This is given by applying the softmax function σ to the output unit activations $\mathbf{a}$:

$$y_c = P(C = c) = \sigma(\mathbf{a})_c = \frac{e^{a_c}}{\sum_{i=1}^{C} e^{a_j}} \tag{2.7}$$

where $c = 1, \dots, C$, and C is the number of output classes of the network, which is equal to the number of units K in the output layer.

The weights of a node may be learned through an iterative process called backward propagation of errors, commonly referred to as back-propagation (Rumelhart, Hinton, Williams, et al., 1988). This iterative process updates trainable parameters based on the error between the prediction of the network $\hat{\mathbf{y}}$ and the target $\mathbf{y}$. This error is referred to as the loss, which is computed by a loss function. More specifically, the weights of a node are updated according to the derivative of the loss with respect to the weights. This derivative is found by using the chain rule to *back-propagate* the errors. For more information about back-propagation you are encouraged to read (Rumelhart et al., 1988).

The choice of the loss function, depends on the task: For regression problems, the mean squared error (MSE) may be chosen, which is given as

$$\mathtt{MSE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{K} \sum_{i=1}^{K} (y_i - \hat{y}_i)^2 \tag{2.8}$$

For classification problems, the cross-entropy loss (CCE) is a good alternative, which is given as:

$$\mathtt{CCE}(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{C} \sum_{c=1}^{C} \Bigg[y_c \log(\hat{y}_c) + (1 - y_c) \log(1 - \hat{y}_c) \Bigg] \tag{2.9}$$

where $\mathbf{y}$ is the true label (encoded as a one-hot vector) and $\hat{\mathbf{y}}$ is the predicted label. For binary classification, the binary cross-entropy loss (BCE) may be used, a specific configuration of the categorical cross entropy. Assuming that the softmax activation function is used for the prediction layer, a two-class classification problem may be represented as a one-class classification problem, as $y_0 = 1 - y_1$. The binary cross-entropy is then given as:

$$\mathtt{BCE}(y, \hat{y}) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})) \tag{2.10}$$

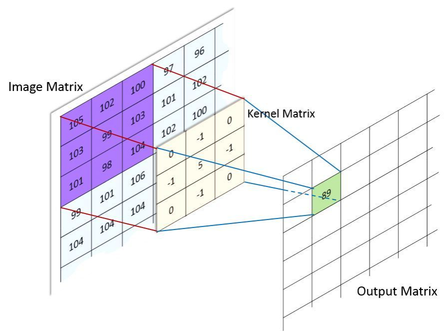


Figure 2.2: Example of a convolution operation in an image.[2]

The weights are then updated such that the loss is minimized. The optimization of weights in a neural network is done by an optimizer such as Stochastic Gradient Descent (SGD) (Kiefer & Wolfowitz, 1952). It has been shown that models learned with optimizers like SGD are able to generalize well to unseen data. This statement holds even when the number of parameters in the network is significantly larger than the number of samples in the training set (Zhang, Bengio, Hardt, Recht, & Vinyals, 2016). This is the main reason why neural networks are known for being good estimators for complex (unknown) functions. However, the drawback of neural networks is that learning the parameters typically require more training data than other traditional classification algorithms such as Support Vector Machines.

2.1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of neural networks that extract features based on patterns in their inputs. They are specifically designed to deal with the variability of 2D objects (LeCun, Bottou, Bengio, Haffner, et al., 1998). This makes CNNs the dominant option for computer vision tasks.

CNNs perform convolutions in their layers, where each node contains a filter that reacts to a specific pattern in the input. This filter is also commonly referred to as a kernel. A convolution is an operation where a filter is passed over an input using a sliding window. The output is defined as the sum of the element-wise multiplication of the values in the sliding window with the corresponding value in the filter. The resulting matrix is referred to as the feature map. A graphical example of a convolutional operation is given in Figure 2.2. For a given filter h with size (j, k) and input f, the convolution is given as

$$G[m,n] = (f * h)[m,n] = \sum_j \sum_k h[j,k] f[m-j, n-k] \tag{2.11}$$

where m, n are the indices of the resulting matrix. The resulting matrix is smaller than the input matrix as the filter center cannot slide over the edges of the input image. For example, when the input matrix f is a 3D image with dimensions $12 \times 12 \times 3$, the result of a convolution with a $5 \times 5 \times 3$ filter is a $8 \times 8 \times 1$ $(12 - 5 + 1)$ matrix.

[2]https://brilliant.org/wiki/convolutional-neural-network/, date accessed: *June 19th, 2019*

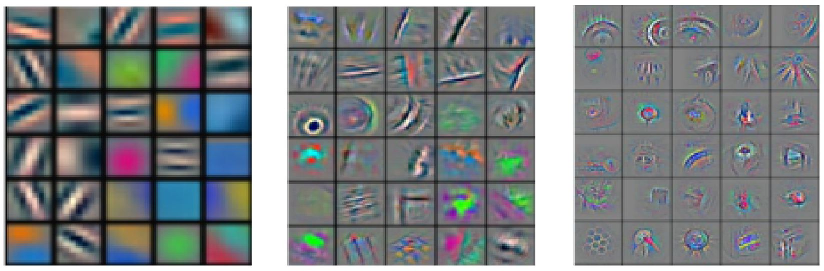

(a) low-level (b) mid-level (c) high-level

Figure 2.3: Feature maps visualizations on different depths of a CNN. (a) feature map in an early layer. Here we see sensitivity to low-level features like lines and gabor filters. (b) feature map in a later layer. Here we see sensitivity to texture-like patterns. (c) feature map of the last convolutional layer. Here we see sensitivity to concrete objects and structures.

Each convolutional layer holds K filters, such that the output of a convolutional layer contains K feature maps. These are then stacked to create a single matrix. Extending the previous example to a layer with 32 kernels would lead to an output matrix of 8 x 8 x 32. However, one may also perform convolutions with padding. Here, the image is padded such that the output dimension is equal to the input dimension. There are multiple methods for padding the input. Typically the input is padded with zeros, as this does not affect the output (since the multiplication with zero remains zero).

Each convolutional layer consists of multiple filters. As every filter contains different values, each filter is sensitive to different patterns in the input. This ensures that the resulting feature maps contain unique information about the input. Since the resulting feature maps are fed through multiple convolutional layers, each feature map reacts to increasingly more complex patterns. For example, feature maps in earlier layers are sensitive to very basic patterns, like lines, color or contrasts. While feature maps in later layers are sensitive to increasingly more complex patterns (Olah et al., 2018), for example structures and faces. This resembles the process of how humans perceive visual stimuli (van Gerven & Bohte, 2018). Here, the early topologies in the visual stream are sensitive to lines of different orientations. While later topologies are specialized to recognize certain objects, for example faces.

To determine what pattern maximally activates certain nodes, the feature maps may be visualized. This is achieved by generating the optimal input f^* for a feature map G, given by

$$f^*_G = \underset{f^*}{\operatorname{argmax}} \sum_m \sum_n G[m,n] \tag{2.12}$$

The solution may be found by taking advantage of back-propagation. Examples of feature-map visualizations with varying complexity may be found in Figure 2.3. This is one of the techniques that will be used in Chapter 6, for analyzing the neural networks in this thesis.

2.1.3 Recurrent Neural Networks

For DNNs and CNNs, the input consists of a fixed number of images, typically one. However, for the classification of photo sets, the input is a sequence of images of arbitrary length. One solution to this problem is to compute the average image, i.e. concatenate all images and take the mean over the image dimension. However, this solution would discard possibly important information, since image-specific features and chronological ordering is lost in the process.

A different solution would be to use a type of neural network that can handle sequences of arbitrary length as an input. One class of neural networks that solve this problem are Recurrent Neural Networks (RNNs). RNNs contain layers which learn spatial information from individual samples (like perceptrons) as well as temporal information from the sequence. This allows RNNs to learn intra- and inter-sequence information (Lipton, 2015).

Nodes in a recurrent layer contain edges with corresponding weights, similarly to perceptrons, with the addition that there may be edges to itself, called recurrent edges. This creates cycles in recurrent layers which transfer information about previously seen samples in the sequence. For the computation of the activations in a recurrent layer, we may combine and extend equations 2.1 and 2.2:

$$\mathbf{z}^{(t)} = \phi(W_{hx}\mathbf{x}^{(t)} + W_{hh}\mathbf{z}^{(t-1)} + \mathbf{b}) \tag{2.13}$$

where the superscript (t) corresponds to the time-step t. W_{hx} corresponds to the non-recurrent weights connecting the input units to the layer units. W_{hh} corresponds to the recurrent weights, connecting the layer units to themselves. Since the input for RNNs are sequences, we write $\mathbf{x}$ instead of x_i.

Finally, the output of the recurrent layer is given by:

$$\hat{\mathbf{y}} = \phi(W_{yh}\mathbf{h}^{(t)} + \mathbf{b}_y) \tag{2.14}$$

The computation of the activations through time is done by unrolling the layer. This is done by computing activations sequentially, where every sample in the sequence is regarded as a time-step. A graphical representation of this process is given in Figure 2.4. Since one forward pass includes multiple time-steps, Backpropagation Through Time (Werbos et al., 1990) was introduced.

Mainstream applications for RNNs are mostly found in Natural Language Processing (NLP), where sentences are depicted as a sequence of words. Examples of RNN applications in NLP are sentiment analysis and Sequence2Sequence problems such as translation from one language into another (Bahdanau, Cho, & Bengio, 2014).

2.2 Related Work

As mentioned, learning a personal quality score for photographs requires two things: first, a method for learning the context of a user; second, a method for incorporating user context into a model. The first method may be rephrased as a classification problem: classifying the type of user given the photoset. When using a neural network for solving the classification task, the feature layer may

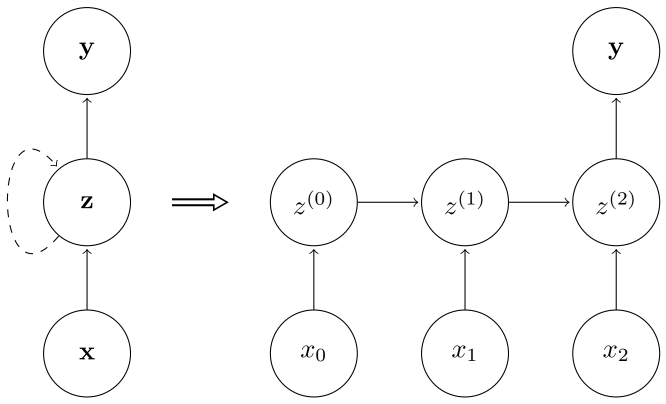


Figure 2.4: Example of unrolling of a Recurrent Layer with three inputs.

be used for extracting high level features from photosets. The second method may be rephrased as an information retrieval problem, where the goal is to summarize photosets.

This section will outline studies that are related to classification using multiple images and photoset summarization.

2.2.1 Multi-image Classification

In multi-image classification, the model is presented with multiple images. Instead of sequentially feeding images, the images are fed at once. From literature there are two prominent approaches: multi-view learning and video classification. In the first approach, images are fed through separate channels. In the second approach, the images are concatenated, creating a new dimension. The new dimension may then represent the temporal dimension of images, which could be leveraged using a RNN.

This section will outline both methods and discusses relevant literature.

Multi-view Learning

Multi-view learning was first used for object recognition. Here, multiple views of the same object were fed to the model. These views may be photographs or scans from different angles, effectively creating a 2.5D scan of the same object.

These approaches are mostly found in the field of medical imaging (Roth et al., 2014; Setio et al., 2016). Here multiple planes of, for example, a lesion are presented to the model. These different views are then processed and combined using a fusion technique, which combines the computed feature vectors. For classifying nodules, late-fusion gave the highest performance (Setio et al., 2016). Late-fusion is a technique where images are processed separately in different streams. The feature vector of each stream is then concatenated into one vector which is used for predictions. Similar successes are achieved in the field of object and face recognition (Su, Maji, Kalogerakis, & Learned-Miller, 2015; Farfade, Saberian, & Li, 2015).

Karpathy et al. extended this approach for a set of multiple images. Here, subsets of images with a fixed length were selected. These images were sequentially processed using convolutional layers and *fused* in different stages. Images fused in early stages were first concatenated and processed using four-dimensional convolutions. Where images fused in later stages are fully processed individually using three-dimensional convolutions and concatenated afterwards. A hybrid approach was also proposed, called soft fusion. Here, images are iteratively processed and merged with their neighbors until there was only one stream. It was found that soft fusion gave the highest performance.

The drawback of these approaches is that the number of streams or images is constant, i.e. the number of input images is static. This may have implications for our application as the number of images in a photoset is variable.

Video Classification

Another possibility is to model our problem as a video classification problem, where the images may be perceived as a fourth dimension. As photograhs in a photoset are shot in chronological order, sequentially going through the photographs may be perceived as looking at a film reel. Stacking all images may thus be seen as a timelapse-video.

A study by Yue-Hei Ng et al. fully exploits the fourth, temporal, dimension. This work focusses on the classification of short video clips. The classification is split into two tasks. First, every frame from the video is processed individually to compute high level features using a pre-trained convolutional neural network. Afterwards, the resulting high level features are concatenated and processed. Two different approaches are given for processing the high level features, either through temporal convolutions or through recurrent cells (as used in Recurrent Neural Networks). It was found that processing the high level features using recurrent cells performed better than using convolutions and pooling.

2.2.2 Information Retrieval

The automatic summarization of a photoset consists of two stages, relevance prediction and reranking. In the first stage, every photograph is individually given a relevance measure that indicates the importance of the photograph in the set. These photographs may then be ranked according to their relevance, where a higher rank indicates a higher importance.

The second stage reranks the photographs. Here the the relevance is judged in the context of the whole photoset, where the co-occurrence of certain photographs may result in a lower rank. An example is duplicate images, as having duplicate images decreases novelty.

Taking the top reranked photographs then gives a representative subset of the photoset which minimizes redundancy and optimizes novelty. The focus of this thesis will be on the first stage, as the aim is to judge the relevance of the photograph based on the preferences of a user.

A classical information retrieval problem consists of two things: a set of documents $D = d_1, d_2, \ldots, d_N$ and an information need, expressed as a query q. The task is to find a subset of documents $d \in D$ which are relevant to the query q. Afterwards the documents are ranked based on some ranking formula.

Bouhini, Géry, and Largeron have introduced the context of the user into judging document relevance. This is done by creating a profile of a user from social annotations. These social annotations are initially provided by the user and may be enriched by annotations from users with similar search queries. The user profile is then integrated into the relevance function to boost documents that are relevant based on the user profile.

There are multiple options for judging relevancy of documents for a given query. In this thesis, the Probability Ranking Principle (Robertson, 1977) is used. This principle states that the overall effectiveness of an IR system will be maximal, if the response to each query is a ranking of documents in order of decreasing probability of relevance. As such, the probability that a document is relevant for a given query q may be used as the relevancy score of a document. The language modelling approach to IR aims to estimate this probability by constructing a Language Model for each document M_d for judging the relevancy $P(R = 1|M_d, q)$. The intuition behind probabilistic ranking models is that models corresponding to relevant documents are assigned higher probabilities (Jones, Walker, & Robertson, 2000).

In Chapter 5, a ranking model will be trained. This model will be trained with and without the incorporation of user context. If the incorporation of user context yields a higher performance than the model without the incorporation of user context, this may be perceived as evidence that the proposed framework is able to model user context.

2.3 Transfer Learning

Deep Neural Networks that are used for image analysis are generally a class of Convolutional Neural Networks. These networks learn to recognize complex patterns that may be found in the input. For most tasks in the field of computer vision, like classification and detection, similar features should be learned in the earlier layers. Examples of these features include edge detection and structure recognition (Figure 2.3). Depending on the task, more specialized features in later layers may also be useful.

When learning different computer-vision-related models, it is observed that the learned feature maps of these models exhibit similar patterns in convolutional filters (Simonyan, Vedaldi, & Zisserman, 2013). Due to this reason, it is convenient to start learning by using a model that has already learned to detect basic features that are applicable to the domain of the task. This can be achieved by using a deep learning model that is trained on domain A and adapting it to domain B. This works when domain A and B share enough similarities. This technique is called Transfer Learning.

2.3.1 Transferring Domain Knowledge

Transfer Learning is best applied when the task on which the pre-trained model is learned, shares knowledge with the task that is to be achieved (Goodfellow, Bengio, & Courville, 2016). The concept of Transfer Learning is that features which are shared between the two tasks, do not have to be re-learned. Instead, the useful features are *transferred*. This makes transfer learning most successful when the learned features from the source task are generalizable to the target task (Olivas, Guerrero, Sober, Benedito, & Lopez, 2009).

The most popular benchmarking task in image classification is classifying images from the ImageNet dataset (Russakovsky et al., 2015). This dataset contains 1000 image categories and a total of roughly 1.5 million images. This dataset is used as a benchmarking tool to evaluate new deep learning architectures that may be used for image classification. New network architectures may then be pre-trained on the ImageNet dataset and made available through popular deep learning frameworks, such as TensorFlow[3] and PyTorch[4].

2.4 Used Dimensionality Reduction Techniques

One sub-task of this thesis is finding groups of similar users. This will be solved by grouping the representation of users into clusters, where every user contained within a cluster shares certain characteristics. These clusters will be found by employing a clustering algorithm, which will be further examined in Chapter 4.

However, these cluster algorithms work best best on low-dimensional data (Lloyd, 1982; McInnes, Healy, & Melville, 2018). To this end, the user representations should be reduced in dimensionality. Dimensionality reduction algorithms typically fall into one of two categories: those who seek to preserve the structure within the data and those who aim to preserve local distances over global distances.

When reducing the dimensionality of data, the chosen dimensionality reduction method has a big impact. For finding the algorithm that best suits the needs of this thesis, three algorithms will be discussed and compared: *Principal Component Analysis*, which belongs to the first category; *t-Distributed Stochastic Neighbor Embedding* and *Uniform Manifold Approximation and Projection*, which both belong to the second category.

In this section, all three algorithms will be examined. They will be explained to an extent, such that the advantages and disadvantages of each algorithm are clear. For further information, it is recommended to read the respective papers. Finally, the three algorithms are applied to an example to choose the most suitable algorithm for this thesis.

2.4.1 Principal Component Analysis

Principal Component Analysis (PCA) (Pearson, 1901) is a dimensionality reduction algorithm that aims to preserve the structure within the data. This means that distances between datapoints in the reduced low-dimensional representation are proportional to the distances between datapoints in the original high-dimensional representation.

PCA decomposes a multivariate dataset into a set of Principal Components. These Principal Components are orthogonal components that explain a maximum amount of variance, where the variance is defined as the expected value

[3]`https://www.tensorflow.org`, date accessed:*June 19th, 2019*
[4]`https://pytorch.org/`, date accessed:*June 19th, 2019*

of the squared deviation from the mean:

$$\mathbf{Var}(X) = E\big[(X - \mu)^2)\big] \tag{2.15}$$

and X is a datapoint in the dataset.

The resulting principal components are sorted, such that the first principal component is the component that explains the most variance. These principal Components may be seen as linear combinations of features that describe the data.

PCA has been successfully applied in multiple fields for reducing dimensionality (Ma & Dai, 2011), including deep learning (Sun, Chen, Wang, & Tang, 2014).

2.4.2 t-Distributed Stochastic Neighbor Embedding

t-Distributed Stochastic Neighbor Embedding (t-SNE) (Maaten & Hinton, 2008) is a manifold approach. This approach is very popular for visualizing high dimensional features that are extracted using deep neural networks (Chan, Rao, Huang, & Canny, 2018). t-SNE aims to find local and global patterns in the data, where similar samples are drawn together and distant samples are pushed apart.

The steps for achieving this are as followed: First, two Gaussian probability distributions are created. The first distribution P models the relationship between points in the original high-dimensional space. The second distribution Q, is a low-dimensional model, which models the relationship between points in the low-dimensional space.

The low-dimensional representation of the high-dimensional data is then determined by minimizing the Kullback-Leibler (KL) divergence between the distributions P and Q. This is given by:

$$\text{KL}(P||Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{2.16}$$

This low-dimensional representation is then optimized through gradient descent until the KL divergence converges to a local minima. An intuitive interpretation of the computed gradients is the following: If the gradient with respect to two points is positive, there is an attraction. If the gradient is negative, there is a repulsion.

The resulting low-dimensional representation is referred to as the embedding. This embedding is a stable local minimum of the optimization. It is worth noting that the KL-divergence is non-convex, such that the resulting minima might be a local optima. However, tricks like *Early Exaggeration* and *Early Compression* try to avoid this (Maaten & Hinton, 2008).

One important hyperparameter of t-SNE is the perplexity. This hyperparameter is a measure which controls how much the global structure should be preserved over the local structure.

While t-SNE is a good tool for reducing dimensionality for existing data, it is unable to transform new data. This is due to the fact that t-SNE does not learn a transformation function[5], instead it directly optimizes the low-dimensional representation.

2.4.3 Uniform Manifold Approximation and Projection

Uniform Manifold Approximation and Projection (UMAP) (McInnes et al., 2018) is a manifold technique, like t-SNE. UMAP aims to find a transformation function for mapping high-dimensional data onto a low-dimensional plane. This is a clear distinction between t-SNE and UMAP as t-SNE directly optimizes an embedding.

The steps for finding the transformation function are as followed: First, a weighted graph is computed for the high-dimensional data W^H that models the local relationship of neighboring points. Second, a weighted graph is computed for a low-dimensional representation W^L, similarly to the high-dimensional neighborhood. The low-dimensional representation is then determined by minimizing the cross entropy, given by:

$$C(W^H, W^L) = \sum_{i \neq j} W_{ij}^H \log\left(\frac{W_{ij}^H}{W_{ij}^L}\right) + (1 - W_{ij}^H) \log\left(\frac{1 - W_{ij}^H}{1 - W_{ij}^L}\right) \tag{2.17}$$

where W_{ij} represents the weight of the edge connecting point i to point j.

The most important hyperparameter for UMAP is the number of neighbors that are considered when constructing the weighted graph. Having a larger number of neighbors produces more global views of the structure, which preserves a more global structure of the data. Another important hyperparameter is the minimum distance between datapoints. A lower (or zero) minimum distance relaxes the constraints on the placement of the points in terms of the distance between neighbors. This typically leads to dense clumps of data. A large minimum distance typically leads to the datapoints being more sparse in the low dimensional plane.

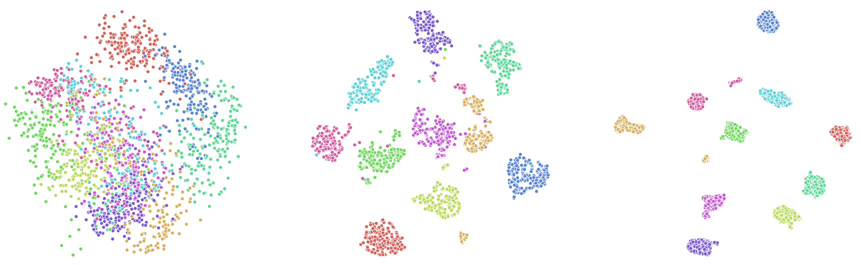
The low-dimensional representation may then be optimized by performing gradient descent to find a stable local minimum. As UMAP indirectly optimizes the embedding through the optimization of the weighted graph, it is suitable for transforming new datapoints.

2.4.4 Comparison

For finding the the best dimensionality reduction technique, the three techniques will be applied to the MNIST dataset. This dataset consists of written digits, where each class corresponds to a different digit. This example will highlight the characteristics of the dimensionality reduction methods. The outcome of the dimensionality reduction methods may then be compared to choose one definitive dimensionality reduction method that will be used throughout this thesis.

In the MNIST dataset, each digit is presented as an $28 \, x \, 28$ black-and-white image. The image may then be flattened, such that every image is represented

[5] Maaten adapted t-SNE to learn a parametric transformation function, called parametric t-SNE (Maaten, 2009). However, at the time of performing the experiments, no robust implementation was available. To this end, parametric t-SNE is not considered in this thesis.



(a) PCA (b) t-SNE (c) UMAP

Figure 2.5: A comparison of the representations found by PCA, t-SNE and UMAP using the MNIST dataset. The hyperparameters were set to their default values. The colours in the plots represent the digits in the dataset. The colour representation is equal for every plot.

by a 1 x 784 feature vector. The goal in this example is to reduce the high-dimensional feature vector to a low-dimensional feature vector containing only two components. The resulting low dimensional space may then be visualized by plotting the two components on the x and y axes.

Figure 2.5 depicts the results of applying these three algorithms to the MNIST dataset, which clearly highlights the difference between the two classes of dimensionality reduction algorithms. The low-dimensional representations produced by PCA contain a lot of overlap between classes. This may be explained by the fact that the images share pixels which are black in every image.
The low-dimensional representations produced by t-SNE and UMAP do not have any overlap. This may be explained by the fact that these types of dimensionality reduction algorithms attract similar samples and push away distant samples.
The differences between UMAP and t-SNE are more subtle. When looking at the relational distance between classes in the PCA representation, these are more similar to the representation produced by UMAP than t-SNE. This highlights the main difference between UMAP and t-SNE: UMAP preserves the global structure of the data better than t-SNE.

However, the choice of dimensionality reduction algorithm depends on more factors than the produced representation. For production systems, scalability and runtime are also important factors. Here every computed representation should be reduced in dimensionality before finding the cluster-membership of the representation. As scalability and runtime is heavily influence by the implementation of the algorithm, we will take a look at the following implementations: `MulticoreTSNE`[6] for t-SNE, the `sklearn`[7] implementation of PCA and the `UMAP`[8] implementation by the original authors of the UMAP paper.

[6]https://github.com/DmitryUlyanov/Multicore-TSNE, date accessed:*June 19th, 2019*
[7]https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html, date accessed:*June 19th, 2019*
[8]https://umap-learn.readthedocs.io/en/latest/benchmarking.html, date ac-

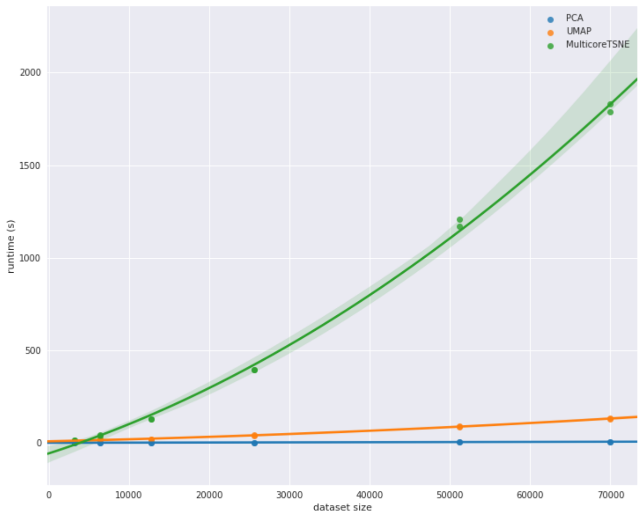


Figure 2.6: A comparison of runtime and scalability of PCA, t-SNE and UMAP.

A comparison between the three algorithms based on these factors may be found in Figure 2.6. In terms of scalability and runtime, PCA performs best, it scales well to large datasets and has a low runtime. This is closely followed by UMAP, which scales just slightly worse than PCA. t-SNE has the worst scaling by far, performing worst out of all three dimensionality reduction methods. Judging on the size of the dataset used within this thesis, it is not feasible to use t-SNE.

For clustering it is desired to generate low-dimensional representations that preferably generate clearly separated groups of samples. Due to efficiency and UMAP preserving global structure better than t-SNE, UMAP will be the chosen method for dimensionality reduction. However, it should be noted that the resulting representations of manifold approaches may generate clusters that are not part of the data but are instead introduced by noise in the data. This is circumvented by carefully tuning the hyperparameters.

cessed:*June 19th, 2019*

2.5 Data Pipeline

This thesis will be using implicit customer feedback. As such, care should be taken when using this data in a machine learning pipeline. This section will outline the retrieval of user data and the preprocessing steps that are performed.

2.5.1 User Data Retrieval

This section is available upon request.

2.5.2 Data Preprocessing

Due to the large collection of photographs in the dataset, pre-processing during runtime is kept to a minimum. To achieve this, photosets are stored in the most compact and readily-available representation possible.

Every image is converted and resized. Afterwards, useful information is extracted from the images, called encodings. It should be noted that storing images in an encoded format, instead of an RGB format, entirely disables the usage of augmentations on the image-level. However, there are a number of augmentations that may be performed on photosets to reduce the risk of overfitting. These will be further examined in Section 3.2.2. Since the encoded format does come with limitations, this format is only used for approaches that use photosets as an input, i.e. Chapter 3 and Chapter 4.

Chapter 3

User Quantification

Before the preferences of a user can be learned, a method for representing a user is required. This representation consist of a list of values that define a user. This list of values will be referred to as the *user vector* $\mathfrak{u}$. In this chapter, two types of user vectors are examined: User vectors containing concrete properties and user vectors containing abstract features. These will be referred to as respectively concrete and abstract user vectors.

Concrete user vectors contain meaningful characteristics of a user. Abstract user vectors contain a series of numbers $\mathfrak{u}_i \in \mathbb{R}$. These numbers may not have any meaning in isolation, but the combination provides a rich representation of the user.

The outline of this chapter is as follows: First, the methodology for computing user vectors is discussed. This is followed by an overview of the data and the training procedure for training the neural networks. The chapter will conclude with the results.

3.1 Methods

In this section, the methodologies will be explained regarding learning the representation of a user.

First, the computation of the two types of user vectors is discussed. Afterwards, the two different types of neural networks will be discussed for learning this representation.

3.1.1 User Vector

For a given user, user characteristics may be known. This information will be used for learning a concrete user vector. As discussed in section 2.1.1, neural networks are excellent feature extractors. To this end, the abstract user vector is extracted from the learned neural networks.

3.1.2 Neural Network Architectures

This section is available upon request.

3.2 Experimental Setup

This section will provide a summary of the dataset and experimental setup that is used for testing the proposed neural networks. First, an overview of the dataset is given. This is followed by a section in which an outline is given for the different steps in the training procedure. Finally, experiments will be presented to test the approach.

3.2.1 Data

The neural networks will be trained to predict the concrete user vector of a corresponding photoset. The dataset consists of sequences of encoded photograph with the corresponding labels. Here, the label corresponds to the concrete user vector. As some user characteristics may be unknown, it was decided to train one model for every feature of the concrete user vector.

3.2.2 Training Procedure

The training procedure is split in two stages, dataset creation and training. First, the photosets for which the label is unknown are removed from the dataset containing all photosets. The resulting dataset is split into three subsets: a training, validation and test set. These contain respectively 62.50%, 25% and 12.50% ($\frac{5}{8}$, $\frac{2}{8}$ and $\frac{1}{8}$) of the dataset. This is done by first grouping the photosets of unique users. Afterwards, N_{val} and N_{test} photosets are drawn from the dataset without replacement. Samples are drawn according to a uniform distribution, where the probability of sampling label c is given by $P(C = c) = \frac{1}{|C|}$, where $|C|$ denotes the number of labels in a class. The subsets are created such that every user is contained in at most one subset. This is done to ensure that the model cannot already have seen the photos of a certain user.

To remove the bias towards over-represented classes, batches from the training set are sampled such that the class distribution in one batch is roughly equal. This is done by sampling according to a uniform class distribution. The epoch is finished when all photosets in the smallest class are sampled once. Note that not every photoset in the largest class is sampled in each epoch, since no sample from the smallest class may be sampled twice. However, through experimentation, it was determined that this sampling method increased performance.

Models were constructed and trained using TensorFlow, an end-to-end machine learning platform (Abadi et al., 2015).

3.2.3 Experiments

There are a number of experiments that may be performed to test the proposed approach. The first experiment compares the proposed architectures. One model is trained for each characteristic.

For categorical classes, the performance measures that will be used are the categorical cross-entropy loss (CCE), given by Equation 2.9, and the categorical accuracy (`acc`). The reported accuracy is the average categorical accuracy of the test set, which is defined as:

$$\texttt{acc}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \sum_{i=1}^{N} \texttt{ca}(\mathbf{y}_i, \hat{\mathbf{y}}_i) \tag{3.1}$$

Where $\mathbf{y}$ denotes the labels and $\hat{\mathbf{y}}$ denotes the predictions, N denotes the number of samples and `ca` is given as:

$$\texttt{ca}(y, \hat{y}) = \begin{cases} 1, & \operatorname{argmax}_c \sigma(y) = \operatorname{argmax}_c \sigma(\hat{y}) \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

where $\operatorname{argmax}_c \sigma(y)$ denotes the label with the highest probability. For example, the $\texttt{acc}\big(\big[[0.1, 0.9], [0.7, 0.3]\big], \big[[0.2, 0.8], [0.4, 0.6]\big]\big) = \frac{1}{2}(1 + 0) = 0.5$

For numerical classes, the mean squared error (MSE) (Equation 2.8) and the correlation ($\mathtt{r}$) between the predicted and actual age will be reported. The correlation is defined as:

$$r(\mathbf{y}, \hat{\mathbf{y}}) = \frac{(\mathbf{y} - \mu_{\mathbf{y}})(\hat{\mathbf{y}} - \mu_{\hat{\mathbf{y}}})}{\text{std}_{\mathbf{y}}\text{std}_{\hat{\mathbf{y}}}} \tag{3.3}$$

The MSE is a measure for how well the model fits the individual datapoints. Whereas the correlation is a measure for the global shape of the data. The MSE punishes large errors in the prediction, making it more sensitive to outliers. The correlation is a measure for the strength of the global relation, where a high correlation indicates a strong relation and a near-zero correlation indicates that there is no relation.

3.3 Results

This section presents the results of the aforementioned experiments. Here, the results will be evaluated to establish the best architecture for the problem of user quantification.

3.3.1 Architecture

This section is available upon request.

Chapter 4

Clustering

For learning the preferences of users, a lot of data is required. Traditionally, the interactions of a user with a system are tracked over a longer period of time. This information may then be used to learn the information needs of a particular user (Bouhini et al., 2016). When the information need is known, the system may be adapted to better suit the needs of the user.

For the application of personalized photosets summarization, it would also be desirable to have multiple interactions per user. When the user receives the result of the initial summarization, the user may include images in or exclude images from the selection. These interactions may then be used to learn which images a user would like to have in their summarization. If these preferences are incorporated in the model, they may be used to select images that would also be selected by the user.

One approach for user modelling is the following: Let a user have a photoset $\mathfrak{P}$. From $\mathfrak{P}$, generate a selection of photographs $s \subseteq \mathfrak{P}$ using model m. The user may then change the selection according to their preferences. The new selection is referred to as the optimal selection s^*. The goal is then to finetune m to generate an optimal model m^* that approximates the optimal selection s^*, given photoset $\mathfrak{P}$.

One disadvantage of this approach is that personalization can only happen after the user has already interacted with the system. Another problematic aspect is the point at which a selection is called optimal. When a user makes changes to selection s, resulting in s', it cannot be determined with certainty that selection $s' = s^*$. To this end, the following assumption is made: a predefined condition exists where it is known that the selection is optimal, iė. $s' = s^*$.

To circumvent these limitations, a different approach must be chosen. In this approach, a user should not be required to have interacted with the system before. This may be achieved by learning the preferences of users that are *similar*. Here, the users may be represented as user vectors $\mathfrak{u}$. Suppose $\mathbb{U}$ denotes the set of all user vectors. The goal is then to generate subsets G_k with similar user vectors. Here, every user in G_k should be more similar to the *typical* user vector $\bar{\mathfrak{u}}_k$ of G_k, than the *typical* user vector $\bar{\mathfrak{u}}_l$ of any other subset. This is formally given by the following property:

$$G_k = \{\mathfrak{u}_i : ||\mathfrak{u}_i - \bar{\mathfrak{u}}_k|| < ||\mathfrak{u}_i - \bar{\mathfrak{u}}_l|| \; \forall l \in \{1, \ldots, K\}, l \neq k\} \tag{4.1}$$

where $||\mathfrak{u} - \bar{\mathfrak{u}}||$ represents the similarity between a user vector and the typical user vector of a subset. Intuitively, the typical user vectors may be seen as the average user vector in a subset. This relies on the assumption that users with a similar representation, have a similar preference.

The typical user vectors are found by employing a clustering algorithm. The resulting clusters that may be found represent the different groups of users with similar preferences.

Clustering concrete representations is an easier task than clustering abstract representations. For clustering concrete representations, one valid approach is found by creating different combination of features. Numerical features may then be converted to categorical features through binning. Users may then be grouped based on these combinations.

Clustering abstract representations is a more complex task, as individual features may not have any meaning when binned. As mentioned in section 2.4, high-dimensional user vectors should first be reduced in dimensionality. This is done by mapping the high-dimensional user vector onto a low-dimensional space by applying UMAP. The resulting low-dimensional representations is then clustered using a clustering algorithm.

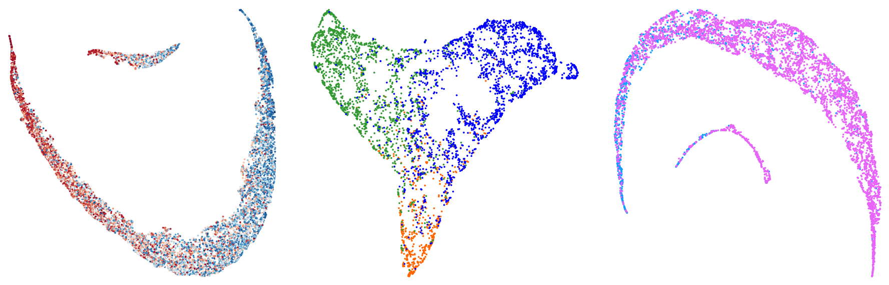
This chapter will focus on the latter: clustering abstract user vectors to find groups of similar users. First, a visualization of the low-dimensional user vectors is given to provide context for the clustering algorithms. Based on this visualization, several suitable clustering techniques are presented. Afterwards, several measures are presented for determining the quality of a given clustering. Finally, the different clusterings that are found by the presented algorithms are discussed and compared.

4.1 User Vector Visualization

In Chapter 3, different models were presented for computing concrete user vectors. Here, each feature was predicted by a different model. The user vectors that are used in this chapter consist of the concatenated user vectors that are found by every model.

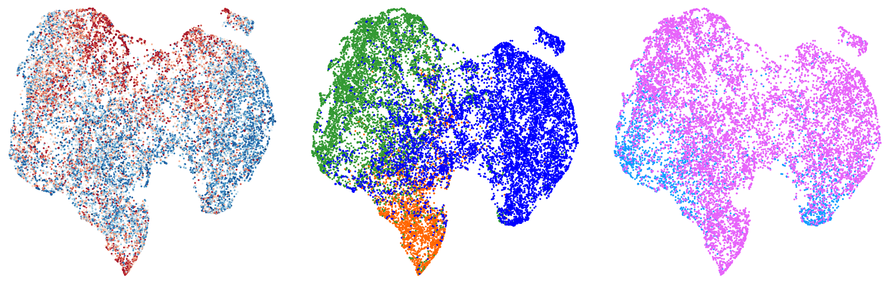
Clustering algorithms are found to yield a higher performance when the number of features is low (Lloyd, 1982). To this end, the high-dimensional user vectors are first reduced in dimensionality. As discussed in Section 2.4, UMAP will be used to transform the high-dimensional user vector to a compact low-dimensional representation. For visualization purposes, the user vector is reduced to two components. However, having more components increases the amount of information that can be represented. To this end, the user vectors are reduced to [redacted] dimensions when performing the experiments.

One important hyperparameter for UMAP is the minimum distance between two datapoints. By enforcing a larger minimum distance in the low-dimensional space, the differences in the local structure may be highlighted, which is beneficial for visualization. To this end, the minimum distance was set to 0.2 for visualization. However, for clustering, it is desirable to have densely distributed



(1) A (2) B (3) C

(a) User Vector Components



(1) A (2) B (3) C

(b) Combined User Vector

Figure 4.1: Visualizations of the user vectors, transformed to a two dimensional space. The colors represent the true labels of the different characteristics. (a) The individual components of the user vectors are separated and transformed in isolation. (b) The user vectors containing all components are transformed.

data (McInnes et al., 2018). To this end, the minimum distance is set to [redacted] when performing experiments.

The combined user vector consists of all characteristics. As every characteristic is learned in isolation, it is possible to transform and visualize each component separately. This highlights the learned features of every characteristic. The produced low-dimensional representations are found in in Figure 4.1a.

One important characteristic that is visible in the visualizations is that classes are localized, i.e. each class may be found in a specific location. For example, characteristic B is somewhat shaped like a triangle where the corners contain a specific class. It is worth noting that UMAP is an unsupervised dimensionality reduction technique, such that labels are not taken into account in the transformation. This indicates that the learned abstract representations correspond to the concrete user vectors.

Characteristic A and C exhibit similar features. Both characteristics are snake-shaped, where the classes are distributed like a spectrum. Here, each end of the snake contains a class. A closer examination of the photosets corresponding to the datapoints in the snake reveal that the position is correlated to the confidence of the network. Datapoints that are closer to a tail have a higher confidence than datapoints closer to the middle.

Another interesting property that is visible in both, `A` and `C`, is the smaller snake-shaped cloud of datapoints in the center. A closer examination shows that photosets in this cloud are distinctly different from photosets in the larger snake.

The visualization of the transformed user vector containing all characteristics may be seen in Figure 4.1b. The phenomenon of class localization that was found in the separate characteristics, is also visible in the visualization of the user vector containing all characteristics. This is especially prominent when examining the color overlay representing the different classes. There is one interesting property that arises from the color overlays: the main divisor of the combined user vector is characteristic `B`. The other characteristics are then localized inside `B`.

4.2 Clustering Algorithms

There are various clustering algorithms that may be used for clustering datapoints into clusters of similar datapoints. This may be done by exploiting the density of the data or by partitioning the data.

Density based clustering algorithms are most effective when clusters arise naturally from the data (McInnes, Healy, & Astels, 2017). An example of this is the UMAP and t-SNE embeddings of the MNIST example, found in Figure 2.5.

The low-dimensional UMAP embedding of the user vectors (Figure 4.1b) do not form distinct clusters. Instead, the embeddings are shaped like a cloud of datapoints. Density based clustering algorithms may find suboptimal clusters using these embeddings. To this end, this thesis only focuses on partitioning-based clustering algorithms. This class of clustering algorithms assigns data points to K nearby components. The goal of partitioning algorithms is to find these components. These components were previously referred to as the *typical* user vectors.

For both algorithms, the number of components K is a hyperparameter, which may be freely chosen. As such, K may be considered a hyper-parameter of this approach. K should be chosen experimentally, however the initial value may be chosen intuitively: Let C be the number of classes. When $K < C$, classes that are similar may be partitioned into the same cluster. When $K = C$, the resulting components may partition the data, such that every class is contained into one subset. When $K > C$, the partitioning becomes more specialized, i.e. partitions are formed within classes. These resulting partitions may then be seen as specific groups within the class.

Computing the optimal partitioning for a given number of datapoints is an intractable problem in Computer Science. To this end, various algorithms are designed to estimate an optimal partitioning instead. These algorithms often rely on the Expectation-Maximization (EM) algorithm (Dempster, Laird, & Rubin, 1977).

In this Section, two variants of partitioning based cluster algorithms will be examined: *Gaussian Mixture Models* (GMM) and *k-Means*. These variants are both iterative processes, where model parameters are updated using the EM Algorithm. To this end, Gaussian Mixture Models and k-means will be explained in the context of the EM algorithm.

For each algorithm, an example will be given where the goal is to partition characteristic `A`. This characteristic is chosen as the shape of the low-dimensional representation highlights defining properties of the algorithms.

The explanations for the algorithms will be given in terms of user clustering, where $\mathbb{U} = \{\mathfrak{u}_1, \ldots, \mathfrak{u}_N\}$ represents the set of all user vectors.

4.2.1 Gaussian Mixture Models

A Gaussian Mixture Models (GMM) (Dempster et al., 1977) is a collection of gaussian components that may be used to model the underlying distribution of the data. GMMs contain two learnable parameters: a set of components $C = \{C_1, \ldots, C_K\}$ and component weights $\theta = \{\theta_1, \ldots, \theta_K\}$. The goal of GMMs is to find K components such that the likelihood for user vectors belonging to the nearest component C_k is maximized. The components may be described as a gaussian distribution $\mathcal{N}$ with mean μ_k and covariance matrix Σ_k. This gaussian is denoted as $\mathcal{N}(\mu_k, \Sigma_k)$. Finding the optimal parameters is done using the EM algorithm.

The EM algorithm does not directly compute the optimal parameters. Instead, the EM algorithm maximizes the likelihood that the learned components generate the data. The reasoning behind this approach is the following: If a set of gaussian components is found from which drawn samples resemble the distribution of the data, then each components is said to be able to model a distinct group of datapoints in the dataset.

The EM algorithm optimizes the parameters using an iterative two-step approach: The first step is called the **E**xpectation step. This step consists of computing an expectation of the assignment for every user vector $\mathfrak{u}_i \in \mathbb{U}$ to each component C_k. This is a soft assignment where each photoset is assigned to each component with a probability. This is given by:

$$\hat{\gamma}_{ik} = \frac{\hat{\theta}_k \mathcal{N}(\mathfrak{u}_i | \hat{\mu}_k, \hat{\Sigma}_k)}{\sum_{j=1}^{K} \hat{\theta}_j \mathcal{N}(\mathfrak{u}_i | \hat{\mu}_j, \hat{\Sigma}_j)} \tag{4.2}$$

Where $\hat{\gamma}_{ik}$ is the estimated probability that user vector $\mathfrak{u}_i$ is generated by component C_k. $\mathcal{N}(\mathfrak{u}_i | \hat{\mu}_k, \hat{\Sigma}_k)$ represents the probability of generating user vector $\mathfrak{u}_i$ by a gaussian with mean μ_k and covariance matrix Σ.

The second step is called the **M**aximization step. This step consists of updating the model parameters to maximize the expectations from the E step. This is done by first updating the component weights, given by:

$$\hat{\theta}_k = \sum_{i=1}^{N} \frac{\hat{\gamma}_{ik}}{N} \tag{4.3}$$

Followed by updating the mean and covariance matrix for every component, given by:

$$\hat{\mu}_k = \frac{\sum_{i=1}^{N} \hat{\gamma}_i k \mathfrak{u}_i}{\sum_{i=1}^{N} \hat{\gamma}_i k} \tag{4.4}$$

$$\hat{\Sigma}_k = \frac{\sum_{i=1}^{N} \hat{\gamma}_i k(\mathfrak{u}_i - \hat{\mu}_k)^2}{\sum_{i=1}^{N} \hat{\gamma}_i k} \tag{4.5}$$

The result of the maximization step is a parameter configuration that is *more likely* to generate the data than the previous configuration. The likelihood that the parameter configuration generates the data may be defined by computing the log likelihood of the data given the parameters. This function is given by:

$$\log p(\mathbb{U}|C, \theta) = \sum_{n=1}^{N} log \left\{ \sum_{k=1}^{K} \theta_k \mathcal{N}(\mathfrak{u}_i | \hat{\mu}_k, \hat{\Sigma}_k) \right\} \tag{4.6}$$

Since the likelihood of the data is a value that may be computed, it is possible to mathematically establish whether the parameter configuration at iteration t is more likely to generate the data than the previous configuration at iteration $t-1$. The EM algorithm is ran until the increase in likelihood is smaller than a certain threshold, denoted by ϵ. Convergence is then established when the following condition is satisfied:

$$\log p(\mathbb{U}|C^{(t)}, \theta^{(t)}) - \log p(\mathbb{U}|C^{(t-1)}, \theta^{(t-1)}) \leq \epsilon \tag{4.7}$$

The component C_k that is most likely to generate user vector $\mathfrak{u}_i$ is given by:

$$C_k^* = \underset{k}{\operatorname{argmax}}\, \gamma_{ik} \tag{4.8}$$

Cluster assignment using GMM is soft, i.e. datapoints belong to a cluster with a probability p. It is then possible to set a threshold α for which a point does not belong to any cluster.

4.2.2 k-Means

K-means (Lloyd, 1982) is the most popular clustering algorithm in the field of data mining. K-means aims to partition datapoints into K clusters. In k-means, the datapoints are hard-assigned to a component. This is done by finding a component configuration that minimize the distance between each point and its closest component.

In k-means, the components are called clusteroids. Clusteroids only contain their position μ_k.

Finding the optimal parameters configuration is done by using the EM algorithm. Here, the E-step consists of assigning user vectors to the nearest clusteroid C_k, given by:

$$S_k^{(t)} = \left\{ \mathfrak{u}_i : ||\mathfrak{u}_i - \mu_k^{(t)}||^2 \leq ||\mathfrak{u}_i - \mu_l^{(t)}||^2 \forall_j, 1 \leq j \leq K \right\} \tag{4.9}$$

where S_k denotes the set of user vectors that are assigned to clusteroid k, $||\mathfrak{u}_i - \mu_k^{(t)}||^2$ denotes the squared euclidean distance between the user vector $\mathfrak{u}_i$ and the clusteroid position μ_k. The superscript (t) denotes the configuration at iteration t.

The M-step then consists of updating the centroid locations μ for every clusteroid. This is given by:

$$\mu_k^{(t+1)} = \frac{1}{|S_k^{(t)}|} \sum_{\mathfrak{u}_i \in S_k^{(t)}} \tag{4.10}$$

where $|S_k|$ denotes the number of user vectors assigned to centroid S_k.

As k-means is not a probabilistic model, a different metric should be used for establishing convergence. A commonly used metric to establish convergence is *inertia*: the sum of the within-cluster sum-of-squares of every centroid. Inertia is given as:

$$\textit{inertia}(\mathbb{U}|C) = \sum_{i=0}^{|\mathbb{U}|} \min_{\mu_j \in C}(||\mathfrak{u}_i - \mu_j||^2) \tag{4.11}$$

Convergence is then established when the following condition is satisfied:

$$\textit{inertia}(\mathbb{U}|C^{(t)}) - \textit{inertia}(\mathbb{U}|C^{(t-1)}) \leq \epsilon \tag{4.12}$$

User vector $\mathfrak{u}$ is then assigned to the nearest centroid C_k^*, given by:

$$C_k^* = \operatorname*{argmin}_k ||\mathfrak{u} - \mu_k|| \tag{4.13}$$

Finding the optimal parameters of k-means may be seen as a specific configuration of a Gaussian Mixture model, where Σ is an identity matrix, multiplied with a very small value β.

This may be shown by integrating this into Equation 4.2:

$$\hat{\gamma}_{ik} = \frac{\hat{\theta}_k \exp\{-(\mathfrak{u}_i - \mu_k)^2/2\beta\}}{\sum_{j=1}^{K} \hat{\theta}_j \exp\{-(\mathfrak{u}_i - \mu_j)^2/2\beta\}} \tag{4.14}$$

When $\lim_{\beta \to 0}$, the component C_k that minimizes $(\mathfrak{u}_i - \mu_k)^2$, approaches zero more slowly than the other components. This results in the expectations for components $\gamma_{i,j \neq k}$ becoming nearly zero and γ_{ik} nearing 1.

The main differences between k-means and GMMs may be found in the E-step: K-means hard assigns datapoints and GMMs soft-assigns datapoints. K-means does not compute the probability of datapoints being generated by a component, but merely computes the distance between every component and the datapoint. This results in k-means being significantly faster than GMMs.

4.3 Determining Partitioning Quality

The evaluation of a partitioning is not a trivial task. Commonly used measures for determining the quality of a partitioning are classification measures, such as completeness, homogeneity and the v-measure. However, in this Chapter, classification is not the goal. Instead, the goal is to partition the data into K clusters.

The completeness score measures the *completeness* of the global partitioning. Here, a *complete* clustering is defined as a clustering where all datapoints belonging to one class are assigned to the same cluster. Intuitively, this is a good measure for penalizing partitions that are *too specific*. Where too specific is defined as a clustering in which users are found which do not generalize well to unseen data. Consequently, this measure penalizes the splitting of clusters.

Another measure that may be examined is the homogeneity score. The homogeneity score is a measure for defining the *purity* of the partitioning. If clusters contain only one class, then the partitioning is called pure. One important property that arises from this definition is that homogeneity does not penalize splitting clusters into smaller clusters. This is a desirable property, as is evident from the previous example. The homogeneity is given by:

$$\textit{homogeneity}(C, K) = 1 - \frac{H(C|K)}{H(C)} \tag{4.15}$$

where $H(C|K)$ is the conditional entropy of a class C given the cluster assignments K, given by:

$$H(C|K) = -\sum_{c=1}^{|C|}\sum_{k=1}^{|K|} \frac{n_{c,k}}{n} \times \log\left(\frac{n_{c,k}}{n_k}\right) \tag{4.16}$$

and $H(C)$ is the entropy of the class, given by:

$$H(C) = -\sum_{c=1}^{|C|} \frac{n_c}{n} \log\left(\frac{n_c}{n}\right) \tag{4.17}$$

where n denotes the total number of samples, n_c and n_k denote the total number of samples respectively belonging to label c and cluster k, and finally $n_{c,k}$ denotes the number of samples with label c assigned to cluster k.

Suppose we have user vectors that correspond to $C = [a, a, b, b]$ and cluster assignments $K_1 = [0, 0, 1, 1]$. This partitioning is perfectly homogeneous as $\textit{homogeneity}(C, K_1) = 1.0$. Whereas the cluster assignment $K_2 = [0, 1, 0, 1]$ yields $\textit{homogeneity}(C, K_2) = 0.0$. It should be noted that splitting cluster assignments K_1 to $K_3 = [0, 1, 2, 3]$ also gives a perfectly homogeneous partitioning, as $\textit{homogeneity}(C, K_3) = 1.0$, which is a *too specific* partitioning. As the completeness score is not a valid measure, a method for determining *too specific* clusters is desired. To this end, the average number of clusters in a given partitioning should also be taken into account. When the number of clusters is small and the homogeneity is high, this might be an indication that the partitioning is *too specific.*

The optimal clustering algorithm and values for K will be determined by performing experiments. These experiments will entail the partitioning of the combined user vector using the two algorithms and various values for K. The data that will be used for clustering consists of the user vectors corresponding to the photosets in the test set of [redacted]. This was done to prevent contamination of the dataset with partitionings that are specific to the training or validation set that may be the result of overfitting.

For every clustering algorithm and k, the test set is split into two subsets: one set for fitting the model, containing 75% of the data, and one set for validation, containing the remaining 25% of the data.

4.4 Clustering Evaluation

This section is available upon request.

Chapter 5

Learning Personal Relevance

This chapter is available upon request.

Chapter 6

Result Analysis

In Chapter 4, a visualization was provided of the user vector space. This visualization provided interesting insights into the modelling of user context. Here, we may see that the learned user vectors are representative for the characteristics of a user.

However, it does not show what features in images are beneficial for a given characteristic. Visualizing the models of different characteristics would provide more insight into the reasoning of a neural network.

Several approaches have been proposed for visualizing and explaining the predictions of a neural network. One such method was examined in Section 2.1.2. Here, the filters of convolutional units were visualized by generating images that strongly activate the filter. In addition to this approach, two other approaches will be examined: visualizing the gradients of the class activations and visualizing the convolutional filters with respect to the image.

In this Chapter a solution will be presented for training a convolutional neural network that uses only one image for classification. This network will then be used for the three aforementioned visualization approaches.

6.1 Training a Neural Network for Visualization

Since the visualization CNN may only use one image for making predictions, the model architecture and training setup requires some changes. These changes will be examined in this section. First, the neural network architecture will be discussed. Afterwards, the new training procedure is discussed. Finally, the performance of the trained network is discussed.

6.1.1 Architecture

For training a neural network for visualization, the input is restricted to only one image. The main interest is the visualization of feature-maps, activations and gradients. To this end, a convolutional neural network architecture is chosen.

The visualization approaches used in this Chapter, rely on gradients that are computed using back-propagation. To this end, the convolutional architecture that will be used is the Inception V3 architecture. The Inception architectures are commonly used for visualization tasks as they provide clear and colorful visualizations (Olah, Mordvintsev, & Schubert, 2017; Olah et al., 2018).

The results from Chapter 3 show that characteristic `A` and `C` yield the highest performance when trained on [redacted] [redacted] [redacted]. Characteristic `C` yields the highest performance when trained on [redacted] [redacted] [redacted]. To this end, an Inception V3 architecture is trained for each characteristics.

The training procedure for training the visualization network is almost identical to that of training the architectures of Chapter 3. The only difference is the sampling of training samples: instead of sampling photosets, images are sampled from a new dataset. The construction of this dataset is discussed in Section 6.1.2. The following image augmentation were applied: Flipping the image vertically, random Gaussian Blurs and cropping followed by resizing. More information on these augmentations may be found in Jung (2018).

The training parameters for this approach are similar to those of Chapter 3.

6.1.2 Dataset Construction

Training a convolutional neural network for visualization requires the input to contain only one image, which requires some changes to the data pipeline presented in section 3.2.2.

One solution to this problem is to randomly select an image from the photoset and use the corresponding label of the full photoset. However, not every image contributes equally to a correct prediction. To this end, a different approach was chosen that includes teacher-student learning. Here we leverage the learned knowledge of a fully trained teacher network to teach the student network. For the teacher network, the architectures from Chapter 3 will be used. The student network is the Inception V3 architecture.

Every photograph $\mathfrak{p} \in \mathfrak{P}$ contributes to the correct prediction p_c of a class c. However, the contribution of images is not equal. The exists a subset of images s that positively contribute to the correct prediction and a subset of images $\bar{s}$ that negatively contribute to the correct prediction, i.e. lowers the confidence of the model.

For training the student network, images from s should be used, instead of $\bar{s}$. Selection s may be found by leveraging the teacher network, denoted by t. This is done by first computing the prediction on true class c using the full photoset $p_c = t(\mathfrak{P})$. Afterwards, every image in $i \in \{1, \dots, |\mathfrak{P}|\}$ may be removed from the photoset, resulting in $\mathfrak{P}_{\bar{i}}$. The prediction on class c, without photo i is then given as $p_{ci} = t(\mathfrak{P}_{\bar{i}})$. We may then construct a function $f(p_{ci}, p_c)$, such that $f(p_{ci}, p_c) = 1$ indicates that image i belongs in s and if $f(p_{ci}, p_c) = 0$, image i belongs in $\bar{s}$. For categorical characteristics, f is given as:

$$f(p_{ci}, p_c) = \begin{cases} 1, p_{ci} \geq p_c \\ 0, p_{ci} < p_c \end{cases} \tag{6.1}$$

since leaving out image i yields a lower probability on class c being the true class. The most influential images s may then be sorted such that $s = [\mathfrak{p}_j : p_{cj} > p_{c(j+1)}]$.

However, the prediction for numerical characteristics is only a single value, such that a comparison would not indicate a less confident prediction. To this end, the subscript c is dropped and f is given as:

$$f(p_i, p) = ||p_i - p|| \tag{6.2}$$

Where higher values of f indicate that leaving out image i leads to a larger error in the prediction, such that photo i has a positive influence for predicting the correct value. s then consists of every image and is sorted as follows $s = [\mathfrak{p}_j : f(p_j, p) > f(p_{j+1}, p)]$.

For training the student network, only the most influential images should be taken from s. For categorical characteristics, s consists of only influential images such that every image may be selected. For numerical characteristics, a boundary ϵ may be chosen to determine the most influential images. This sampling method will be referred to as `(A)`.

A different method for selecting the most influential images is to choose the first K images from the sorted list s, where $K = 0.1 \times |\mathfrak{P}|$. This ensures that from all influential images, only the most influential images are used for training. This sampling method will be referred to as `(B)`.

Finally, the dataset for training is constructed by joining all influential images in set s for every photoset $\mathfrak{P}$.

6.1.3 Results

This section is available upon request.

6.2 Generating Optimal Image Inputs

Convolutional Neural Networks are driven by convolutional layers. As explained in Section 2.1.2, convolutional layers contain a number of filters. In the convolutional layers, a convolution of the input and a filter is performed for every filter in the layer, resulting in H feature maps.

One method for visualizing the optimal activation for a specific filter is the following: First, generate a random image, containing pseudo-random noise. Second, feed the image to the network to compute the feature map of the corresponding filter. Third, change the input such that the feature map is maximized. Changing the image is done by computing the gradients of the feature map with respect to the input image. The second and third steps are then repeated for a number of iterations.

The resulting input that optimally activate a certain filter is also commonly referred to as a *deep dream*. More information about this process is found in Olah et al. (2017) and Olah et al. (2018).

The Inception V3 architecture contains 16 blocks of convolutional layers. Every block responds to increasingly more complex patterns in the data. In the following example, filters at different depths will be visualized. Here, a depth of 1 refers to the first block and a depth of 16 refers to the 16th block.

The feature map visualizations of three networks at various depths are given in Figure 6.1. At depth 1, the filters of the three networks seem to be activated by colors. At depth 5, more advanced patterns arise. Here, the three networks react to lines in different directions. Depth 9 is where the three networks start to deviate. The filter of the characteristic `C` reacts to complex shapes. The filters of characteristic `A` and `B` react to similar looking patterns. Starting at depth 13, all networks start to exhibit specialized filters for solving the corresponding classification problem. At depth 16, which corresponds to the last block in the network, the filters react to specific patterns in the data. This block contains the last convolutional layer before the prediction layer. The features that are found in this block are the features that are used for making predictions.

6.3 Saliency Maps Generation

We now discuss how regions in the image relate to the predictions of the neural network, to better understand the learned features of the convolutional neural network. This is done by computing saliency maps. Saliency (Simonyan et al., 2013) represents the relative importance of the inputs for computing the gradient of the output class. This is done by computing the gradients of class activation S_c with respect to the input image I, given by:

$$\frac{\partial S_c}{\partial I} \tag{6.3}$$

The result is a heatmap that highlights the pixels that are important for making the prediction. Here, pixels that are more important have a higher color intensity.

Due to non-differentiable activation functions, the resulting heatmap contains noise. To this end, the softmax or sigmoid activation on the last layer is replaced with a linear activation function ($f(x) = x$). To combat the noise of the remaining ReLU activation functions, Guided Backpropagation (Springenberg, Dosovitskiy, Brox, & Riedmiller, 2014) is used. Guided Backpropagation tries to remove noise from the produced activation map by preventing the flow of negative gradients from units that decreases the activation of the feature that is visualized. This step is performed during back propagation, where gradients that have a negative effect on the class activation are set to zero.

For visualizing the saliency maps, we consider the three images in Figure 6.2.

6.4 GradCAM

The second approach for visualizing the model activations, is visualizing the gradients directly. This is done using GradCAM (Selvaraju et al., 2016). GradCAM is a generalization of the CAM algorithm (Zhou, Khosla, Lapedriza, Oliva, & Torralba, 2015).

GradCAM is similar to visualizing saliency maps. The difference between the two methods is the layer for which the gradients should be computed. Instead of computing the gradients of the output layer with respect to the input, the gradients of the last convolutional layer, before activation, are computed with respect to the input.

As discussed in Section 2.1.2, convolutional layers consist of H filters. The gradients should be computed for every filter G_h in the convolutional layer with respect to the input image I. This is given by:

$$\frac{\delta G_h}{\delta I} \tag{6.4}$$

The gradients are then stacked and normalized. Normalization is performed by dividing the stacked gradients with the maximum gradient in the stack. Afterwards the mean is computed over all maps, resulting in a gradient map with dimensions (j, k). The resulting gradient map is then multiplied with the output of every filter $G_h \in G$ and activated using ReLU activation, resulting in the GradCAM heatmap. Finally, the heatmap is upscaled to match the dimensions of the input image, $(224, 224)$.

In Figure 6.3, the GradCAM maps are presented for the same three images that were used for computing the saliency maps in Section 6.3. Here, a higher color intensity denotes a larger gradient which corresponds to more important pixels.

Feature Map Visualizations

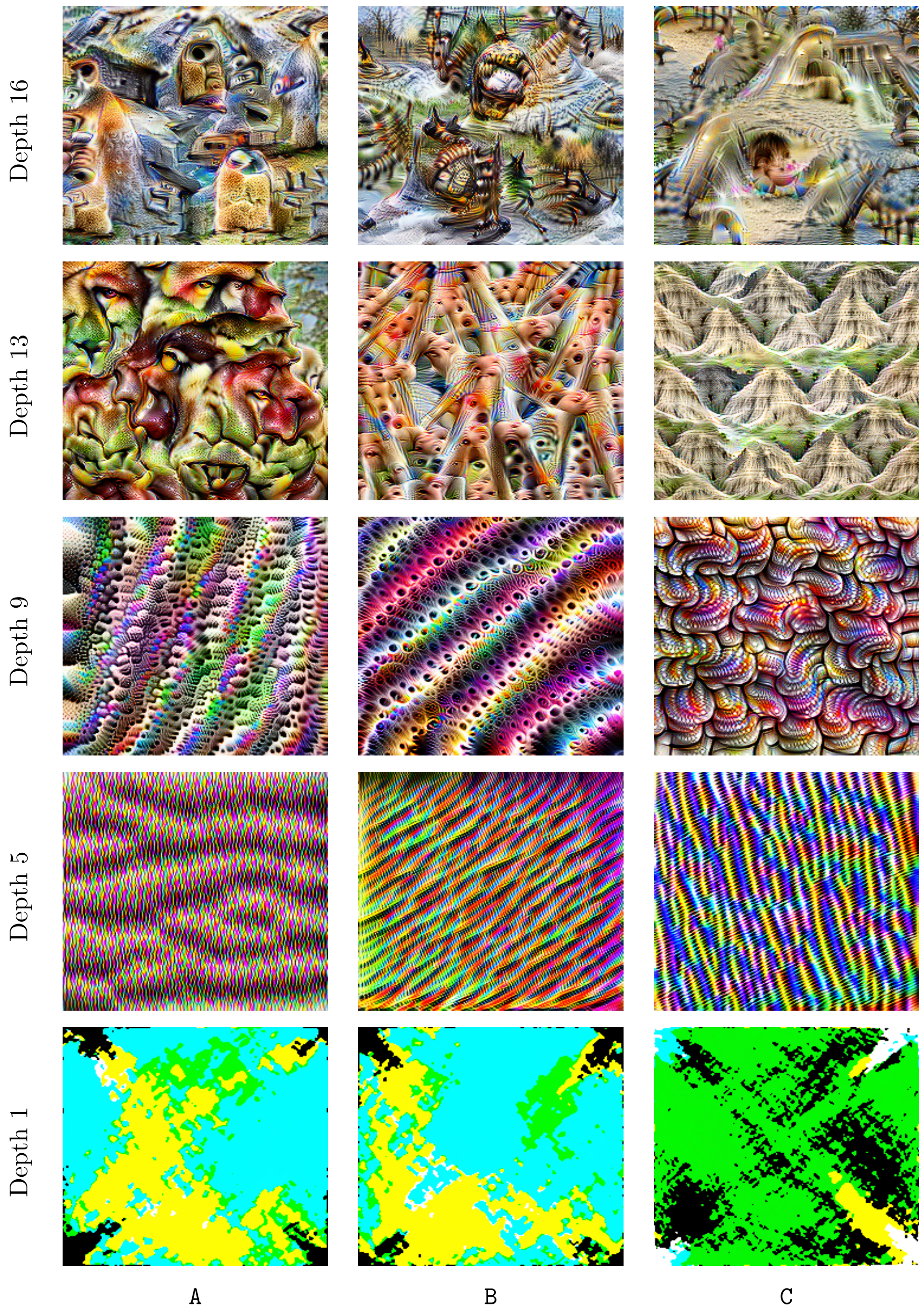


Figure 6.1: Feature map visualizations on varying depths.

Saliency

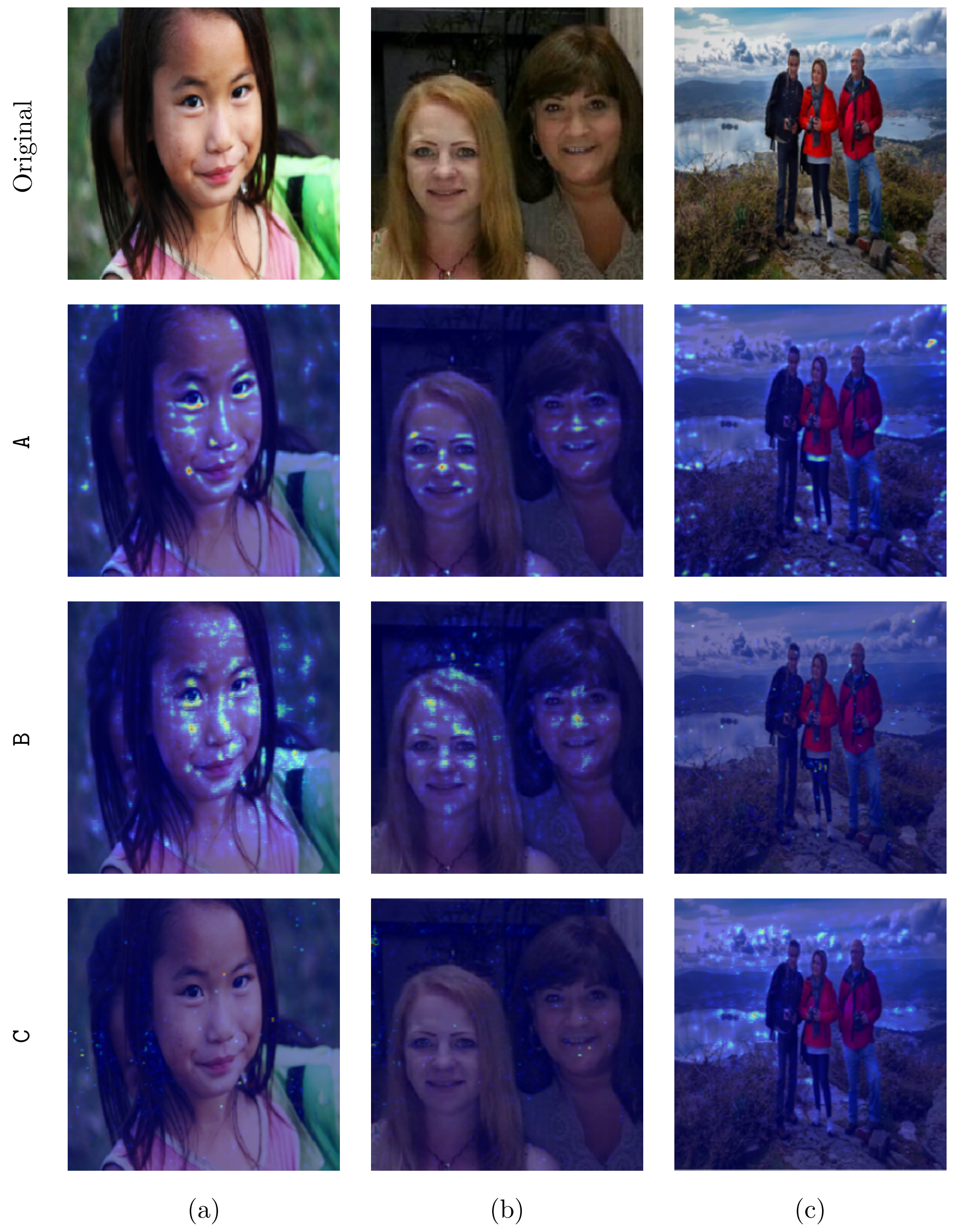


Figure 6.2: Examples of saliency maps for three images. A higher saliency corresponds to a higher color intensity. Every row corresponds to a different class. Every column corresponds to a different image.

GradCAM

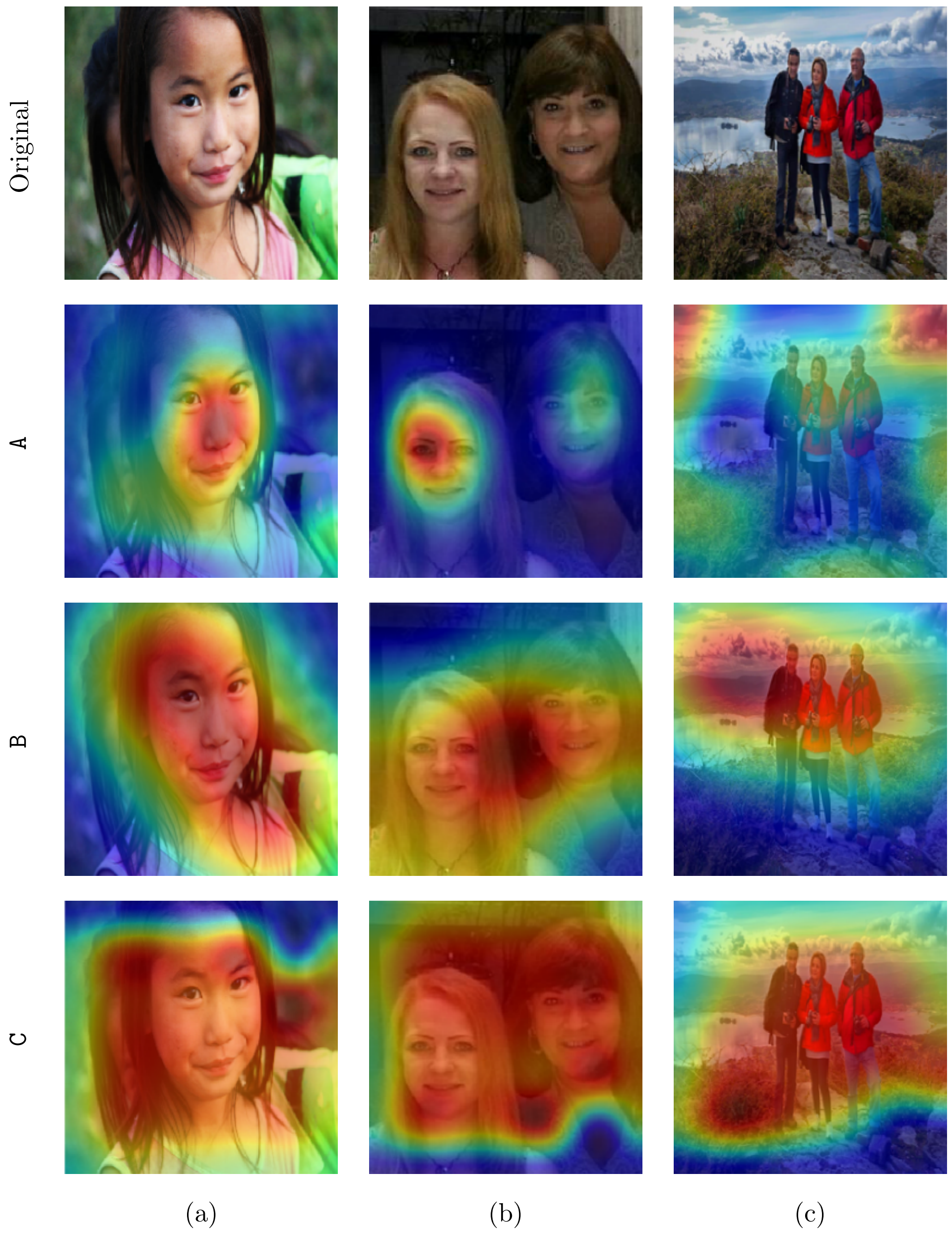


Figure 6.3: Examples of GradCAM maps for three images. A higher color intensity corresponds to a larger gradient. Every row corresponds to a different class. Every column corresponds to a different image.

Chapter 7

Discussion

This chapter is available upon request.

Chapter 8

Conclusion

This chapter is available upon request.

Glossary

Notation	Description	Symbol	Page List
convergence	Training a model until convergence means that the model is trained such that a given (loss) metric does not change anymore.		16
convex	Optimization functions that are convex have only one optimal solution, i.e. a global optima or minima. When a function is non-convex, more than one solution exists. Not every solutions found by a non-convex optimization may be optimal.		16
data pipeline	All steps that are taken for retrieving the data up to using the data for a machine learning model.		5
feature	Neural Networks learn abstract patterns in the data, these are called features.		5
feeding	In the context of neural networks, feeding an input is referred to presenting the neural network with some input and computing an output.		6
finetuning	Finetuning a neural network involves training a neural network on data using a very low learning rate. This ensures that the original knowledge is not lost but is specialized towards the data used for finetuning.		25
multivariate dataset	Datasets with two or more features.		15
one-hot	A one-hot encoded vector is a vector containing only zeros except for one element, which is a 1. For example, if the maximum value is 5 and the number 2 should be one-hot encoded, the one-hot encoding is given as $[0, 0, 1, 0, 0, 0]$.		8
overfitting	When a model has overfitted, it has fitted the training data to an extent that is does not generalize well to unseen data.		20

Notation	**Description**	**Symbol**	**Page List**
photoset	A collection of photographs of a user.	$\mathfrak{P}$	5
ranking	A list that is explicitly ordered from best to worst.		5
representation	A defining description.		5
spatial information	This refers to the information that may be gained from one sample, e.g. the visibility of a dog in an image.		11
temporal information	This refers to the information that may be gained from looking at the ordering of photographs in a photoset.		11
user	The person creating a summarization.	u	5

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems.* Retrieved from http://tensorflow.org/ (Software available from tensorflow.org)

Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M. S., ... Asari, V. K. (2019). A state-of-the-art survey on deep learning theory and architectures. *Electronics*, *8*(3), 292.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bishop, C. M. (2006). *Pattern recognition and machine learning.* springer.

Bouhini, C., Géry, M., & Largeron, C. (2016). Personalized information retrieval models integrating the user's profile. In *2016 ieee tenth international conference on research challenges in information science (rcis)* (pp. 1–9).

Chan, D. M., Rao, R., Huang, F., & Canny, J. F. (2018). t-sne-cuda: Gpu-accelerated t-sne and its applications to modern data. *CoRR*, *abs/1807.11824*. Retrieved from http://arxiv.org/abs/1807.11824

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, *39*(1), 1–22.

Edunov, S., Ott, M., Auli, M., & Grangier, D. (2018). Understanding back-translation at scale. *CoRR*, *abs/1808.09381*. Retrieved from http://arxiv.org/abs/1808.09381

Farfade, S. S., Saberian, M. J., & Li, L.-J. (2015). Multi-view face detection using deep convolutional neural networks. In *Proceedings of the 5th acm on international conference on multimedia retrieval* (pp. 643–650).

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning.* MIT press.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, *abs/1704.04861*. Retrieved from http://arxiv.org/abs/1704.04861

Hu, J., Shen, L., & Sun, G. (2017). Squeeze-and-excitation networks. *CoRR*, *abs/1709.01507*. Retrieved from http://arxiv.org/abs/1709.01507

Jones, K. S., Walker, S., & Robertson, S. E. (2000). A probabilistic model of information retrieval: development and comparative experiments: Part 2. *Information processing & management*, *36*(6), 809–840.

Jung, A. B. (2018). *imgaug.* https://github.com/aleju/imgaug. ([Online; accessed 30-Oct-2018])

Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., & Fei-Fei, L. (2014). Large-scale video classification with convolutional neural networks. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 1725–1732).

Kiefer, J., & Wolfowitz, J. (1952, 09). Stochastic estimation of the maximum of a regression function. *Ann. Math. Statist.*, *23*(3), 462–466. Retrieved from https://doi.org/10.1214/aoms/1177729392 doi: 10.1214/aoms/1177729392

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Lipton, Z. C. (2015). A critical review of recurrent neural networks for sequence learning. *CoRR*, *abs/1506.00019*. Retrieved from http://arxiv.org/abs/1506.00019

Lloyd, S. (1982). Least squares quantization in pcm. *IEEE transactions on information theory*, *28*(2), 129–137.

Ma, S., & Dai, Y. (2011). Principal component analysis based methods in bioinformatics studies. *Briefings in bioinformatics*, *12*(6), 714–722.

Maaten, L. v. d. (2009). Learning a parametric embedding by preserving local structure. In *Artificial intelligence and statistics* (pp. 384–391).

Maaten, L. v. d., & Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, *9*(Nov), 2579–2605.

McInnes, L., Healy, J., & Astels, S. (2017, mar). hdbscan: Hierarchical density based clustering. *The Journal of Open Source Software*, *2*(11). Retrieved from https://doi.org/10.21105%2Fjoss.00205 doi: 10.21105/joss.00205

McInnes, L., Healy, J., & Melville, J. (2018). Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (icml-10)* (pp. 807–814).

Nwankpa, C., Ijomah, W., Gachagan, A., & Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*.

Olah, C., Mordvintsev, A., & Schubert, L. (2017). Feature visualization. *Distill*, *2*(11), e7.

Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K., & Mordvintsev, A. (2018). The building blocks of interpretability. *Distill*. (https://distill.pub/2018/building-blocks) doi: 10.23915/distill.00010

Olivas, E. S., Guerrero, J. D. M., Sober, M. M., Benedito, J. R. M., & Lopez, A. J. S. (2009). *Handbook of research on machine learning applications and trends: Algorithms, methods and techniques - 2 volumes.* Hershey, PA: Information Science Reference - Imprint of: IGI Publishing.

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *2*(11), 559–572.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). *Language models are unsupervised multitask learners.*

Richter, F. (2017, 8). Smartphones cause photography boom. *Statista*. (https://www.statista.com/chart/10913/number-of-photos-taken-worldwide/)

Robertson, S. E. (1977). The probability ranking principle in ir. *Journal of documentation*, *33*(4), 294–304.

Roth, H. R., Lu, L., Seff, A., Cherry, K. M., Hoffman, J., Wang, S., ... Summers, R. M. (2014). A new 2.5 d representation for lymph node detection using random sets of deep convolutional neural network observations. In *International conference on medical image computing and computer-assisted intervention* (pp. 520–527).

Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, *5*(3), 1.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... others (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, *115*(3), 211–252.

Selvaraju, R. R., Das, A., Vedantam, R., Cogswell, M., Parikh, D., & Batra, D. (2016). Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, *abs/1610.02391*. Retrieved from http://arxiv.org/abs/1610.02391

Setio, A. A. A., Ciompi, F., Litjens, G., Gerke, P., Jacobs, C., Van Riel, S. J., ... van Ginneken, B. (2016). Pulmonary nodule detection in ct images: false positive reduction using multi-view convolutional networks. *IEEE transactions on medical imaging*, *35*(5), 1160–1169.

Simonyan, K., Vedaldi, A., & Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.

Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2014). Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*.

Su, H., Maji, S., Kalogerakis, E., & Learned-Miller, E. (2015). Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the ieee international conference on computer vision* (pp. 945–953).

Sun, Y., Chen, Y., Wang, X., & Tang, X. (2014). Deep learning face representation by joint identification-verification. In *Advances in neural information processing systems* (pp. 1988–1996).

van Gerven, M., & Bohte, S. (2018). *Artificial neural networks as models of neural information processing.* Frontiers Media SA.

Werbos, P. J., et al. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550–1560.

Wiener, J. (2014, 10). Facebook's top open data problems. *Facebook Research*. (https://research.fb.com/facebook-s-top-open-data-problems/)

Yue-Hei Ng, J., Hausknecht, M., Vijayanarasimhan, S., Vinyals, O., Monga, R., & Toderici, G. (2015). Beyond short snippets: Deep networks for video classification. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 4694–4702).

Zhang, C., Bengio, S., Hardt, M., Recht, B., & Vinyals, O. (2016). Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*.

Zhou, B., Khosla, A., Lapedriza, À., Oliva, A., & Torralba, A. (2015). Learning deep features for discriminative localization. *CoRR*, *abs/1512.04150*. Retrieved from http://arxiv.org/abs/1512.04150