Duplicate Products Detection in E-Commerce

Pavel Ponomarev pavponn@gatech.edu
Sharath Nataraj snataraj3@gatech.edu
Joe Kraemer jkraemer6@gatech.edu
Brandon Alves balves6@gatech.edu

Abstract

Finding near-duplicates in large datasets is an important problem for many online businesses. Millions of product listings are created daily on e-commerce platforms such as Amazon, Shopee or eBay. Then, the challenge for online shops is to identify similar products despite having completely different images or descriptions. Finding resembling products allows e-commerce platforms to recommend customers better postings and help them find a better deal, increasing the chances of purchasing and facilitating the listing process for retailers. This project aims to implement a solution for duplicate products detection exploring multiple approaches based on image and text embeddings and analysing their performance.

1. Introduction/Background/Motivation

Similarity or (near-)duplicates detection is a fundamental problem in Computer Science theory, which has many applications in the real-world.

Undoubtedly, the problem of duplicate products detection is especially relevant for online marketplaces such as Amazon and Shopee. A solution could benefit both buyers and retailers, as it would improve recommendation systems and thus reduce time spent searching for products, which increases marketplace revenue [24] and allow sellers to price their products more competitively.

Detecting duplicate products on e-commerce platforms poses unique challenges that require specific solutions. Unlike general similarity detection problems, e-commerce platforms deal with massive volumes of products, necessitating scalable solutions that avoid exhaustive comparisons. Moreover, identical products can be listed in various ways by different retailers or vendors, making the problem more complex. Lastly, some retailers may intentionally provide misleading information to boost sales, which can further complicate the detection of duplicate products. As such, the objective of this project is to develop an effective solution to detect duplicate products on e-commerce platforms.

Classical ways to deal with similarity detection between two entities, and in particular products, commonly resort to rule-based methods and hand-crafted features such as string similarity measures [18, 20, 27]. Other common approaches for detecting similarity include usage of different hashes for text and images, such as Nilsimsa [19], pHash [26], PhotoDNA [3], PDQ [2] and others. However, while these technologies help to fight well spam and other integrity violations such as child exploitation, they don't generalize well to the problems of product matching: local sensitivity hashes like Nilsimsa suffer from inability to capture context while perceptual hashes can be attacked using object removal and addition techniques [26].

Recent advances of deep learning in computer vision and natural language processing suggest to use embeddings produced by convolutional neural networks [17, 30] and transformers [9, 21, 25] to match similar entities. Some works [23, 28] also directly apply similar techniques to the product matching: they fine-tune existing pre-trained BERT [12] architectures using Contrastive [13] and Triplet [14] losses. More recent practical solutions suggest that losses like ArcFace [10] and CurricularFace [15] can be applied.

We envision our solution for duplicate products detection problem consisting of two parts: (i) embeddings generation and (ii) retrieval/ranking. For (i) we plan to start with a baseline approach based on image embeddings extracted from a pre-trained CNNs on ImageNet, tf-idf embeddings for product titles/descriptions. Then we will try to improve the results by fine-tuning image and text embeddings using different loss functions mentioned above that showed promising results on related problems. We will attempt to improve text embeddings using one of the pre-trained neural language models (e.g., BERT) and adapt it for the product matching problem by performing transfer learning in a similar way as for images. For (ii), to perform similarity search effectively on big volume of data, we will use nearest neighbors indexes implementations like Faiss [16] that will help us to avoid direct 1 to 1 comparison and ensure fast performance. In addition, we aim to try ranking techniques like combination of embeddings and neighborhood blending [5].

If project is successful, existing marketplace platforms could incorporate our solution into their production systems.

The Shopee Product Matching dataset was created for the purpose of the Shopee Product Matching competition hosted on the Kaggle platform in 2021. The dataset was provided by Shopee, which has a large and diverse product catalog spanning multiple categories and brands. The attributes of the dataset are as follows:

- `posting_id`: A unique identifier for each product

posting in the dataset;
- `image`: The filename of the product image associated with the posting;
- `image_phash`: A perceptual hash of the product image, used for image matching tasks;
- `title`: The title or name of the product associated with the posting;
- `label_group`: A label or category assigned to the product, used for evaluation and matching tasks.

The dataset used in this work contains over $32,400$ product images, is self-contained and complete, and does not rely on external resources or contain confidential or offensive content. The dataset was already clean and structured, requiring minimal cleaning before use in deep learning models. The preprocessing we performed involved resizing the images to a uniform size, their normalization and then multiple random augmentations were applied during the training.

A brief exploratory data analysis showed that the dataset is composed of $11,014$ unique labels which corresponds to approximately 32% of the dataset. The number of images associated with each label group ranges from 1 to 51, with a median of 2 images per label group. This means that there are roughly equal numbers of images per label group, although there is still some variation.

2. Approach

This section provides an overview of our approach to address the problem of detecting duplicate product listings. We describe our methodology for generating both image and text embeddings in Section 2.1, and elaborate on techniques to improve retrieval accuracy for duplicate postings in Section 2.2.

Our implementation was done using PyTorch, and all of our authored code is available on GitHub[1]. Note that this section solely presents the approaches we employed in our research. For information on performance measurement, metric selection, experimental results, and best hyperparameters, please refer to Section 3.

2.1. Embeddings Generation

To establish similarity between two product listings, we adopt a classical approach of relying on the similarity between image and text (product title) embeddings. Initially, we adopt a baseline approach where we extract embeddings from pre-trained image and text models, followed by fine-tuning using multiple loss functions that showed promising results in related work [5, 6, 23, 28]. To get the best results, we performed a hyperparameter search using techniques such as grid and random search, with the help of the Ray Tune framework [4] (refer to to Section 3 for results and hyperparameters discussion).

2.1.1 Baseline

As a first step, we employed pre-trained models without specific optimization for our task. We experimented with two CNNs to generate image embeddings: ResNet-18 and DenseNet-121. We chose ResNet-18 due to its relatively small size, with only 18 layers and approximately 11 million parameters. We also selected DenseNet-121 to provide a comparison point with ResNet-18, given its superior classification performance on the ImageNet dataset, while still maintaining a comparable size of approximately 8 million parameters. Following this, we extracted 512-dimensional embeddings from both models. Please, find the implementation for both ResNet-18 and DenseNet-121 baselines in the `ResNet18Baseline.ipynb` notebook.

For the text embeddings, we first started with TF-IDF vectorizer to convert the text into a numerical representation by assigning weights to each term based on its frequency and inverse document frequency. As a baseline model for training, we started with BERT [12] which is a powerful language model that can learn contextual representations of words in a text and is commonly used as baseline for NLP tasks. We went further and explored an advanced sentence transformer model called paraphrase-xlm-r-multilingual-v1 [25] which is a model that maps sentences and paragraphs to a 768 dimensional dense vector space and is designed to work well on data from different languages. This might be extremely beneficial for us give that our data is from the platform operating mostly in Southeast Asia and Taiwan. While BERT has 12 transformer blocks and 110 million parameters, XLM has 12 transformer blocks and 550 million parameters and both generate 768 dimensional vector space as outputs. The notebbooks that can be referenced for these are `textBaseline.ipynb`

2.1.2 Contrastive Loss

Contrastive loss is another loss function that we tried to fine-tune text and image embeddings of similar products. This function is commonly used loss in siamese network architectures [8], where two input instances are compared and their similarity or dissimilarity is quantified through the loss value. The aim of contrastive loss is to increase the distance between dissimilar instances while minimizing the distance between similar instances in the embedding space. Formally, given a dataset sample consisting of two object embeddings X_1 and X_2 and label Y ($Y = 1$, when objects are similar, and 0 otherwise) the contrastive loss was defined by Hadsell *et al.* [13] as follows:

$$L_{contrastive} = (1-Y)\frac{1}{2}D^2_{X_1,X_2} + Y\frac{1}{2}\{\max(0, m - D_{X_1,X_2})\}^2,$$

[1] https://github.com/pavponn/dl-ecommerce-duplicates

where D_{X_1,X_2} is a distance between X_1 and X_2 and m is a predefined margin.

We believed that by using contrastive loss, we would be able to improve the discriminative power of the embeddings, which would facilitate better clustering of similar products.

To implement this approach we needed to build new pairwise training and validation datasets, create a module that would create a siamese network given an arbitrary neural network that produces embeddings as well as the neural network itself. On top of pre-trained networks, we also add additional layers that we trained.

As we elaborate further in Section 3, we use cosine similarity to determine whether two embeddings correspond to duplicate listings or not. Thus, we decided to stick to cosine distance for D_{X_1,X_2}.

One can find this approach implemented for images and text in `ImageContrastiveLossMethod.ipynb` and `TextContrastiveLossMethod.ipynb` respectively.

2.1.3 Triplet Loss

Triplet loss [14] is a type of loss function used in training deep learning networks for learning to embeddings that maps (images/texts) into a space where distance between points correspond to their similarity. In triplet loss, we use anchor, positive and negative embeddings. The anchor is the embedding that is being learned, positive embedding is that of a positive sample which is similar to anchor and negative embedding is a sample which is dissimilar to the anchor. The goal of the loss is to ensure that distance between anchor and positive is smaller than the distance between anchor and negative. During training, the network learns by producing embeddings that satisfy this condition so that semantically similar embeddings are obtained. Formally, given embeddings X_a (anchor), X_p (positive) and X_n (negative), Triple loss function is defined as follows:

$$L_{triplet} = \max(D_{X_a,X_p} - D_{X_a,X_N} + m, 0),$$

where m is a predefined margin between positive and negative pairs and D_{X_1,X_2} is a distance between embeddings X_1 and X_2.

Similar to the Contrastive loss approach, to implement this method, we created a new dataset consisting of triplets and setup additional layers on top of the pre-trained networks. The notebooks can be referenced in `TextXLMTripletLoss.iynb` and `ImagetResNetTripletLoss.ipynb`

2.1.4 Cross-entropy Loss

Cross-entropy is a versatile loss function often used in Machine learning/Deep learning for classification tasks. It is basically a measure of how well the model's prediction matches the true labels. This is measured as the sum of the negative log likelihood of each class. It is also differentiable making it nice to work with gradient descent in neural networks. We applied Cross-entropy loss on top of both text and image baseline models before proceeding with ArcFace and this can be referenced in the following notebooks: `TextBERTArcfaceLoss.ipynb` `TextXLMArcfaceLoss.ipynb` and `ImageCrossEntropyLossMethod.ipynb`.

2.1.5 ArcFace Loss

The idea behind ArcFace [11] is to learn discriminative features by optimizing the softmax loss with an additive angular margin. The angular margin encourages the network to learn features that are more separable, making it easier to distinguish between different faces.

The ArcFace loss is defined as follows:

$$L_{arcface} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{s\cdot\cos(\theta_{y_i}+m)}}{e^{s\cdot\cos(\theta_{y_i}+m)} + \sum_{j=1,j\neq y_i}^{n} e^{s\cdot\cos\theta_j}}$$

where N is the number of training samples, y_i is the true class label of the i-th sample, n is the total number of classes, θ_j is the angle between the feature vector and the weight vector of the j-th class, and m is the angular margin, which is a hyperparameter that controls the degree of angular separation between the different classes. s is the scaling factor which is used to amplify the feature embedding.

The only difference between the SoftMax and the ArcFace loss function is the logit, i.e. the power of the SoftMax, and the margin is added to ground truth.

In essence, the ArcFace loss adds an angular margin to the SoftMax loss, making the network more effective at discriminating between similar classes.

To implement ArcFace, we used a pre-trained model as the backbone for the model. On top of it we also added batch normalization and dropout layers. During training, the model is fed images and their corresponding labels. The loss is calculated using the ArcFace loss function, which compares the predicted features with the ground truth labels.

One can find this approach implemented for images and text in `ImageArcFaceLossMethod.ipynb`, `TextBERTArcFaceLoss.ipynb` and `TextXLMArcFaceLoss.ipynb` respectively.

2.1.6 CurricularFace Loss

CurricularFace is an adaptation of loss function called Curriculum Loss for training facial recognition models to increase the feature margin between different classes [15, 22]. Curriculum Loss functions schedule and organize

the samples that are used. At the beginning of the training, easier samples are given higher weighting. This importance factor is then reduced to gradually prioritize more difficult samples. In traditional Curriculum Loss, the curriculum is established at the beginning of training where all samples are ordered based on their difficulty. CurricularFace is different because the samples are randomly selected from a mini-batch and the curriculum is created with each random sampling. Additionally, the importance of hard samples is adjustable and will change depending on the specific difficulty of each mini-batch. The formulation is similar to ArcFace, but only modulates the positive cosine similarity, where CurricularFace modulates the negative cosine similarity as well. This is performed with the function N.

$$L_{CF} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{s\cdot\cos(\theta_{y_i}+m)}}{e^{s\cdot\cos(\theta_{y_i}+m)}+\sum_{j=1,j\neq y_i}^{n}e^{s\cdot N(t,\cos\theta_j)}}$$

$$N(t,\cos\theta_j) = \begin{cases}\cos\theta_j, & T(\cos\theta_y i)-\cos\theta_j \geq 0\\ \cos\theta_j(t+\cos\theta_j), & T(\cos\theta_y i)-\cos\theta_j < 0\end{cases}$$

Where N is the number of samples, m is the margin, y_i is the class label, n is the total number of classes, θ_j is the angle between the feature vector and the weight vector, and m is the angular margin, which is a hyperparameter that controls the separation between the different classes. s is another hyperparameter which controls the magnitude of the feature embedding.

To integrate CurricularFace, we created a CurricularFace PyTorch module and then added this layer to the baseline BERT and XLM models for text embeddings and the baseline ResNet and DenseNet model for image embeddings. Like the other loss functions we add additional batch normalization and dropout layers. A grid search was run to find optimal parameters for s, m, batch size, and learning rate. The training and analysis of CurricularFace can be found in `CurricularFaceLossImage.ipynb` and `CurricularFaceLossText.ipynb`.

2.2. Retrieval & Ranking

After generating text and image embeddings, they are used to search for other product matches. These embeddings can also be combined in different ways to provide a more complete context in order to then search for matching products. For all of our retrievals, we use faiss [16] to search for similar embedding vectors.

2.2.1 Combining Image & Text Embeddings

The simple approach we started with was to simply concatenate the text and image embeddings. We added the image/text vectors together to create combined embeddings. The intuition here is to strengthen matches that have both close text and image embeddings and weaken matches where either titles or images are similar, allowing higher accuracy of duplicates detection. The notebook for the same can be referenced in `Combine_text_and_image_predictions.ipynb`. This was tried with and without normalizing the individual text/image embeddings.

2.2.2 Neighborhood Blending

Neighborhood Blending is a technique that helps to make similar queries more distinct and closer to their neighbors. This results in more coherent clusters. This happens by removing nodes that don't meet the similarity threshold criteria.

This approach came from one of the methods implemented for Kaggle competition organized by Shopee [5]. The idea is to retrieve potential matches from the image and text embeddings alone, as well as the concatenated embeddings. Following this, the matches were then all placed through an iterative neighborhood blending pipeline. We used a python library to implement neighborhood blending [1]. The notebook for Neighborhood Blending is `NeighborhoodBlending.ipynb`.

3. Experiments and Results

Approach/Loss	Base Model	F1-score (train)	F1-score (valid)	F1-score (train+valid)
Baseline (no loss)	RN-18	.663@.92	.685@.91	.655@.92
Baseline (no loss)	DN-121	.657@.92	.675@.91	.650@.92
Cross-entropy	DN-121	.621@.89	.637@.86	.616@.89
Contrastive	RN-18	.628@.92	.646@.90	.623@.92
Triplet	RN-18	.481@.60	.454@.60	.446@.60
ArcFace	**DN-121**	**.914@.78**	**.740@.68**	**.856@.78**
CurricularFace	DN-121	.446@.98	.669@.80	.473@.98

Table 1: Comparision of approaches to generate image embeddings. RN-18 is for ResNet-18 and DN-121 for DenseNet-121.

Approach/Loss	Model	F1-score (train)	F1-score (valid)	F1-score (train+valid)
Baseline (no loss)	TFIDF	.629@.60	.725@.50	.612@.65
Baseline (no loss)	BERT	.532@.99	.539@.99	.529@.99
Baseline (no loss)	XLM	.569@.80	.621@.75	.560@.80
CrossEntropy	XLM	.822@.80	.721@.65	.763@.80
Contrastive	XLM	.535@.93	.572@.91	.532@.93
Triplet	XLM	.634@.95	.729@.95	.616@.95
ArcFace	**XLM**	**.902@.65**	.770@.55	**.833@.65**
CurricularFace	BERT	.704@.97	**.839@.98**	.679@.97

Table 2: Comparison of approaches to generate text embeddings.

In this section, we report the results of our experiments based on the approach described in Section 2. We evaluate the performance of various text and image embeddings in

Section 3.1 and analyze the effectiveness of our retrieval and ranking strategies using the best models in Section 3.2.

We split our dataset into two parts: training ($\approx 80\%$ of available dataset) and validation ($\approx 20\%$), such that each label group falls exactly into one of them. This prevents us from leaking any training data into the validation set and also facilitates the results comparison.

As the problem of detecting duplicate products requires high precision and recall, the primary metric we aim to optimize is the average (per posting) F1-score.

3.1. Comparison of Embeddings Generation Approaches

We outline the comparison of the approaches we produced image and text embeddings with in Table 1 and Table 2 respectively. For each approach (loss function) we provide best F1-scores (at corresponding similarity thresholds) that we were able to achieve on training, validation and training + validation datasets using a given model. While for baseline we list all models, for each other approach we only provide a result from the best model and setup of hyperparameters due to the shortage of space.

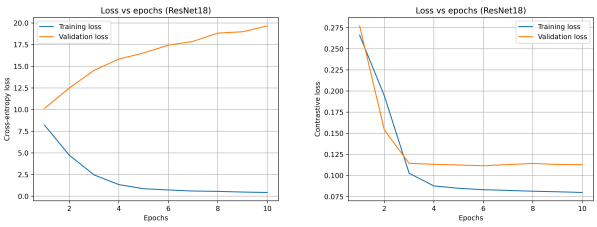
As a similarity metric between two embeddings, we use cosine similarity, which is commonly preferred for high-dimension vectors over the similarity based on euclidean distance. To calculate F1-score, we first produce and normalize the embeddings for all postings in the dataset. Then, we instantiate an `IndexFlatIP` index from the `faiss` framework with the produced embeddings. Next, we query the k-most similar embeddings for each posting using the corresponding embedding ($k \approx 100$). Finally, we iterate over the list of thresholds and leave only the ones that have a similarity higher than the threshold. We then calculate F1-score based on the group labels.

We conducted hyperparameter search for each combination of model and loss function, but due to limited space, we do not list all the tried combinations. The tables show the best results obtained for each setup (with the best set of hyperparameters) after 10 training epochs. Our repository includes charts of loss curves and F1-scores [2].

Both image baselines (ResNet-18 and DenseNet-121) demonstrated relatively good results without any specific training, achieving F1-scores greater than 0.65. This suggests that pre-trained CNNs already capture the notion of similarity to some degree.

The text baselines (TFIDF, BERT, XLM) gave good results. Without any training, the XLM model outperformed BERT. TFIDF also gave realy good results of F1 scores greater than 0.6.

Introducing cross-entropy loss on top of BERT and XLM significantly increased F1 scores for BERT and XLM. The F1 scores with cross entropy is greater than 0.7 for all three datasets. At the same time, this approach did not work well for images and we were barely able to achieve results close to baseline.



(a) Cross-entropy Loss (b) Contrastive Loss

Figure 1: Comparison of Cross-Entropy and Contrastive loss curves for the setups that achieve similar F1-score results.

Unfortunately, despite rigorous hyperparameter tuning, we were unable to improve the baseline results for images and titles using either contrastive or triplet loss. We tried using additional layers on top of the given CNN and Transformer architectures, as well as freezing different numbers of layers in the pre-trained models. We also experimented with the learning rate, its scheduler, dropout probability, batch size and augmentations (for images). However, for contrastive loss it only brought us to comparable but slightly worse F1-scores than the baselines, and for triplet loss we missed the baselines scores substantially.

The model based on DenseNet-121 with ArcFace loss achieved the highest F1-scores compared to other approaches. The F1-score on the train set was very high at 0.914, indicating that the model was able to learn the training set very well. The F1-score on the validation set was 0.740, which is still a good score, although much lower than the training set, indicating potential overfitting or limitations in generalizing fully to unseen data.

Arcface loss with XLM and BERT outperformed the loss functions when it comes to text embeddings. Similar to images, the F1 scores on training was 0.902 while that of validation was 0.770, similarly indicating potential overfitting or limitations in generalizing fully to unseen data. While this pattern is similar for Arcface, but still this conclusion is by evaluating F1 scores of one validation split. To obtain more conclusive results and to verify the extent of overfitting, it is ideal to perform a 10-fold cross-validation. This stands true for all other losses as well.

CurricularFace shows worse performance compared to ArcFace on images embeddings. In general, CurricularFace generalizes ArcFace and should be able to meet its performance. Given the complexity of the loss function and the model, its possible that the hyperparameter tuning was better for ArcFace than CurricularFace. Perhaps with more

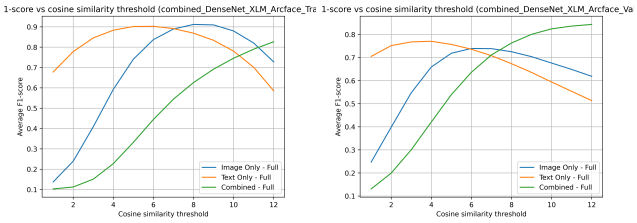
[2] https://github.com/pavponn/dl-ecommerce-duplicates/tree/master/final-charts

time, CurricularFace could surpass ArcFace. Because of this large performance difference, we chose to move forward with ArcFace.

Looking at the text embeddings, CurricularFace shows performance that is much closer to ArcFace indicating that the hyperparameter tuning was better for text than images. Although CurricularFace was similar, ArcFace seems to show more stable performance across all three data configurations and we decided to move forward with ArcFace.

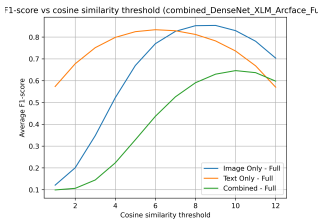
One feature to note is that Cross-entropy, CurricularFace and ArcFace training losses on class labels. Meaning that they are attempting to classify the class label in the dataset and optimize difference between then produced embeddings only implicitly. This differs from Contrastive and Triplet loss which instead try to maximize the similarity between positive samples and minimize the similarity between negative samples *per se*.

Thus, when comparing the training loss and the validation loss, it looks like the Cross-entropy, CurricularFace and ArcFace are overfitting to the training data, assuming solve a general classification problem. This happens because of the way we split the dataset: each label group belongs either to training set or to validation dataset. Nevertheless, from Table 1 and Table 2, it is evident that using loss functions like ArcFace and Cross-entropy still achieve the primary training goal of improving the expressiveness of embeddings produced by models even for unseen label groups. We don't see this behaviour for Contrastive and Triplet losses since they are based on embeddings themselves and not class labels, thus the validation loss should decreases with training. We illustrate this in Figure 1.



(a) Training set

(b) Validation set



(c) Training and validation sets

Figure 2: F1-scores vs cosine similarity threshold for concatenation of image and text embeddings.

3.2. Comparison of Retrieval & Ranking Strategies

We used a simple concatenation of image and text embeddings as the baseline strategy for combining embeddings. This gave us really good results with F1 scores for combination embeddings outperforming the individual F1 scores. This is illustrated in Figure 2. This kind of concatenation works especialy in our context because it helps in learning the relationship between these two modalities.

Results from the Neighborhood Blending (NB) were worse than expected and did not exceed the trivial concatenation of text and image embeddings. This is partially because there are 10 thresholds that need to be tuned that are all interdependent. If a threshold early in the pipeline is set too high, not enough samples are passed through the pipeline and the NB pipeline does not add any additional grouping information. Given more time and more extensive hyperparameter searches, we believe that NB could provide F1 score improvements. Additionally, we didn't have time to add a Min2 constraint. This was discussed in one of the solutions for Shopee Kaggle competition [5] and is based on the fact that there is always at least one match (with itself). Using this constraint, we could add overrides to the threshold limit that would allow for a more robust flow of matches. Sometimes the only match for a given pair is dropped because it doesn't meet the threshold.

4. Challenges

Our team did not have prior extensive experience in deep learning projects of this size, so we faced many practical issues that were new to us. This includes but is not limited to: memory errors in PyTorch when using cuda (GPU), timeouts in Amazon SageMaker and Google Colab, slow training for Contrastive and Triple losses, especially for images and in particular for DenseNet-121 (given its number of layers).

5. Concluding Remarks

In this project, we explored various techniques to detect duplicate products using image and text embeddings, fine-tuning them with different losses, and testing multiple retrieval and ranking techniques. Future research could further investigate the effectiveness of using the CurricularFace loss and improving neighborhood blending. Moreover, additional image features such as OCR captions, YOLOv4 [7] labels, and generated image captions [31] could be extracted to improve the system's performance followed by a more sophisticated ranking model [29].

We assert that the primary accomplishment of this project lies in the extensive knowledge and skills acquired during the course of its implementation, owing to the considerable scope and complexity of the task at hand.

Student Name	Contributed Aspects	Details
Pavel Ponomarev	Implementation, Analysis and Manuscript Writing	Prepared initial project structure, built multiple reusable util modules, implemented and analysed baseline for images, implemented and the fine-tuned image and text embeddings using contrastive loss and analysed the results. Additionally, researched usage of a simpler loss function such as cross-entropy loss for images. Structured the report.
Sharath Nataraj	Implementation, Analysis and Manuscript Writing	Focused on the text embeddings and created the all the baselines for text. Implemented Triplet loss modules and trained on text and image data. Trained text data Cross entropy,arcface and triplet loss respectively. Researched better baseline models to narrow down on [25]. Worked on combining the results from text/image embeddings.
Joseph Kraemer	Implementation, Analysis and Manuscript Writing	Implemented SuperLoss module and training, CurriculumLoss, Integration of CurriculumFace module with basemodels (BERT, XLM, ResNet, DenseNet), hyperparameter search for all 4 CurriculumFace text and image model combinations, training for final models for all 4 CurriculumFace text and image model combinations, Created Neighboorhood Blending notebook, Created baseline BERT module
Brandon Alves	Implementation, Analysis and Manuscript Writing	Implemented image embeddings with ArcFace loss and analysed the results. Implemented ArcFace loss and CurricularFace loss modules. Fine-tuned and trained image embeddings using ArcFace loss. Worked on exploratory data analysis.

Table 3: Contributions of team members.

6. Work Division

We outline work division and responsibilities in Table 3. Overall, we believe that contribution was equal and each team member spent at least 40 active hours of work on the project.

References

[1] Neighborhood blending - github repository. https://github.com/sonisanskar/Neighborhood-Blending. Accessed: 2023-05-01. 4

[2] Open-sourcing photo- and video-matching technology to make the internet safer. https://about.fb.com/news/2019/08/open-source-photo-video-matching/. Accessed: 2023-04-29. 1

[3] Photodna. https://www.microsoft.com/en-us/photodna. Accessed: 2023-04-29. 1

[4] Ray tune: Hyperparameter tuning. https://docs.ray.io/en/latest/tune/index.html. Accessed: 2023-05-01. 2

[5] Shopee - price match guarantee: From embeddings to matches. https://www.kaggle.com/competitions/shopee-product-matching/discussion/238136. Accessed: 2023-04-29. 1, 2, 4, 6

[6] Shopee - price match guarantee: Matching prediction by gat & lgb. https://www.kaggle.com/competitions/shopee-product-matching/discussion/238022. Accessed: 2023-04-29. 2

[7] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-yuan Liao. Yolov4: Optimal speed and accuracy of object detection, 04 2020. 6

[8] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a "siamese" time delay neural network. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, page 737–744, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc. 2

[9] Daniel Cer, Yinfei Yang, Sheng yi Kong, Nan Hua, Nicole Lyn Untalan Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Céspedes, Steve Yuan, Chris Tar, Yun hsuan Sung, Brian Strope, and Ray Kurzweil. Universal sentence encoder. In *In submission to: EMNLP demonstration*, Brussels, Belgium, 2018. In submission. 1

[10] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4685–4694, 2019. 1

[11] Jiankang Deng, Jia Guo, Jing Yang, Niannan Xue, Irene Kotsia, and Stefanos Zafeiriou. ArcFace: Additive angular margin loss for deep face recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(10):5962–5979, oct 2022. 3

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. 1, 2

[13] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742, 2006. 1, 2

[14] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In Aasa Feragen, Marcello Pelillo, and Marco Loog, editors, *Similarity-Based Pattern Recognition*, pages 84–92, Cham, 2015. Springer International Publishing. 1, 3

[15] Yuge Huang, Yuhan Wang, Ying Tai, Xiaoming Liu, Pengcheng Shen, Shaoxin Li, Jilin Li, and Feiyue Huang. Curricularface: Adaptive curriculum learning loss for deep face recognition. *CoRR*, abs/2004.00288, 2020. 1, 3

[16] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *CoRR*, abs/1702.08734, 2017. 1, 4

[17] Gregory R. Koch. Siamese neural networks for one-shot image recognition. 2015. 1

[18] Pradap Konda, Jeff Naughton, Shishir Prasad, Ganesh Krishnan, Rohit Deep, Vijay Raghavendra, Sanjib Das, Paul C., AnHai Doan, Adel Ardalan, Jeffrey Ballard, Han Li, Fatemah Panahi, and Haojun Zhang. Magellan: Toward building entity matching management systems. *Proceedings of the VLDB Endowment*, 9:1197–1208, 08 2016. 1

[19] Jesse Kornblum. Identifying almost identical files using context triggered piecewise hashing. *Digital Investigation*, 3:91–97, 2006. The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06). 1

[20] Hanna Köpcke, Andreas Thor, and Erhard Rahm. Evaluation of entity resolution approaches on real-world match problems. *PVLDB*, 3:484–493, 09 2010. 1

[21] Yuliang Li, Jinfeng Li, Yoshi Suhara, AnHai Doan, and Wang-chiew Tan. Deep entity matching with pre-trained language models. *Proceedings of the VLDB Endowment*, 14:50–60, 09 2020. 1

[22] Yueming Lyu and Ivor W. Tsang. Curriculum loss: Robust learning and generalization against label corruption, 2020. 3

[23] Ralph Peeters, Christian Bizer, and Goran Glavas. Intermediate training of bert for product matching. In *DI2KG@VLDB*, 2020. 1, 2

[24] Nikolaos Polatidis and Christos Georgiadis. Recommender systems: The importance of personalization in e-business environments. *International Journal of E-Entrepreneurship and Innovation*, 4:32–46, 10 2013. 1

[25] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. pages 3973–3983, 01 2019. 1, 2, 7

[26] Priyanka Samanta and Shweta Jain. Analysis of perceptual hashing algorithms in image manipulation detection. *Procedia Computer Science*, 185:203–212, 2021. Big Data, IoT, and AI for a Smarter Future. 1

[27] Andreas Thor. Toward an adaptive string similarity measure for matching product offers. pages 702–710, 12 2010. 1

[28] Janusz Tracz, Piotr Wójcik, Kalina Jasinska, Riccardo Belluzzo, Robert Mroczkowski, and Ireneusz Gawlik. Bert-based similarity learning for product matching. 12 2020. 1, 2

[29] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Y. Bengio. Graph attention networks. 10 2017. 6

[30] Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. Learning fine-grained image similarity with deep ranking. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1386–1393, 2014. 1

[31] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. 02 2015. 6