# Goal of the Image Search Engine

1. Searching for similar images to an input image (Image → Image)
2. Searching images using text (Text → Image)
3. Generating tags for images (Image →Text)

# Image-to-Image, searching for similar images

Solution -

1. We can build a model which can provide us pairwise similarity score when two images are given to it as an input, but this leads to scalability issue. Let's assume our model is CNN we have more than a dozen images, this becomes too slow to be even considered.

2. We can make a model which produces embeddings or feature for image. Comparing these embeddings for similarity we can get the similar images.

# Solution - Image embedding

1. The idea is to find a real valued vector representation similar to word embedding that encodes the meaning of the image such that vectors of similar images are closer in the vector space and vectors of completely different images are far from each other in the vector space.

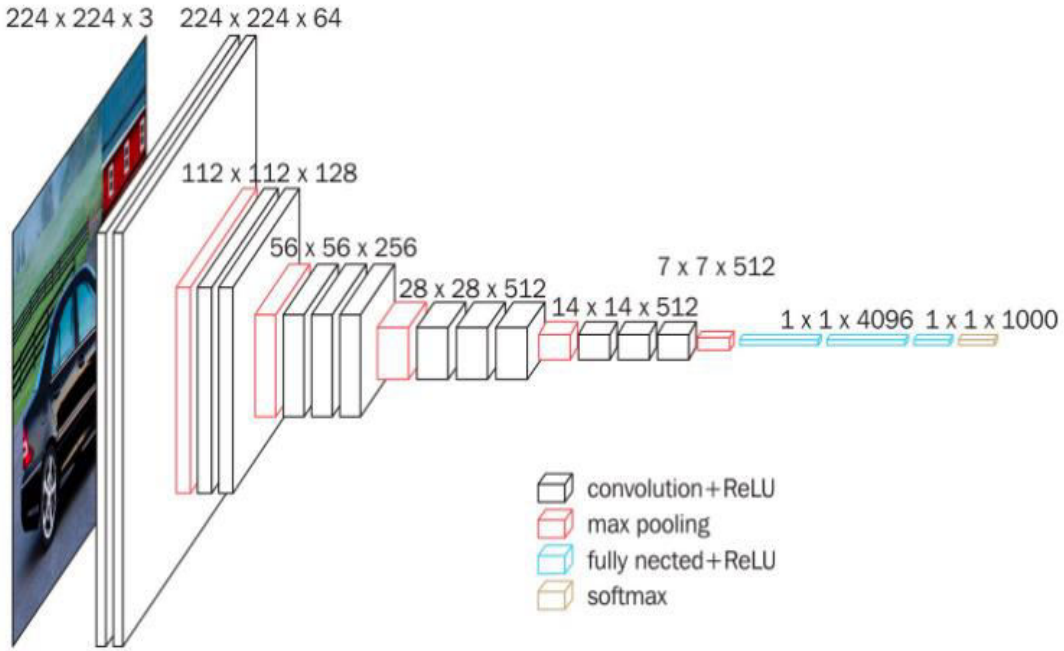
2. We can find similarity between two vectors using cosine similarity which is measure of similarity between two non-zero vectors.

3. Using Image embedding solution makes our model more scalable.

# VGG16

1. We used a CNN model - VGG16 (shown on right side) which is pre-trained on a large image dataset (Imagenet) available online to find embeddings for our images.

2. We use this model upto penultimate layer to find embeddings. Each embedding is 4096 vector.



# Cosine Similarity

1. Cosine Similarity is a metric used to measure how similar the vectors are.

2. Mathematically, it measures the cosine of angle between two vectors projected in a multi-dimensional space. The cosine similarity is advantageous because even if two similar vectors are far apart by the Euclidean distance (due to difference in sizes of documents), chances are they may still be oriented closer together. The smaller the angle, higher the cosine similarity.

# Using Annoy for Indexing

1. We can build a fast index of embeddings using Annoy which will allow us to very quickly find the nearest embeddings to any given embedding. We can store this index of embeddings in the disk. (which makes it scalable)

2. Annoy is a library used for finding approximate nearest neighbours in search space (using cosine similarity).

3. Now we can simply take an image, get its embedding and look in our fast index to find similar embeddings.

# Image→Image: on Indexed Images

```
image_to_image_search(image_embeddings[51],fp[51])
```

```
Search image:
```

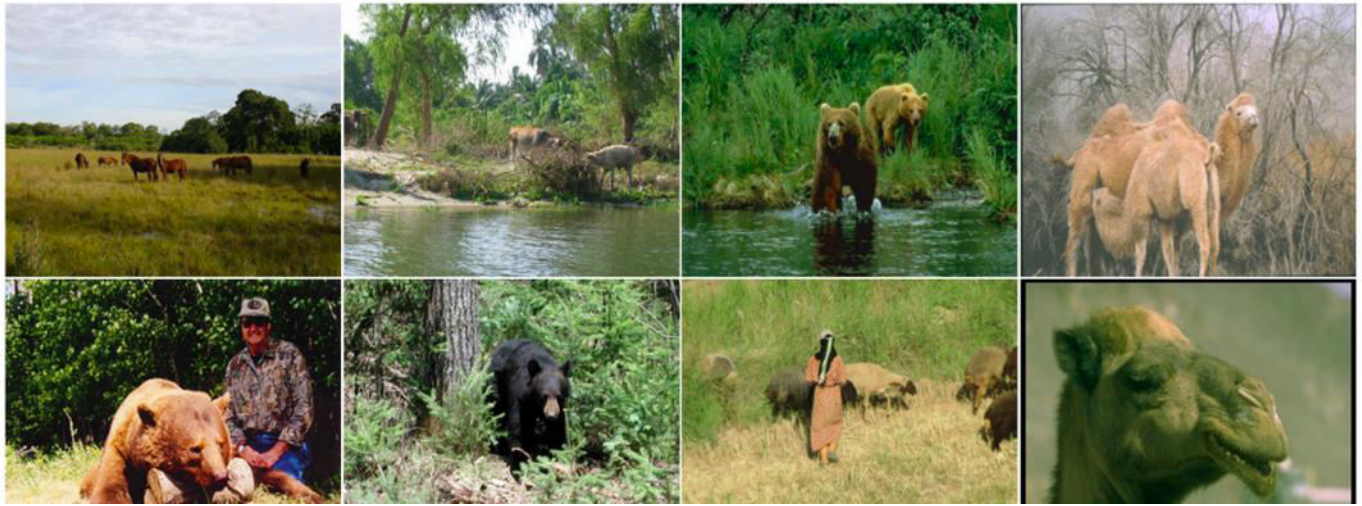
```
Similar images-
```

# Image→Image: on Unseen Image Samples (non indexed)

```
image_to_image_search(t_image_embeddings[17],t_fp[17])
```

Search image:

Similar images-



# Word Embeddings

1. To find embeddings for each word we use pre-trained vectors from GloVe which were obtained by crawling through wikipedia and learning the semantic relationships between words in that dataset.

2. GloVe is a model for distributed word representation. This is an unsupervised learning algorithm for obtaining vector representations for words. It maps word into a meaningful space where the distance between words is related to their semantic similarity.

3. We can find similar words by finding similarity between word embeddings.

# Text ⇔ Image

1. Now we have an embedding space for images and an embedding space for words, to achieve text ⇔ image we need a common embedding space for both images and words, then we can simply find images with similar embeddings to the embedding of a word and vice-versa.

2. So our task is to combine these two embeddings and have a common embedding.

3. First of all, the size of vectors in both the embeddings is an issue, as for words we have embedding vectors of size 300 and for images we have embedding vectors of size 4096.

4. Secondly, let's say if size was not the issue, then we can clearly see that these two embedding spaces are obtained from completely different models which have no relation with each other.

# Solution

1. In VGG16 model, the last layer is of size 1000 representing the probability of each class.

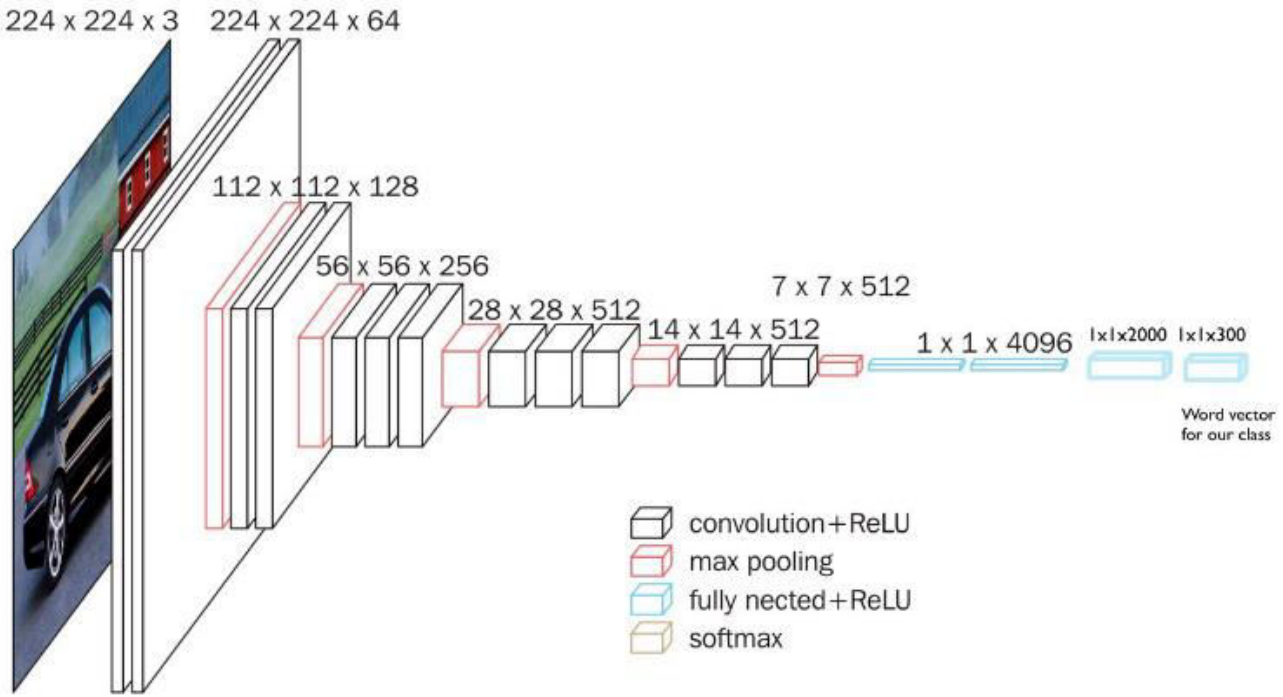
2. To make the hybrid model, we will replace this last layer of the model with the word vector of our category, this will allow our model to learn to map semantics of images with semantics of words.

3. We do this by adding two layers:
   (i). One intermediate layer of size 2000.
   (ii). One output layer of size 300.

4. We will train our model to learn to predict word vector associated with the label of the image but we will only train the last 2 layers.

5. Our model will now predict a semantically rich word vector of size 300 for input image.

# Model




Now our final model looks like this.

Dataset : 32 classes with approx 50 images each

Training-Test Split : 20%

Unseen label Dataset : 5 classes with 10 images each

**Accuracy Metric :**

> **“flat” hit@k** – the percentage of images for which the model returns the one true label in its top k predictions.

**Model Parameters:**

```
================================================================
Total params: 143,064,044
Trainable params: 8,798,900
Non-trainable params: 134,265,144
________________________________________________________________
```

Loss Function : Cosine Similarity

Optimizer: SGD

Epochs: 60

# Word to Image search :

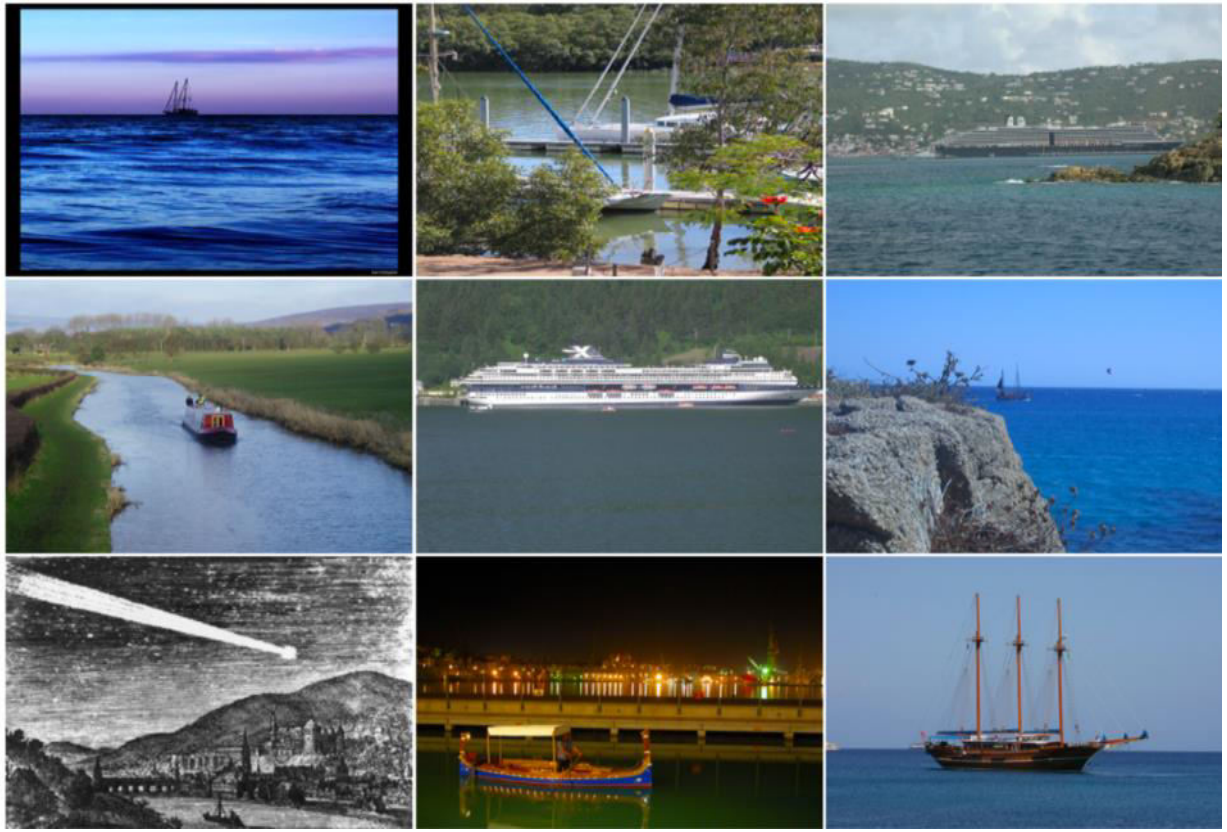
searching word :

‘**ocean**’

**Flat hit @k**

| **k** | Training set | Test Set | In Paper |
|---|---|---|---|
| 2 | **96.3%** | **60.9%** | **65%** |
| 3 | **96.3%** | **62.5%** | **70%** |

```
word_to_image_search('ocean')
```

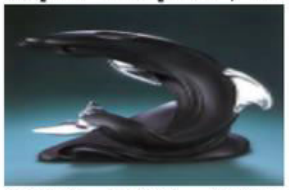


# Generating tags for images (of unknown labels):

1. A distinct advantage of our model is its ability to make reasonable inferences about candidate labels it has never visually observed.( Zero Shot Learning)

2. For 16 out of 50 unseen label image, exact label was generated.

dolphin -> biplane , sailboat , pony , pigeon , reptile , tractor , dolphin , cuckoo ,



dolphin -> fish , bottle , eaten , chicken , eating , duck , stuffed , yellow , dish ,

elephant -> horse , horses , cat , sheep , elephant , deer , dog , animals , rabb

elephant -> horse , horses , sheep , cow , dog , herd , cat , elephant , animals

# References


- **DeViSE: A Deep Visual-Semantic Embedding Model**
  Link - https://papers.nips.cc/paper/2013/file/7cce53cf90577442771720a370c3c723-Paper.pdf

# Thank You