2AMM10 Assignment 1

Omniglot character recognition

Authors:

Danylo Vasylyshyn	1815709
Sam van der Velden	1017871
Stan Meijerink	1222737

Problem formulation

To make an analogy to human performed tasks, the given task is similar to recognizing handwriting. In this task one has to recognize and identify characters to known characters of an alphabet, while also dealing with noise resulting from individual differences. Furthermore, this skill of analyzing similarity between characters extends to unknown characters as well, such as hieroglyphs.

In machine learning, performing a similar task where glyphs need to be seperated can be formulated as an image retrieval task. A neural network will implement this task by creating an embedding for similarities between images in continuous vector space. By comparing the embeddings a judgement on similarity can be made. A cutoff value is then used to come to the binary conclusion on whether the character is identical to the query image or not.
This particular approach to this problem, where a single reference query image is used, is called weakly supervised learning.

The end goal is to identify whether characters are the same or not, as mentioned before, this is a binary result, which can be quantified in an accuracy score. This can be done using the ratio of correct identifications to the amount of total trials. There is no specific reason to use precision or recall in this model specifically, as there is no reason why both false positives and false negatives could not be tolerated. Therefore accuracy was chosen to use as the metric for determining the performance of the neural network.

Before this can be done however, it is needed to first determine a threshold that serves as the aforementioned cut off for the final evaluation. This threshold is defined by first gathering for all queries the furthest euclidean distance of the image that is still labeled as identical, as well as the closest image that is labeled as different. An iteration of 100 equal steps between these two values is performed and the accuracy is measured each time. Finally, the best performing threshold, based on its accuracy, is chosen.

Model formulation

To execute the image retrieval task an triplet model will be used as presented in the paper[1]. In this triplet model each triplet will have an anchor, a positive and a negative, where the positive denotes the image that is the same as the query, while the negative is different. This will allow the model to determine if a handwritten character is similar to the anchor (aka positive) or different from the anchor (aka negative). This will then facilitate metric learning where these triplets will be optimized so that the distance from a positive to the anchor is always shorter than the distance from a negative to the anchor.
To optimize the loss of the triplets the following loss formula will be used:

$$TripletLoss(x_a, x_p, x_{cn}) = max(0,\ m + \|f(x_a) - f(x_p)\|^2 - \|f(x_a) - f(x_{cn})\|^2)$$

[1] https://arxiv.org/pdf/1503.03832.pdf

In the tripletLoss the function $f()$ retrieves the embeddings for the corresponding handwritten characters, m denotes the margin, x_a is the anchor (the query image), x_p is the positive, and x_{cn} is the closest negative (the negative closest to the anchor).

The reason the triplet model is a good fit for the problem formulation is that while the handwritten characters, which are given as input, are not necessarily known by the model. The triplet model only checks which sets of features are present in a character and compares that to the given query image, which will work on any possibly new characters. The improvement to the triplet loss is also taken into account by taking the closest negative as mentioned in the paper[2].

A convolutional neural network will be used while the handwritten characters can experience transformations however the output should remain the same, thus invariance is needed. The model consists of 5 convolution layers and a linear layer at the end. Each convolution layer has 16 filters (except for the last one which has 10 filters) and has kernel size of 5, padding 1, and stride 1. After the second convolution layer a normalization function is performed and before the linear layer an average pool function is performed with a filter size of 4.

The input will consist of 5 times 28x28 images of each having 1 binary layer. For each image the corresponding embedding will be calculated using the model. The output will be a 150 dimensional vector for each of the 5 input images. To determine if an image should have the same label as the query image, the following distance measure is used to calculate the euclidean distance between the query image and any other image. When the distance is lower than a set threshold determined by the model, it will be labeled the same as the query image. In the distance calculation function $f()$ retrieves the embeddings of the images. In this formula x_a is the anchor (the query image), and x_i is any other image from the sample.

$$Distance(x_a, x_i) = \sum ||f(x_a) - f(x_i)||^2$$

Implementation

The neural network takes as an input batch of images, and produces an embedding (n-dimensional vector) for each provided image. The neural network consists of 5 convolutional layers and a single linear layer. It was observed that the optimal number of convolutional layers is 4 - 5 (the experiments in the "**Experiments and Results"** section will display data for this number of layers). Networks with lower number of layers were less accurate, and those with more than 5 convolutional layers took longer to train, but did not give any increase in accuracy. Each convolutional layer is followed by ReLU activation function, which is chosen to deal with the vanishing gradient problem. After the activation in the second layer there is a batch normalization. It was observed that the batch normalization layer results in ~0.04 accuracy increase.
The complete model architecture can be seen in table 1.

[2] https://arxiv.org/pdf/1503.03832.pdf

Table 1 - Model Architecture

Layer number	Layer description
1	Convolution(filters=16, kernel=5, stride=1, padding=1)
	ReLu()
2	Convolution(filters=16, kernel=5, stride=1, padding=1)
	ReLu()
	BatchNormalization()
3	Convolution(filters=16, kernel=5, stride=1, padding=1)
	ReLu()
4	Convolution(filters=16, kernel=5, stride=1, padding=1)
	ReLu()
5	Convolution(filters=10, kernel=5, stride=1, padding=1)
	AvgPool(kernel=4)
6	Linear_layer(256, 150)

As can be seen in table 1, each convolution layer has 16 filters. As the color of the images can be interpreted as binary, the input channel for the first layer is 1. All of the convolutional layers have kernel size of 5, padding 1, and stride 1. A kernel size of 5 is more effective than a kernel size of 3, since a higher degree of freedom caused by larger kernel size provides the advantages of capturing some more complex patterns in the earlier layers of the network ending up with a more detailed description of the image in the output. Although it is expected that with the increase in the degrees of freedom, a larger number of filters would be needed to capture the necessary patterns. 16 filters is found to be substantial and a further increase in the number of filters does not result in any substantial accuracy increase because the figures in the dataset are primitive. A stride of 1 was chosen as increasing the stride to 2 would result in too little data as the pixel count is already quite small.

The last layer of a neural network is a linear layer. Its purpose is to produce the embedding based on the features in the last convolutional layer. A 150 dimensional embedding is used. Experimentally it was shown that by increasing the dimensionality of the embedding the accuracy increases. But a too high dimensionality would introduce unnecessary complexity to a model, especially given such primitive figures.

The decisions taken for the different parameters of in the model are based on the results of the experiments described in the section “**Experiments and Results”** as well as the reasoning that will be explained for those decisions in the following section.

Experiments and results

For the experiments a number of factors are going to be changed to see if the model could be improved. The conducted experiments and the corresponding results will be presented below in the table below.

Before the experiments the loss and accuracy of the default model after 7 epochs were:

- Loss (training) = 0.0908
- Loss (testing) = 0.1057
- Accuracy (testing) = 0.901

As mentioned before, the model performs 7 epochs. The table below shows the performance over epochs. As can be seen in table 2, the accuracy does not increase anymore from the 6th epoch onwards. To balance the accuracy and the running time of the model it was decided that 7 epochs would be sufficient to train the model.

Table 2 - Performance over epochs

Epoch	1	2	3	4	5	6	7	8	9	10
Loss (training)	0.126	0.106	0.099	0.097	0.094	0.093	0.091	0.089	0.088	0.086
Loss (testing)	0.117	0.107	0.111	0.105	0.105	0.102	0.106	0.102	0.103	0.101
Accuracy(testing)	0.858	0.887	0.882	0.896	0.901	0.901	0.901	0.902	0.900	0.907

In the experiments below the following changes were compared to the default model as described above.

- Changing the depth from 5 layers to 4 and 6
- Changing the kernel from 5 to 3 at: the first layer, all layers and the only the last layer
- Changing the filter count from 16 to 8 and 32
- Changing the margin of the loss function from 0.2 to 0.1, 0.3, and 0.4
- Changing the number of output embeddings from 150 to 50, 100, 200

Table 3 - Experimentation network depth

Network depth	Depth of 4 layers	Depth of 5 layers (default model)	Depth of 6 layers
Loss (training)	0.0879	0.0908	0,0905
Loss (testing)	0.1043	0.1057	0.0993
Accuracy (testing)	0.901	0.901	0.9023

Table 4 - Experimentation kernel sizes

Kernel sizes	Kernel size 3 (first layer)	Kernel size 3 (all layers)	Kernel size 3 (last layer)	Kernel size 5 (all layers)
Loss (training)	0.0880	0.0921	0.0876	0.0908
Loss (testing)	0.0991	0.1000	0.0963	0.1057
Accuracy (testing)	0.9067	0.9007	0.9057	0.901

Table 5 - Experimentation amount of filters

Amount of filters	8	16 (default)	32
Loss (training)	0.0960	0.0908	0.0871
Loss (testing)	0.993	0.1057	0.0961
Accuracy (testing)	0.896	0.901	0.9092

Table 6 - Experimentation margin value

Margin	0.1	0.2 (default)	0.3	0.4
Loss (training)	0.0447	0.0908	0.1319	0.1803
Loss (testing)	0.0493	0.1057	0.1455	0.1906
Accuracy (testing)	0.9060	0.901	0.9016	0.9022

Table 7 - Experimentation number of output embeddings

Number of output embeddings	50	100	150 (default)	200
Loss (training)	0.0909	0.0913	0.0908	0.0894
Loss (testing)	0.1004	0.0997	0.1057	0.1018
Accuracy (testing)	0.903	0.898	0.901	0.903

Results and analysis

As can be seen in the experiments on the model depth in table 3, the model of 6 layers does not have an (significant) improvement in accuracy and loss, and thus 5 layers will be the maximum for the model.

Changing the kernel size to 3 on the first, last, or all layers proved to give no significant improvements to the model as can be seen in table 4, the variance was within run to run variance of the default model. A change in stride was not experimented upon while the images were of low quality, thus increasing the stride to 2 would have meant that a lot of detail in the images would get lost which would worsen the ability of the model to recognize the characters.

Changing the amount of filters to either 32 or 8 showed no increase in performance of the model as can be seen in table 5, and was within run to run variance of the default model. A change in margin or number of embeddings also showed no improvements for the model compared to the default as can be seen in tables 6 and 7.

As can be seen in the experiments, none of the performed changes had any performance improvements regarding the default model. In some experiments the accuracy was slightly higher, however this was within the run to run variance the default model has. the run to run variance is ~1%.

Conclusion

Our model performs reasonably well at ~90% accuracy. As for limitations, the training and test data do not have backgrounds, limiting the model's generalizability to either highly situational or preprocessed data. Besides that, the images are low in pixel count, leading to anomalous features that might not have been present in the high quality reference image. If a high quality image was used, a higher stride and padding count could be considered. Third, the image contains only two colors, the model is naturally not applicable for cases in which a third color is added.

As conclusion, the used convolutional neural network is adequately performing a weakly supervised metrics learning task as presented in this assignment. The combination with triplet loss for performing such a task leads to a high accuracy while the triplet loss is a state-of-art method as presented in the paper on triplet loss[3].

While our model performs reasonably well at ~90% accuracy, there are a few improvements that could be considered in the future implementations that could likely increase the performance of the model.

[3] https://arxiv.org/pdf/1503.03832.pdf

1. **Data augmentation:** currently the model is being trained on the same data during each epoch. However making some little random adjustments to the training datapoint could help the model generalize better. Possible adjustments to the figures in the dataset could include:
   - **Some amount of rotation:** However by rotating the initial picture a lot we risk to train model to be rotationally invariant, which is not desirable, since a flipped picture of a certain symbol could resemble a different symbol.
   - **Scaling the pictures:** It is a desirable property for a given problem to train a model to be invariant to the symbol. By introducing such augmentation, it is likely that the accuracy of the model will increase.
2. **Batch sampling:** The current training process is going through the whole dataset in the same sequence each epoch. For the given task, sampling is partially performed by the loss function (since the value of the loss is calculated for a different negative example every time). However introducing random sampling may further disable the model to learn correlations in the dataset sequence.
3. **Higher quality images**: The dataset could be changed to higher quality images for less noise in the training dataset, where the features of the actual glyphs are better represented.
4. **Real Images:** The training dataset could be trained on real images of handwriting under various lighting conditions for better generalizability to real world applications.

Work division

Danylo: code implementation + implementation part in the report + conclusion in the report + parameter experimentation
Stan: Model formulation + experiments and results + results and analysis + tweaks in the code + layout and consistency of the report + parameter experimentation
Sam: Problem formulation + conclusion in the report + layout and consistency of the report + tweaks in the code + parameter experimentation