2AMM10 2021-2022 Assignment 1: Group 47

Ahmet Ayrancioglu (1678620) Ricardo Andrade (1725882) Ruben Wolters (1342355)

May 23, 2022

1 Problem formulation

Image retrieval is the task of finding and retrieving an image from a collection of images based on the characteristics of a designated query image. These characteristics range from a simple visual resemblance to more complex patterns, like a related style or quality between two images. This problem has been extensively researched in the last few years and several different models have been developed and tuned to improve the results obtained for the image retrieval problem.

In this report, a model is showcased that is capable of recognizing characters from alphabets from around the world. The provided dataset is organized in tuples containing a query image x_q and support set images $\{x_i\}$ where $i \in [0, 4]$. For each tuple, the model finds the set of images from the support set that corresponds to the character given in the query image. More formally, the goal is to find the set $\{j\}$, where $j \in [0, 4]$ and $1 \leq |\{j\}| \leq 3$.

To solve this problem, first, a suitable way to represent the original images is found in such a way that reflects the characteristics of the images followed by the computation of a distance measured between the images. Finally, the desired results mentioned before are obtained.

2 Model formulation

Following the previous section and the problem at hand, a model architecture known as the Convolutional Siamese Neural Network[2] is used. A Siamese Network consists of two or more identical subnetworks that produce feature vectors for each input and compare them. An example of a Siamese Network can be seen in Figure 1. Siamese Networks have the particularity that the subnetworks that compose them share the same weights and parameters. The symmetry of the network means that when two distinct images are pushed through each of the twin networks, the top layer will compute the same metric as if the same two images were pushed through the opposite twins.

The reason for using a Siamese Network is that the dataset does not have enough examples for every character. This makes training a classification model that would recognize every character extremely hard. However, as the focus is not on identifying each character individually, the model can be trained to learn the characteristics behind characters and distinguish between characters accordingly. By using this approach a distinction between characters that the model has not seen during training can be made, which makes it ideal for this assignment.

Siamese Networks are usually paired with Triplet Loss or Contrastive Loss. For this assignment, Triplet Loss is used. In its essence, the Triplet Loss compares an Anchor (initial input) to a Positive (positive input) and a Negative (negative input) with the objective of reducing the distance between the Anchor and Positive while increasing the distance between the Anchor and the Negative. The formal definition of the Triplet Loss is the following:

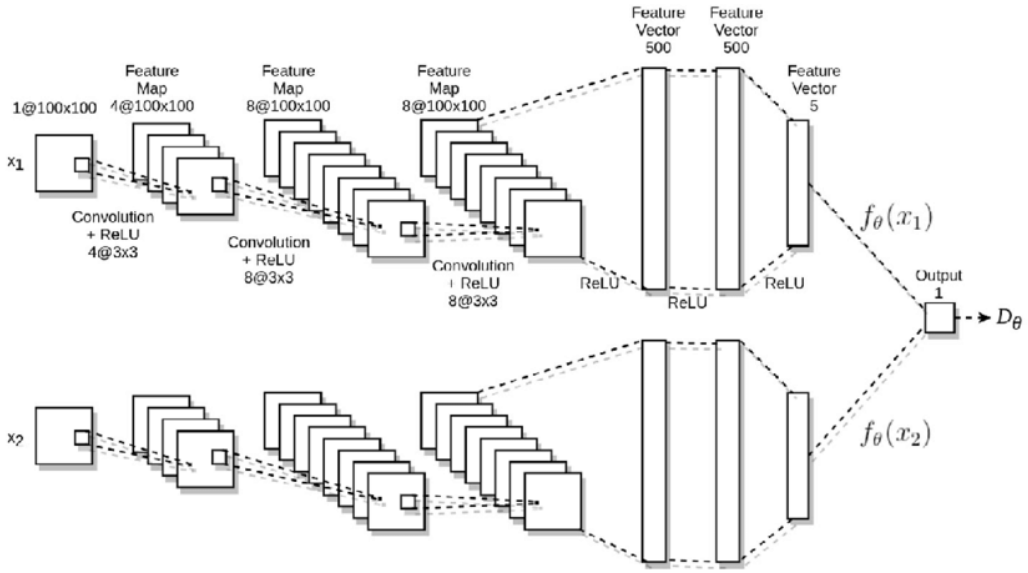

Figure 1: Example of a Siamese Network[1]

$$\mathcal{L}(A, P, N) = \max(0, ||f(A) - f(P)||^2 - ||f(A) - f(N)||^2 + \alpha),$$

where $f(A)$, $f(P)$, and $f(N)$ are the feature embedding for the Anchor, Positive, and Negative respectively, and α is the margin that is used to extend the distance between positive and negative pairs.

Since the model is trained in batches, the loss function can be defined as

$$\mathcal{L}(A, P, N, M) = \frac{1}{M} \sum_{i=1}^{M} \max(0, ||f(A_i) - f(P_i)||^2 - ||f(A_i) - f(N_i)||^2 + \alpha),$$

where M is the size of the batch. The loss that is propagated backward through the network is the mean of all the losses of the triplets in the batch.

3 Implementation and training

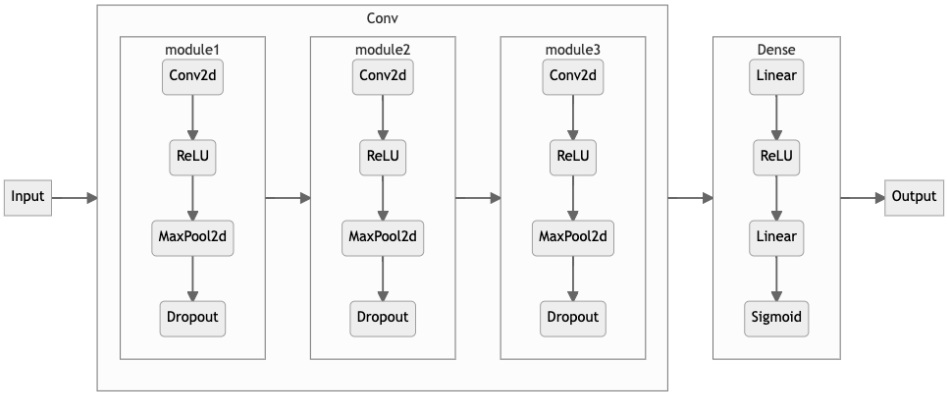

Figure 2: Diagram of the used model.

The model consists of 2 parts, the convolution layers, and the dense layers. A high-level representation of the model can be seen in Figure 2. The convolution layers follow a simple approach of duplicated modules, where the same module composed of multiple layers is duplicated 3 times. The only difference between the modules is the number of filters in the Conv2D layer, which start from 32 for the first module and is doubled for each consecutive module, ending with 128 for the last layer. Each module consists of 4 layers. The exact description of these 4 layers can be seen in Table 1.

Table 1: Table showing the layers used in each of the sequential convolution blocks

Layer	**Type**
1	Conv2D: kernel size=3x3, stride=1x1, padding=same
2	ReLU
3	MaxPool2D: kernel size=2x2, stride=2x2, padding=none
4	Dropout: p=0.3

For the dense layers again a simple architecture is used where only 4 layers are used in order to create the embeddings. The exact description of the dense layers can be seen in Table 2.

Table 2: Table showing the layers used in the final dense block

Layer	**Type**
1	Linear: input=1152, output=512
2	ReLU
3	Linear: input=512, output=128
4	Sigmoid

For the loss function, Triplet Loss was chosen, as the goal of training this model is to distinguish between different characters, and Triplet Loss closely reflects the semantics of the goal in terms of mathematics. Additionally, by using Triplet Loss a more efficiently trained model can be made, as with each back-propagation step the negative examples are pushed further away, while positive examples are pulled closer at the same time. A margin of 1 is used for the Triplet Loss.

For the optimizer, AdamW is used, which is the implementation of the popular Adam optimizer with weight decay. Weight decay helps models to be more generalizable as it is a regularization technique. The parameters for the AdamW optimizer are as follows:
betas=(0.9, 0.999), eps=1e-08, lr=0.001, weight decay=0.01.

For training the model, a specialized PyTorch dataset was created. In the initialization step of the creation of the dataset, all possible triplets are produced in the form (anchor, positive, negative) for each row r_i in the original dataset. Then, whenever an item from row r_i is requested from the dataset, a random triplet from all the triplets that were created from row r_i is picked and returned. With this approach, the model is not trained on all possible triplets, which would be inefficient. Additionally, as a random triplet is returned every time row r_i is requested, the model will see all triplets given enough epochs. Then a random split in this dataset is made into training and validation sets using a ratio of 0.1 for the validation set.

After the loss and optimizer are initialized, the model is trained for 60 epochs on the Google Colab GPU with an early stopping procedure in order to avoid overfitting. In each epoch, the batch with size 32 triplets is passed through the model using a forward pass. Then the loss is calculated from the triplets and back-propagation is applied. The early stopping procedure works as follows: if the validation loss fails to improve 10 times in a row, the training is stopped to save time and prevent overfitting. In addition to this early stopping procedure, the model that achieves the lowest validation loss during training is saved and is used for inference. Figure 3 shows the loss curve for both training loss and validation loss during the training phase.

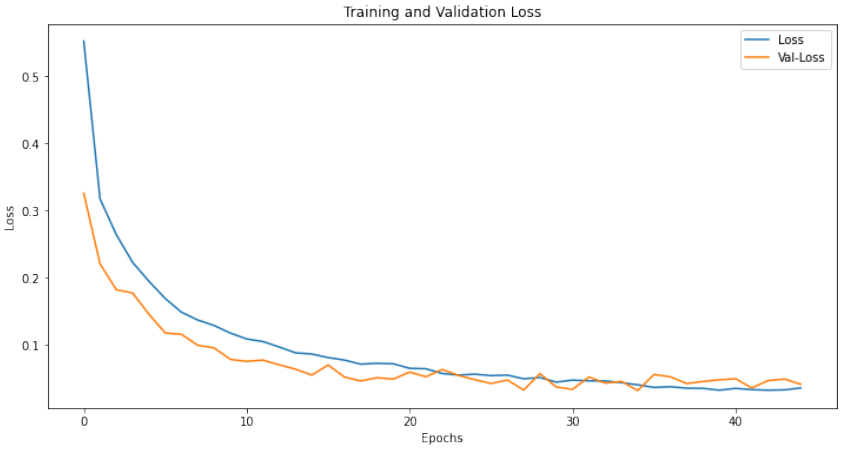

Figure 3: Loss curve of the model.

4 Experiments and results

As previously stated, the model is trained using the Triplet Loss concept. From Figure 3, it can be seen that training loss continues to decrease until around epoch 40. The training loss drops rapidly until epoch 10 and then slows down. Towards the end of the training, there is little to no improvement in the training loss. A similar scenario happens for the validation loss, but the validation loss curve is bumpier than the training loss. This is kept in mind for the early stopping procedure.

In order to evaluate the model, for each image in the test set, a prediction is made whether it is similar to the query image or not. However, as described in earlier sections, the model is only learning a representation of the images and ranking them according to how close they are to the query image. This is a good approach but it is not enough to label a particular image as 0 (different) or 1 (same). In order to go from ranking the images to classifying them, a threshold value is needed that will be used to determine a label. This threshold value is not learned and is simply a calculated estimate from the training set. After training the model, every image in the training set is passed through the model and its distance to the corresponding query image is computed. Then, the mean distance of all instances with a label 1 and the mean distance of all instances with a label 0 is calculated. Finally, the mean of these 2 values is taken and that value is set as the threshold. Any image with a distance to the query image lower than the threshold are classified as 1 (same).

The labels for every image in the test set are predicted using the threshold procedure described above. Then by using the predicted labels and the ground truth labels, the model is evaluated using pairwise metrics. The metrics used are as follows:

- Precision $= \frac{TruePositives}{TruePositives+FalsePositives}$, measures how much of the images retrieved that are relevant to the query image. Outputs values between 0 and 1.
- Recall $= \frac{TruePositives}{TruePositives+FalseNegatives}$, measures how much of the images that are relevant to the query image are retrieved. Outputs values between 0 and 1.
- F1-Score $= 2 \cdot \frac{Precision \cdot Recall}{Precision+Recall}$, is the weighted harmonic mean of precision and recall, and measures the balance between precision and recall. Outputs values between 0 and 1.
- Accuracy $= \frac{TruePositives+TrueNegatives}{TruePositives+TrueNegatives+FalsePositives+FalseNegatives}$, measures how well the model correctly classifies the samples provided. Outputs values between 0 and 1.

For all metrics, a value closer to 1 means that the model is performing well, while a value closer to 0 means the opposite.

Table 3: Model evaluation

Metric	Positive Class	Negative Class
Precision	0.90	0.96
Recall	0.94	0.94
F1-Score	0.92	0.95
Accuracy	0.94	

As can be seen in Table 3, the model performs very well. It accurately predicts for each tuple which images from the support set are similar to the query image and which ones are not.

5 Conclusion

From this assignment, a conclusion can be drawn that the implementation of the Siamese Network is effective at retrieving a particular query image from a set of supporting images. This remains the case even on the test set, which is impressive as the characters in the test set are not present in the training set. This means that the model achieves the goal of learning how to distinguish characters instead of learning individual characters.

However, this does not mean that the model cannot be improved. The process of determining the threshold could be improved by integrating the threshold into the learning procedure and letting the model learn the threshold as well. Another place where improvements to the model can be made is the loss function. Instead of training on randomly selected triplets, a smarter selection of triplets can be made, with emphasis on hard to separate triplets. This is called SemiHardTripletLoss [3], where the negative image is farther from the anchor than the positive image, but the value of the calculated loss is still bigger than 0.

References

[1] Aditya Deshpande, Ali Minai, and Manish Kumar. One-shot recognition of manufacturing defects in steel surfaces. *Procedia Manufacturing*, 48:1064–1071, 01 2020.

[2] Gregory R. Koch. Siamese neural networks for one-shot image recognition. 2015.

[3] Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2015.