Image Similarity Engine for Visual Recommendations

Prathamesh Pawar, Marko Krstulovic, Sahana Rajashekara

Khoury College of Compute Science

Northeastern University

Boston, MA

pawar.prath@northeastern.edu

krstulovic.m@neastern.edu

rajashekara.s@northeastern.edu

1. Objectives and significance

In today's business landscape, companies frequently need images for a variety of uses, from marketing and branding to content creation and web design. However, not all businesses have the right to use certain images, which can pose legal risks and limit their options. This project aims to create a solution to recommend visually similar, copyright-free images as alternatives to licensed or copyrighted images.

We aim to develop an ML pipeline that utilizes Convolutional Neural Networks along with Siamese Networks to extract the feature embeddings for each image. We use Siamese pairs of images to refine the weights of the CNN model, producing final feature embeddings for each image in the training dataset. We will implement a similarity algorithm and, given an input or test image, compares feature embeddings using metrics like cosine similarity, and provides copyright-free alternatives.

2. Background

2.1. Convolutional Neural Networks (CNN):

Convolutional Neural Networks (CNNs) are currently seen as one of the most efficient ways to classify images in the field of computer vision. They are modeled after the function of biological visual cortices, influenced by a study from Hubel and Wiesel that identified two types of cells [1]. This was used by Fukushima to model the first basic implementation of a CNN called the Neocognitron [2]. The Neocognitron used unsupervised learning for training through two types of layers modeled after Hubel & Wiesel's work. The "S-layer," or convolutional layer, in the model was a shared-weights receptive-field layer that incorporated units that emphasized or inhibited certain features (such as vertical or horizontal lines). The units of the convolutional layer derive different feature weights depending on the behaviors of these filters or kernels. Data passes through several convolutional layers before reaching the next part of the model: the "C-layer," or pooling layer. This layer covers segments of units from the previous convolutional layer and computes some aggregation across them, pooling together units deemed to be alike. This mitigates effects on training due to objects being shifted or moved within the images.

A major model improvement was back-propagation, making it possible to determine kernel coefficients during training instead of manually defining them. Lecun et al., while observing how to interpret hand-written zip-codes, were able to utilize gradient descent to determine an error rate for the previous epoch and utilize it to continuously refine the kernel [3]. A further advancement of the CNN was the max-pooling filtering operation pioneered by Yamaguchi et al., which calculated the maximum value of a region in the neural network and propagated it [4]. Modern CNNs (*Figure 1.*) employ an input layer to process an image, several convolutional layers, a pooling layer, a ReLU layer, and fully-connected layers resulting in a loss layer, noted by Sharma et al. [5]. This ReLU, or Rectified Linear Unit, layer serves to increase stability by replacing negative values from pooling with zero, avoiding results getting stuck at 0 or exploding to infinity. This maintains the gradients used within the CNN to a better degree than a sigmoid activation function, observed by Indolia et al. [6].

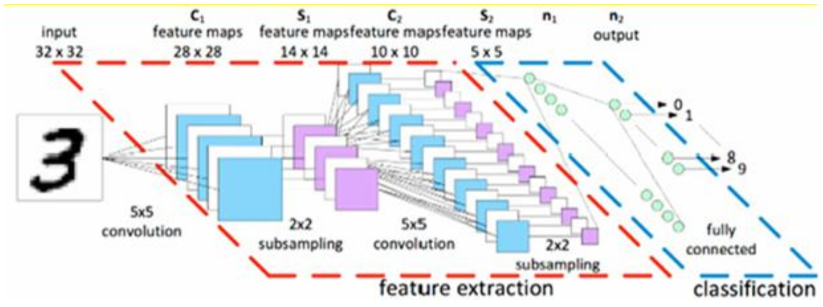

Figure 1. Different Layers of a CNN, visualized in Sharma et al. [5]

CNNs face several challenges to their efficiency. Not much is currently understood as to how they make decisions, especially as deeper networks are developed, a fact noted by Younesi et al. [7]. Furthermore, CNNs do not usually incorporate domain expertise or customization, making it more difficult for specific domain trends to be accounted for. Data scarcity and a lack of ability to handle feature translation result in classifications that are uninformed or eagerly classified. Finally, CNNs have a tradeoff between overfitting and misclassification, so data must be pruned in the pooling layer of the CNN, but this lost data could be relevant to decision making.

2.2 Triplet Networks and Embeddings:

CNNs attack the problem of image classification but can struggle with determining similarities and dissimilarities between images. A proposed solution to this is the use of Siamese networks, a combination of identical CNNs that process different combinations of data that come in the form of embeddings. Embeddings describe the way in which inputs are processed in comparison to each other, according to Song et al. [11]. Contrastive embeddings attempt to minimize the distance between a pair of similarly classified inputs, while a triplet embedding also incorporates a third input that is dissimilar to the original (*Figure 2.*). A Siamese network utilizing triplet embeddings additionally trains a second different CNN that develops dissimilar feature maps around what is chosen as a single anchor image. This network instead attempts to maximize the distance between the anchor image and the image considered dissimilar to it. This results in an output of both similar and dissimilar feature maps that are then run through a triplet loss function. This loss function supports finding triplets with dissimilar images that have greater distance than similar pairings.

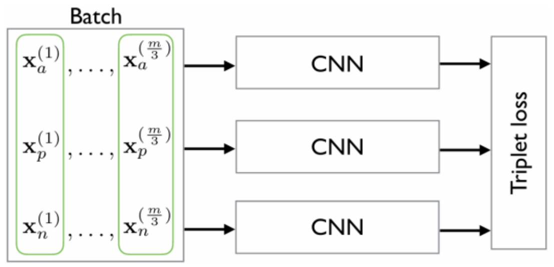
Batch
$\mathbf{x}_a^{(1)}, \ldots, \mathbf{x}_a^{(\frac{m}{3})}$
$\mathbf{x}_p^{(1)}, \ldots, \mathbf{x}_p^{(\frac{m}{3})}$
$\mathbf{x}_n^{(1)}, \ldots, \mathbf{x}_n^{(\frac{m}{3})}$
CNN
CNN
CNN
Triplet loss

Figure 2. Siamese Network with Triplet Embedding around anchor image $\mathbf{x_a}$*, where (a, p) are similar and (a, n) are dissimilar by Song et al. [11]*

2.3 Prior Work and Key Differences:

Most approaches utilize Siamese networks to determine image similarity through scoring of the image pairs. In our approach, we transform the images into embeddings capturing intrinsic similarities and we designate the original image as an anchor. Then, we would choose a Siamese image and a dissimilar image for a given anchor. We optimize the embedding using triplet loss, which minimizes the distance between an anchor and a positive pair and maximizes the distance between an anchor and a negative pair. This would generate double the optimal weights and feature maps of the contrastive approach.

3. Methods

3.1 Data

For our project, we used the CIFAR-10 dataset [10], which offers ten classes of image categories. This dataset is particularly well-suited for training our models due to its comprehensive collection of labeled images. The CIFAR-10 dataset comprises a total of 60,000 images, each measuring 32x32 pixels, with the dataset split into 50,000 training images and 10,000 test images. Each class contains 6,000 images. The classes feature a variety of vehicles and animals, including airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks.

In the Triplet Model, we focus on a subset of this data by choosing 5000 training images and 1000 testing images. This allows us to capture optimal weights and feature embeddings necessary for identifying visually similar images while managing computational resources. We ensured that each training batch had a balanced representation of different classes, with an equal number of samples from each. This approach eliminates class imbalance and its negative impact on the model's accuracy. Additionally, we selected semi-hard negatives to compute the triplet loss during training. These are images from a different class that are somewhat similar to the anchor image, making them useful for training the model to distinguish between similar looking images and truly matching ones. These strategies allowed the model to train effectively and achieve robust performance. We converted images to tensor representations using the ToTensor operation, which normalizes pixel values, minimizing the influence of lighting and intensity variations. Furthermore, this operation formats the images into a structured tensor representation ensuring compatibility with neural network architectures.

For our baseline comparison CNN model, we used the same CIFAR-10 dataset and ensured class balance by creating a subset of the dataset with 1,000 images sampled uniformly from each class. This step mitigated the risk of class imbalance, which could negatively impact the model's performance. To enhance model training, we applied data augmentation techniques, including random horizontal flipping and random cropping, followed by normalization. These transformations improved the model’s ability to generalize by introducing slight variations to the training data. When interacting with images, neural networks encounter complications with data scarcity. Data augmentation involves applying different image transformations and manipulations in order to widen a dataset via operations such as rotation, flipping, erasing specific pixels, or photometric transformations (changing the entire image’s color), explained by Wang et al. [8].

3.2 Triplet Model:

We propose a Triplet architecture to model image relationships by learning an embedding space that reflects both similarities and dissimilarities. The architecture comprises:

3.2.1. Embedding Network (EmbeddingNet):

A convolutional neural network (CNN) that extracts feature embeddings from input images.

- **Convolutional Layers:** Two Conv2d layers (kernel size 5) with PReLU activations and MaxPool2d for spatial down-sampling.
- **Fully Connected Layers:** A series of linear layers reduce the feature dimensions to the desired embedding size.

3.2.2. Triplet Network (TripletNet):

Processes triplet inputs (anchor, positive, and negative images) using the EmbeddingNet. It outputs embeddings for each input, (f(a), f(p), f(n)), enabling comparison in the embedding space.

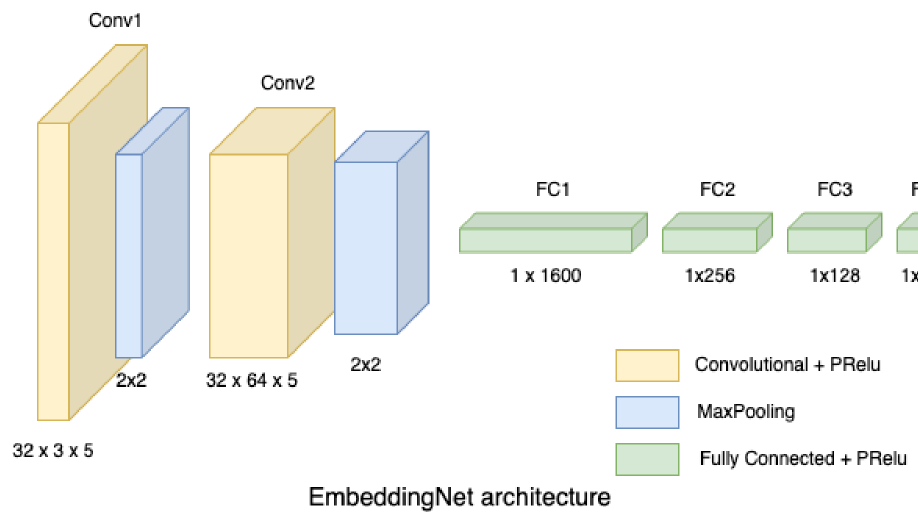

Figure 3. A model of the Triplet Network that we devised and employed in our experiments.

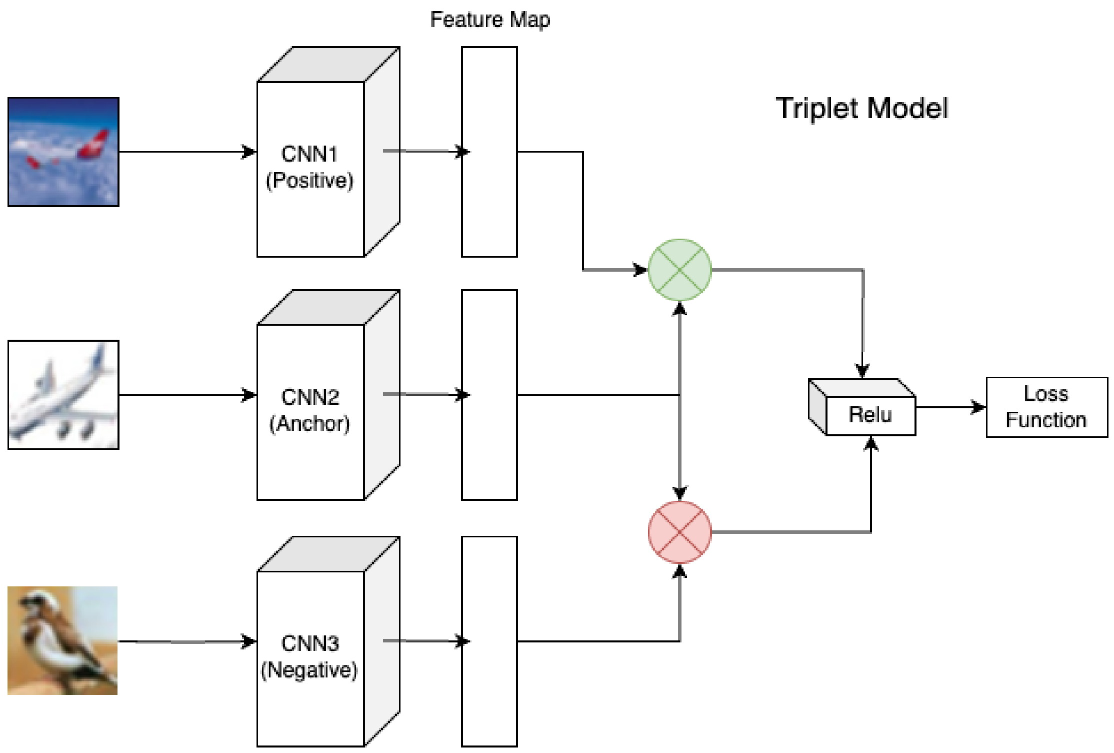

Figure 4. A model of how the Triplet Network processes anchors, positive matches, and negative matches.

3.2.3 Loss Function Description

In the Triplet Network, we process three images—anchor, positive, and negative—through three identical CNNs. Each CNN receives one image and generates an embedding or feature map as output. The outputs are combined into a list of three embeddings corresponding to the inputs.

To train the network, we employ a custom **Online Triplet Loss** function designed as follows:

1. **Positive Distance:**
   Compute the Euclidean distance between the embeddings of the anchor and positive images:
   $d_{pos} = ||f(a) - f(p)||$
2. **Negative Distance:**
   Compute the Euclidean distance between the embeddings of the anchor and negative images:
   $d_{neg} = ||f(a) - f(n)||$
3. **Loss Calculation:**
   Calculate the triplet loss by subtracting the negative distance from the positive distance and enforcing a margin α\alphaα to ensure sufficient separation:
   $$L_{triplet} = \max\{(0, ||f(a) - f(p)||^2 - ||f(a) - f(n)||^2 + \alpha\}$$
   This ensures that d_{pos} remains smaller than d_{neg} by at least the margin α\alphaα, encouraging the model to correctly differentiate between similar and dissimilar pairs.
4. **Activation:**
   Apply the Rectified Linear Unit (ReLU) function to maintain non-negative loss values and prevent trivial solutions.
5. **Backpropagation:**
   The loss is propagated back through the network to optimize the parameters of the individual CNNs within the Triplet Network.

This flexible, nested architecture allows for independent tuning of the individual CNNs, enabling tailored feature extraction for each image while jointly optimizing the triplet-based embedding space.

3.3 Evaluation Strategy

Evaluating a non-classification task in ML can be tricky. We intend to compare our model against a baseline classifier in order to assess our model's performance and efficacy. In order to do this, we intend to leverage both manual evaluation and textual evaluation. We use torchvision to convert images to flattened tensor arrays to process and PyTorch to handle our different models.

Our triplet network outputs not image classes from the CIFAR-10 database, but rather tensor feature maps. These feature maps should be capturing the most important salient features within the images (courtesy of the CNN architecture and Triplet embeddings). From this, we can devise a library of images from CIFAR-10's testing dataset that the model has not been trained on and instead derive feature maps for all these images. At the same time, we intend to use a simple CNN classifier trained on the CIFAR-10 dataset to derive a predicted class for these images. In doing so, we develop a baseline comparison between our triplet network and effectively a random selector. Once we have this library, we randomly sample ten images from CIFAR-10's testing dataset to serve as iterations of evaluation. From these ten images, we derive a feature map and a predicted class from the triplet network and simple classifier respectively.

3.3.1 Manual Evaluation

To manually evaluate the image similarity, we refer to the evaluation strategy used by Bell et.al.[12] and develop a logical evaluation metric for Evaluation Score. For each test image Ti, our model generates a set of 10 candidate images $R_i = \{R_{i1}, R_{i2}, \dots, R_{i10}\}$. We manually evaluate each candidate in R_i, marking it as either 1 (correct) or 0 (incorrect) according to Bell et.al [12]. An image is marked correct if and only if it is deemed to fall into the same higher level or subcategory of the test image.

$$\text{Mark}(R_{ij}) = \begin{cases} 1, & \text{if } R_{ij} \text{ is a correct match} \\ 0, & \text{if } R_{ij} \text{ is an incorrect match} \end{cases}$$

The score for each test image T_i is determined by summing the marks in Ri.

$$\text{Score}(T_i) = \sum_{j=1}^{10} \text{Mark}(R_{ij})$$

To decide if T_i is ultimately a correct prediction, we compare total score with a threshold of 5:

$$\text{Final Mark}(T_i) = \begin{cases} 1, & \text{if Score}(T_i) > 5 \\ 0, & \text{if Score}(T_i) \leq 5 \end{cases}$$

The overall evaluation score is the average of the final marks across all test images N:

$$\text{Evaluation Score} = \frac{1}{N} \sum_{i=1}^{N} \text{Final Mark}(T_i)$$

3.3.2 Textual Evaluation

We also consider an alternate approach utilizing textual representations of images and intend to compare these model evaluations against a manual evaluation.

Now, for each image (both in the test set and the library), we generate a list of keywords describing the image. This is done by utilizing the BLIP library in Python and pulling from a pretrained image captioning base provided by Salesforce for the library. Using this processor and model, we generate a caption for each of the images. We then use the natural language toolkit ("nltk") to prune the caption of stopwords (words deemed insignificant for natural language processing and that are filtered out) and associate up to five of these keywords to each image.

After this, for each test image, we search the library for the ten most similar images. For the simple classifier, we simply select ten images that share the same predicted class. For the triplet network, we compute the cosine similarities of the test image's feature map to each potential feature map in the image library. We then select the ten images with the smallest cosine similarity. We calculate the overlap of keywords between the test images and similar images. For each test image, we derive a similarity score based on the ratio of the cardinality of the intersection between the set of keywords for the test and model selected images compared to the number of total keywords for all images. Comparing these should give us a sense of how similar the test images are to the model-selected "similar images". The similarity score, $S_{text}(K_{test}, K^i_{model})$ for each test image is computed as follows, where K_{test} is the set of keywords representing the test image, K^i_{model} is the set of keywords representing images from the image library, and n is the number of model selected images chosen (in our case 10).

$$S_{text}(K_{test}, K^i_{model}) = \frac{\sum_{i=1}^{n} |K_{test} \cap K^i_{model}|}{\sum_{i=1}^{n} |K^i_{model}|}$$

4. Results

4.1 Triplet Network Results

The Triplet model was trained using the **CIFAR-10 dataset**, a collection of 60,000 images across ten classes. To create triplet samples for training, we used the following process:

1. **Triplet Construction:**
    - **Anchor and Positive:** Two images belonging to the same category were selected, with one designated as the anchor image and the other as the positive image.
    - **Negative:** An image was randomly chosen from a different category as the negative image.
2. **Dataset Creation:**
    - A total of **5,000 triplets** were generated for the training set.
    - An additional **1,000 triplets** were created for the validation set.

3. **Training Configuration:**
    - The model was trained for **50 epochs** with the following parameters:
        - **Learning Rate (lr):** 1×10^{-3}
        - **Optimizer:** Adam optimizer
        - **Number of Epochs (n_epochs):** 50
        - **Log Interval:** Model progress and metrics were logged every **50 steps** during training.

This setup ensures the Triplet model learns meaningful embeddings, leveraging the diverse classes of the CIFAR-10 dataset while maintaining a balance between training and validation datasets.

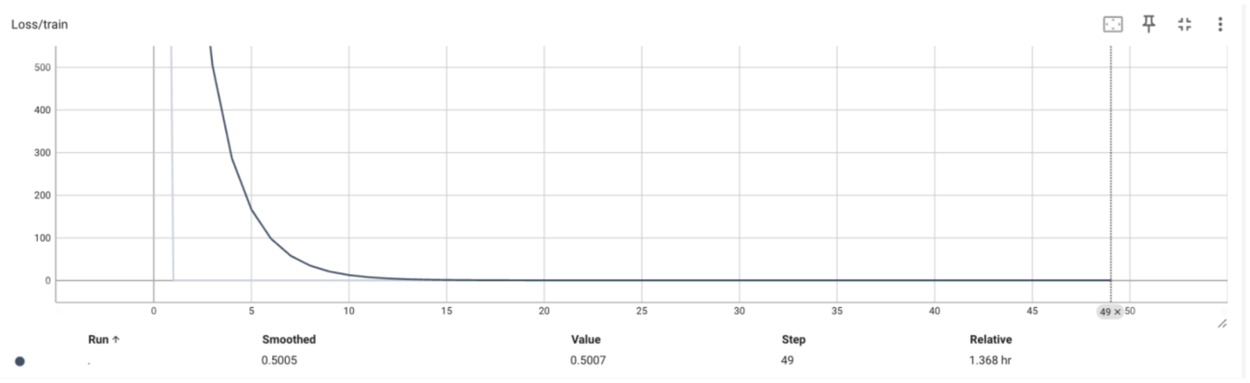

Figure 5. Training loss for our Triplet Network across 50 epochs of training.

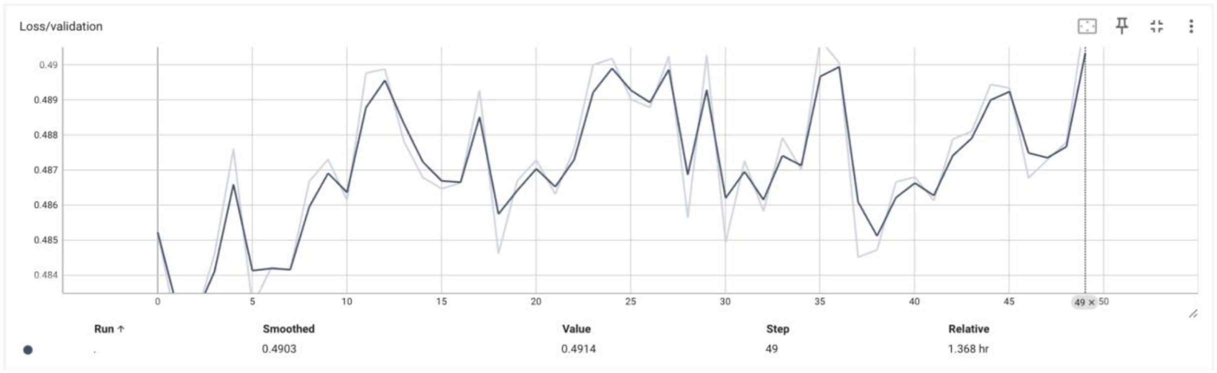

Figure 6. Validation Loss for our Triplet Network across 50 epochs of validation.

In *Figure 5*, we see a steady decrease in the training loss indicating our model has performed well on the training dataset. We also observe that there is a drastic drop in the initial few epochs of the training. After which the loss becomes constant.

The Validation loss is in stark contrast to the training loss, visible in *Figure 6*. We observe that there is no consistent trend for the loss. This indicates that the model is not performing well and might be overfitting the training data.

4.2 Simple CNN Classifier Results

As a baseline, we decided to set up a simple CNN classifier to compare against our Triplet network. Our intention was to compare our model to what amounts to a class predictor and random selector, making decisions on similarity solely on the basis of class membership. We built a simple CNN classifier through PyTorch. For capturing features, we used a 2D Convolutional layer with 3 inputs and 64 outputs feeding into a ReLU layer, followed by another Convolutional layer (now with 64 input and outputs) and ReLU layer. This ended with a Maximum Pooling layer with a dropout of 0.25. This structure was repeated one more time, scaling the outputs to about 128 features. For the classifier, we used a Sequential container applying linear transformations and passing outputs through ReLU layers three times. We minimized cross-entropy loss for multiclass predictions, visible in the function below. In this loss function, $y_{i,j}$ represents the true class label j for instance i, C represents the number of classes, and $p_{i,j}$ is the predicted probability that that instance i belongs to class j.

$$CrossEntropy = -\frac{1}{N}\Sigma_{i=1}^{N}\Sigma_{j=1}^{C}(y_{i,j}.log(p_{i,j}))$$

We trained this network on the CIFAR-10 training dataset, applying random horizontal flips and cropping transformations, and then converting to Tensors and normalizing. We developed a balanced subset of 1000 images for each of the 10 classes included in CIFAR-10. From here, we trained the model over **50 epochs** with a learning rate of 1×10^{-3}. We also utilized K-Fold cross validation, using 5 folds of training data. We included early stopping criteria if there was no improvement across 5 epochs. We optimized using the Adam optimizer.

Over the 50 epochs, we observed the training and validation loss across several folds. In *Figure 7*, we show the gradual decrease in training loss for the classifier across 50 epochs. We note that this graph shows the decrease in loss across all folds of the K-Fold split. This means that for the training data, the classifier model managed to reach a state where it fit the training data relatively well. Additionally, when viewing the validation loss, shown in *Figure 8*, we observed that the classifier steadily improved on non-training data, implying that it avoided overfitting and had decent generalization. Again, it loops back to the start of the graph due the presence of several folds of validation but shows an overall gradual improvement throughout training.

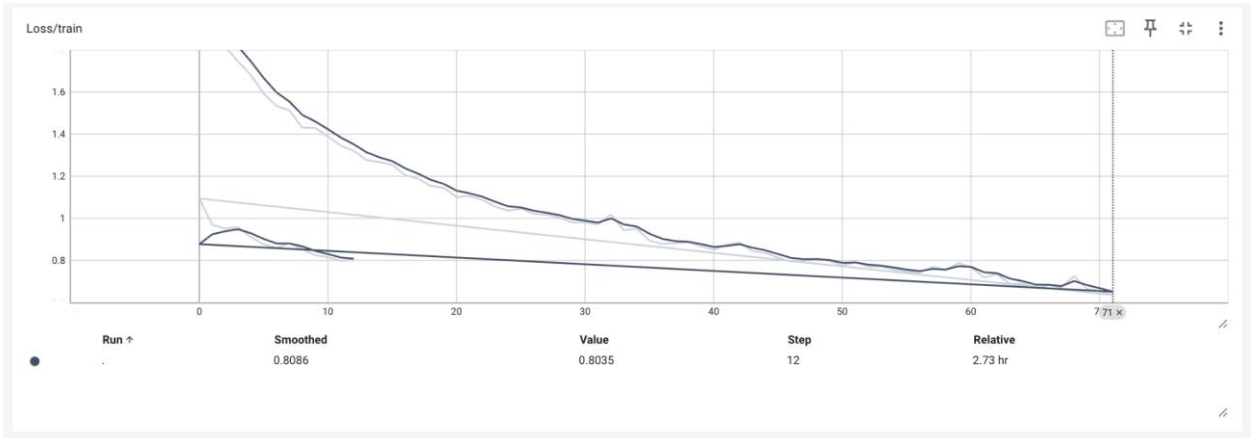

Figure 7. Training loss for our simple CNN classifier across 50 epochs of training, for each of 5 folds of K-Fold Cross Validation, highlighting the 2 most recent folds of validation.

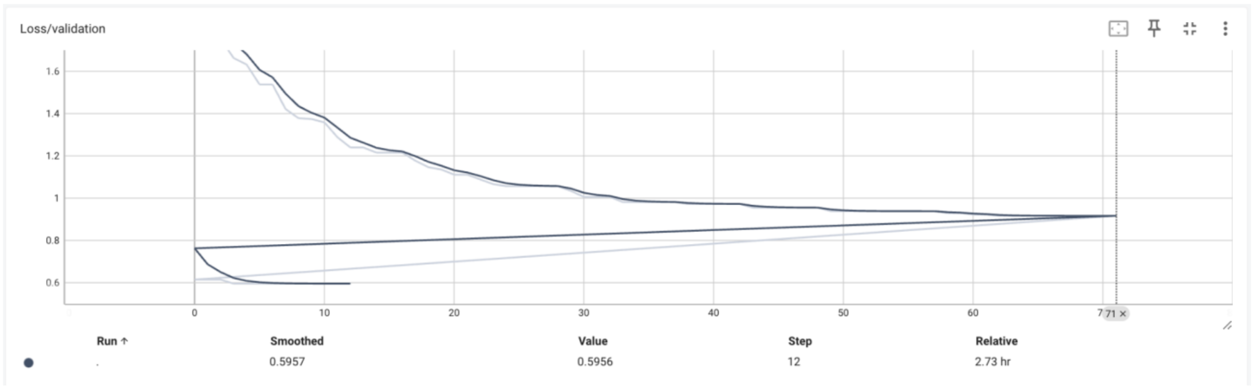

Figure 8. Validation loss for our simple CNN classifier across 50 epochs of training, for each of 5 folds of K-Fold Cross Validation, highlighting the 2 most recent folds of validation.

4.3 Model Evaluation Results

To get to the point of model evaluation, we sampled a library of images from the CIFAR-10 test dataset, obtaining 50 images for each of the 10 classes. Then, for each of these images, we generated feature maps using the Triplet Network, predicted classes using the CNN classifier, and a set of keywords using the BLIP library. Here, we converted images between Tensors and PIL Images and normalized/denormalized whenever required to put into networks (this was necessary since BLIP expected denormalized PIL images as inputs and saving images required PIL images). We then randomly sampled ten test images from the same dataset. These represent our trials and inputs to the networks. We obtained feature maps, predicted classes, and keywords as per the prior methods. Next, we chose ten similar images from the library for each of the test images using each model. For the CNN classifier, we randomly choose ten images that share the same predicted class; for the Triplet network, we selected ten images by choosing the ten images with the smallest Euclidean distance between feature maps. Model performance was compared and evaluated across two main metrics.

The first was human performance, where similar images were obtained from both the Triplet and Classifier networks. We observed and made matches to the input image based on two main criteria. The first was that if the input and output images belonged to the same exact category, they were considered a match. The second criterion was that if the output image was a subcategory or belonged to the same higher-level category, it was considered a match. An example is shown in *Figure 9*, evaluating a randomly obtained test image according to these criteria. In this example, matches were considered for trucks (exact category match), specific types of trucks like dump trucks (subcategory match) and immediate higher level matches (heavy-duty vehicles).

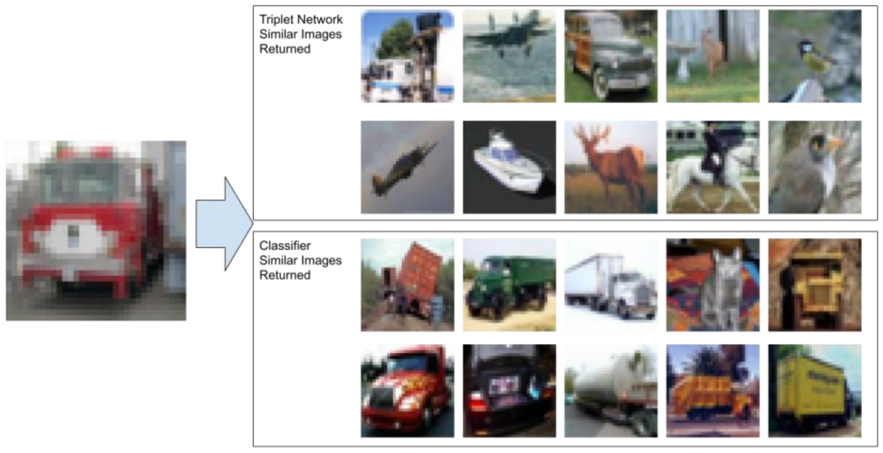
Triplet Network Similar Images Returned
Classifier Similar Images Returned

Figure 9. An example output of both networks, corresponding to test image 9.

We then proceeded to evaluate the two models according to the scoring metric. *Table 1* shows the results for the scores out of 10 for each model per test image. These scores represent the number out of ten similar images that were considered matches according to the prior mentioned criteria. The Triplet Network performed much less accurately compared to the random class-based selections carried out for the simple CNN classifier. The classifier randomly selected images that it predicted had the same class, and this showed in the trials where the comparisons suffered (mainly for Images 4, 5, and 7). Nonetheless, the triplet network struggled to properly identify most of the images, with a final *Evaluation Score* of 0, compared to the evaluation score of 0.6 for the CNN classifier. This metric, representing the number of test images that had at least five human-matched similar images, is measured in the interval [0, 1]. A score of 1 represents that the model had at least five matches among its similar images. This means that the performance of the Triplet Network was noticeably worse to the effectively random selection of images by the simple classifier.

	Score(T_i) for Sampled Test Images									
Image	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Triplet Network	3	5	3	5	2	2	5	4	5	4
Simple Classifier	7	6	4	7	1	2	9	2	9	9

Table 1. Results obtained from manual evaluation of matching images to test images.

We then proceeded to evaluate the images using our textual evaluation model based on keyword overlap. *Table 2* shows these results based on keyword intersection. This metric was evaluated based on the total keywords among similar images that were present in the keyword set of the corresponding test image, measured out of the total keywords among similar images. Our goal with this model was to capture the effectiveness of the examined models in learning salient features or concepts of images that can give more abstract matches. However, due to image quality as well as some of the weakness in the consistency of natural language processing, it was hard to generate a single concrete set of keywords for each of the images. We elected to use image captioning over class-based labeling to try to utilize these more abstract concepts. From the results we obtained, we observed that the CNN classifier still outperformed our Triplet Network six out of ten times. The CNN classifier obtained an average S_{text} of 0.11011 compared to that of the Triplet Network, which had an average S_{text} of 0.06143.

	$S_{text}(K_{test}, K^i_{model})$ for Sampled Test Images									
Image	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Triplet Network	0	0.235	0	0.026	0.151	0.059	0.029	0.059	0.028	0.026
Simple Classifier	0.147	0.269	0	0.067	0.034	0.026	0.182	0	0.214	0.161

Table 2. Results obtained from our textual evaluation model based on overlapping keyword sets.

Overall, analysis of model performance revealed that the Triplet network we developed and trained was massively outperformed by the random selection of images based on the prediction of the simple CNN classifier. Both in human evaluation and textual keyword overlap, we found our model struggled or misidentified images as similar even with stark differences. Interestingly, some cases saw the Triplet network outperform the CNN classifier in textual evaluation, especially when both models struggled to find the ideal similar images. For example, with images 4, 5, and 7, the Triplet network did better at finding underlying similarities than the CNN classifier. It is possible that the Triplet network is able to pick up some of the more abstract concepts that come up when the keywords are more abstract and less material or class-based.

5. Conclusion:

The Triplet Network was trained using the CIFAR-10 dataset to learn meaningful embeddings for image similarity but struggled in evaluation tasks compared to a simple CNN classifier. Manual evaluation revealed that the Triplet Network often failed to identify accurate matches, with a lower overall Evaluation Score compared to the CNN classifier. Using class-based predictions consistently outperformed the Triplet Network in both human evaluation and textual similarity. The Triplet Network achieved an average Evaluation Score of 0.0, while the CNN classifier scored 0.6, indicating the classifier's superior performance. In textual evaluation, the CNN classifier showed a higher average keyword similarity score ($S_{text} = 0.11011$) compared to the Triplet Network ($S_{text} = 0.06143$). Additionally, the Triplet Network occasionally outperforms the classifier in edge cases, suggesting it may capture abstract features overlooked by the classifier.

Therefore, we conclude that the Triplet network was not able to achieve sufficient performance to be considered a viable model to image recommendations. While accessing the shortcomings of the model we found that the model could have been trained on a larger dataset to increase the diversity in the data. This was not possible due to the limitations of computing power available at the time. The loss function design could represent the model learning properly and some variations of the function could be experimented with to evaluate the loss function.

Furthermore, more research could be done in developing our evaluation strategy. Currently, there are no standardized evaluation metrics for Image similarity. While comparing the performance of a recommender model with the one of the classifier model, the recommender model performs poorly. More objective evaluation strategies could be developed. Potentially switching to a more comprehensive textual comparison model could have been more adequate. Using full captions instead of keywords could have caught more the implied relationships between the keywords and what was being shown in the image. This could also have been examined to see if there was an underlying trend in the features that the Triplet network was identifying when selecting similar images.

In conclusion, the Triplet Network built with our current training architecture failed to produce the desired results. However, we claim that it shows promise and could be developed further with more data, hyper-parameter tuning, and a more complex model architecture to extract niche similarities between images.

6. Individual tasks:

In addition to all of us working on this report, we each made important contributions to the core of the project:

1. **Prathamesh:** Prathamesh worked on the main Triplet model development and coding, contributing the general architecture of the custom model, and the key training and testing frameworks. This included coding, training, and testing of the model, as well as organizing the key results of our experiments. He developed the evaluation strategy in collaboration with me, contributing to the brainstorming process and the general framework for model evaluation.
2. **Marko:** Marko worked on a slightly customized CNN classifier for the CIFAR-10 dataset to use as a baseline for comparison against the Triplet network. He worked on coming up with our evaluation strategy with Prathamesh. Also, he implemented the evaluation strategy in our evaluation model and fine-tuned it to evaluate both networks. Lastly, he conducted the manual evaluation of the image results from our testing framework and evaluation model.
3. **Sahana:** Sahana focused on data preparation, triplet formation and implementing data augmentation strategies for the model. This included our original manual forming of anchor, positive, and negative pairs to optimize triplet loss computation and ensure diverse training scenarios. She checked to make sure we were using balanced class representation in each batch, addressing potential class imbalance issues. Additionally, she researched how to apply data augmentation techniques to enhance the dataset's variability, improving the model's training.

7. References

[1] Hubel, D. H., & Wiesel, T. N. (1959). Receptive fields of single neurones in the cat's striate cortex. The Journal of physiology, 148(3), 574–591. https://doi.org/10.1113/jphysiol.1959.sp006308

[2] K. Fukushima, "Visual Feature Extraction by a Multilayered Network of Analog Threshold Elements," in IEEE Transactions on Systems Science and Cybernetics, vol. 5, no. 4, pp. 322-333, Oct. 1969, doi: 10.1109/TSSC.1969.300225

[3] Y. LeCun et al., "Backpropagation Applied to Handwritten Zip Code Recognition," in Neural Computation, vol. 1, no. 4, pp. 541-551, Dec. 1989, doi: 10.1162/neco.1989.1.4.541.

[4] Yamaguchi, K., Sakamoto, K., Akabane, T., Fujimoto, Y. (1990) A neural network for speaker-independent isolated word recognition. Proc. First International Conference on Spoken Language Processing (ICSLP 1990), 1077-1080, doi: 10.21437/ICSLP.1990-282

[5] Sharma, Neha & Jain, Vibhor & Mishra, Anju. (2018). An Analysis Of Convolutional Neural Networks For Image Classification. Procedia Computer Science. 132. 377-384. 10.1016/j.procs.2018.05.198.

[6] Indolia, Sakshi & Goswami, Anil & Mishra, S.P. & Asopa, Pooja. (2018). Conceptual Understanding of Convolutional Neural Network- A Deep Learning Approach. Procedia Computer Science. 132. 679-688. 10.1016/j.procs.2018.05.069.

[7] A. Younesi, M. Ansari, M. Fazli, A. Ejlali, M. Shafique and J. Henkel, "A Comprehensive Survey of Convolutions in Deep Learning: Applications, Challenges, and Future Trends," in IEEE Access, vol. 12, pp. 41180-41218, 2024, doi: 10.1109/ACCESS.2024.3376441.

[8] Wang, Z., Wang, P., Liu, K., Wang, P., Fu, Y., Lu, C., Aggarwal, C.C., Pei, J., & Zhou, Y. (2024). A Comprehensive Survey on Data Augmentation. ArXiv, abs/2405.09591.

[9] Takahashi, R., Matsubara, T., & Uehara, K. (2018). Data Augmentation Using Random Image Cropping and Patching for Deep CNNs. IEEE Transactions on Circuits and Systems for Video Technology, 30, 2917-2931.

[10] Krizhevsky, A. (2009). Learning multiple layers of features from tiny images (Technical Report). University of Toronto.

[11] Song, H.O., Xiang, Y., Jegelka, S., & Savarese, S. (2015). Deep Metric Learning via Lifted Structured Feature Embedding. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 4004-4012.

[12] Bell, S., & Bala, K. (2015). Learning visual similarity for product design with convolutional neural networks. ACM Transactions on Graphics (TOG), 34, 1 - 10.

[14] Koch, G.R. (2015). Siamese Neural Networks for One-Shot Image Recognition.

[15] Chen, T., Kornblith, S., Norouzi, M., & Hinton, G.E. (2020). A Simple Framework for Contrastive Learning of Visual Representations. ArXiv, abs/2002.05709.

[16] Schroff, F., Kalenichenko, D., & Philbin, J. (2015). FaceNet: A unified embedding for face recognition and clustering. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 815-823.