Using Siamese Neural Networks to Generate Image-Based Travel Recommendations

Samantha Vaca

Abstract

Planning a trip is hard and time-consuming, but thanks to recommenders, it doesn't have to be. Today, recommenders are used in practically every industry, making the user experience simple and stress-free. However, most recommenders have access to large volumes of data, something that is not always available when it comes to recommending travel destinations, leading to what is known as the cold start problem. For example, when recommending hotels or restaurants, recommenders can make matches based on user reviews. Recommending a destination presents a different kind of problem. There are little to no user reviews for an overall city, beach, or destination. This is why a different approach must be taken: using images to recommend travel destinations. Similar images of various types of destinations can be analyzed by a neural network in order to determine their similarity. But even then, a large amount of images would be needed in order to train an effective neural network capable of predicting accurate recommendations. So in order to minimize the amount of images needed, a Siamese Neural Network will be implemented. The model will be implemented using the TensorFlow framework with a Triplet Loss function. The dataset being used to train the network is a small custom-built dataset of city and beach images gathered from Wikipedia and Wikimedia Commons. The demonstrated Siamese model was able to accurately identify similar images of similar places.

Introduction

Recommender systems seem to be implemented in almost every industry. The most prominent among these industries is the entertainment industry. Recommenders are in all of the popular streaming services, and they usually give people precisely what they want to see. And from this stems a very interesting question, how can we implement a recommender for the tourism industry that is both effective and relies on little data? When it comes to recommending a destination, there are no reviews, no ratings, and no way of knowing what the user liked or disliked. The only thing constantly present among all destinations is images. Because of this, a different kind of recommender must be implemented: one that will recommend destinations with images. In order to construct this recommender, a neural network capable of distinguishing between images of different types of destinations must be built.

Neural networks usually require large amounts of data, but some destinations may not have many published images. The best solution to this problem is a Siamese Neural Network (SNN) [6, 15]. This type of neural network requires little training data to make accurate predictions [1].

An SNN is a neural network that uses two or more subnetworks to learn similarities [4, 12]. These subnetworks must always be identical, having the same weights and parameters. An

example of the network can be seen in Figure 1. An SNN is able to learn the similarity among inputs even when given very little data, which is very beneficial for when data is scarce [2, 3].

The network utilizes a loss function. Most commonly, this is the Triplet Loss function. The Triplet Loss function takes in three inputs: an anchor, a positive, and a negative. The anchor is the original input, the positive input is in the same category as the original input, and the negative input is unrelated to the original input [16, 17].

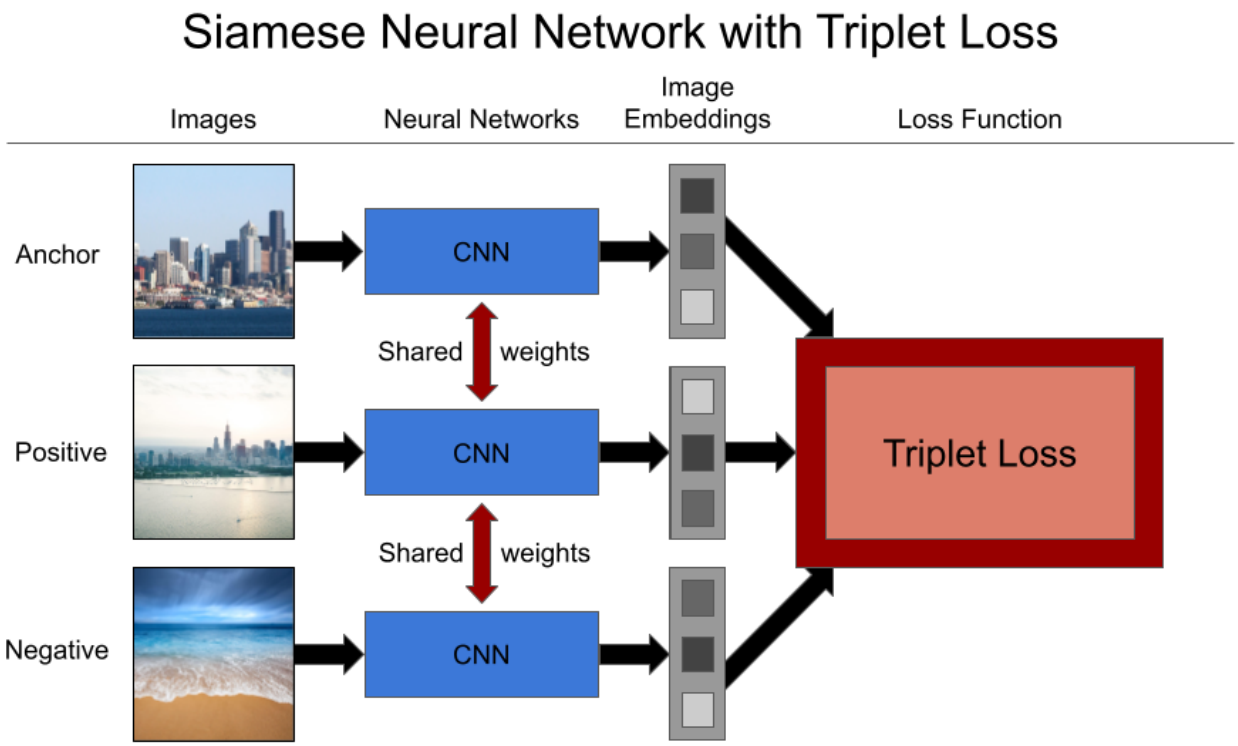

Figure 1: A Model of a Siamese Neural Network

Using an SNN with a Triplet Loss function, a recommender will then be built to predict accurate travel recommendations for users.

Methods

First is the data input. A custom dataset will be implemented using a web scraper that gathers images for different cities from their Wikipedia entries. Each image inputted into the neural network has to be vectorized. From there, it is inserted into a dataset for the neural network to train on. After this process has been completed, image triplets are generated from the images in the dataset. These image triplets help the SNN understand the difference between images of the same and different classes.

The model is then initialized using the Tensorflow framework. Once the model has been initialized, the model begins its training. A validation data set is used to tune the hyperparameters and make the predictions more accurate. Once the training has been finalized, the model is used to predict image classes and displays the predictions on a graph.

Data Gathering And Initialization

An important step in the process of building an SNN is the gathering and implementation of data [14]. Because an SNN requires little data to produce accurate results, the data that is used for the training of the model must reflect the task at hand as well as possible.

The data for this project is images that are gathered from Wikipedia and Wikimedia Commons. These images are of touristic and popular cities across the world. The images are web scraped. Once all of the images have been downloaded, they are iterated over and resized to fit the proper dimensions for the model. A separate list of labels is also created corresponding to the category of each image. Once the images and the labels have been finalized, they are all saved into a NumPy file, where they are stored in their respective training and validation arrays [5]. The final training set consists of about 2,500 images, and the validation set consists of about 250 images.

Creating Triplets

In order for an SNN to work properly, pairs must be created. Pairs guide the model in the right direction by categorizing images of the same and different classes. Pairs that are positive are images of the same class, while pairs that are negative are images of a different class [9].

With the Triplet Loss function, triplets are created. Within each triplet is an Anchor, a Positive, and a Negative input. The Triplet Loss function uses an Anchor to compare images. The Anchor is the original image, the Positive is an image of the same class, and the Negative is an image of a different class. The images that belong to the Positive and Negative classes are compared to the original image, or the Anchor, to determine whether they are similar or different [7, 8].

In the model, there are images of two classes: Cities and Beaches. An example of image triplets can be seen below in Figure 2.

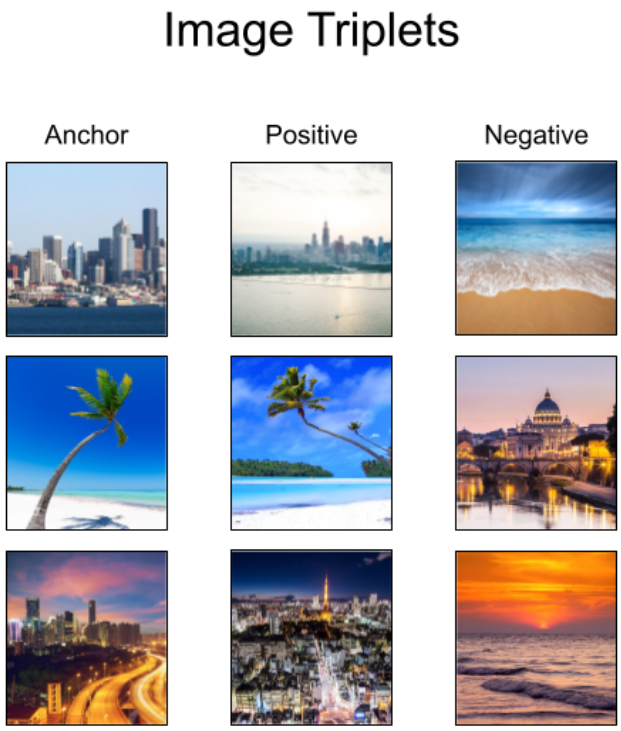

Figure 2: An Example of Image Triplets

Building The Network

In order to build the Siamese model, the network must first be configured. In this configuration, the image shape, batch size, and number of epochs are determined [10, 14]. The save path of the file is also determined here so that the model can be ready for use later.

Next is building the Siamese model [11, 13, 18]. The first step is setting the input shape, which in this case is the shape of the images from the dataset. The next step is defining the layers of the model: the convolutional layers, the dense layers, the pooling layers, and the dropout layers.

The convolutional layer is one of the main parts of any convolutional neural network. Inside the layer, there is a set of learnable filters. The repeated exposure to images during the convolutions results in a feature map. In other words, this layer helps the model learn highly specific features in images that can then be detected on testing and input images.

The dense layer is a layer of neurons that is deeply connected. Each neuron in the layer receives the input of the neurons from the previous layers. The dense layer is used to classify images based on convolutional layer outputs.

The pooling layer serves to generalize the features learned in the convolutional layer. This helps the model to be able to more easily recognize features on different images, allowing it to learn features independent of their location on each image.

The dropout layer works to prevent overfitting in a model. Overfitting occurs when the model focuses too closely on the little details of the model to the point that the features are insignificant and not reflective of the image or the category it belongs to.

After several attempts of training and tweaking, the model ended up with a total of 10 convolutional layers, 4 pooling layers, 2 dense layers, and 2 dropout layers.

Finally, the Siamese model is built using the images as the input and the image embeddings as the output.

Compiling The Model

The final step before training the model is compiling it. In order to compile it, the loss function must be defined. The Triplet Loss function is utilized for this project. This function reduces the distance between the Anchor and Positive inputs while increasing the distance between the Anchor and Negative inputs.

Training The Model

The model is finally ready to be trained once it has been initialized and compiled. While training, the SNN learns the features of the images in order to be able to attribute each image to its respective class. Training the model is necessary to be able to make predictions and apply the model for travel recommendations.

The model uses the triplets defined in the steps above. It is trained on 30 epochs in order to maximize the accuracy of the predictions.

The predictions of the validation image set are shown in Figure 3 below.

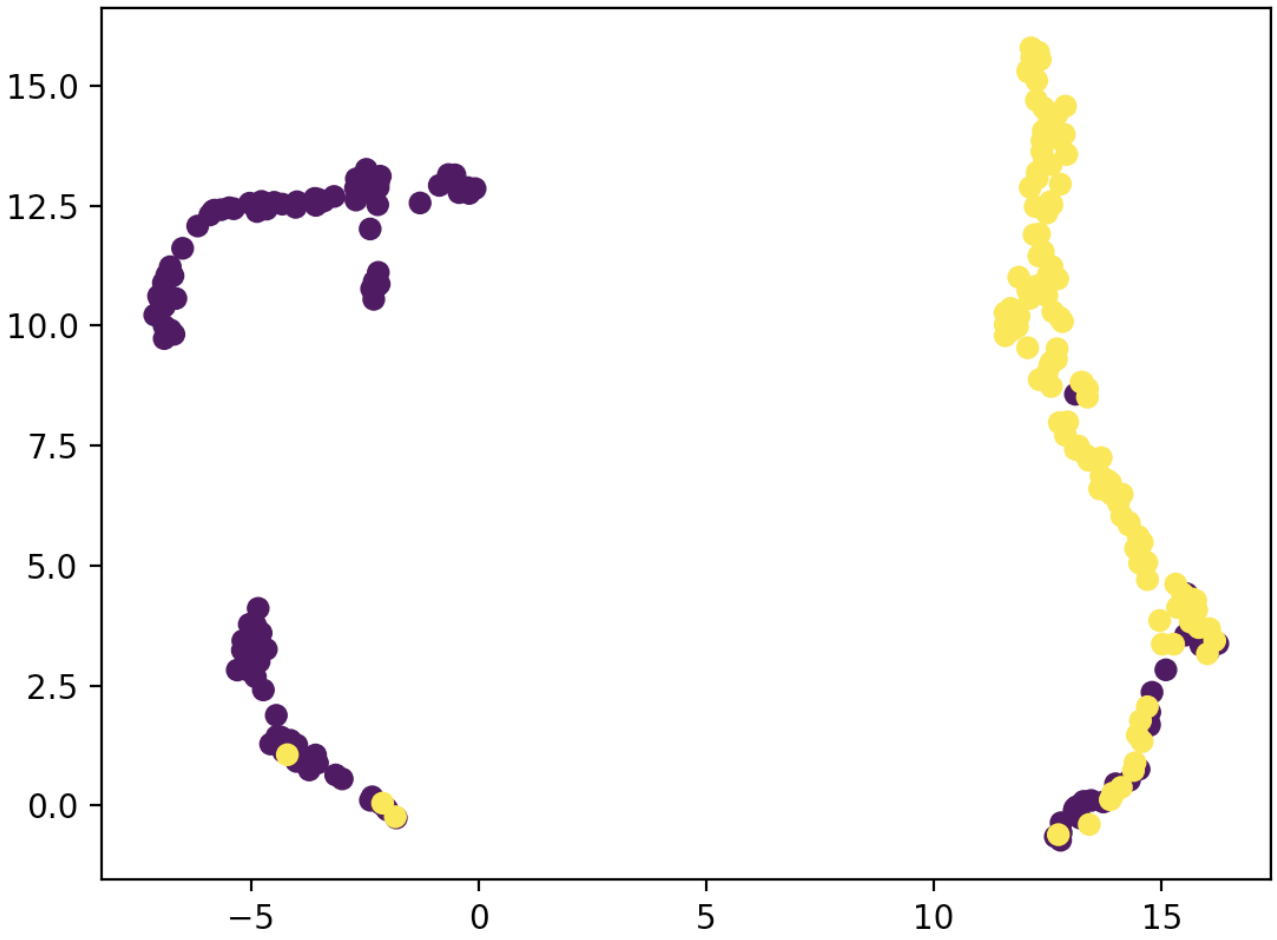

Figure 3: Graphed Validation Image Predictions

As shown above, the model has learned how to mostly distinguish between images of different classes. The colors of the graphed image embeddings indicate the actual class of the embedding: either Beach or City. Embeddings that are clustered together are identified as more similar by the model.

The final model has a loss of 0.60 with a margin of 1.

Saving The Model

Once the model has finished training, it is saved to the disk so that predictions can be made [10]. This allows for easy access of the model without the need to train it each time.

Testing The Model

In order to visualize the results of the model, a testing set of about 50 images is implemented. From here, images from the testing set are randomly picked. The randomly chosen images are grouped into pairs so that they can be compared [11]. Next, the saved model is loaded from the disk and prepared for use.

The model is finally put to the test when the images are compared for similarity. Each image pair is loaded in and resized to fit the size requirements of the model. Channel and batch dimensions are both added to the images before rescaling the pixels in the images. The model is then used to make predictions on the image pairs.

The Euclidean distance function is used to measure the similarity between the images. Once the features have been extracted from the images and put into vectors, the vectors are separated, and the sum of their squared distance is calculated [13]. This calculated distance is used to determine the similarity of the images. An example of the Euclidean distance can be seen in Figure 4.

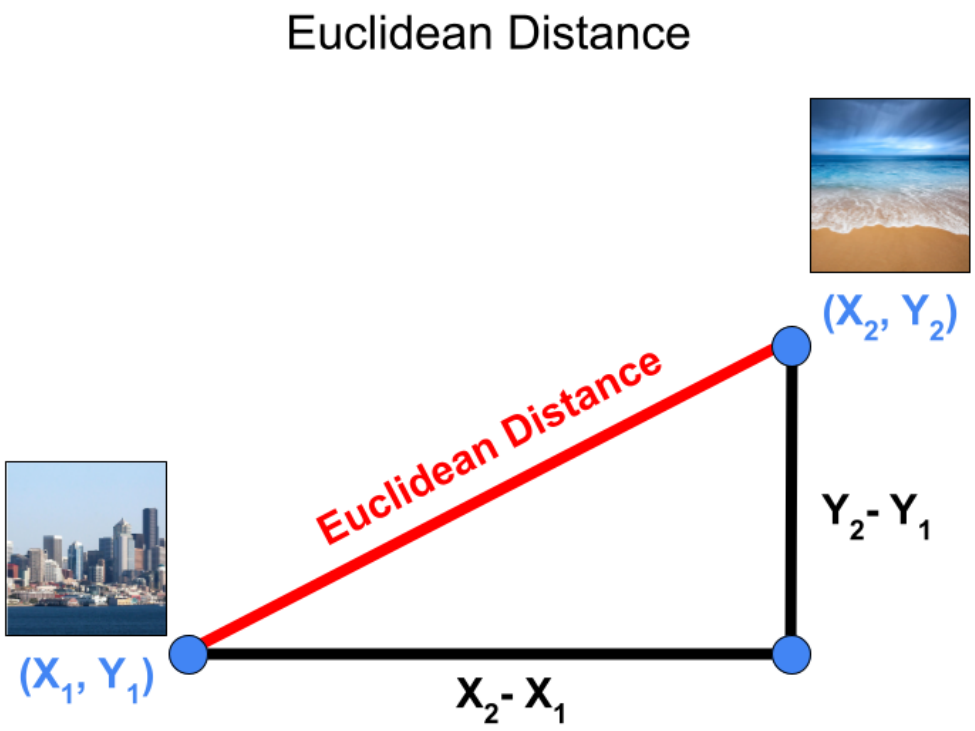

Figure 4: The Euclidean Distance

Baseline Models

A baseline model is used to compare a custom model to an already well-known one. This helps researchers understand how well their model is doing. The baseline model that will be used to compare the finished SNN model is ResNet.

ResNet

ResNet is a neural network with hundreds of layers. These layers are used to analyze the images and generate embeddings [18]. A loss of approximately 0.59 with a margin of 1 was found after implementing the ResNet model. This loss is pretty similar to the custom SNN's loss of 0.60. The ResNet model also results in more clusters, as seen in Figure 5 below.

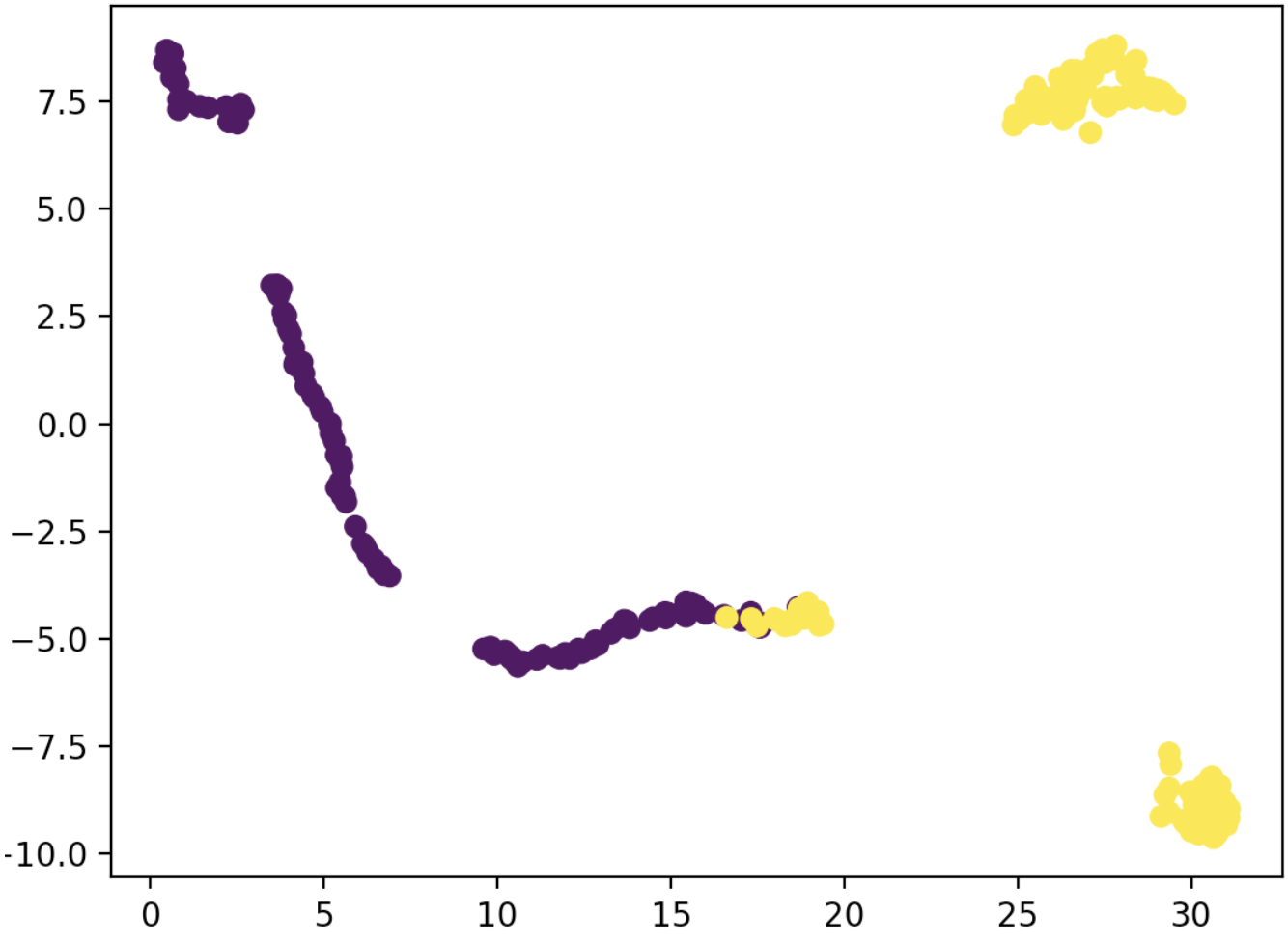

Figure 5: Graphed ResNet Validation Image Embeddings

Compared to the custom SNN model being implemented throughout this research, the ResNet model seems to cluster images by focusing on very specific details, resulting in more clusters than actual classes. The loss for the ResNet model on 5 epochs seems to do about as well as the custom-built SNN model on 30 epochs.

Results

The SNN model predicts similarity between two images by calculating the Euclidean distance between the image embeddings. The closer the predicted similarity score is to 0, the more similar the two images are.

The majority of the time, the model is able to accurately predict whether two images are of the same class, as shown in Figure 6 and Figure 7.

Figure 6: Same Class Prediction

Figure 7: Different Class Prediction

Sometimes though, the model fails to accurately predict. The image in Figure 8 below predicts a high similarity between images of two different classes.

Figure 8: False Prediction

Check out the model here: https://github.com/samanthavaca/Travel_Recommender

Model Implementation

In the future, this model will be implemented in a system that predicts travel recommendations for users. This recommendation system would find similar travel destinations for the user based on previous travel destinations and locations of interest.

How The Recommender Would Work

For each destination, the system would be given an image. A similarity score between that image and each image in the recommender dataset would then be calculated using the SNN. From there, the images that produce the highest similarity score would be stored as the recommendations. Finally, the recommendations that produce the highest scores would be displayed for the user.

Conclusion

After gathering images, building a custom Siamese Neural Network with a Triplet Loss function, and computing similarities between image embeddings, the model has begun to distinguish between images of different classes. In the future, more classes of data would likely be implemented for the model to match the real-world expectations of the many types of destinations found in the travel industry. In summary: using a Siamese Neural Network with a Triplet Loss function to distinguish between travel images shows promise and, with further research and time, allows for bright possibilities.

References

1. A. Faroughi and P. Moradi, ”MOOCs Recommender System with Siamese Neural Network,” 2022 9th International and the 15th National Conference on E-Learning and E-Teaching (ICeLeT), 2022, pp. 1-6, doi: 10.1109/ICeLeT55619.2022.9765439.
2. J, Sean Benhur. “A Friendly Introduction to Siamese Networks.” Medium, Towards Data Science, 2 Sept. 2020, https://towardsdatascience.com/a-friendly-introduction-to-siamese-networks-85ab17522942.
3. Dutt, Aditya. “Siamese Networks Introduction and Implementation.” Medium, Towards Data Science, 11 Mar. 2021, https://towardsdatascience.com/siamese-networks-introduction-and-implementation-2140e3443dee.
4. Vassallo, Thomas. “Calculating Audio Song Similarity Using Siamese Neural Networks.” Medium, Towards Data Science, 28 Aug. 2020, https://towardsdatascience.com/calculating-audio-song-similarity-using-siamese-neural-networks-62730e8f3e3d.

5. Trotter, Cameron. “How to Train Your Siamese Neural Network.” Medium, Towards Data Science, 19 Feb. 2021, https://towardsdatascience.com/how-to-train-your-siamese-neural-network-4c6da3259463.
6. Salakhutdinov, Ruslan, et al. “Siamese Neural Networks for One-Shot Image Recognition.” Carnegie Mellon University School of Computer Science, 2015, https://www.cs.cmu.edu/rsalakhu/papers/oneshot1.pdf.
7. Moindrot, Olivier. “Triplet Loss and Online Triplet Mining in Tensorflow.” Olivier Moindrot Blog, 19 Mar. 2018, https://omoindrot.github.io/triplet-loss.
8. Guha, Sunny. “Custom Loss Function in Tensorflow 2.0.” Medium, Towards Data Science, 6 Jan. 2020, https://towardsdatascience.com/custom-loss-function-in-tensorflow-2-0-d8fa35405e4e.
9. Rosebrock, Adrian. “Building Image Pairs for Siamese Networks with Python.” PyImageSearch, 23 Nov. 2020, https://pyimagesearch.com/2020/11/23/building-image-pairs-for-siamese-networks-with-python/.
10. Rosebrock, Adrian. “Siamese Networks with Keras, Tensorflow, and Deep Learning.” PyImageSearch, 30 Nov. 2020, https://pyimagesearch.com/2020/11/30/siamese-networks-with-keras-tensorflow-and-deep-learning/.
11. Rosebrock, Adrian. “Comparing Images for Similarity Using Siamese Networks, Keras, and Tensorflow.” PyImageSearch, 7 Dec. 2020, https://pyimagesearch.com/2020/12/07/comparing-images-for-similarity-using-siamese-networks-keras-and-tensorflow/.
12. Khandelwal, Renu. “One-Shot Learning with Siamese Network.” Medium, The Startup, 28 Jan. 2021, https://medium.com/swlh/one-shot-learning-with-siamese-network-1c7404c35fda.
13. Great Learning Team. “Similarity Learning with Siamese Networks.” Great Learning, 17 Dec. 2020, https://www.mygreatlearning.com/blog/siamese-networks/.
14. Prasad, Krishna. “Siamese Networks - Line by Line Explanation for Beginners.” Medium, Towards Data Science, 21 June 2020, https://towardsdatascience.com/siamese-networks-line-by-line-explanation-for-beginners-55b8be1d2fc6.
15. Ramachandran, Abhi. “Using Triplet Loss and Siamese Neural Networks to Train Catalog Item Embeddings.” DoorDash Engineering Blog, 8 Sept. 2021, https://doordash.engineering/2021/09/08/using-twin-neural-networks-to-train-catalog-item-embeddings/.
16. Das, Shibsankar. “Image Similarity Using Triplet Loss.” Medium, Towards Data Science, 16 July 2019, https://towardsdatascience.com/image-similarity-using-triplet-loss-3744c0f67973.
17. Shrestha, Enosh. “Triplet Loss and Siamese Neural Networks.” Medium, 24 Oct. 2019, https://medium.com/@enoshshr/triplet-loss-and-siamese-neural-networks-5d363fdeba9b.
18. Essam, Hazem, and Santiago L. Valdarrama. “Image Similarity Estimation Using a Siamese Network with a Triplet Loss.” Keras, 25 Mar. 2021, https://keras.io/examples/vision/siamese_network/.