university of groningen / faculty of mathematics and natural sciences / artificial intelligence

Dynamic Label Propagation in a Lifelong Machine Learning Context

Bachelor's Project Thesis

Arjan Jawahier, a.jawahier@student.rug.nl,
Supervisor: prof. dr. L.R.B. Schomaker

Abstract: Transcription of historical handwritten documents is an important domain within machine learning. However, it is far from a solved problem. Monk is an engine used for transcribing handwritten texts, and the spotting of words in those texts. It uses the input from volunteers to train its classifiers, getting better with each label. However, the labeling process can still be improved. In this thesis, a new method of acquiring word labels is developed and tested. A convolutional neural network is created to rate lists of word instances on their visual structure. This rating is then used together with a sampling technique called the Metropolis-Hastings algorithm to find out which word instances to label for the biggest increase in average F_1-score. Using this technique, probability distributions are created and subsequently used in sampling and labeling simulations. The convolutional neural network has learned how to rate lists of word instances well, with a reasonable mean squared error and a low standard deviation. The results of the sampling and labeling simulations show that using the generated distributions is not very effective. Uniform random guessing is better than using the generated distributions.

1 Introduction

Automatic transcription of historical handwritten documents, also known as the process of handwriting recognition, is an important domain within machine learning. Although numerous character classifiers have been developed (Philbin, Chum, Isard, and Sivic, 2011; Ramel, Sidère, and Rayar, 2013), handwriting recognition is often not as simple as classifying each individual character and concatenating them all to form a textual transcription. This follows from the fact that many historical writing styles contain extensive use of contractions and loops (e.g. see Figure 1.1), which makes mapping characters to letters a lot harder (van Oosten and Schomaker, 2012). This observation leads to another natural approach: Whole word classification, where the total textual element is recognised as being a member of a word class.

Figure 1.1: Some examples that illustrate why character classification and concatenation does not work in practice. The loops and contractions shown here are common occurrences, not the exception.

Word classification from images is usually done in several steps. First, a page from a document is segmented into line images, which are processed in their entirety. Usually, a convolutional neural network (CNN) is used to extract sequential fea-

tures from the 2D shape of line images. A (bidirectional) Long Short Term Memory Neural Network (LSTM or BLSTM) (Graves and Schmidhuber, 2005) connected to a connectionist temporal classification (CTC) output layer then uses those sequential features to classify the words occurring within the input line. (Graves, Fernández, Gomez, and Schmidhuber, 2006). This form of handwriting recognition is often called handwritten text recognition (HTR). This approach is attractive, because of the possible compositionality: Character sequences can be recognised, if they were not part of training set. However, this is only possible given a large training set and a linguistic text corpus.

One method of obtaining a decent HTR system is utilizing a computer-assisted transcription (CAT) framework (Serrano, Giménez, Civera, Sanchis, and Juan, 2014). Such a framework might allow a user of the system to correct the transcriptions made by the classifier with minimal effort. One HTR system that is comparable to the system described by (Serrano et al., 2014) is called *Transkribus*[*]. *Transkribus* learns word labels efficiently by using the input from its many users. It is a downloadable transcription system that users can use to transcribe the handwritten documents that they possess. The transcriptions made by the users are then used to train the recurrent neural network under the hood, making it better at recognizing words, abbreviations and other information in handwritten documents.

Techniques from the domain of active learning can be used to augment the CAT framework, as is done by (Serrano et al., 2014). These techniques allow systems without an abundant amount of labeled training data to choose which unlabeled data to present to the user, which - when labeled - would have a relatively large impact on the remainder of the learning algorithm (Settles, 2009).

Systems that attempt to automatically transcribe historical handwritten documents often receive new data. This happens for example when new documents become available to the system. Moreover, it can happen that different users of the system fail to agree on the class of a word. Thus the label of a word instance can change over time. In such cases, many traditional machine learning techniques require a retraining of their model. This is usually computationally expensive and costs a lot of time, especially in cases where the transcription systems have access to large amounts of data. Because of this reason, combined with the ever increasing need for more labels, illustrates why traditional machine learning algorithms are not the answer to this problem. A way to tackle this problem is to make use of techniques from the domain of lifelong machine learning (LML) (Liu, 2017).

Systems that make use of LML can be said to learn continuously, have an internal storage of knowledge (i.e. a knowledge base) and use that knowledge in future learning tasks (Liu, 2017; Chen, Ma, and Liu, 2015). LML has been shown to be useful in a considerable number of recent studies, for learning tasks within the domains of supervised learning, unsupervised learning and reinforcement learning (Silver, Yang, and Li, 2013). The challenge posed to artificial intelligence (AI) researchers working with LML is to combine learning algorithms with knowledge representations. However, if this is done right, the superiority of LML over traditional machine learning algorithms can often be shown (Silver et al., 2013).

The system described by (van der Zant, Schomaker, Zinger, and van Schie, 2009) uses a CAT framework and LML for the annotation of handwritten words. Their system (called *Monk*) is at its core a word retrieval system. Users of *Monk* are interested in a specific string of characters that might be present in a historical document. Annotating the text beforehand is a necessary step in order to satisfy the needs of the end user. In one of the training processes, *Monk* tries to find certain words based on a query string. It then returns a so called "hit list" of word images from the document to the user. This hit list contains word images that were classified as the word in the query according to the distance to the separating boundary. The word images were then ranked according to their prototypicality, so the best hits are shown first (van Oosten and Schomaker, 2012). The user can either annotate or choose to agree or disagree with the current label of each word image, depending on the training mode. This data is then immediately used in the learning process, which propagates labels to other similar word images.

One strength of *Monk* is that it uses whatever technique yields the best results. It is not only

[*]For more information about *Transkribus*, visit https://transkribus.eu/Transkribus/

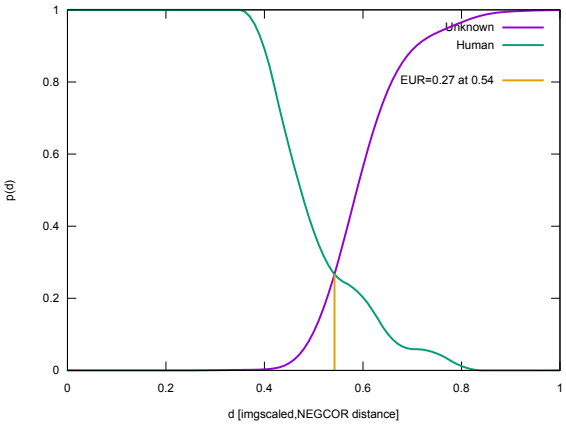


Figure 1.2: The False Reject Rate (FRR) and False Accept Rate (FAR) curves for word images in the hit list of the word "because". On the horizontal axis, the distance to the prototype of "because" is shown. On the vertical axis, the probability of either falsely rejecting (green curve) or falsely accepting (purple curve) are shown. Note that when the horizontal distance between the two curves increases, the separability of the word increases as well.

performing the line of text recognition done in HTR systems. Word image classifiers and character image classifiers (e.g. Chinese characters) are also present within the system (He and Schomaker, 2018). This means *Monk* does not suffer from the requirement that a lot of data is needed to create an accurate linguistic data model. For a more descriptive overview of the *Monk* system, the interested reader is referred to (van der Zant et al., 2009; Schomaker, 2016; van Oosten and Schomaker, 2012).

Active learning based on distances of candidates in a hit list (Figure 1.2) or on False Acceptance Rate/False Rejection Rate curves showed promising use for active learning within the *Monk* system. However, the returned hit lists often presented word images with low distances to the prototype, which were obviously of a different word class. This could be the case because of a number of reasons. To increase accuracy, or better yet, the F_1-score of a classifier, more labels are required. A way of harvesting labels in a more effective way could be a good start for this.

When a user queries a word, *Monk* outputs a ranked hit list in which the order of the images shown is based on the prototypicality of each word image (i.e. the degree to which a given instance of a word is similar to a canonical prototype) (van Oosten and Schomaker, 2012). Based on this approach, the order of the word images shown will influence the label propagation step in the next step of the learning process. For instance, should a labeler focus on words that are already well recognized and close to the canonical prototype, or alternatively, should they focus on hits further down the hit list? The problem lies in finding influential word instances. The aim of this study is to implement an algorithm in *Monk*, such that the word images presented to the user will be highly influential in the label propagation algorithm, if they are annotated. The underlying research question is: How can word images be sampled from a hit list, such that they exhibit the highest possible impact on the lifelong learning process when they are labeled?

To this end, the main aim of this study is to develop a sampler that can present the user with influential word samples. If the user so chooses, the sampler could present the user with raw and unlabeled data only, such that labeling these single images would increase the classifier's ability to recognize handwritten words.

2 Methods

The sampler will be described in detail in the next section. Two major components that were needed for the development and evaluation of the sampler are described in this section. These components are: A CNN that takes an input hit list image and outputs a rating that signifies how structured the hit list seems, and a *Nearest Centroid Classifier* that computes which label should be assigned to an input word image.

The main idea behind the use of the CNN is that the given ratings can be used as indicators of places in a hit list where the most influential word images reside. Influence can be defined in numerous ways. However, the following definition is used in this study: If f stands for the average F_1-score of all hit lists, then the influence of labeling instance A is higher than the influence of labeling instance B, iff. the labeling of instance A elicits a bigger increase

in f than labeling instance B. The goal is to sample from a hit list with a given rating, such that the increase in average F_1-score is maximal (Equation 2.1):

$$\delta_{best}\left(\frac{1}{n}\sum_{i=1}^{n} f(h_i)\right) \geq \delta\left(\frac{1}{n}\sum_{j=1}^{n} f(h_j)\right) \quad (2.1)$$

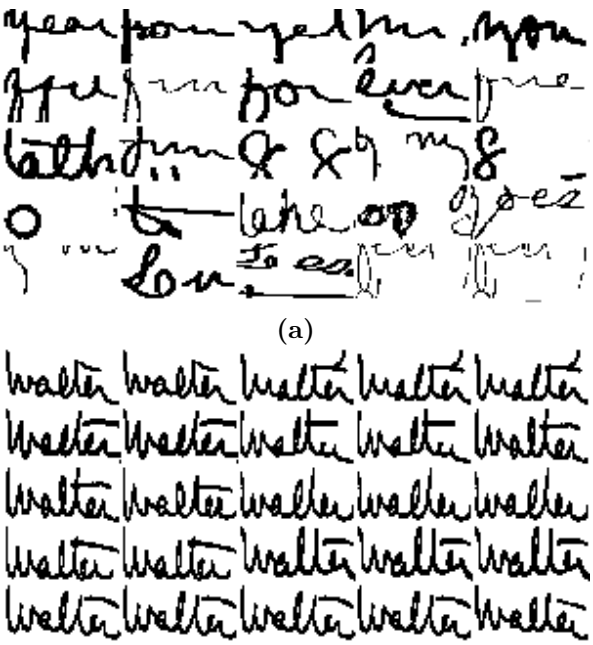
In this section, two CNNs are described. Both were tested on their ability to rate a hit list image. The best CNN was used in subsequent experiments. The task of the CNN presented here can be summarized as an image regression task. The input for the model is an image, and the output should be a number. It is interesting to note that the input labels are all integers, but the output labels can be any real number. This is no problem, because the output should reflect the quality of the hit list, which is treated in this paper as a continuous range of values. Whereas the input labels are within the range [1, 25], The outputs of the neural networks are not restricted to be within that range. A hit list with a given output of e.g. 27.34 is regarded as more structured than an hitlist with a given output of 25.0.

2.1 Dataset

Hit lists generated by a word classifier from the *Monk* system could be considered rich in potential if the instances in the hit list look similar to each other. However, since calculating the similarities of the instances within a hit list is essentially how those hit lists are made in the first place, the approach used here was different. Instead, each hit list was given a number which signifies how many human-labeled instances were in that hit list. The number of human-labeled instances can be regarded as a measure of quality. Conceptually, if hit list A has more human-labeled instances than hit list B, hit list A should visually appear to have more similar instances than hit list B. This idea is illustrated in Figure 2.1.

Hit lists generated by a word classifier from the *Monk* system can have a varying number of word instances. Because simple convolutional neural networks cannot handle a variable image input size[†],

[†]CNNs that can handle variable input sizes do exist, but are outside the scope of this study.



(a)

(b)

Figure 2.1: a) A hit list with a low number of human-labeled instances. b) A hit list with a high number of human-labeled instances. Hit lists with a higher number of human-labeled instances appear more structured than hit lists with fewer human-labeled instances.

the instance count per hit list has been set to 25 (a 5×5 grid, as can be seen in Figure 2.1).

Each instance was strongly down-scaled to a size of 60×30 pixels (i.e. a width of 60 pixels and a height of 30 pixels). This was done to create hit list images with a size of 300×150 pixels, which is small enough to allow for both fast network training and bigger batch sizes, but big enough to detect the most telling patterns. Although some patterns within each word image are lost, it is expected that the same patterns are lost in similar word images, such that the overall similarity between the word images stays the same. It is worth noting that the images used to create the hit list images were already binarized beforehand by the preprocessing procedures in *Monk*.

The dataset consisted of 15000 generated hit list images made up of word images from a book written by a single writer. Each image was created by first inserting a random number of correct instances and filling the remainder with random instances from randomly chosen words. The hit lists were reranked according to the *Nearest Centroid Classifier*, to make the hit lists more realistic. Each image was automatically labeled with a target integer ranging from 1 to 25. This is the number of correct instances in the hit list.

To use all of the data in this dataset for training and testing, k-fold cross validation was used to measure the performance of the networks. The value of k was chosen to be 10. The dataset was shuffled before this process.

2.2 Convolutional Neural Nets

CNNs usually consist of a convolutional part and a non-convolutional part, where the non-convolutional part consists of fully connected layers like a *Multilayer Perceptron* (MLP). Often, an MLP uses input-features extracted by hand to reason about some problem. The non-convolutional part of a CNN uses the features from a *feature map* generated by the preceding convolutional part (Sudholt and Fink, 2016). The convolutional part generates these features by convolving a number of filters over an input feature map (e.g. an image for the lowest layer). Convolutional layers are often followed by dimensionality reducing *Max Pooling* layers, which help reduce the number of parameters within the model, which in turn helps reduce computation time and overfitting (Romanuke, 2017).

It is well known that CNNs do well in image processing tasks, thanks to the many successes published in the literature: e.g. (Cireşan, Meier, and Schmidhuber, 2012; Krizhevsky, Sutskever, and Hinton, 2012; Sudholt and Fink, 2016). This is the case because of several facts: (1) Convolutions preserve spatial information. (2) Convolutional layers use far fewer parameters than fully connected (i.e. dense) layers. (3) Convolutional layers are equivariant to translation, thanks to the receptive field operating on local regions. These three benefits of CNNs are attractive for a task which involves extracting features from an input image and reasoning about those features, such as the hit list rating task presented in this paper.

Two types of CNNs have been tested on their ability to ascertain the quality of a given hit list image. For both of these types, the models were compiled using the well-known RMSprop optimizer with an initial learning rate of 0.001, which was multiplied by 0.9 at the end of every epoch. The loss measured in both models was the mean squared error.

2.2.1 Pre-trained CNN

The first type made use of a pre-trained feature extracting module called Inception v3. Inception v3 is essentially a complete CNN by itself, published in (Szegedy, Vanhoucke, Ioffe, Shlens, and Wojna, 2016). The model has been trained on a subset of the popular dataset ILSVRC-2012-CLS (ImageNet). Inception v3 has gotten great results on tasks involving the ImageNet dataset, reaching a minimal top-1 error of 21.2% and a minimal top-5 error of 5.6% (Szegedy et al., 2016). The full network is shown in Figure 2.2.

Although the dataset used in this paper is very different from the ImageNet dataset, the idea was to let the Inception v3 module recognize the features in the images and use those features to calculate the visual quality of hit lists. To this end, the last layer (i.e. output layer) of the original Inception v3 model was replaced with two custom layers. The new pre-final layer was a dense (i.e. fully connected) layer with 256 neurons and a rectified linear unit (ReLU) activation function. The last layer had to be a fully connected single neuron with a linear

activation function. This is the recommended setup for the last layer in a regression problem. The custom layers were then trained using the training set from the dataset containing hit list images, while the layers from the Inception v3 module retained their pre-trained weights.

It is interesting to note that the Inception v3 module only accepts input images of size 299×299 pixels. Because of this reason, the hit list images were scaled to the fixed input size, resulting in loss of data.

2.2.2 Custom CNN

The second type of CNN used in this research was one made from scratch. The idea was that by training the whole neural network on the dataset instead of only the last few layers, the neural network could learn the features of hit list images better than the pre-trained CNN. The network is shown in Figure in Table 2.1. This network was built using the standard approach of building a convolutional neural network, where convolutional layers are followed by max pooling layers to reduce the dimensionality of the data. The number of max pooling layers and the positions of those layers were determined by following recommendations made in (Romanuke, 2017).

The convolutional-max pooling layer pairs are followed by a flatten layer to make the data one-dimensional. At this point, the data is represented by a one-dimensional feature vector of 7840 values. This vector is passed through two dense layers of 200 and 100 neurons respectively, to the output layer. All layers except the last one have a rectified linear unit activation function, which is widely used in neural networks and commonly regarded as superior to other activation functions for hidden layers, mainly thanks to its faster computation time (Krizhevsky et al., 2012). The output layer uses a linear activation function.

To combat overfitting, two regularization methods were used. The first one is the usage of dropout layers after the dense layers with 200 and 100 neurons. This technique sets a fraction of the neurons to 0 each weight update cycle, essentially forcing the network to find more general ways of representing patterns in the input data. The fraction of neurons dropped is commonly set to 0.5 for dense layers with enough neurons. This approach has been used here as well.

The architecture described in this section has been designed to perform better at the hit list rating task than the pre-trained CNN. Furthermore, the smaller architecture has the added benefit of faster training and predicting procedures.

2.3 Nearest Centroid Classifier

A simple *Nearest Centroid Classifier* has been developed, which could classify input word images based on their distance to the class centroids. The word images were classified as the words whose feature centroid was closest to the features of the input image. This classifier has been constructed for two purposes:

1. Re-ranking hit lists from the dataset used for the CNNs.

2. Being a part of the closed loop training phase of the sampler, which is described in section 3.

The classifier consists of two parts. First, the Inception V3 feature extractor module (Figure 2.2, without the last layer) has been used to extract features from word images. The input to the Inception V3 module is a 299×299 pixels image. The output of this feature extractor is a one-dimensional vector with 2048 values. In the second part of the classifier, this feature vector is compared to an array of class centroids. These class centroids were created by computing the average feature vector of instances of a class that were labeled by human volunteers. It might be worth noting that the centroids themselves also consist of 2048 values. For each input feature vector, the Euclidean distance between that feature vector and each centroid is computed. The classifier outputs a label for the input image, based on the centroid with lowest Euclidean distance to the feature vector.

3 Sampler system

As mentioned in the introduction, the sampler is at the core of this study. The goal of the sampler is to sample from hit lists in an effective manner (i.e. to pick the most influential spot in a hit list to label). The approach used here boils down to simply generating empirical probability density functions (PDF) for each hit list rating, which could be used

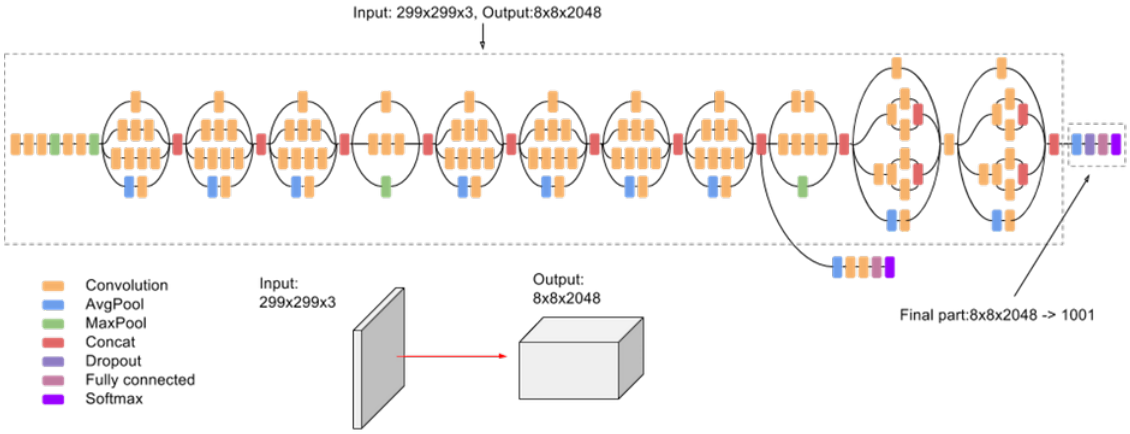


Figure 2.2: The standard Inception v3 network. For the purposes of hit list rating, the last fully connected layer with the softmax activation function has been replaced by two fully connected layers, one with 256 neurons and one with 1 output neuron. The layer with 256 neurons used a ReLU activation function and the the last layer used a linear activation function.

Table 2.1: The custom CNN made for hit list rating. The dropout layers cause the network to find various pathways to fit the data, generalizing more and thus preventing overfitting. Here, p stands for the fraction of neurons that get dropped (i.e. set to 0).

Layer type	Neurons	Kernel / Stride	Output shape
Conv2D	24	5×5 / 1	(146, 296, 24)
MaxPooling2D		2×2 / 2	(73, 148, 24)
Conv2D	32	3×3 / 1	(71, 146, 32)
MaxPooling2D		2×2 / 2	(35, 73, 32)
Conv2D	48	3×3 / 1	(33, 71, 48)
MaxPooling2D		2×2 / 2	(16, 35, 48)
Conv2D	64	3×3 / 1	(14, 33, 64)
Conv2D	80	2×2 / 1	(13, 32, 80)
Conv2D	96	2×2 / 1	(12, 31, 96)
MaxPooling2D		2×2 / 2	(6, 15, 96)
Conv2D	112	2×2 / 1	(5, 14, 112)
Flatten			(7840,)
Dense	200		(200,)
Dropout ($p = 0.5$)			(200,)
Dense	100		(100,)
Dropout ($p = 0.5$)			(100,)
Dense	1		(1,)

to determine each next word image for the labeling process.

However, the range of possible hit list ratings given by the convolutional neural network is continuous. Therefore, intervals of hit list ratings had to be chosen to generate PDFs for, as only a finite number of distributions could be obtained via the methods laid out in the next subsection. These intervals were arbitrarily chosen to be: $[-\infty, 5)$, $[5, 9)$, $[9, 13)$, $[13, 17)$, $[17, 21)$ and $[21, \infty]$. The idea is that these PDFs can be used in the sampling of the next influential word image. For example, if

a hit list with a predicted rating of 14.05 were to be given to the sampler, it should sample the most likely samples to elicit a surge in F_1-score by looking up the posterior PDF appropriate for hit lists with a rating in between 13 and 17.

3.1 Generating PDFs

The generation of the PDFs was done according to the Metropolis-Hastings algorithm (Hastings, 1970). This algorithm is one of the most commonly used Markov Chain Monte Carlo methods. It allows for the sampling from some unknown posterior distribution by specifying a prior distribution, a conditional likelihood, and a proposal distribution. Essentially, this method moves the proposal distribution with every accepted sample. By sampling from the proposal distribution and accepting the move with probability

$$p = min\left(1, \frac{L(M_{new}) * P(M_{new})}{L(M_{old}) * P(M_{old})}\right), \qquad (3.1)$$

a sample of the posterior distribution will be generated. In Equation 3.1, L stands for likelihood, P stands for the prior distribution, M_{new} stands for the new proposal point and M_{old} stands for the old proposal point (i.e. the previous state).

3.1.1 Prior distributions

Choosing proper prior distributions is essential if the available computation time is limited. By choosing prior distributions far away from the actual posterior distribution, the time needed to obtain a good sample from the posterior distribution increases. For the prior distributions, Beta distributions were chosen. These distributions are defined over the range [0, 1], and have different shapes depending on their two parameters, α and β. The mean of the Beta distribution is equal to

$$\mu(Beta(\alpha, \beta)) = \frac{\alpha}{\alpha + \beta} \qquad (3.2)$$

and can thus easily be set by choosing proper values for α and β. The priors were chosen with the intervals of hit list ratings in mind. For each interval, the mean of the Beta distribution belonging to that interval was chosen to be the mean of the interval divided by 24 (for normalizing to a range of [0,1]), with the exception of the intervals involving negative and positive infinity. The actual parameter values are laid out in Table 3.1.

3.1.2 Likelihood distributions

To generate a likelihood distribution for each rating interval, the following approach was used. First, 114 realistic hit lists from 114 different word classes were generated. These word classes all had the property that there were 50 or more instances per class. These hit lists would be used to establish a base average F_1-score and test the effect of labeling lots of single word images on that average F_1-score. The idea was to create an array I where each index represented the position of the instance in the hit list, and each value represented how influential a word image at that position is. Array I was assumed to be a realistic representation of the influential spots in a hit list.

The likelihood generation algorithm consisted of a number of iterations equal to the number of realistic hit lists generated beforehand: 114. In each iteration, a hit list was selected and the top 25 words were labeled one by one, after which the word instances were reclassified and reranked. Subsequently the change in F_1-score was measured. After the change in F_1-score was established, it was added to the the correct value in array I (i.e. corresponding to the position of the labeled hit) and the 114 realistic hit lists were reverted to the way they were before labeling the hit. Lastly, array I was converted into a PDF that served as the likelihood in the Metropolis-Hastings algorithm. Any negative values of I were set to 0, and then the area under the curve was normalized to 1.

3.2 Experiment

As mentioned before, the generated PDFs were used in the sampling of word images from hit lists. To determine the value of using this approach, the following experiment was conducted: Eight sampling methods were compared on how well they influence the increase in average F_1-score. The first method used all generated posterior distributions. The second method used no generated posterior distributions, but merely used uniformly random sampling. This was done to test whether using a posterior distribution improved the performance at all. The remaining six methods used only one pos-

Table 3.1: The values used for parameters α and β for each hit list rating interval.

Hit list rating interval	α	β	Mean of Beta distribution
$[-\infty, 5)$	3	21	0.125
$[5, 9)$	7	17	0.292
$[9, 13)$	11	13	0.458
$[13, 17)$	15	9	0.625
$[17, 21)$	19	5	0.792
$[21, \infty)$	23	1	0.958

terior distribution for sampling. This way, it could be tested whether using all posterior distributions based on the hit list ratings of the hit lists was beneficial over just using a single posterior distribution.

The core of the experiment was the same for each method. Each method started out with the same set of testing hit lists, constructed using a random number of correct instances and a number of filler instances, similar to the description in section 2.1. There were three key differences:

1. Every hit list made in this dataset had a unique word label connected to it.
2. The actual class labels of the incorrect instances within each hit list were known. This allowed the labeling to be performed automatically.
3. The filler instances in these hit lists were not randomly selected. Instead, the hit lists were obtained through classifying all word images and reranking them.

The experiment itself consisted of 115 iterations. In each iteration, the 8 aforementioned sampling methods were tested once. At the start of each method, the F_1-score averaged over the hit lists in the dataset was calculated and stored. Secondly, all suitable hit list images in the dataset were rated. It is important to note that whereas all hit lists were used in the calculation of the average F_1-score, only the hit lists that had 25 or more instances were rated and sampled from. By taking the correct posterior distribution (if any) for the rating, a sampling of a single word image was done for every hit list. After labeling this word image, the class centroids were recalculated, all word images were reclassified and reranked. Lastly, the increase in F_1-score was computed by computing the average F_1-score again and subtracting the starting F_1-score.

4 Results

4.1 Convolutional neural networks

As can be seen in Figures 4.1 and 4.2, the custom CNN performed much better in the hit list rating task than the pre-trained CNN. The mean squared error was about 5 times as high for the pre-trained network. One of the causes for this could be the destructive, non-uniform scaling of the hit list images, which had to be done for the pre-trained CNN. Seeing as the custom CNN performed better, it was used in the sampling experiments.

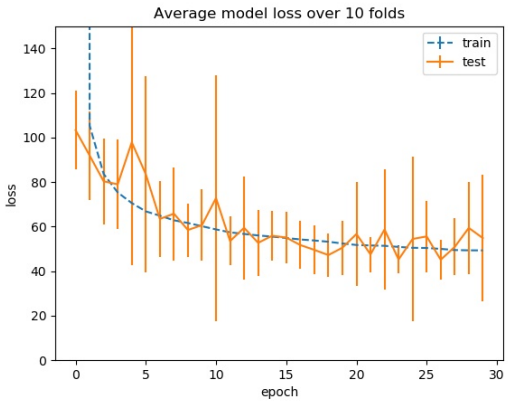
One thing to note about Figure 4.1b is the proximity of the training and testing errors, indicating almost no overfitting. This can be attributed to the effectiveness of the dropout layers used in the CNN. Without those, the model would have been overfitted to a larger extent.

4.2 Posterior distributions

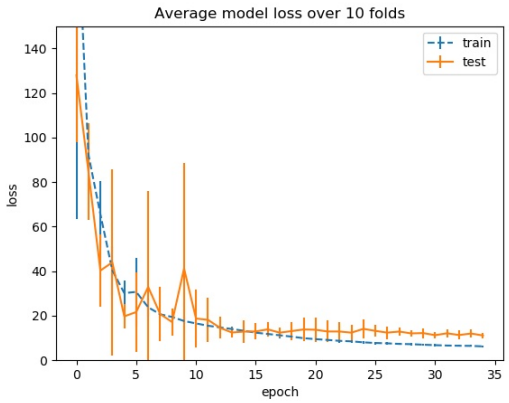
In Figure 4.3, the generated posterior distributions are shown. These probabilities were used in the sampling of instances from hit lists. The figure helps in interpreting the results gotten from the sampler, which are given in the next subsection. It can be seen that the posterior distributions closely resemble the prior distributions specified in Table 3.1.

4.3 Sampler

Figure 4.4 shows the box plot of the data generated by the sampling algorithm. Although the expectation was that the mean of the "All posteriors" method would be the highest, it is clearly visible in the plot that this is not the case. Instead, it seems that using only the sixth posterior distribution is beneficial over any other method. Even uniformly randomly picking instances to label from a hit list

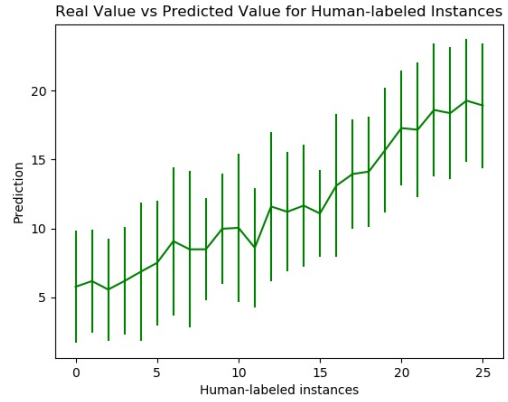


(a) **Train and validation losses of the pre-trained CNN after each epoch of training, averaged over 10 folds.**

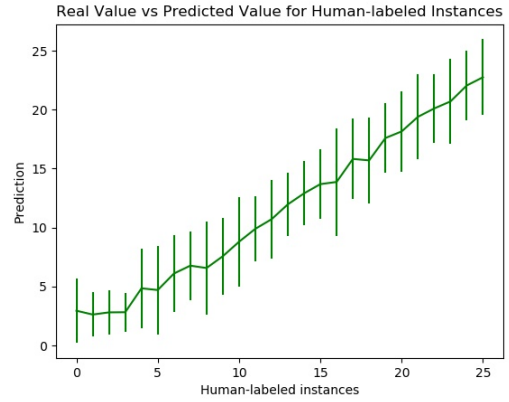


(b) **Train and validation losses of the custom CNN after each epoch of training, averaged over 10 folds.**

Figure 4.1: Average model losses for both CNNs. It can be seen that the custom CNN has learned the task of rating hit lists better than the CNN including Inception V3. The model is also much more stable, depicted by the low standard deviation at the end of the training phase. Since the custom CNN was better at the given task, the custom CNN was used in the rest of the algorithms as the hit list rater.



(a) **The mean and standard deviation of the predicted number of human-labeled instances plotted against the real number of human-labeled instances in the test set of the 10th fold for the pre-trained CNN.**



(b) **The mean and standard deviation of the predicted number of human-labeled instances plotted against the real number of human-labeled instances in the test set of the 10th fold for the custom CNN.**

Figure 4.2: The ability to predict the value of human-labeled instances in a hit list of both CNNs. Once again, it is clear to see the custom CNN is more stable and better able to rate hit lists.

seems to work better than using the generated posteriors.

A Shapiro-Wilk Test was done to check for normality in the data. It became clear that the data is not normally distributed, with p-values ranging in the order of 10^{-11} to 10^{-13}.

Because of the non-normally distributed data, a Mann-Whitney U Test was performed for each

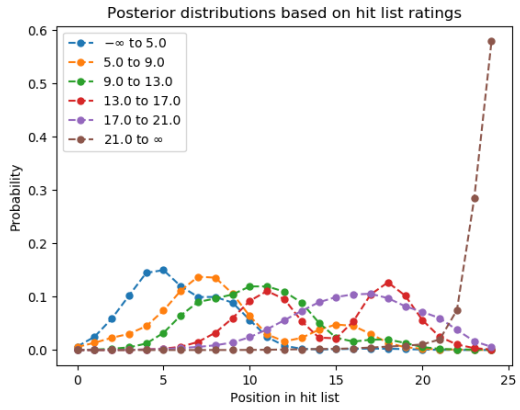


Figure 4.3: The posterior probability distributions generated by the Metropolis-Hastings algorithm. The probabilities are defined at integer positions. The legend shows which hit list rating interval corresponds to which posterior distribution.

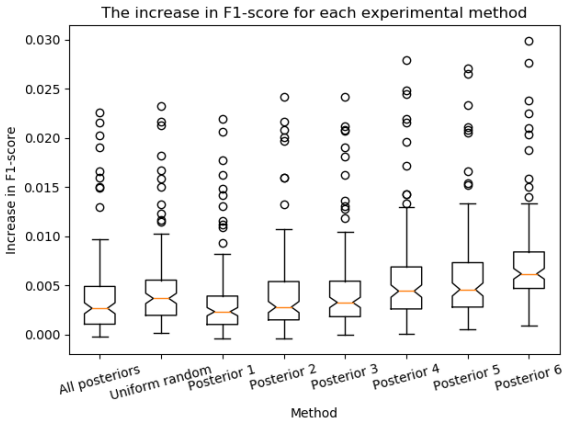


Figure 4.4: A boxplot of the measurements of the increase in F_1-score for each of the 8 methods. The methods "Posterior 1" to "Posterior 6" use a different notation than before. But they correspond to the order of the legend shown in 4.3 (i.e. "Posterior 1" is the posterior for rating interval [-∞, 5)). The increase in F_1-score was measured after a single labeling from each known word class in the dataset. The yellow lines represent the medians. The notches represent 95% confidence intervals for the medians.

pair of methods to test for significant differences in means of increase in F_1-score. The results are displayed in Table 4.1. The test confirms that there is a significant difference between the distributions of using all posteriors and uniform random sampling. In other words, the proposed algorithm performs worse than random guessing with these posteriors. Furthermore, there is no significant difference between using all of the posteriors and only using posterior 1 or posterior 2. However, the methods involving the last 4 posteriors do show a statistically significant difference with the usage of all posteriors. And the last 3 methods show a statistical difference with the random guessing method. In particular, using only posterior 6 seems to be better than all other methods.

5 Discussion

In this paper, an attempt has been made to answer the research question: How can word images be sampled from a hit list, such that they exhibit the highest possible impact on the lifelong learning process when they are labeled? A series of algorithms were developed in order to answer the research question. Firstly, a convolutional neural network was created, which could rate lists of word instances on their visual structure. Secondly, a nearest centroid classifier was developed that could classify individual word images based on the Euclidean distance to a word prototype. Thirdly, the Metropolis-Hastings algorithm was used to generate probability distributions which could be used in the sampling of word images from a list of word images. Lastly, the effectiveness of those probability distributions was measured in a series of simulations.

Whereas the test results of the CNN show that it can accurately rate hit lists on their visual structure, the results of the sampling and labeling simulations have shown that the proposed generated distributions in their current forms are not a good way of sampling from hit lists. Therefore, the answer to the research question would be: It is currently unknown, but uniform random sampling is a good start.

Table 4.1: The p-values computed with a Mann-Whitney U Test on the data for each pair of methods. The values printed in boldface denote statistical significance between the distributions of the simulation data on the 5% level. Most statistically significant values denote a larger mean in the data of the method presented in the first column of the row, compared to the data of the method presented in the first row of the column, with one exception; the value also printed in italics denotes that the mean in the data of method presented in the first column of the row is actually lower than the mean of the method presented in the first row of the column. This can also be seen in Figure 4.4.

	All posts	Uniform	Post 1	Post 2	Post 3	Post 4	Post 5
Uniform random	$\mathbf{4.00\times10^{-3}}$						
Posterior 1	1.85×10^{-1}	$\mathbf{\textit{1.42}\times10^{-4}}$					
Posterior 2	1.21×10^{-1}	6.66×10^{-2}	$\mathbf{2.02\times10^{-2}}$				
Posterior 3	$\mathbf{4.04\times10^{-2}}$	1.94×10^{-1}	$\mathbf{2.74\times10^{-3}}$	2.62×10^{-1}			
Posterior 4	$\mathbf{1.23\times10^{-5}}$	$\mathbf{2.30\times10^{-2}}$	$\mathbf{1.10\times10^{-7}}$	$\mathbf{4.39\times10^{-4}}$	$\mathbf{3.42\times10^{-3}}$		
Posterior 5	$\mathbf{7.49\times10^{-7}}$	$\mathbf{4.58\times10^{-3}}$	$\mathbf{3.82\times10^{-9}}$	$\mathbf{5.22\times10^{-5}}$	$\mathbf{5.09\times10^{-4}}$	2.46×10^{-1}	
Posterior 6	$\mathbf{2.89\times10^{-14}}$	$\mathbf{1.65\times10^{-9}}$	$\mathbf{1.38\times10^{-17}}$	$\mathbf{4.68\times10^{-12}}$	$\mathbf{1.09\times10^{-10}}$	$\mathbf{5.04\times10^{-5}}$	$\mathbf{7.54\times10^{-4}}$

5.1 Ineffectiveness of the first 3 probability distributions

The expectation was that using all generated probability distributions is the best strategy in order to get the highest F_1-score the quickest. However, as can be seen in Figure 4.4, this is not the case for these probability distributions. The same figure also shows that later probability distributions are better. Using only posteriors 4, 5 or 6 is significantly better than uniform random sampling, but using the other posteriors is not. Looking at Figure 4.3, it can be seen that posteriors 4, 5 and 6 primarily sample from the last few positions in a hit list. This can be largely attributed to the choice of prior distributions given in Table 3.1. Apparently, these later prior distributions are more effective than earlier prior distributions. Perhaps a completely uninformative prior distribution (i.e. a uniform distribution) would be more effective than multiple Beta distributions.

If the prior distributions were improved, the methods "Posterior 1", "Posterior 2" and "Posterior 3" would consequently also be improved. Combined with the fact that most hit lists are given a rating between 1.0 and 13.0 by the CNN (i.e. in the ballpark of the first 3 posteriors), the usage of the proposed "All posterior" algorithm would be more promising.

5.2 Limitations

One of the hand-picked limitations that were put in use was the number of words in a hit list image. The value was set to 25, but in reality, this value varies considerably. Because of this restraint, hit lists with fewer than 25 word instances could not be properly rated, and were excluded from the CNN training phase, the likelihood distribution generation algorithm and the sampler simulations. Furthermore, hit lists with more than 25 instances were cut down to a size of 25, thereby reducing the amount of data used considerably. This restraint was chosen for simplicity, as traditional convolutional neural networks cannot handle variable input size. After all, to increase the number of words in an image and still keep most of the patterns, the image would need to increase in pixel-size as well. To get more accurate results this restraint could be removed by using network architectures that can handle inputs with varying sizes. For example, PHOCNet (Sudholt and Fink, 2016) is able to handle input images of varying size.

Another limitation in this paper was the usage of the nearest centroid classifier. It was created in order to speed up the simulations. However, in the real *Monk* engine, other types of classifiers are used. Using these classifiers, reality would have been modeled better. Doing this would take more time and computational effort, so the restraint has been put into place for simplicity's sake.

The number of hit list rating intervals used in this research was an arbitrary choice. This value was chosen to be 6. It represents a trade-off between speed and simplicity on the one hand, and more realistic results on the other.

For the likelihood generation algorithm, included

word classes were selected based on whether they had more than 50 ground truth instances. This was done to speed up the likelihood generation, as it was one of the most computationally heavy algorithms. This decision left out a considerable number of word classes that did not have that many instances. Perhaps this decision led to a decline in realism, as word classes with fewer instances might have different optimal locations to label at. The combination of not using these word classes in the likelihood generation, but subsequently using them in the final simulations, may have led to a drop in performance of the proposed algorithm.

References

Zhiyuan Chen, Nianzu Ma, and Bing Liu. Lifelong learning for sentiment classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pages 750–756, 2015.

Dan Cireşan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. *arXiv preprint arXiv:1202.2745*, 2012.

Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5-6):602–610, 2005.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM, 2006.

W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. 1970.

Sheng He and Lambert Schomaker. Open set chinese character recognition using multi-typed attributes. *arXiv preprint arXiv:1808.08993*, 2018.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Bing Liu. Lifelong machine learning: a paradigm for continuous learning. *Frontiers of Computer Science*, 11(3):359–361, 2017.

J Philbin, O Chum, M Isard, and J Sivic. Interactive layout analysis, content extraction and transcription of historical printed books using agora and retro. *Digital Humanities 2011*, page 358, 2011.

Jean-Yves Ramel, Nicolas Sidère, and Frédéric Rayar. Interactive layout analysis, content extraction, and transcription of historical printed books using pattern redundancy analysis. *Literary and linguistic computing*, 28(2):301–314, 2013.

Vadim V Romanuke. Appropriate number of standard 2× 2 max pooling layers and their allocation in convolutional neural networks for diverse and heterogeneous datasets. *Information Technology and Management Science*, 20(1):12–19, 2017.

Lambert Schomaker. Design considerations for a large-scale image-based text search engine in historical manuscript collections. *it-Information Technology*, 58(2):80–88, 2016.

Nicolás Serrano, Adrià Giménez, Jorge Civera, Alberto Sanchis, and Alfons Juan. Interactive handwriting recognition with limited user effort. *International Journal on Document Analysis and Recognition (IJDAR)*, 17(1):47–59, 2014.

Burr Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

Daniel L Silver, Qiang Yang, and Lianghao Li. Lifelong machine learning systems: Beyond learning algorithms. In *2013 AAAI spring symposium series*, 2013.

Sebastian Sudholt and Gernot A Fink. Phocnet: A deep convolutional neural network for word spotting in handwritten documents. In *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 277–282. IEEE, 2016.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

Tijn van der Zant, Lambert Schomaker, Svitlana Zinger, and Henny van Schie. Where are the search engines for handwritten documents? *Interdisciplinary Science Reviews*, 34(2-3):224–235, 2009.

Jean-Paul van Oosten and Lambertus Schomaker. Separability versus prototypicality in handwritten word retrieval. In *Frontiers in Handwriting Recognition (ICFHR), 2012 International Conference on*, pages 8–13. IEEE (The Institute of Electrical and Electronics Engineers), 9 2012. ISBN 978-1-4673-2262-1. 10.1109/ICFHR.2012.269.