CS4099: Major Software Project

Script Identification with Ancient Egyptian Hieroglyphs

Akshit Talwar
140014861

Supervisors: Dr. Mark-Jan Nederhof & Dr. Juan Ye

School of Computer Science

University Of St Andrews

1 Abstract

Egyptian Hieroglyphs are an ancient writing system that dates back to 3000 B.C. It is a figurative writing style that can be read phonetically. The translation of these ancient scripts help in understanding the culture and history of the Egyptian civilisation. With the help of technology, transcription of these texts can be automated.

The aim of this project is to extend an existing Egyptian Hieroglyph OCR tool. The functionality to distinguish between ancient and modern text makes the tool more powerful. This added functionality is designed to work on texts of Urkunden IV and Kitchen Ramesside Inscriptions.

The implemented approach uses a Convolutional Neural Network (CNN), which is a deep learning neural network to classify the text. In this paper, traditional neural networks are briefly described, followed by a detailed explanation of the components of CNNs — architecture, training, transfer learning, and hyperparameter optimisation. With the limited amount of training data available, the classification model predicts labels with an accuracy of 96.875%.

2 Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated. The main text of this project report is 6,932 words long, including project specification and plan. In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona fide library or research worker, and to be made available on the World Wide Web. I retain the copyright in this work.

Contents

3 Introduction

Egyptian hieroglyphic writing dates back to around 3000 B.C making it the earliest authentication of the principle of phonetic transfer. [13] Their meaning was completely lost around 400 A.D. but the discovery of the Rosetta Stone in 1799 presented the opportunity to translate the script. Alongside hieroglyphic text, the stone contained a Greek translation but it still proved difficult to decipher the text. Fortunately, Jean-Francois Champollian, in 1822, discovered that each symbol represents a sound and multiple symbols form a word. This helped in understanding the writing system which revealed a lot of information about the culture and civilisation.

Optical Character Recognition (OCR) is the process of digitising images into text encoding. The input images contain typed or handwritten text and can be in the form of scanned documents, photos of documents, etc. Having access to a machine-readable format of ancient text has considerable advantages.

- Linguistic studies:
  It is the study of language as expressed in examples of real-world text. It may focus on different aspects of language, such as lexicography [1] or grammar [4]. [14]
- Non-linguistic studies:
  Annotated text can be used by historians who cannot read hieroglyphic. Digital translations can be searchable and contentious translations can also be easily verified. [14]
- Learning and teaching:
  Self-study is a major advantage of having Ancient Egyptian texts freely available. The St Andrews corpus contains several dozens of classical texts [12] and one of its main objectives is to support self-study. This is particularly helpful for people without access to the very few centres that teach this language. The addition of a transcription tool can only aid in this process.

Essentially, the OCR consists of two processes — segmentation and recognition. Segmentation is the process of extraction of individual glyphs from the input image. Recognition is the process of classification of these segmented glyphs. These functions have already been implemented in the provided OCR tool. It has a reported classification accuracy of 91.3%. [14]

The aim of this project was to devise a method to distinguish between Egyptian Hieroglyphs and modern text. The problem at hand may be easy for the human eye, but algorithmically defining the differences between symbols is difficult. Feature extraction is the process of specifying the differentiating features between the classes. Two approaches were considered, the second being the successful one. The first approach involved manual feature extraction and the second approach involved deep learning.

Machine learning is essentially teaching a program to learn by example. A set of examples with the correct classification is provided as training data, along with features to observe. After training is complete, the model can predict the class of an input with a certain level of accuracy. However, the process of feature extraction is very cumbersome and there is no guarantee that the extracted features would provide the best accuracy. It is highly dependent on the programmer's skill level and the problem at hand. A more novel solution to this problem is through the use of Deep learning.

Deep learning is an extension of machine learning in which the features are automatically learnt by

the neural network. This results in a generally higher accuracy of classification as the most optimal features are extracted. There are different types of deep learning network but in this project a Convolutional Neural Network (CNN) is used.

The report is structured by first outlining the objectives, describing the design behind the initial and final approach, implementation of the final approach, a description of the experimental setup, along with results and a critical evaluation, finally proposing future work prospects.

4 Context Survey

4.1 Convolutional Neural Network & LeNet

Yann LeCun et. al used a gradient-based learning technique to classify high-dimensional patterns such as handwritten characters. [10] A CNN was compared to other methods in a standard handwritten digit recognition task. The main argument of this paper was that better pattern recognition can be achieved if the task of feature extraction was automated. Essentially, hand-crafted feature extraction can be replaced by automatic learning.

4.2 Transfer Learning

Jason Yosinki et. al proved that transferability is negatively effected by two issues — splitting of a network the middle of fragilely co-adapted layers and the specialisation of higher layer features to the original task. [20] They were able to quantify the effect of fine tuning or transfer learning based on the layers change in the base network.

4.3 Limitations of Related Work

The main limitation of the literature is that it does not address the problem of having less training data. It is mentioned by Jason et. al that features transferred from distant tasks are still better than random weights. [20] However, this holds true when the training data size is large.

5 Software Engineering Process

The development methodology followed a pattern similar to scrum. An adaptive methodology was needed for this project as it is a research project. Flexible weekly goals were set that encouraged incremental progress. Goals were divided into immediate, short, and long-term goals. Immediate goals referred to tasks that needed to complete a short-term goal. Short-term goals were weekly progress milestones and long-term goals were overall project goals.

Scrum is a methodology which accounts for continuously changing requirements. It supports an evidence-based approach and assumes that the exact progress path will not be known in advance. Rather, as the project progresses, understanding of the project increases and implementations become more clear. Since there was only one contributor to this project, there was no need for a product owner or scrum master.

6 Ethics

This project does not involve any people, therefore preliminary ethics self-assessment form covers all the ethical concerns that could be related to this project.

7 Design

This section contains a brief explanation of the initial attempt, followed by a detailed explanation of the final implementation.

7.1 Initial Implementation

Feature extraction is the process of extracting differentiating features between class images. The initial attempted approach involved manually searching and specifying these features. The intuition was that the pixel density of modern images would be concentrated towards the bottom of the image, whereas for the hieroglyphs they would be more spread out. The modern images generally contain cursive handwriting. It can be seen that the connections between letters is mostly made near the bottom of the text. On the contrary, hieroglyphs generally had an almost equal distribution of pixels throughout the image. These observations were true for a subset of the samples. Some progress was made when the radial distribution of pixel was mapped on *Octave*. However, as more samples were tested, there were too many exceptions to handle. There had to be considerations for noise, distortion, style variation, translation and rotation. This approach did not seem very practical. As no robust distinguishable features were apparent, this approach was stopped. Research into automated ways of feature extraction was started.

7.2 Final Implementation

The implementation is a deep learning neural network. First, a traditional neural network is explained briefly. Followed by an in-depth explanation of a Convolutional Neural Network. This includes the basic architecture and training process. Some additional concepts such as Transfer learning and computational considerations are also described.

7.2.1 Traditional Neural Network

A neural network consists of layers of interconnected *neurons*. Here, a neuron (also called perceptron) is, in some ways, similar to the biological notion of a neuron. It takes multiple inputs $x_1, x_2, x_3...$, does some computation and produces a result which can be either 0 or 1. Each neuron contains a set of *weights* $w_1, w_2, w_3...$ which denotes the 'importance' of the respective input values. The output is determined by checking if the calculated *weighted sum* is above a certain threshold. If it is, then the neuron 'fires' (output is 1).

The input layer is the first layer in which this decision-making occurs. The output layer is the final layer which produces an output. The layers in between are called hidden layers. Each neuron's output is transmitted as input to each neuron in the following layer. In a feed-forward network, the neurons do not send their output to previous layers.

Learning of a network takes place by continuously changing the weights and biases of the network to produce correct outputs on the training set. When a small change in these values is made,

a small change in the output is expected. However, the network design above is flawed. Since the output of a neuron can only be 0 or 1, a small change may result in a significant alteration in the network. It will change the weights to correctly classify the current input but may result in a significantly changed behaviour of the network. This means that the classification of other images may be incorrect. As a result, the network cannot be fine tuned accurately.

This is where *sigmoid* neurons are useful. A sigmoid neuron is similar to a perceptron but instead of limiting its input and output to 0 and 1, it considers all float values in between as well. The output firing function is also different to accommodate for the continuous distribution of possible values. Now, during training, a small change in weights or biases would result in a small change in the network behaviour.

Given the working of a neural network, it is still not clear how classification is done. The missing component is how the images are distinguished from each other. The process of finding distinguishing features is called feature extraction. In traditional neural networks, features have to be manually selected. The network then learns the weights for these features which would classify the images. This process of feature selection is cumbersome and it is not an ideal way to classify images as it may result in sub-optimal classification. Automation of feature extraction would potentially fix both of these issues. Deep learning does exactly this.

7.2.2 Convolutional Neural Network

Convolutional Neural Networks (CNNs) are a type of deep learning neural network that are specifically designed to handle input in the form of images. They are mainly used for visual recognition tasks such as segmentation and classification. Traditional neural networks can be used for image recognition, however, they have a few drawbacks. Since each neuron is connected to each other, high-resolution images suffer from the *curse of dimensionality*. Given a $256 \times 256 \times 3$ image, the number of parameters in a traditional neural network would be 196,608. This could lead to overfitting. Moreover, spatial context is not considered. This means that pixels that are not within the same vicinity are still treated as if they are next to each other.

They combine three architectural ideas to ensure some degree of shift, scale, and distortion invariance: local receptive fields, shared weights (or weight replication), and spatial or temporal sub-sampling. [10]

- The neurons in CNN layers support 3 dimensions — width, height and depth which reduce the number of parameters required. As we will see below, neurons are only connected to a small spatial region of its previous layer. This is called a receptive field.
- Each filter is computed across the entire image. Since the weights and bias used, is the same, the weights are shared.
- Local connectivity between neurons of adjacent layers results in learnt filters to represent small parts of the input image. These are then combined to represent larger areas. In this way, spatial context is preserved.

In these ways, drawbacks of traditional neural networks are addressed by CNNs.

The architecture of CNN consists of 4 main layers — convolution, rectified linear unit, pooling, and classification. Each of them is described below.

Convolution

Mathematically, convolution is an operation on two functions that expresses the amount of overlap between the two functions. With this in mind, the convolutional layer processes different spatial areas of the image and then uses this data for feature extraction. It learns image features using small squares of input data and preserves the spatial relationship between pixels. [18]

Each image can be represented by a matrix of numbers where each element refers to a pixel. The values of these pixels depend on the channel of the image. A 3 channel input refers to 3-base coloured images (usually RGB), whereas a 1 channel input refers to a grayscale image. The depth in a CNN input refers to the number of channels of the input. The height and width are the actual height and width of the input images.

The convolution layer's parameters consist of a set of learnable filters. There are three hyper-parameters in the convolution layer — *filter size*, *stride*, and *zero padding*.

A filter is a spatially smaller image that extends through the full depth of the input volume. Intuitively, it can be viewed as being a smaller matrix of the original image matrix. During a forward pass, the filter slides across of the image's width and height at a rate called the *stride*. If the stride is 1, the filter is slid by 1 pixel. This process is called convolving. Each value in the filter matrix is multiplied by the input pixel value associated with the filter. This results in a 2-dimensional feature map, also known as an activation map. Each output image is essentially an extracted feature. The smaller the filter size, the more features are extracted. In Figure 1, it can be seen that a filter of size 3×3 is used. The size of zero-padding is the final hyperparameter that

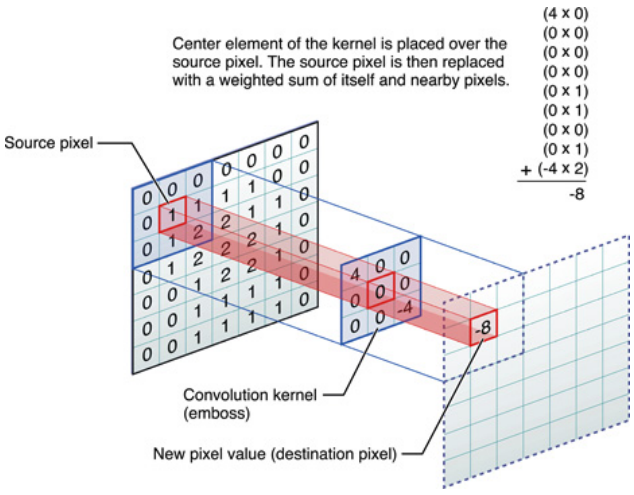

Figure 1: Convolution layer example from [2]

specifies the size of padding of zeroes around the border. This can help control the spatial size of the output volume.
The size of the output can be mathematically defined as,

$$Output = \frac{(W - F + 2P)}{S} + 1$$

where W: Width of input, F: Filter size, P: Zero padding, and S: String length.

To illustrate the effectiveness of the convolutional layer, consider the example of a high-dimensional input. Assume the input is of size $256 \times 256 \times 3$ ($Width$, $Height$, $Channels$). In a regular neural network, the number of weights required would be $256 * 256 * 3 = 196,608$. It can clearly be seen that this number would increase significantly with the increase in image size. In a CNN, with a filter size of 20×20, stride of length 2 and no padding, using the formula described above the output size would be $118 \times 118 \times 3$. This translates to 41,772 weights, which is already a significant reduction. This local connectivity is a key feature of CNNs.

Parameter sharing is the sharing of parameters under the assumption that if a feature at a certain spatial position is useful, then it is also useful at a different position. This concept is used to control the number of free parameters. The weights and bias at a given depth, called depth slice, are shared. These shared parameters are called a filter and result in the formation of an activation map. The translation invariance of a CNN is because of this feature.

Rectified Linear Unit

This layer, also known as ReLU, is used to introduce non-linearity to the network. It applies a non-saturating activation function to each element which takes the maximum of 0 and the value. Effectively, it replaces negative values with 0. This max function results in a CNN training several times faster that their equivalents with $tanh$ units. [9]

Pooling

This layer is used to reduce the dimensionality (down-sampling) of each depth slice. This is done to reduce the number of parameters, thus reducing the chances of overfitting. This also reduces the amount of computation needed. Though each depth splice is resized spatially, usually using the MAX operation, the important information is retained. Other operations include L2-norm pooling and average pooling, though generally, max pooling returns better results. The pooling operation is not restricted to the MAX operation. It can be L2-norm pooling, average pooling, etc. however, MAX pooling has better results in general. [16] It has two hyperparameters — spatial extent and stride.

Classification

Fully connected layer is used as a classifier. It is called a fully connected layer as it is completely connected to each activation in the previous layer. Therefore, it can be seen as a regular neural network. It is identical to the convolution layer but the only difference is the absence spatial extent. In the convolution layer, the dot product is applied on a local region of the input by the neurons, whereas in the classification layer applies it to the complete input.

Training of CNN

Before the learning takes place, the training data set needs to be initialised. This is described in Section 7.2.3.

The next step is to initialise the weights of the network. These can be randomly assigned to a network that is being trained from scratch. A pre-trained base network can also be used which is called Transfer Learning. It is discussed in more detail in the Section 7.2.2.

The learning algorithm is essentially a problem of minimising a function. Gradient-based learning draws on the fact that it is generally much easier to minimise a reasonably smooth, continuous function than a discrete function. [10] Here, the data loss needs to be minimised. In a classification problem, it refers to the deviation between a predicted and ground truth label. The total data loss is the average of data losses for each sample.

The loss function can be minimised by estimating the impact of small variations of the parameter values on the loss function. This is measured by the gradient of the loss function with respect to the parameters. Efficient learning algorithms can be devised when the gradient vector can be computed analytically (as opposed to numerically through perturbations). This is the basis of numerous gradient-based learning algorithms with continuous-valued parameters. [10]

There are two ways to calculate the gradient of a function. A numerical gradient is easy to implement but it is slow and tends to be approximate. On the other hand, the analytical gradient is fast and exact, however, it tends to be more error-prone.

It should be noted that the loss should follow a pattern similar to the one shown in Figure 2. Loss that is asymptotically reduced is the sign of a good learning rate.

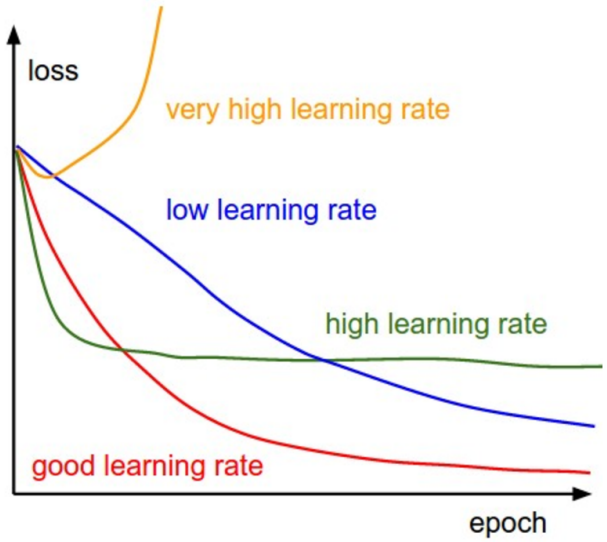

Figure 2: Sketch of learning rates graph from [5]

For accuracy, the difference between the training and validation accuracy should be low. If the difference is large, then it is a sign of overfitting.

A solver orchestrates model optimisation by coordinating the network's forward inference and backward gradients to form parameter updates that attempt to improve the loss. The responsibilities of learning are divided between the Solver for overseeing the optimisation and generating parameter updates and the Net for yielding loss and gradients. [19]

To optimise the process of training, training data is divided into batches. This is because when a huge dataset is considered, perhaps in the order of millions, computing the loss over the entire dataset would be wasteful. Instead, when the gradient is computed over batches, the efficiency of

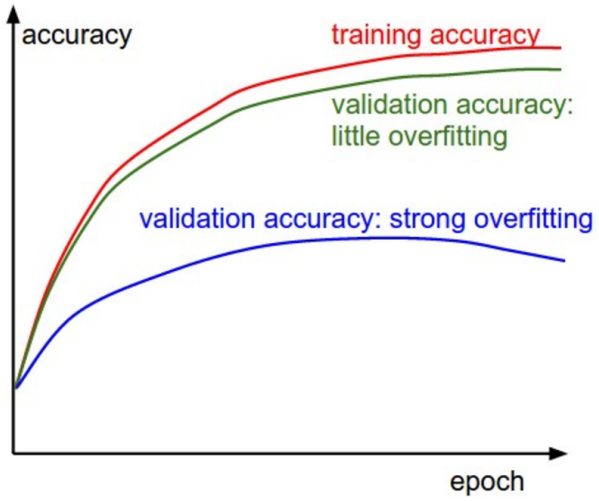

Figure 3: Sketch of accuracy graph from [5]

the program increases. *Stochastic Gradient Descent* does exactly this, but the batch size consists of only one example.

Adaptive gradient (AdaGrad) algorithm incorporates knowledge of the geometry of the data observed in earlier iterations to perform more informative gradient-based learning. Informally, the procedures give frequently occurring features very low learning rates and infrequent features high learning rates, where the intuition is that each time an infrequent feature is seen, the learner should 'take notice'. Thus the adaption facilitates finding and identifying very predictive but comparatively rare features. [6] This strategy often improves convergence performance over standard stochastic gradient descent in settings where data is sparse and sparse parameters are more informative.

Nesterov's Accelerated Gradient (NAG) can be a very effective method for optimising certain types of deep learning architectures. It is slightly different than Stochastic Gradient Descent in the way the weight setting is calculated. In NAG, the gradient on the weights is taken and the momentum is added, however, in SGD only the gradient on the current weight is considered.

There is no ideal solver type for a network, though theoretically some solvers can be determined to better than others. This may or may not translate to practical results. Instead, an experimental approach needs to be taken to determine the best solver for the given task

The gradient only returns the direction in which the function has the steepest rate of increase. Learning rate or step size is a hyperparameter in training the neural network which specifies how far from the current point to step. If a small step size is chosen, the progress will be consistent but will take longer. If the step size is high, there is chance of overstepping which will effect the loss negatively.

Another variable required when training a CNN is epoch. An epoch is a single pass through an entire training set. Therefore, the number of epochs is the total number of cycles the model is trained. It is model and data dependent, therefore there is no fixed ideal number for the number of epochs. Since it cannot be predicted, experimentation is required to find a suitable value. This

value would mean that the model is trained 'enough' but is not over trained (which will result in overfitting). In other words, the loss function is minimised and accuracy on the validation set is maximised.

Transfer Learning

When a deep neural network is trained on images, they all tend to learn first-layer features that resemble either Gabor filters or colour blobs. [20] Since CNN is a type of deep neural network, it can be extended that the initial layers of the network would *generally* contain similar looking features. The first-layer features are called *general*. As the later layers of the network are reached, the features become more and more specific to the training data. Both of these observations can be taken advantage of by training a base network on a huge amount of data and then transferring that network to a different dataset which is perhaps smaller. This potentially prevents overfitting to the smaller dataset.

Usually, when the transfer learning approach is taken, a pre-trained network is used as the base network. The final n layers' weights are randomised. The errors from the learning can then be backpropogated to fine-tune the previous layers of the network, or the transferred feature layers can be left frozen. [20] Some considerations regarding the size of data sets and number of parameters need to be kept in mind. If the target dataset is small and the number of parameters is large, fine-tuning may result in overfitting.

Transfer learning has many variations. Once a base network is trained, the are several possible ways of training the target network. Initial n layers can be kept frozen and the remaining layers can be reset to random weights, or vice-versa. There are two main issues to consider in transferability: optimisation difficulties related to splitting networks in the middle of fragilely co-adapted layers and the specialisation of higher layer features to the original task at the expense of performance on the target task. [20]

The paper concludes that transferring of features and then fine-tuning them results in networks that generalise better than those trained directly on the target dataset. [20] This also holds true if the target dataset is large.

7.2.3 Training Data

This consists of two steps — accumulating a well-balanced set of images and modifying the images to reduce variance. The collection of data is described in Section 8.1.1.

The images need to be modified to achieve maximum accuracy. The changes include normalisation and whitening balance. Normalisation refers to making the images the same size. Adjusting the white balance of an image is used for both colour and greyscale images. There are several ways of normalising, but I have chosen to squash the images to the same size. This also solves the problem of zero-centering the images, which as the name suggests, involves centering the subject in the image. Moreover, due to the assumption made for parameter sharing, features at one location may be useful at some other location as well. Also, in our case, the images are binary images to begin with. They are converted to greyscale images with 1 channel to be accepted by the CNN. Hence, adjusting the white balance is redundant.

K-fold cross validation was considered as a model validation technique. In this, a validation set is randomly created, in each iteration, from the training set. This helps in preventing overfitting. This is mainly used to prevent overfitting to the training data. It is also used when the training data available is very less. If a standard train-validation split were to be used, there is a high chance that modelling or testing capabilities would be lost. However, there is a drawback — the training time increases significantly. This is a major concern for CNN as the training using a standard split already takes long.

7.2.4 Deep Learning Framework

There are many deep learning toolkits available which provide similar functionality but through different methods. Toolkits like Tensorflow, Caffe, Theano, Torch, etc. are popular amongst the community. Each of them has their own advantages and disadvantages, however, Caffe was chosen as the framework of choice because of the following reasons:

- Highly optimised and abstracted code - easy to use.
- Large community [3]
- It is C++ based, therefore it is cross-platform.
- Many plug-ins (such as DIGITS) [3]

The system on which this project was implemented did not have a GPU. Therefore, the training of the model was run on the CPU. Torch followed by Theano perform the best for CPU-based training, however Caffe was a close third. [3]

DIGITS is a webapp, built by NVIDIA, for training deep learning models. It supports multiple deep learning frameworks with image classification, segmentation, and object detection tasks as its main aim. It provides an all-in-one interface to manage data sets, train models, test them and view performance. It provides real-time graph of training progress that plot the accuracy and loss against the number of epochs. The test results provide a layer-by-layer detailed visualisation with values of each hyperparameter. When using this interface, the main focus is on creating the model rather than programming and debugging of low-level code.

7.2.5 DL4J for Java Implementation

I decided to train the network in Caffe and then port the pre-trained model to Java due to the above reasons. Keeping the model training separate from the OCR also kept the OCR tool light.

This did require some extra work as the model needed to be transformed into a DL4J acceptable format. DL4J accepts *Keras* which is an open-source deep learning tool for Python. It allows models to be imported from other deep learning frameworks, including Caffe. To aid in this conversion, *caffe2keras* was used.

The given java code required *Apache Ant* to build. To include DL4J into the project, *Maven* is required which is a dependency manager in addition to a build system. *Ant* is powerful as it provides complete control of the build process and Maven manages dependencies well. A combination of these would provide the best of both and this why I decided to merge them. To achieve this, two different frameworks were tried — *maven-ant-tasks* and *aether-ant-tasks*. The documentation

for both of them is quite sparse, therefore a lot of hit and trial was required. Maven ant tasks was successfully set up [15], however one dependency of DL4J (*jai-image-core-io*) did not support Maven 2 [8]. Aether ant tasks [7] uses Maven 3 and this managed to fix the error.

8 Implementation

The objective entails the construction of a classification model. There are 2 classes — *Ancient* and *Modern*. *Ancient* refers to all the Egyptian Hieroglyphs and *Modern* contains all the modern languages.

This section contains final implementation details such as database creation, classification model creation and Java implementation.

8.1 Final Implementation

As discussed above, Caffe and DIGITS were used for machine learning and Java was used to extend the existing project. Installation of a deep learning framework requires work as there are usually a lot of dependencies. This was no exception, though once the installation was complete, the project ran smoothly.

The implementation involved several aspects — creation of the database, construction of model architecture, training the model, and testing. This process was repeated several times till the desired result was achieved.

8.1.1 Creation of Database

The glyph images exported by the OCR tool are binary images i.e. they consist of either black or white pixels. Since convolution networks only support colour and greyscale images, the image type for the database was chosen to be greyscale.

The images need to be normalised to the same size. Different sizes (16×16, 28×28, 64×64 and 256×256) were tested and the size which returned the best accuracy was chosen. The images were squashed to comply with the size as it accounted for the whole glyph. The other option was to crop a segment of the image to comply with the size. However, this option is not ideal. There are certain hieroglyphs that contain many connected 'cursive' lines, as seen in Figure 4. If a crop of this image was considered, it could be misconstrued as modern text by the network. An additional variable of what region should be cropped also needs to be considered. If the middle region of the image should be chosen, then glyphs such as (find glyphs) would not be represented accurately. If either side region were selected, (find other glyph) would cause the same problem.

Figure 4: Glyph with similar properties to cursive handwriting

To create a training dataset, images needed to be collected and segregated into their respective classes.

There were two approaches taken to create a training data set — manual and using a prototype database. The manual approach involved the collection of data by using the OCR tool on documents. The intuition was that if a range of symbols was collected that were different enough, the CNN model would be able to predict the distinction accurately. There were 386 ancient images and 176 modern images collected. The number of modern images is significantly lower as the transcribed documents contained fewer modern text glyphs.

The second approach was to use the prototype database in the OCR tool. In the original OCR tool, a prototype refers to a classified example of a sign. Initially, the prototype database is empty. The user has to manually classify each sign encountered. These are then added to the database and used to predict the next occurrence of the sign. During the next classification attempt, prototypes for the database are used and each of them is compared to the current glyph. The closest glyph to a prototype is labelled. If the classification is incorrect, then it can be manually corrected. In this way, the database keeps growing and the accuracy of the OCR tool increases. This database contains several prototypes for a single sign. It is to account for minor handwriting that may result in misclassification. Using this as a training dataset for machine learning reduces chances of overfitting as the network becomes less sensitive to minor changes in a glyph. The extracted features are more robust.

In total there are 2,388 images. 20% of the images are designated to be the validation set. Therefore, the remaining 1,911 images result in 1620 ancient images and 291 modern images.

The data stored in the back-end can be of 2 types — LMDB and HDF5. Lightning Memory-Mapped Database Manager (LMDB) is a BTree-based database management library. The entire database is exposed in a memory map, and all data fetches return data directly from the mapped memory. It is extremely high performance and memory-efficient. [11] Hierarchical Data Format 5 (HDF5) is a data model, library and file format for storing and managing data. It supports an unlimited variety of data types, and is designed for flexible and efficient I/O and for high volume of complex data. [17]

Since HDF5 is more suitable for large amounts of data, LMDB was chosen to store the database.

8.1.2 Creation of Classification model

As previously described, transfer learning is very useful to train networks on sparse data without overfitting. This is assuming that the base network is trained on a similar problem to the target network. Due to the nature of the task at hand, finding a similar network was not possible. Transfer learning or fine tuning was not used for the final network because of this. However, the architecture of an existing network, called LeNet, is used. The model problem, though different than the current task, shares some image similarities — handwritten, grayscale and the size. The architecture is simple enough to address our problem of binary classification.

The general architecture is

$$input -> conv1 -> pool1 -> conv2 -> pool2 -> ip1 -> ReLU -> ip2 -> output$$

as can be seen in Figure 5. It accepts images of size 28×28 and contains 7 layers, not including

the input layer. As discussed, the images are squashed to the accepted size.

Hyperparameter values need to be optimised to maximise the accuracy and minimise the loss of a network. The process of optimisation is subjective and depends on the expertise level of the programmer. It is essentially a trial based approach that is cumbersome and is not easily reproducible. It may even result in sub-optimum networks. To deal with these issues, implementing an automatic way to test the hyperparameters is needed.

A random search or grid search finds the best hyperparameter configuration randomly. Also, the trials are independent of each other. A better approach is to model the relations between the hyperparameters and predict a smarter value. Doing this, on average, reduces the number of trials required in finding the most optimal configuration. Bayesian optimisation is an example that uses a Gaussian process to optimise the expected improvement between trials. *Spearmint* is a parameter search via Bayesian optimisation implementation. It has been used to carry out the hyperparameter optimisation.

The hyperparameters that have been optimised are listed below:

- Number of filters in $conv1$
- Number of filters in $conv2$
- Number of filters in $ip1$
- Filter size in $conv1$
- Filter size in $conv2$
- Base learning rate
- Weight decay

The layers of the CNN, along with the hyperparameter values have been listed below. Layer $conv1$ in the first convolutional layer has hyperparameters with values:

- Number of filters: 10
- Size of filter: 3
- Stride: 1

Therefore, the size of the feature maps is 26×26. It contains 100 trainable parameters,

Layer $pool1$ is the first pooling layer with hyperparameters

- Spatial extent: 2
- Stride: 2

The input of this layer is the output of $conv1$. Using the formula previously described, the output of this layer is of size 13×13.

Layer $conv2$ is the second convolution layer with hyperparameters:

- Number of filters: 70
- Size of filter: 3

- Stride: 1

The output feature maps of this layer are 11×11 and there are 6,370 learned parameters.

Layer *pool*2 is the second pooling layer with hyperparameters:

- Spatial extent: 2
- Stride: 2

The output of this layer is 70 filters each of size 6×6.

Layer *ip*1 is the first fully connected layer with number of filters as 700. It multiplies each input with the number of filters of this layer, i.e $700 * 70 * 6 * 6 = 1,764,700$ to get the number of parameters. This number is quite large, however, the model has been tested for overfitting and it returned false. Therefore, it can be said that the extra number of parameters increase the accuracy of the model without leading to overfitting. This is further demonstrated in Section 9

Layer *ReLU* is a rectified unit layer that removes the negative values.

Layer *ip*2 is the second fully connected layer with the number of filters as 2. These numbers represent the classification labels as specified in the training set.

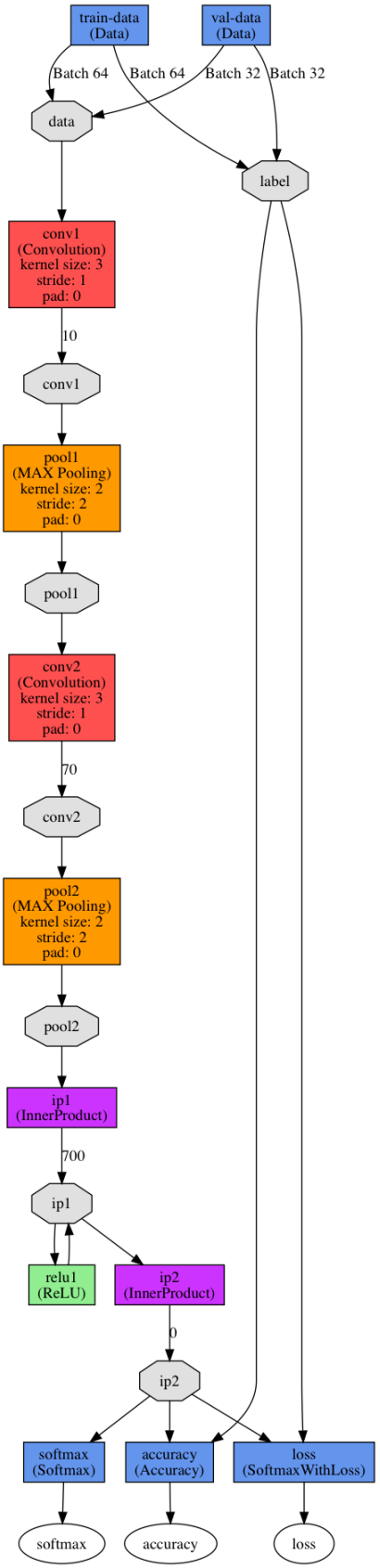

Figure 5: Model

The testing and evaluation of this model are in Section 9.

8.1.3 Java Implementation

A new java class was created `CNNDeepLearning` in which the Keras CNN model is initialised. It also specifies a function which uses this model to classify an image. This class is only initialised once per program run. This is important as the initialising process of a CNN is slightly computation heavy. The network does not change when different parts of the OCR are accessed, which makes reinitialisation redundant.

A design decision as to how the predicted labels were to be displayed was required. Simply predicting all the labels within a given line is not ideal as it would be very slow, and sometimes unnecessary. Instead, when a glyph is in focus i.e. it is clicked, the label appears above the selected glyph, as can be seen in Figure 6.

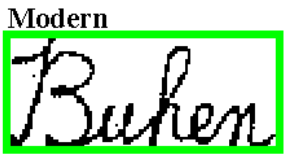

Figure 6: Glyph with label

9 Testing & Results

This section describes the testing datasets, processes, and the results.

9.1 Datasets

Caffe CNN models were tested for 3 accuracies — training, validation and testing accuracy. The training accuracy is the accuracy of the model on the data used to train the model. Validation accuracy is the accuracy on the validation set used during the training process to monitor the progress. Finally, test accuracy is the accuracy on a manually collected test set.

The training set contains 1,911 images— 1620 ancient and 291 modern images. The ratio of ancient to modern images is maintained in both the validation and test set. This is to ensure that there is a fair accuracy measurement. The details of the training set data has been described in Section 7.2.3.

The validation set is a randomly selected subset of the training set amounting to 20%. The ratio is still maintained. As discussed previously, k-fold cross-validation was not implemented.

Data for the test set was collected from the provided *Urkunden IV* and *Kitchen Ramesside Inscriptions*. The model is intended to classify images from these scripts, therefore collecting test data from these was most logical. The data was collected in an unbiased manner. An attempt was made to reduce the number of similar images. The test set contains 83 ancient images and 15 modern images, thus maintaining the original ratio. An additional text file was created which contains the ground truth labels for each of the images to check the accuracy. It should be noted that this data

set is not used in any phase of the model creation. Therefore, the test result accuracy is a good estimate of how the model would generalise.

9.2 Python

A python script was written to automate the process of testing several models on the given datasets. The images are loaded from a specified directory. A mean image is applied to them if it exists. Different networks are then tested on these datasets and the accuracy is calculated. The networks used are described below:

- *OldDataset*- Model trained on old dataset containing manually collected images
- *NotOptimised*- Model with manually set hyperparameters
- *FinalSGD*- Final optimised model using SGD solver
- *FinalNAG* - Final optimised model using NAG solver
- textitFinal - Final optimised model using ADAGRAD

9.3 Results

Since there is no paper published that tackles a similar problem, a very basic **baseline** was created for demonstration purposes. The training data set contains 2,388 images, out of which 2,023 are ancient and 365 are modern. If an algorithm were to classify each image it sees as ancient, it would have an accuracy of **84.7%**. This is assumed to be the baseline.

As discussed previously, a CNN model is trained to maximise the accuracy and minimise the loss. Figure 7 is a representation of these values during the training process. It can be seen that the training loss (blue line) follows the trend of a 'good learning rate' line, as seen in Figure 2. This means that the learning rate value for this model is appropriately set. The orange line indicates the validation accuracy of the model (**94.79%**). Though it stabilises quite early, the loss keeps reducing.

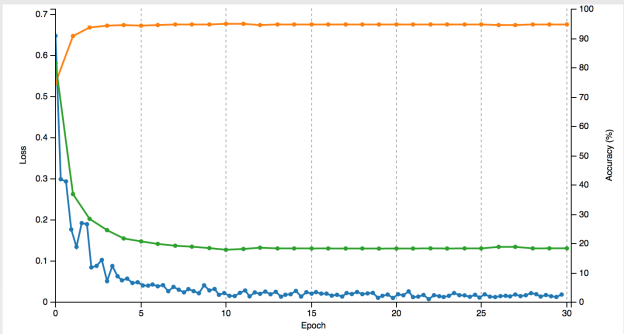

Figure 7: Final model training graph in DIGITS

Using the python code, the training dataset accuracy on the model is **98.91%**, whereas the testing dataset accuracy is **96.875%**. For the given training data and network architecture, these values are the highest possible accuracies. This is because bayesian hyperparameter optimisation was used, which ensures the best configuration for the model.

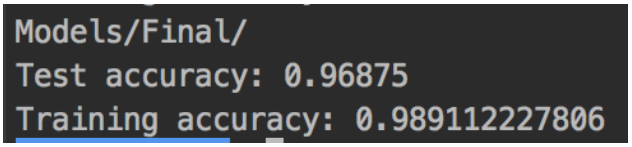

Figure 8: Final model train and test accuracy

10 Future work

With the deep understanding of CNNs gained while doing this project, there are potentially many uses within the current OCR tool. The current classification algorithm can be upgraded to use CNNs despite the low number of training images. In my opinion, there are two possible solutions to this problem — use a pre-trained network and employ transfer learning or expand the current training dataset by applying image manipulations to cause slight variances. In either case, since the dataset is small, to prevent overfitting cross-validation can be used without worrying about time or computation cost.

CNN also has very strong support for image recognition. The task of identifying different elements on the page, such as headers, footnotes or even line direction specifying glyphs can potentially be recognised with a CNN model.

11 Conclusion

The task of classification of ancient and modern images has been successfully implemented into the existing OCR tool. A novel approach of deep learning has been used which optimises the performance of the classification. An accuracy of 96.875% has been achieved from a baseline of 84.7%. Caffe (deep learning framework) along with multiple plug-ins such as DIGITS have been understood and used. The model from this C++ framework has been imported into Java using DL4J. The imported model is then used in the OCR tool to classify images based on where the user clicks.

References

[1] Ancient Egyptian Dictionary Project. Thesaurus linguae aegptiae. `http://aaew.bbaw.de/tla/`. Accessed: 2017-04-04.

[2] Apple. Performing convolution operations. Accessed: 2017-04-04.

[3] Soheil Bahrampour, Naveen Ramakrishnan, Lukas Schott, and Mohak Shah. Comparative study of caffe, neon, theano, and torch for deep learning. *CoRR*, abs/1511.06435, 2015.

[4] (cf. Ramses. Ramses. `http://ramses.ulg.ac.be/`. Accessed: 2017-04-04.

[5] cs231n. Cs231n convolutional neural networks for visual recognition. Accessed: 2017-04-02.

[6] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[7] Hendy Irawan. Aether/ant tasks. https://wiki.eclipse.org/Aether/Ant_Tasks. Accessed: 2017-04-04.

[8] jameyg42. invalid version range in java8-and-higher profile. https://github.com/jai-imageio/jai-imageio-core/issues/23. Accessed: 2017-04-04.

[9] 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.

[10] Yann Lecun, Lon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[11] The LMFDB Collaboration. *Lightning Memory-Mapped Database Manager*, 2013. available from http://www.lmfdb.org/.

[12] (Mark-Jan Nederhof. St andrews ancient egyptian texts. https://mjn.host.cs.st-andrews.ac.uk/egyptian/texts/. Accessed: 2017-04-04.

[13] Richard Mattessich. The oldest writings, and inventory tags of egypt. *The Accounting Historians Journal*, 29(1):195–208, 2002.

[14] Mark-Jan Nederhof. Orc of handwritten transcriptions of ancient egyptian hieroglyphic text. *Altertumswissenschaften in a Digital Age: Egyptology, Papyrology and beyond, Leipzig*, 2015.

[15] Peter Thomas. Why you should use the maven ant tasks instead of maven or ivy. https://ptrthomas.wordpress.com/2009/03/08/why-you-should-use-the-maven-ant-tasks-instead-of-maven-or-ivy/. Accessed: 2017-04-04.

[16] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. *Artificial Neural Networks–ICANN 2010*, pages 92–101, 2010.

[17] The HDF Group. Hierarchical data format version 5. https://support.hdfgroup.org/HDF5/. Accessed: 2017-04-04.

[18] ujjwalkarn. An intuitive explanation of convolutional neural networks. Accessed: 2017-04-02.

[19] Yangqing Jia and Evan Shelhamer. Caffe. http://caffe.berkeleyvision.org/tutorial/solver.html. Accessed: 2017-04-04.

[20] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014.