Video Retrieval Using Dual Encoder

Arjun Naga Siddappa, Atsushi Shimizu, Karthik Udhayakumar

New York University

Abstract

In today's world media is generated in huge amounts every second. Most of it in the format of videos. With rise of social media TikTok videos, Instagram Reels, Youtube Shorts etc. have been used by millions across the world and churn up huge amount of data. Searching through these videos is consequentially an imminent task. We develop a video search system that can search through a data base of videos with just a text query. To implement them we use Dual Encoders of text and image data and a Milvus a vector database to search through the embeddings generated by these encoders. We conduct comprehensive experimentation to obtain the best set of parameters for the model.

Introduction

The video search system requires a text query input and would return the top 10 videos that match the query in the database. This system has the Dual Encoders at the heart of its operation. The main idea is that we train a dual encoder of image and text to generate embeddings. Through training these the encoders learns to map embeddings for an image and video close together in the embedding space if the image and text are related to each other. For example "Dog playing in the park with a ball" and an image of a dog playing in the park would be two different vector points very close to each other in the vector space. After this given any text we can pass it through the text encoder to get an embedding for it that represents a point in this vector space. We would then extract the closest points (image and text) around this point. These points in the dataset are images and texts extracted and linked to the videos in the database. Obtaining these points will lead us to picking out the respective videos.

To train the encoder we use an image captioning dataset which contains images along with descriptive texts. Then we do the search on another database. We extract the images and texts from the video database and pass them through the trained encoders to get their embeddings these embeddings along with the video info are stored in a vector database called Milvus. We can then query this vector database using query text embedding to get matching points and in turn the matching videos. This is basically our implementation of the system.

To find the best set of hyper parameters for the Dual Encoder we conducting several experiments which we will detail in this report. The Dual Encoder is made up of a an Image Encoder and a Text Encoder. There are several pertained models we could use for these two encoders. We then create "EncoderHeads" that would be attached on top of these pertained models that would learn to generate embeddings. We use CosineEmbeddingLoss to train the encoders. The following are the list of hyper parameters and their possible values.

Hyperparameter	Possible Values
Image Encoder	InceptionV3, Xception, ResNet, YOLO
Text Encoder	BERT , GloVe
Embedding Size	[10,1024]
Learning Rate	[0.00002, 0.0125]
Optimizer	SGD, Adam
Activation	BatchNorm, Tanh
Negative Pairs Rate	[1,20]
Loss Margin	[-0.1, 0.5]

The following sections will in detail explain the experiments and the results. After which we will give more details about the system.

Related Work

The idea of comparing data from different modalities by embedding them into the same space was introduced in [Wu et al. 2017]. After embedding, we can apply common similarity measurements, such as cosine similarity and dot product. In their paper, they present the dual-encoder model, with one encoder for each modality. This model is trained on data set of positive entity pairs (a, b). With positive pairs, we can teach the model what data should be mapped close to each other in the embedding space. In addition, to show what pairs should not be mapped close to one another, we generate negative pairs (a, b^-) by picking irrelevant b^- for a. Note that the number of negative pairs (k) we create for one positive pair is a hyperparameter. They studied this on Freebase 15k dataset and showed that $k = 50$ is optimal. This may be different for different problems, but one takeaway is that k being too small or too large would degrade the model performance. They also discussed the wide range of applications, including text classification, content-based recommendation, document search, learning word embeddings, etc.

Among many applications, one common model structure takes image and text as two modalities. In [Marin et al. 2018], the data set is pairs of food images and their recipes, which are further classified into ingredients and instructions. They deployed VGG-16 and ResNet-50 for the image encoder and LSTM-based model for the text encoder. To better learn the embedding, semantic regularization is introduced in this paper, which is an additional loss coming from classifying the image or recipe into one of the food-related semantic categories. With this regularization, embedding vectors from the same high-level food category are mapped closer to each other than vectors from different food categories. For instance, the American sandwich and the Italian panini will form a loose cluster in the embedding space while they are completely separated from the Chinese hot pot. Dut to this semantic regularization, the model became more robust to the unseen data. The combination of video and text has also been researched. In [Dong et al. 2020], they proposed a video encoder, which processes each frame in the video by CNN, followed by RNN with GRU to combine the output of CNN. Also in this paper, the regularization technique is introduced, which is called hybrid space learning. In addition to learning embedding which is denoted as learning a latent space in the paper, the model learns a concept space simultaneously. To learn the concept space, the model does a multi-label classification task. Thus, the underlying idea is quite similar to the semantic regularization.

Our model consists of an image encoder and a text encoder, and we try different state-of-the-art architectures for both encoders. We choose pre-trained Inception V3 for our base model, which deploys multiple convolution filters with different combinations in each layer to extract features from the image efficiently. We also consider another image encoder based on pre-trained Xception model introduced in [Chollet 2016], which is inspired by the Inception model. Xception is considered to have more flexibility than Inception model due to its filter-decomposition type concept in the channel direction. As third option, we try ResNet-101, first introduced in [He et al. 2015]. It was known that deeper nets tend to produce better performance but deeper nets have issues in training due to their deep structure such as vanishing or exploding gradients. The ResNet overcomes this disadvantage by utilizing the so-called skip connection, which efficiently passes back the gradient from the loss.

On the text encoder's end, word embeddings are widely used to convert text into a numerical representation. We implement our encoder with GloVe word embeddings followed by bidirectional LSTM to understand the semantic meaning of the text. This type of word embedding is considered context-free because it assigns the same embedding vector without paying attention to the context that a word appears. In [Peters et al. 2018], they presented the context-based embedding, called ELMo. It first encodes text into context-free embedding, which is fed into multiple-layer bidirectional LSTM and trained on supervised NLP tasks. The pre-training is done this way, and this model, given a text, outputs multiple hidden states from different layers for each word. In the paper, they argue that syntactic information is better represented at lower layers while semantic information is captured at higher layers. Thus, they recommend fine-tuning how to mix up these outputs depending on downstream tasks. Though these RNN-based models have shown strong performance, they have disadvantages such as not good at capturing the dependency of words being placed in distant locations in a text. To address this issue, [Vaswani et al. 2017] presented a Transformer, a model architecture eschewing recurrence and instead relying entirely on an attention mechanism. Like the multi-layer RNN structure in ELMo, we can construct a model with multi-self-attention layers where pairwise dependencies are captured in the lowest layers while semantic information coming from all the words in the text is stored in the higher layers. The pre-trained Transformer model is called BERT, and [Devlin et al. 2018] also recommends fine-tuning the outputs from different layers for each downstream task. In our implementation, however, we only use the output from the highest layer corresponding to the first token. In BERT, the first token is always [CLS] and, its outputs are used for classification tasks. Thus, this can be considered as the representation of the entire text.

Data Set

Our model needs to learn both visual representations and textual infomation and their mappings. Hence, we opted to use Nocaps dataset [Agrawal et al. 2019] which consists human-generated captions describing images. For each image, we have 10 captions in this data set. From this dataset we generated a image-caption pair data which we use to train and test our model.

Since our data set is not so large, and most of the computation happens in the pre-trained image and text encoders, we implemented our original Dataset class that stores not the raw images and texts but the outputs of pre-trained nets. By this implementation, the time for training one epoch decreases from 15 minutes to 1 minutes approximately, enabling us to explore different model settings.

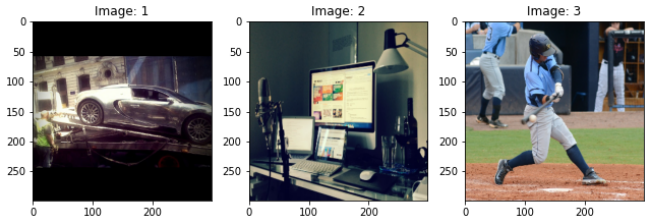

Figure 1: Sample Images

Text 1 (pos-pair): A sports car is on the back of a truck.
Text 2 (pos-pair): A laptop in front of a computer monitor.
Text 3 (neg-pair): Two men ride on top of a tank.

Base Model

For the ablation study, we define a base model as in Table 1. The loss function we used is called cosine embedding loss defined as follows.

$$\mathcal{L}(x, y) = \begin{cases} 1 - \cos(x_{image}, x_{text}) & \text{if } y = 1 \\ \max(0, \cos(x_{image}, x_{text}) - \text{margin}) & \text{if } y = -1 \end{cases}$$

Hyperparameter	Optimal Value
Image Encoder	InceptionV3
Text Encoder	BERT
Embedding Size	256
Learning Rate	0.0005
Optimizer	Adam
Activation	Tanh
Negative Pairs Rate	4
Loss Margin	0.1

Table 1: Base Model Setting

Test Loss	MedR	R@10
0.1256	43 / 1,350	0.1847

Table 2: Base Model Performance

Our goal is to map positive pairs as close as possible while mapping negative pairs orthogonal. Although we have only d mutually orthogonal vectors in d-dimensional space, we have $\mathcal{O}(2^d)$ nearly orthogonal vectors. By the margin factor, we are letting negative pairs nearly orthogonal, not strictly orthogonal. We say two vectors x_i and x_j are nearly orthogonal if

$$-\epsilon \leq \frac{\langle x_i, x_j \rangle}{\|x_i\|_2 \|x_j\|_2} \leq \epsilon$$

for $\epsilon \in [0, 1]$. We replaced the last layer of the Image encoders with 2 fully connected layers with ReLU and Tanh activation functions between the layers to produce embeddings. The text encoder is also connected to the same structure fully connected layers.

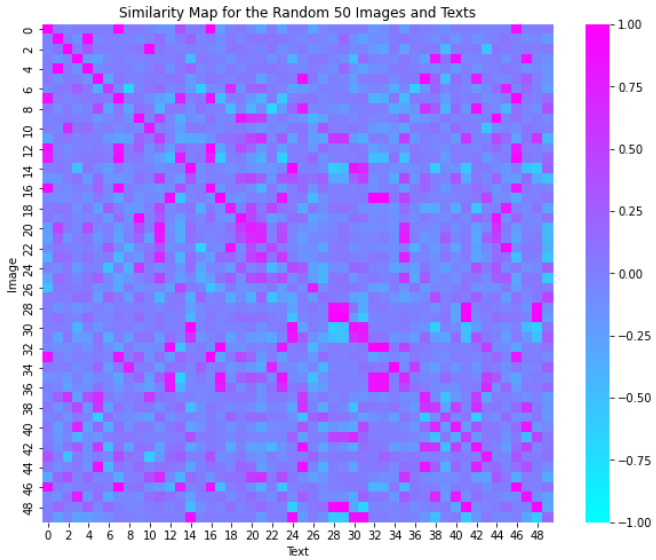

Figure 2: Cosine Similarity

Evaluation

To evaluate the model performance, we use 3 criteria, test loss, median rank (MedR), and recall rate at top K(R@K).

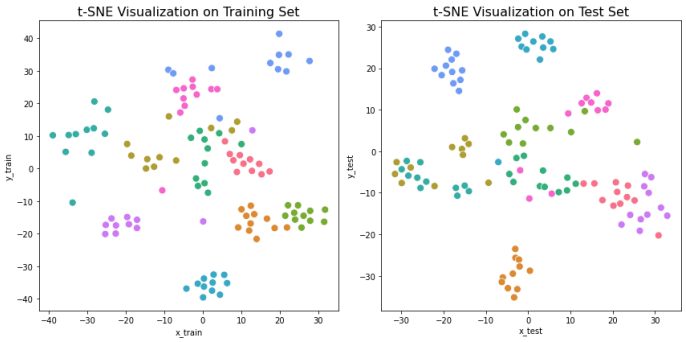

Figure 3: Visualization of Embedding Space by t-SNE

The MedR is the median of the rank of the true image over all the test data. So, smaller MedR indicates that the model works well. In R@K, we take only top K images for each query and check if the true image is included. By checking for all test data, we can calculate the probability that the true image is successfully retrieved. Thus, higher R@K is better. We set $K = 10$ in this work. The performance of our base model after 20-epoch training was in Table 2. The MedR and R@10 is on image retrieval task. The Figure 2 shows the cosine similarity between randomly chosen image-text pairs. If the model is perfect, we should see 1.0 in the diagonal entries and less than 0.1 (margin) otherwise, but there are several false positives. In Figure 3, we randomly choose 10 positive pairs, which consist of one image and 10 texts, and plot the embedding vectors using t-SNE. Though it's not perfect, we see some clusters in this plot.

Ablation Study

In this section, we make a small change to the base model and see its impact by comparing the modified model to the base model. The setting of the model is the same as the base model if not otherwise specified.

Xception

As a first attempt, we replace the Inception V3 in the image encoder with Xception. Surprisingly, the Xception outperforms the Inception by a significant margin as shown in Figure 4. Both of the models we used was pre-trained on ImageNet, and thus it is not that one is trained on data set that is closer to our domain.

ResNet

Continuing our focus on Image encoder, we replaced Inception V3 with another image classification architecture, ResNet. Pretrained ResNet101 and ResNet 50 models were used as image encoders. The intuition behind experimenting with ResNet101 is due to the fact that it is a deeper model with large number of trained parameters and we expected it to capture the image features in the embedding it produces. The observed initial results were similar to the performance of model with Xception image encoder. Since we used a pretrained ResNet model with 44.5M parameters, we wanted to

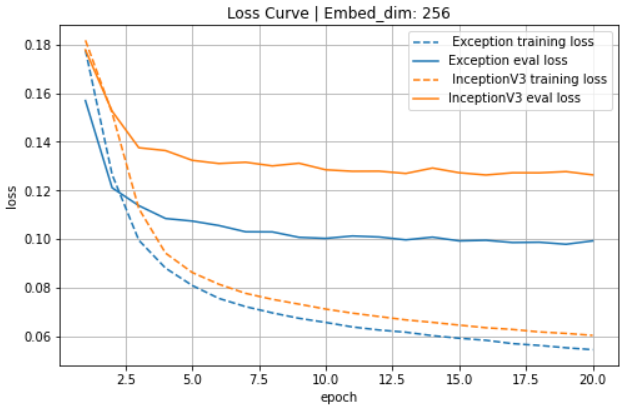

Figure 4: Xception

experiment with higher embedding dimensions to make better use of the additional trained parameters in the ResNet model. We increased the embedding dimensions from 256 to 384, 512, 1000. However, upon observing the results, increasing in the embedding size seems to have a negative impact on the models performance. Models with higher embedding dimensions had no significant performance improvement when compared the the base ResNet model with embedding size of 256. Similarly, ResNet50 and ResNet101 has similar performance.

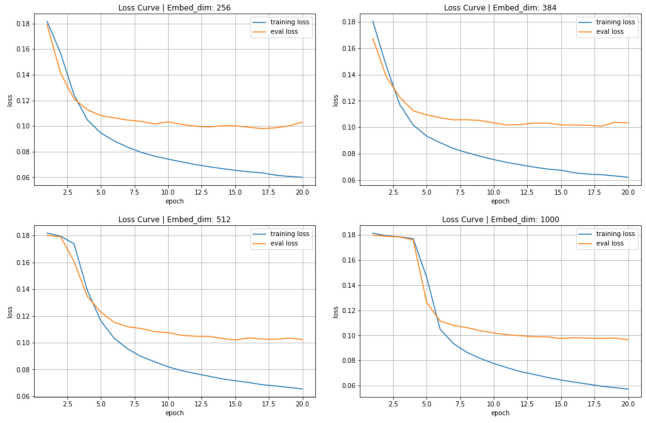

Figure 5: ResNet

YOLO

We also wanted to experiment with YOLO as the Image Encoder. Unlike the rest, YOLO is an Object Detection model and not an Image Classification model. For this reason we expected YOLO to give a better performance than the other models as an Object Detection model has the ability to recognise multiple objects/classes. We experimented using the first 10 layers of the YOLO(made available by the referred repo) along with BERT as the text encoder. Surprisingly, we observed poor performance from YOLO. Even after adding convolutional layers on top of the pretrained model the performance did not improve. After reducing to a large test loss, the model started to overfit. YOLO is a very complex model and the repo that we used for the pretrained model and weights was complex with many wrapper code making it difficult to modify and experiment more with YOLO. Thus we could not use YOLO as the image encoder.

GloVe

In this subsection, we replace the BERT text encoder with GloVe word embeddings. These embeddings are first processed in single layer bidirectional LSTM, then projected into the embedding space by succeeding fully-connect layers. The major difference between the two is i) BERT uses Transformer while GloVe model depends on sequential ordering of the text, and ii) BERT uses the WordPiece tokenization which oftern break a word into multiple tokens while there is a one-to-one correspondence between a word and a embedding in GloVe. As in Figure 6, the model struggles to get optimized even after 50 epochs. The test loss might be decreasing if we train more epochs, but it was beyond our computational resources. We reason that this might be because it is too difficult to learn the good embeddings while also learning how to extract semantic information from word vectors.

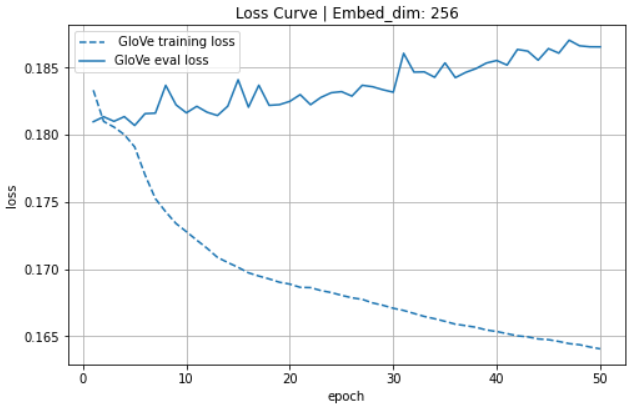

Figure 6: GloVe

Optimizer

In our base model, we chose Adam as our optimizer. In many supervised learning tasks, the loss function space is stable in terms of data set $D = \{(x_i, y_i)\}_i$. In other words, given data set D, there is a one-to-one correspondence between the loss function $\mathcal{L}(\theta)$ and the parameters θ. With this stable function space, the momentum in Adam gets us to the optima efficiently. In our cosine embedding loss, however, the loss function space is not stable. Given two embeddings emb_{image} and emb_{text}, we compute the loss and backpropagate it to update parameters θ_{image} and θ_{text}. θ_{image} is updated so that it outputs emb_{image} that is as close as possible to emb_{text} if $y = 1$ and nearly orthogonal to emb_{text} if $y = -1$. We say the function space is dynamic because after the update, the target emb_{text} also shifts due to the update of θ_{text}. Moreover, there exists an infinite number of optimal θ^*_{image} and θ^*_{text}. These are obtained by randomly rotating the coordinate system of the embedding space. Therefore,

we question the effectiveness of momentum in our problem. To test the impact of Adam, we trained one model with Adam and the other with SGD. The result is shown in Figure 7, indicating that Adam is still powerful in our setting. Though we cannot conclude which model attains the smaller loss from this experiment, Adam achieves faster learning than SGD, due to its momentum acceleration.

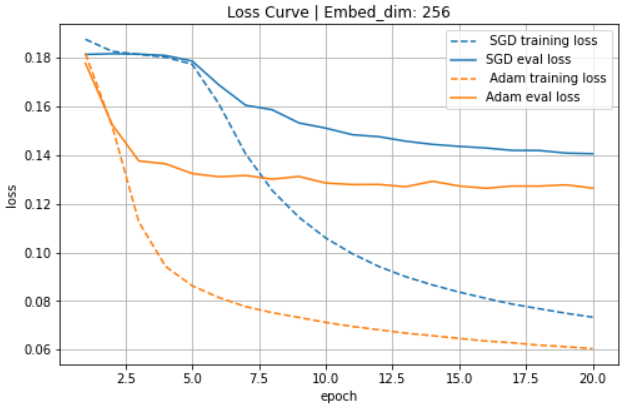

Figure 7: SGD Optimizer

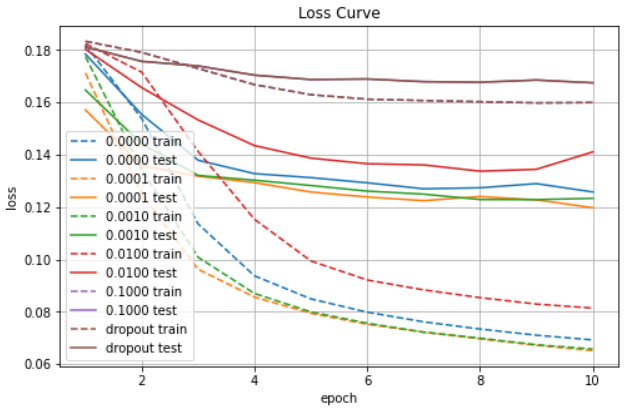

Figure 8: Impact of Regularization

Regularization

In the training result of the base model, we see a huge gap between training loss and test loss. This implies that using regularization techniques could improve the model performance on the test set. So, we tried ℓ_2 regularization and dropout with the rate of 0.35. The result is shown in Figure 8, which suggests that adding a small weight decay factor might improve the model performance. However, we have smaller training losses as well, and thus, better performance with regularization might be due to some randomness. On the other hand, dropout couldn't improve the model performance. In simple classification tasks, we train the model with random connection cuts but use the full network in prediction, working as an ensemble. Though this ensembling effect gives additional confidence in usual classification tasks, this is not considered as additional confidence but shifting the embedding vectors in an unwanted way in our problem.

Negative Pairs Rate

[Marin et al. 2018] studies the impact of the negative pairs rate, which is the number of negative pairs we produce for one positive pair. They suggest that 50 is optimal for their task. As this is dependent on the setting of the task, data, the size of embedding space, etc., we also need to search for the best value for the negative pairs rate for our problem. We try different values and the result is shown in Table 3. Note that with a higher rate, the model sees more data in each epoch. To adjust this impact, we train the models with different epochs. As in our base model, we have the rate of 4 and 20 epochs training, the epochs are adjusted as $\frac{20(1+4)}{1+\text{rate}}$. The result suggests that rates of 8-12 are the best for our task. The higher the rate is, the more model care about avoiding the penalty from negative pairs, resulting in a high loss from positive pairs.

Rate	Image Ret. MedR	Avg. Loss from Single Pair	
		Positive Pairs	Negative Pairs
1	63 / 1,350	0.3163	0.0979
2	49 / 1,350	0.3555	0.0666
4	50 / 1,350	0.4379	0.0529
8	28 / 1,350	0.4900	0.0303
12	27 / 1,350	0.6305	0.0178
16	32 / 1,350	0.7943	0.0072
20	675 / 1,350	0.9001	0.0000

Table 3: Impact of Negative Pairs Rate

Learning Rate

Our base model is trained with a learning rate of 0.0005. It seems that the learning rate for dual-encoders and cosine similarity loss is typically very small, [Dong et al. 2020] uses 0.0001 for learning rate for instance. In this subsection, we study the impact of the learning rate on our model. The result is shown in Figure 9. Obviously, 0.0025 and 0.0125 are too large for this task and there is no learning happening. The smallest learning rate of 0.00002 looks taking time to converge simply because of the small step size. This experiment suggests that if we want to train more epochs to obtain a better model, one strategy would be starting training with a learning rate of 0.0001, then changing it to 0.00002 after having no improvements for a predefined number of epochs. To better understand the actual learning process, we visualize the first two parameter update results for two learning rates in Figure 10. The faint and small markers are the initial similarities, and we used darker and larger markers to represent the parameter updates. With a good learning rate of 0.0001, the cosine similarity for positive pairs increases in each update while the similarity for negative pairs increases in the first update and then dropped in the second update. An intuitive explanation would be the loss was only from positive pairs in the first update due to the margin. In the second update, however, the loss was also from negative pairs, penalizing the negative pairs. In the bottom plot, both positive and negative pairs behave nearly the same in each update. In

the first update, the loss was generated only from the positive pairs again, and the corresponding update made all pairs close to each other. Then in the second step, the loss was dominated by the ones from negative pairs, forcing all the pairs to separate. Though further investigation is needed to fully understand the learning process, this is our explanation of why training is successful with a small enough learning rate and why training never converges with a too-large learning rate.

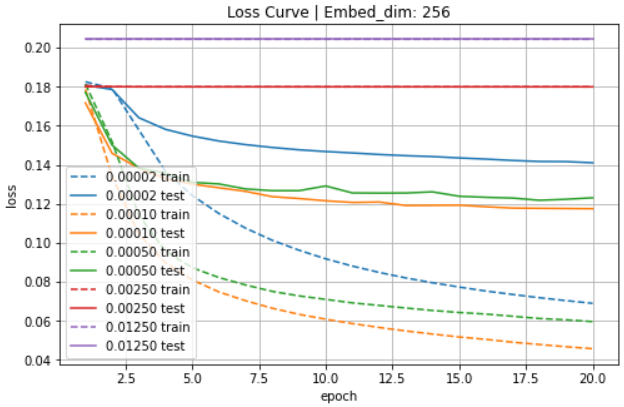

Figure 9: Impact of Learning rate

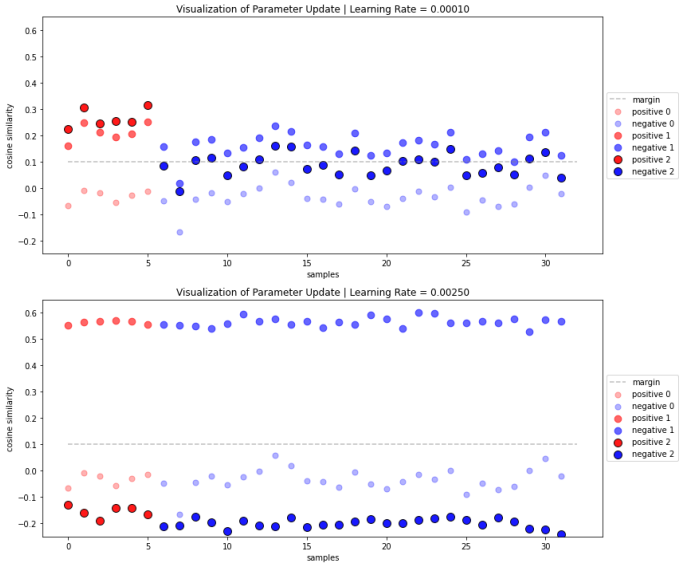

Figure 10: Learning Process Visualization

Dimensionality of Embedding Space

As we can accommodate $\mathcal{O}(2^d)$ nearly orthogonal vectors in d-dimensional space, there should be a strong connection between the dimensionality of embedding space and the number of positive pairs in the training set. Our experiment result is shown in Figure 11. From this plot, 16 is too small for the embedding space, at least struggling to find a good projection. On the other end, increasing the dimensionality doesn't improve the model performance.

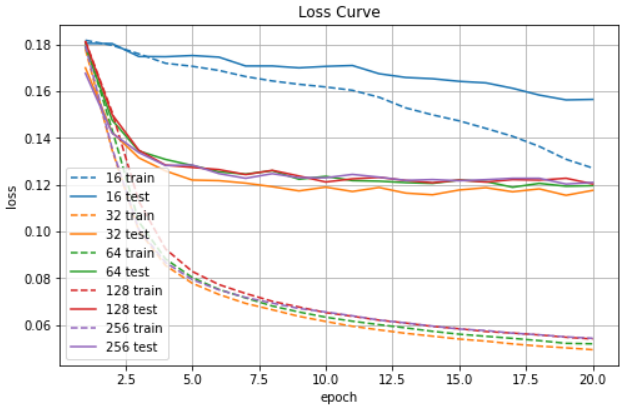

Figure 11: Impact of Dimensionality of Embedding Space

Margin Factor in Cosine Embedding Loss

In this subsection, we study the impact of margin on our loss function. The margin affects only negative pairs. Recall that ideally, we want to project positive pairs to be identical while negative pairs to be orthogonal. However, we could accommodate only d mutually orthogonal vectors in d-dimensional space. Thus, we need to relax it to *nearly* orthogonal by allowing negative pairs to have a small cosine similarity. The margin works as a threshold that decides the cosine similarity of negative pairs is small enough. We tried five different values for the margin. As this margin is related to the size of the embedding space, we set it to 18, which turned out to be a little tight for a margin of 0.1 from the previous subsection. The results are in Table 4. With -0.1 margin, the median rank is 676, which indicates the model is just random. From the loss, we see this model gives up optimizing for positive pairs due to relatively heavy loss from negative pairs. By increasing the margin, we see decreases in both losses from positive and negative pairs. The losses from negative pairs are relatively small, reflecting there are 4 times more negative pairs than positive pairs and the difficulty of reducing the loss for positive pairs. The median rank is almost the same for margins of 0.3 and 0.5, while we have a higher median rank for a margin of 0,1. This result is consistent with the observation in the previous subsection as a margin of 0.1 is too tight for 18-dimensional space, and relaxing the margin results in better projection. Once the margin is loosened enough, further relaxation doesn't improve the performance. Note that the average loss from positive pairs of 0.28 in a 0.5 margin indicates that the average angle of positive pairs is about 44.7 degrees, suggesting there is more room for improvement for positive pairs.

Margin	Image Ret. MedR	Avg. Loss from Single Pair	
		Positive Pairs	Negative Pairs
-0.1	676 / 1,350	1.1005	0.0000
0.0	72 / 1,350	0.3935	0.1009
0.1	62 / 1,350	0.3927	0.0766
0.3	47 / 1,350	0.3421	0.0438
0.5	46 / 1,350	0.2886	0.0242

Table 4: Impact of Margin

Negative Pairs Shuffling

So far, we create the negative pairs before training. One way to teach the model what pairs of vectors should be mapped separately is to show more negative pairs. Though this can be controlled by negative pairs rate, our study shows that having a too-high rate deteriorates the model performance as the model focuses only on mapping negative pairs correctly. In this subsection, we consider another approach of shuffling the negative pairs generation in each epoch. By doing this, we can show more negative pairs to the model while keeping the negative pairs rate unchanged. This might improve the robustness of our model on unseen data. The outcome is shown in Figure 12. We see an increase in training loss due to the shuffling because the model cannot decrease the training loss by overfitting to the fixed negative pairs. As a result, our model achieves better robustness, resulting in better test loss.

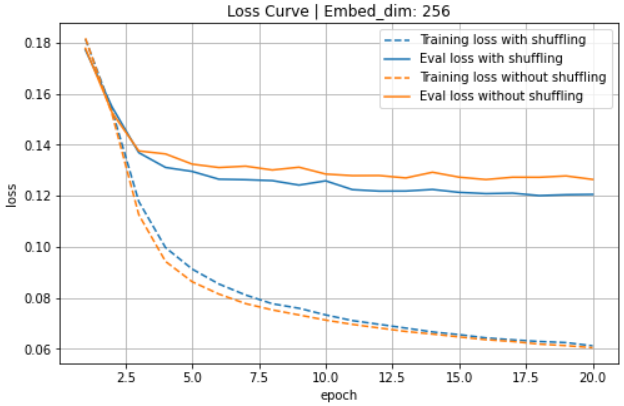

Figure 12: Negative Pairs Shuffling

BatchNorm instead of Tanh

In our base model, we use the tanh activation after the fully-connected layer. As the output of this activation is our embedding, one-sided activations, such as ReLU or sigmoid, utilize only one single quadrant out of 2^d quadrants. We choose tanh because it's two-sided and symmetric. Another approach that [Dong et al. 2020] uses is Batch Normalization, which enforces the outputs to have Gaussian distribution. The comparison of the two methods is shown in Figure 13, and the Batch Normalization results in a better test loss. To further analyze this result, we normalize the embedding vectors and draw an histogram of the entries in Figure 14. Because of its design, BatchNorm produces a Gaussian distribution while tanh generates a two-peak curve. Under the assumption that embedding vectors are constructed by randomly drawn from these distributions, it is provable that the resulting vectors are unbiased if we use the Gaussian distribution. Let $\mathbf{x} \sim \mathcal{N}(0, I_d)$ and $\bar{\mathbf{x}} = \mathbf{x}/\|\mathbf{x}\|_2$. Then, for the distribution of $\bar{\mathbf{x}}$, we have

$$F(\bar{\mathbf{x}}) = \prod_{i=1}^{d} \left(\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\bar{x}_i^2} \right) = \left(\frac{1}{\sqrt{2\pi}} \right)^d e^{-\frac{1}{2}\|\bar{\mathbf{x}}\|_2^2}$$

This indicates that the distribution of $\bar{\mathbf{x}}$ only depends on the scaling factor and thus, is rotationally invariant. Therefore, the distribution is equivalent to the distribution uniformly sampled from a unit ball. Recall that we want to embed the image embedding vectors mutually nearly orthogonal. In this sense, the BatchNorm makes embedding vectors unbiased, and uses the embedding space efficiently. This analysis matches the result we get.

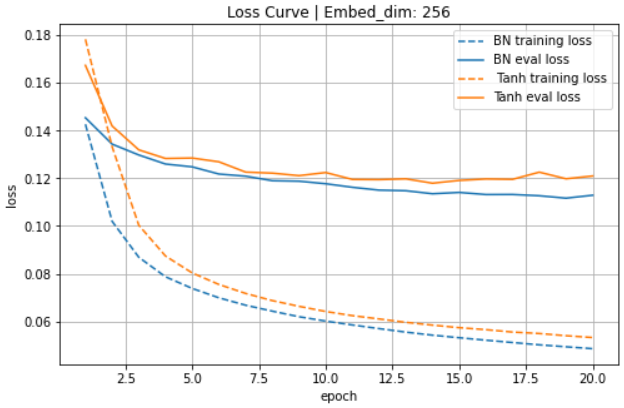

Figure 13: Tanh v.s. BatchNorm

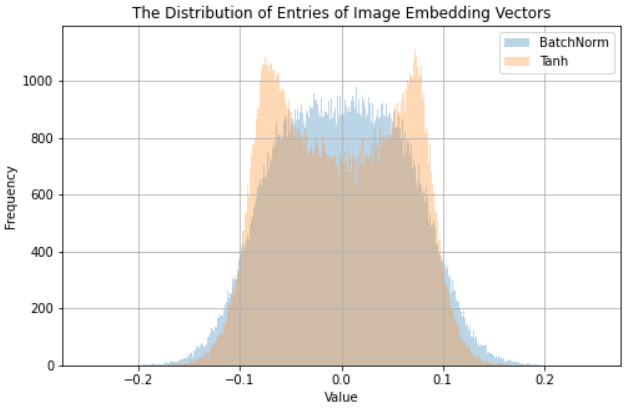

Figure 14: Normalized Vector Entries Distribution

Best Model

After all of the experimentation we found the following configuration of hyperparameters as the best performing:

Hyperparameter	Optimal Value
Image Encoder	Xception
Text Encoder	BERT
Embedding Size	256
Learning Rate	0.0005
Optimizer	Adam
Activation	BatchNorm
Negative Pairs Rate	8
Loss Margin	0.1

The System

Now that we have the best dual encoder model we train it completely with the nocaps dataset. the training process is shown in figure 15.

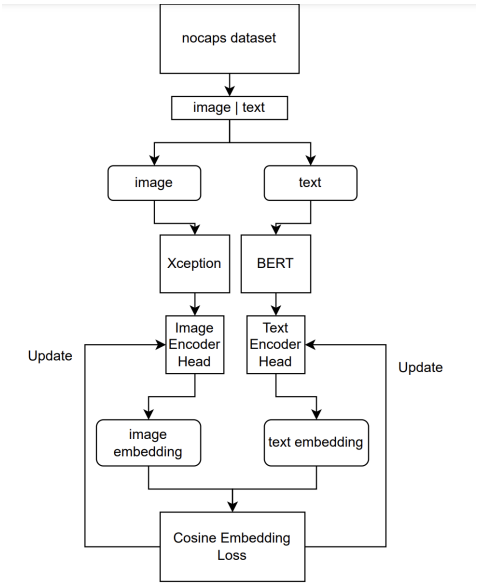

Figure 15: Training Process

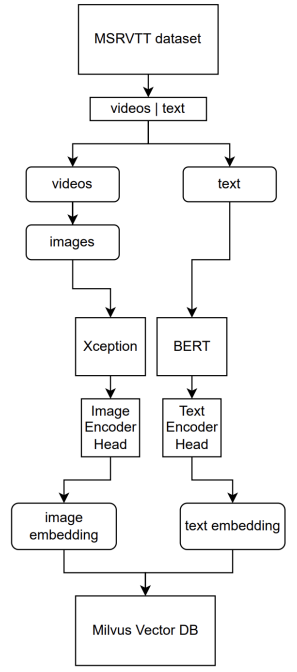

Figure 16: Loading MSRVTT Data

Once we have our dual encoders we create a vector database of points from the MSRVTT dataset. It contains 10K web video clips with and 200K clip-sentence pairs in total, covering the most comprehensive categories and diverse visual content. Since the video clips are short, averaging around 12 seconds, we sample one image per video, map it with the list of captions associated with the video and create a vector database using Milvus. Each video in the dataset is associated with 10 captions. We extract one image from each video and take the 10 captions and create 11 points for each video in the database by passing through the encoders and inserting in the DB. this process is shown in figure 16.

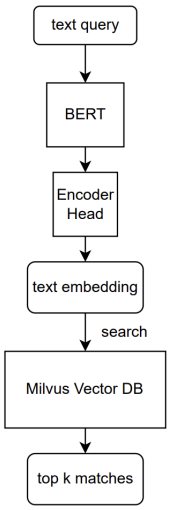

Figure 17: Querying Process

Since the model learns to recognize a wide variety of visual concepts in image and associate them with textual data, We can use it to extract visual representations from natural language. The system is now ready for video search. Given a text query we obtain the embedding from the text encoder and the text encoder head and query the vector database with this embedding. This returns the top k matches in the database. These points are basically images or texts inserted before. These points contain the information(name and path) of the video. We now display the relevant videos. The querying process is shown in figure 17.

Future Work

To attain a better performance, we can use some advanced regularization techniques introduced in [Marin et al. 2018] and [Dong et al. 2020]. Their use of classification tasks as an additional error shows a better generalization on unseen data. Due to this regularization, images and texts similar at high-level forms a loose cluster in the embedding space and thus, the chances of retrieve a totally unrelated vector decrease. Apart from traditional CNNs, Transformer architecture for vision based tasks can be used as an Image Encoder although it requires lot more fine tunings to acheive better results [Dosovitskiy et al. 2020].

Conclusion

Using the concept of Dual Encoders we developed a Video Search System that given a text query returns the matching videos from the database. We determined the best Dual Encoder model of Image and text after various experimentation in hyperparameter tuning. We used a vector database to store the embedding vectors and query them. We learnt and implemented multiple deep learning models such as InceptionV3, Xception, Resnet, YOLO, BERT and GloVe. We developed a full-fledged system using deep learning concepts for a very

realistic use case.

References

Agrawal, H.; Desai, K.; Wang, Y.; Chen, X.; Jain, R.; Johnson, M.; Batra, D.; Parikh, D.; Lee, S.; and Anderson, P. 2019. nocaps: novel object captioning at scale.

Chollet, F. 2016. Xception: Deep Learning with Depthwise Separable Convolutions.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.

Dong, J.; Li, X.; Xu, C.; Yang, X.; Yang, G.; Wang, X.; and Wang, M. 2020. Dual Encoding for Video Retrieval by Text.

Dosovitskiy, L.; Beyer, A.; Kolesnikov, D.; Weissenborn, X.; Zhai, T.; Unterthiner, M.; Dehghani, M.; Minderer, G.; Heigold, S.; Gelly, J.; Uszkoreit, N.; and Houlsby, N. 2020. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep Residual Learning for Image Recognition.

Marin, J.; Biswas, A.; Ofli, F.; Hynes, N.; Salvador, A.; Aytar, Y.; Weber, I.; and Torralba, A. 2018. Recipe1M+: A Dataset for Learning Cross-Modal Embeddings for Cooking Recipes and Food Images.

Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; and Zettlemoyer, L. 2018. Deep contextualized word representations.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Poloshkhin, I. 2017. Attention Is All You Need.

Wu, L.; Fisch, A.; Chopra, S.; Adams, K.; Bordes, A.; and Weston, J. 2017. StarSpace: Embed All The Things!

Appendix

Figure 18: Results for ”Man driving a car”

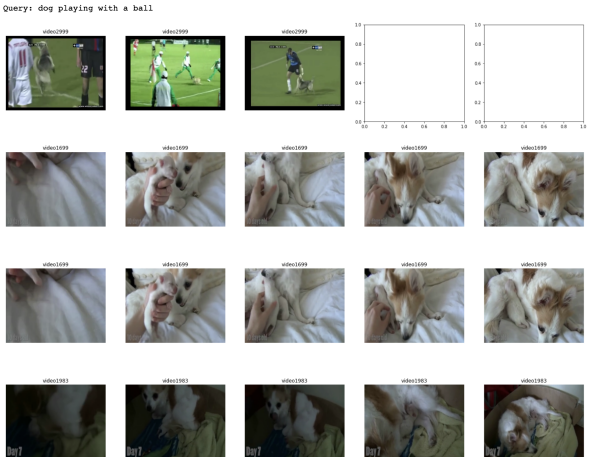

Figure 19: Results for ”Dog playing with a ball”

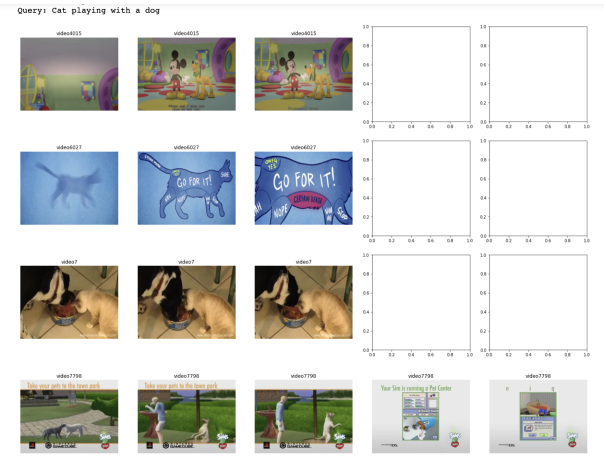

Figure 20: Results for ”Cat playing with a dog”

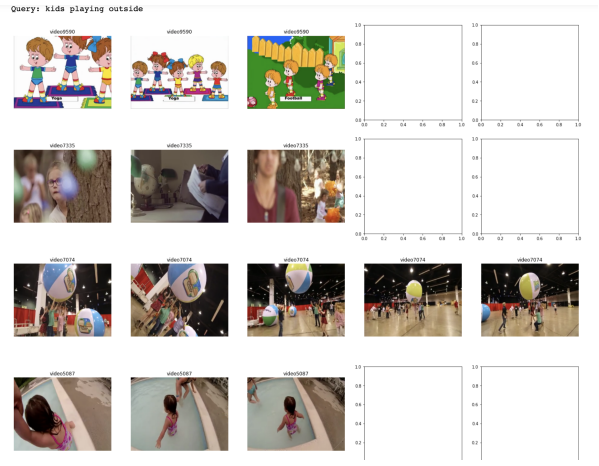

Figure 21: Results for ”kids playing outside”