Recent Advancements in Computing power, Deep learning [1] based algorithms are able to grasp/learn the hidden pattern from large pile of data .Neural Network in particular inspired from human Brain, helps to identify hidden pattern in the data.

Previously Machine learning models, required manual feature engineering to get the decent/ benchmarking accuracy with the advent of deep learning (coined by Hinton), feature engineering was also taken care by the neurons in the neural architecture. Manual feature engineering for Image and NLP demands Domain knowledge in the particular field which will be overcomed by the Deep Neural Networks

In this project/proposal, I implemented Neural Network architecture for both Images and Text data, Using Convolutional Neural network for image (which acts as Image feature synthesizer) and Recurent Neural network (LSTM -Long short term memory) for Text based data (which is used for sequence data)

Image Captioning -Show and Tell published in 2016,uses Encoder and Decoder Architecture, for getting text output from the image input.

I am also using Encoder-Decoder based Architecture[2] for this use case,where the Encoder takes image as the input and produced the Embeddings as the output-(which captures the image related encodings) and this embeddings are feed to the Decoder which is RNN(LSTM) and produces the word embedding output (300 dimension) ,from this word Embeddings ,I will run Beam search to get most relevant sentence for the given image

I am planning to implement the Show and tell architecture by adding Attention mechanisms[3] between the Encoder and decoder architecture which is like -show ,attend and tell architecture.

I tried different variants of architecture starting from Feed forward neural network to RNN for text -and CNN for images.

I preprocessed the text with word embeddings and trained the RNN for initial set of weights and then image was given as the initial state to the Decoder (RNN) and each previous word was given to predict the current word this words forming a tree structure, produces the output sentence for which combination of words there exists the maximum probability.

Complete Flow:

Introduction:

The Emergent tool helped me to understand the importance of hidden layers in deep neural networks to predict a image that is hard to learn. Now, In this assignment I did some literature work on the basics of neural network and worked on image captioning project. The network trains itself from a data set containing 8000 images and gives caption to each image.

Artificial Neural Network :

A neural network is simply the artificial working of a human brain. In a human brain each neuron takes input processes and send it to the terminal node as an electrical pulse through the axioms. This network is simulated in a artificial environment. This technology is the basis for machine learning.

Deep Neural Network :

As the word says a neural network with many layers of hidden layers is termed as deep neural networks. More the depth of hidden layers more the system can be accurate. This is called deep learning.Unlike machine learning where the labelled data structure format is the basic for detection deep learning uses a series of convolution layer to predict the output .

Activation layer :

The activation layer takes the input from the first layer and produces a intermediate output that is understandable only for the network and the next layer will take decision based on this output. It is used to create a non-linear transformation of the input data .

Fully connected layer :

When a layer has access to all the inputs and activations from the previous layer then it is a fully connected layer.

Epoch :

It is the number of times a learning algorithm will work through the entire data set. One epoch means each variable in the training dataset had an opportunity to update its value. The number of

epochs is usually large it is allowed till the system learns properly. It is simply one forward pass and one backward pass of all training examples.
Iteration:
It is the number of times a batch of data has passed through the algorithm.It is both the forward pass and backward pass, Batches are subdivisions of input . we can't send the entire input to the neural network in one stage so we divide them into batches.
Supervised and unsupervised learning:
Most machine learning algorithm uses supervised learning . Where you have an input (x) and an output (y)an algorithm is used to learn the mapping .The algorithm guides the learning process when there is an error like a teacher and is complete when a level of acceptance is achieved so called as a supervised learning.ex.classification and regression.
In unsupervised learning there is no teacher it has only inputs and the algorithm is left to itself for the process without any supervision ex.clustering and association.
Back propogation:
A basic neural network has input and output in between this each neuron selects some weights in the process of predicting the output . If there is a deviation from the expected output this error (deviation)is propogated again to the input for the neurons to predict a correct weight. Through a process of optimization via gradient descent.
Forward propogation:
The process of forwarding data from input to the output is termed forward propogation. Every neural network will run a number of forward and backward propogations till it reaches a minimal or zero error value.

Image captioning :
As the name says it is the process of giving a caption to an image based on training sets of data.
Encoder and decoder :
One of the recent acheivements in the neural network area is the use of encoder and decoders. Neural networks are defined by CNN and RNN both are different and require different algorithms. Using both in one system is very complicated. Now, the encoder is made to do the role of CNN and the decoder plays RNN and they form a same system.This makes the training of datasets easy and efficient . The encoder takes the input processes and gives an intermediate output for the input taken (eg. Image or video) The decoder takes the intermediate result and processes for the original output(words in our case).

The CNN is state invariant and the RNN is timing invariant.

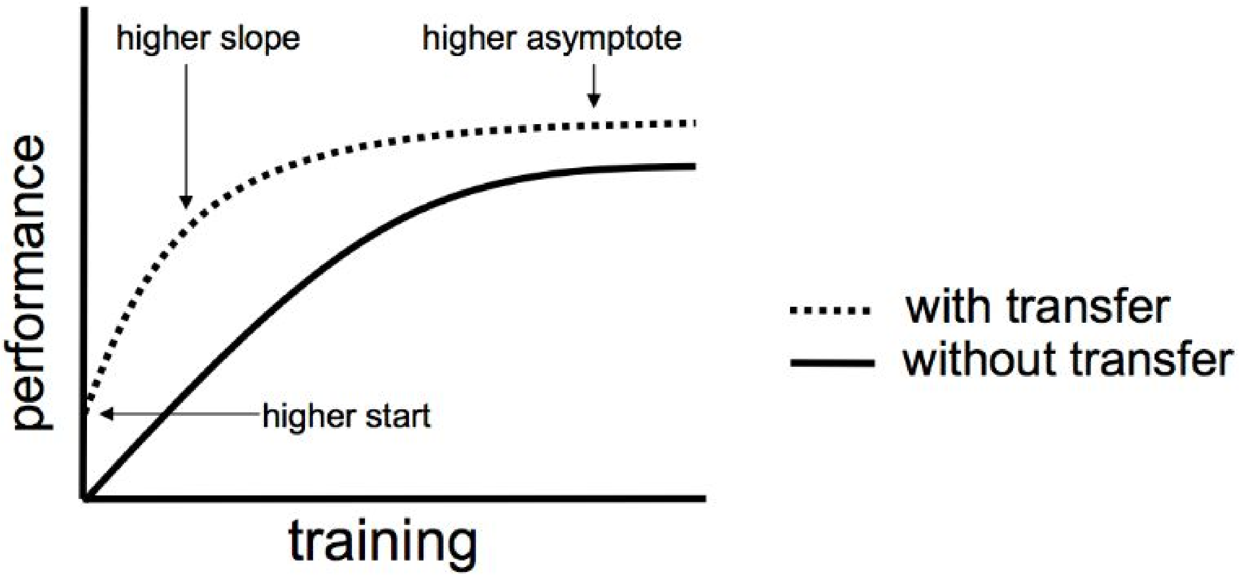
Long Short Term Memory Units (LSTM):
It is a good at learning dependencies between two points in a timeline that are separated far timely.

For example learning to predict a word in a long sentence where the word strongly depends on some other word that occurred much much before in the same sentence.
LSTM is a major development in the RNN algorithm. The RNN basically has a track of previous information access to predict the present output . A simple RNN is a collection of many neural networks with dependencies . Following diagram shows the basic structure of a RNN in unrolled fashion.
But, the problem arises when the length of the dependencies becomes very long. To solve this problem LSTM is used. This is capable of detecting long term dependencies.

(say some 20 lines about LSTM with a pic)

Optimizer :
If the predicted output is wrong then the network back propogates the output and the optimizer that seperates part of network will make it correct depending on number of errors.More the number of pause then matrix weights will be adjusted . I used ADAM optimizer in this project .
(What optimer are we using)

Transfer learning :
It is very complex and time consuming to develop and train a full convolution neural network from the scratch. From the initialization it will consume a lot of time the size of data is also very large. Transfer learning helps us to solve this problem. Here a model trained for one task can be used to train another task to serve another process.This optimization helps the second task to perform faster with good performance .
Lisa Torrey and Jude Shavlik describes three possible uses for transfer learning.



Higher start : The initial skill is higher than a normal model without transfer learning

Higher slope : The rate at which the skill learning is improved is steeper.
Higher asymptote: The skill of trained model is better as compared to the other.
In this project I have used a pretained model of transfer learning scenario.

Imagenet :
This is the source from where I used the images (transfer learning) from Vgg16 .It is an open source there are many competitions happening here for the image classification accuracy.I have used these images for my image captioning analysis.
Modern ConvNets take about 2-3 weeks to train across multiple GPUs on a ImageNet, it is common to see people release their final ConvNet checkpoints in the open source tool so that others can use the networks for fine-tuning.
I used caffe library that has a model zoo where anyone can share their network weights.

Image captioning Architecture
“”””
Citations papers
1: A Survey on Transfer Learning
https://www.cse.ust.hk/~qyang/Docs/2009/tkde_transfer_learning.pdf
2: VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION
https://arxiv.org/pdf/1409.1556.pdf
3: Visual Geometry Group
http://www.robots.ox.ac.uk/~vgg/
4:Where to put the Image in an Image Caption Generator
https://arxiv.org/pdf/1703.09137.pdf

“”””

Image captioning is combining the two worlds in the Deep learning ecosystem which is Image and Text.Image captioning as in is Neural Architecture which takes image as input and outputs the caption as sequence.

Training the Image classification or Language model as separate network was easy , the errors from the output layer can be easily back propagated into the interior layers , but Training image captioning model as a standalone network was not so straight forward and getting Descent accuracy needs extensive tuning of model hyper-parameters because while Back propagating the errors from the Text sequence model into Image model the gradients will vanish therefore update will not occur in the image CNN layers this problem is called as Vanishing gradients

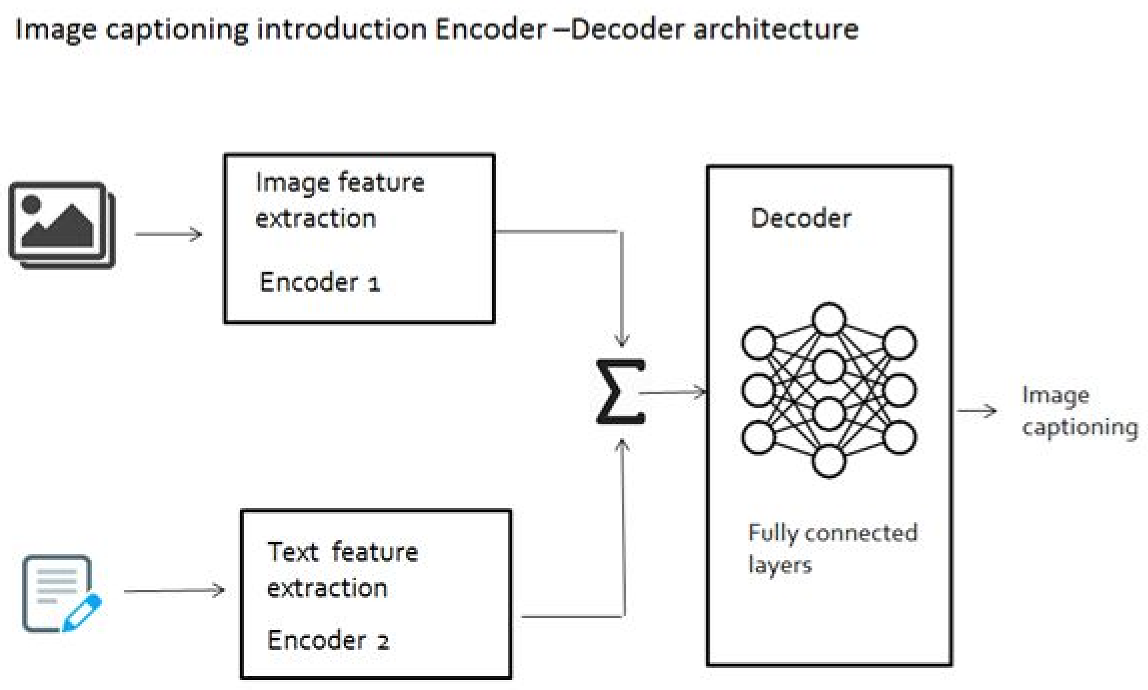


Image captioning Architecture components

- Image Feature Extraction
- Text feature Extraction
- Combining techniques
- Decoder
- Training the model
- Prediction /inference

Image Feature Extraction:

To extract the feature from the images we used concept of transfer learning which helps to move knowledge obtained from one domain into another domain and enjoys lot of traction and success this days according to the paper [1]

Transfer learning

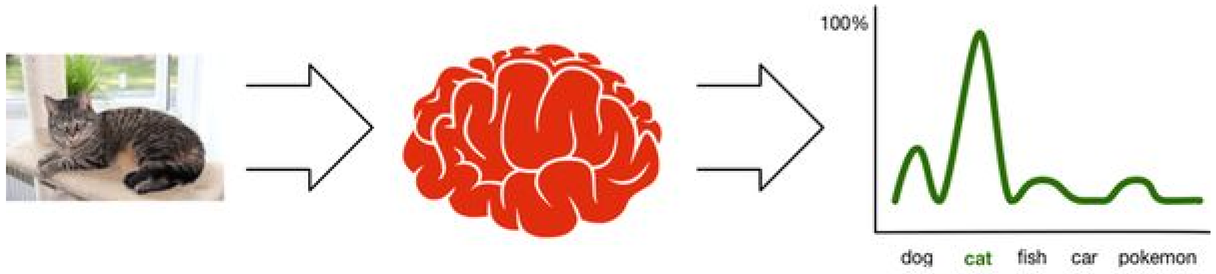
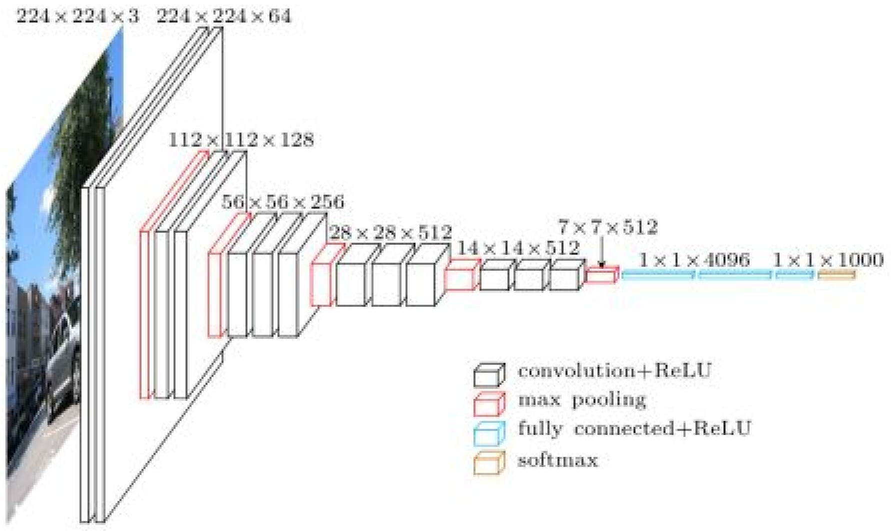


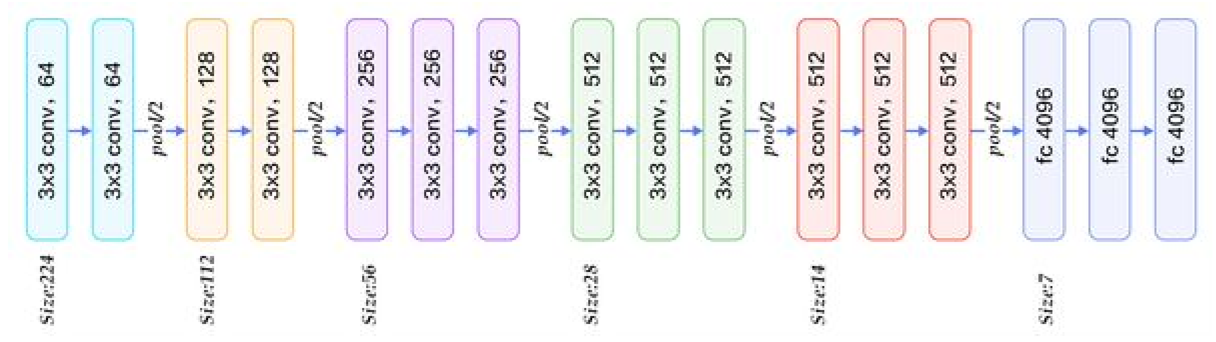
Image classification task

We used Pretrained Neural network model , which was trained on image classification task ,(taking image as input and outputs a particular task as output) .
We used particular network called VGG16 [2] which was built by : Visual Geometry Group [3] ,which consists of 16 layers   of

- Convolution layers,
- Max-pooling layers
- Fully connected layers



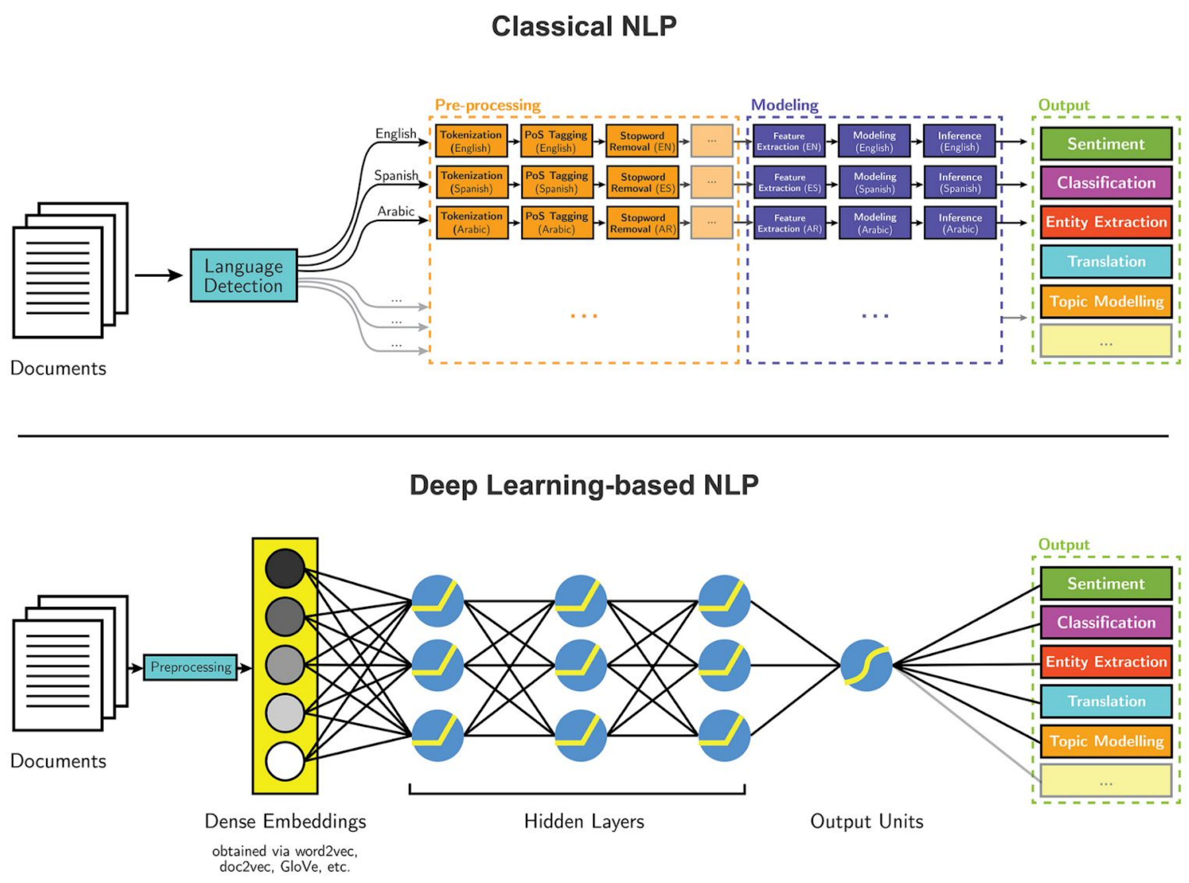
View of VGG



Layers Overview

VGG16 image classifier network was used to extract the features from the image .i.e we get the neural weights which is of (1,4096) embedding, from the last layer of the VGG16 (we exclude the Softmax layer at the end)
Output of the Image feature extraction step is vector of length (1,4096)

<u>Text feature Extraction</u>



To extract the features from the text /caption , we can use same transfer learning methodology for text instead of using machine learning based manual feature extraction .

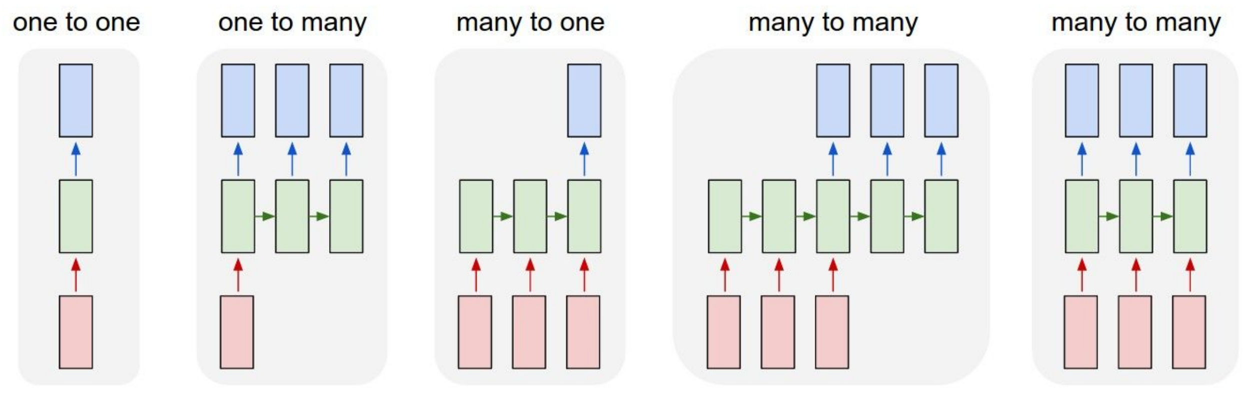
CNN and RNN network cells can be used to extract text features from the caption dataset, In this dataset, we used RNN to extract features from the captions.

Steps to extract features from the Captions:

- Tokenize the captions (sentences)
- Map the word tokens into the unique id
- Using this word ids frame the input as sequence
- Input the sequence into the RNN network
- Extract the features from the RNN layers

Inspirations to use RNN in our architecture stems from this amazing blog written by the "Unreasonable effectiveness of RNN" http://karpathy.github.io/2015/05/21/rnn-effectiveness/

RNN can take input sequence of any form into output sequence of any form.



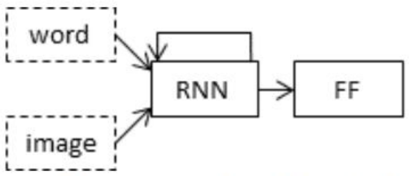
Seq to seq mapping using RNN

Caption generation can be implemented :

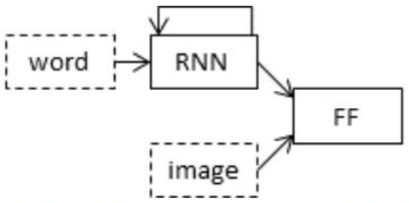
- One to one (Input :[image +one word] ,output :one word per step)
- Many to many (input : [image+many words] ,output :many words per step)

Combining techniques
There are two basic ways of combining techniques[4] which combines the image and text features and produces caption given a image .

- Conditioning by injecting the image
- Conditioning by merging the image



(a) Conditioning by injecting the image means injecting the image into the same RNN that processes the words.

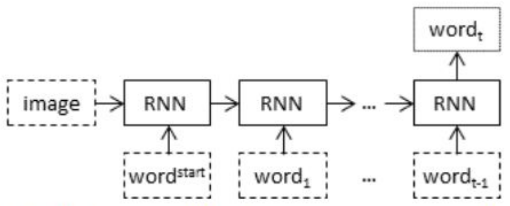


(b) Conditioning by merging the image means merging the image with the output of the RNN after processing the words.

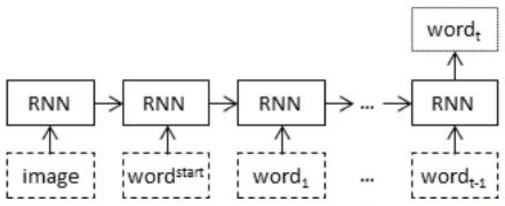
In 'inject' architectures, the image vector (usually derived from the activation values of a hidden layer in a convolutional neural network) is injected into the RNN, for example by treating it on a par with a 'word' and including it as part of the caption prefix

In 'merge' architectures, the image is left out of the RNN subnetwork, such that the RNN handles only the caption prefix, that is, handles only purely linguistic information. After the prefix has been encoded, the image vector is then merged with the prefix vector in a separate 'multimodal layer' which comes after the RNN subnetwork. Merging can be done by, for example, concatenating the two vectors together. In this case, the RNN is trained to only encode the prefix and the mixture is handled in a subsequent feedforward layer

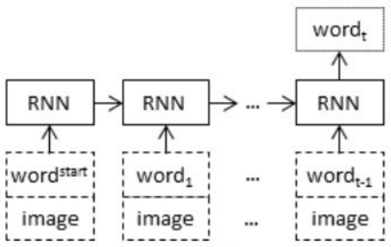
If an RNN's hidden state vector is somehow influenced by both the image and the words then the image is being injected, otherwise it is being merged.



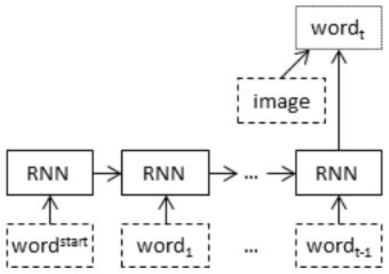
(a) Init-inject: The image vector is used as an initial hidden state vector for the RNN.



(b) Pre-inject: The image vector is used as a first word in the prefix.



(c) Par-inject: The RNN accepts two inputs at once in every time step: a word and an image.



(d) Merge: The image vector is merged with the prefix outside of the RNN.

(sub types of the Combining techniques)

In this report we used Merge architecture as our image captioning network.,

Decoder:

Since we used merge architectures which combines the feature output of both the networks VGG16 (CNN) and RNN (LSTM) network ,we used feedforword layer at the end Therefore feedforword layer act as the decoder which combines the two layers features and Identifies the pattern between the two networks

Training

Training the models which are different from each other(CNN and RNN) was difficult, and while training we faced memory issues ,therefore we made the image features and text features separately
and merged the features using the fully conected layers ,therby escaping the memory issue

prediction

at prediction time ,When a new image was passed to our network ,our image featurizer (VGG16) produces the
features and we pass START_SEQ to our text featurizer, thereby we have image and text features ,and our model
produces the text output and this text output again passed as input into the text encoder model

this continues until the model produces the END_SEQ

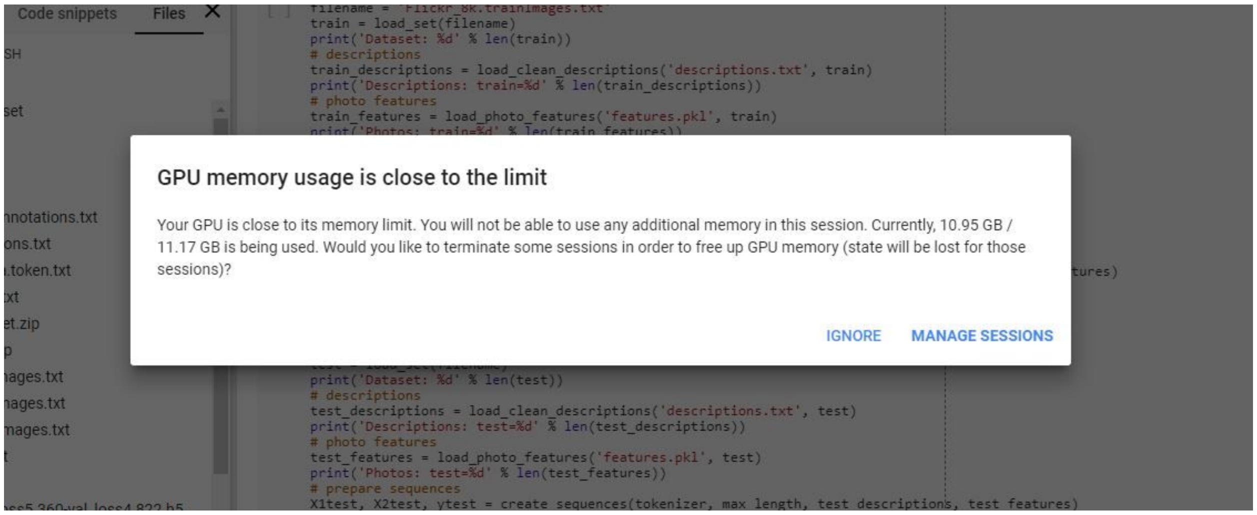
Experiments

Trail 1: (ludwig_notebook)

Training the image ,text ,and combiner models in a one shot :

challenges:

Not able to fit the model in System/machine RAM thereby we are not able to train the models



```
filename = 'Flickr_8k.trainImages.txt'
train = load_set(filename)
print('Dataset: %d' % len(train))
# descriptions
train_descriptions = load_clean_descriptions('descriptions.txt', train)
print('Descriptions: train=%d' % len(train_descriptions))
# photo features
train_features = load_photo_features('features.pkl', train)
print('Photos: train=%d' % len(train_features))
```

GPU memory usage is close to the limit

Your GPU is close to its memory limit. You will not be able to use any additional memory in this session. Currently, 10.95 GB / 11.17 GB is being used. Would you like to terminate some sessions in order to free up GPU memory (state will be lost for those sessions)?

```
print('Dataset: %d' % len(test))
# descriptions
test_descriptions = load_clean_descriptions('descriptions.txt', test)
print('Descriptions: test=%d' % len(test_descriptions))
# photo features
test_features = load_photo_features('features.pkl', test)
print('Photos: test=%d' % len(test_features))
# prepare sequences
X1test, X2test, ytest = create_sequences(tokenizer, max_length, test_descriptions, test_features)
```

Trail 2: (keras)

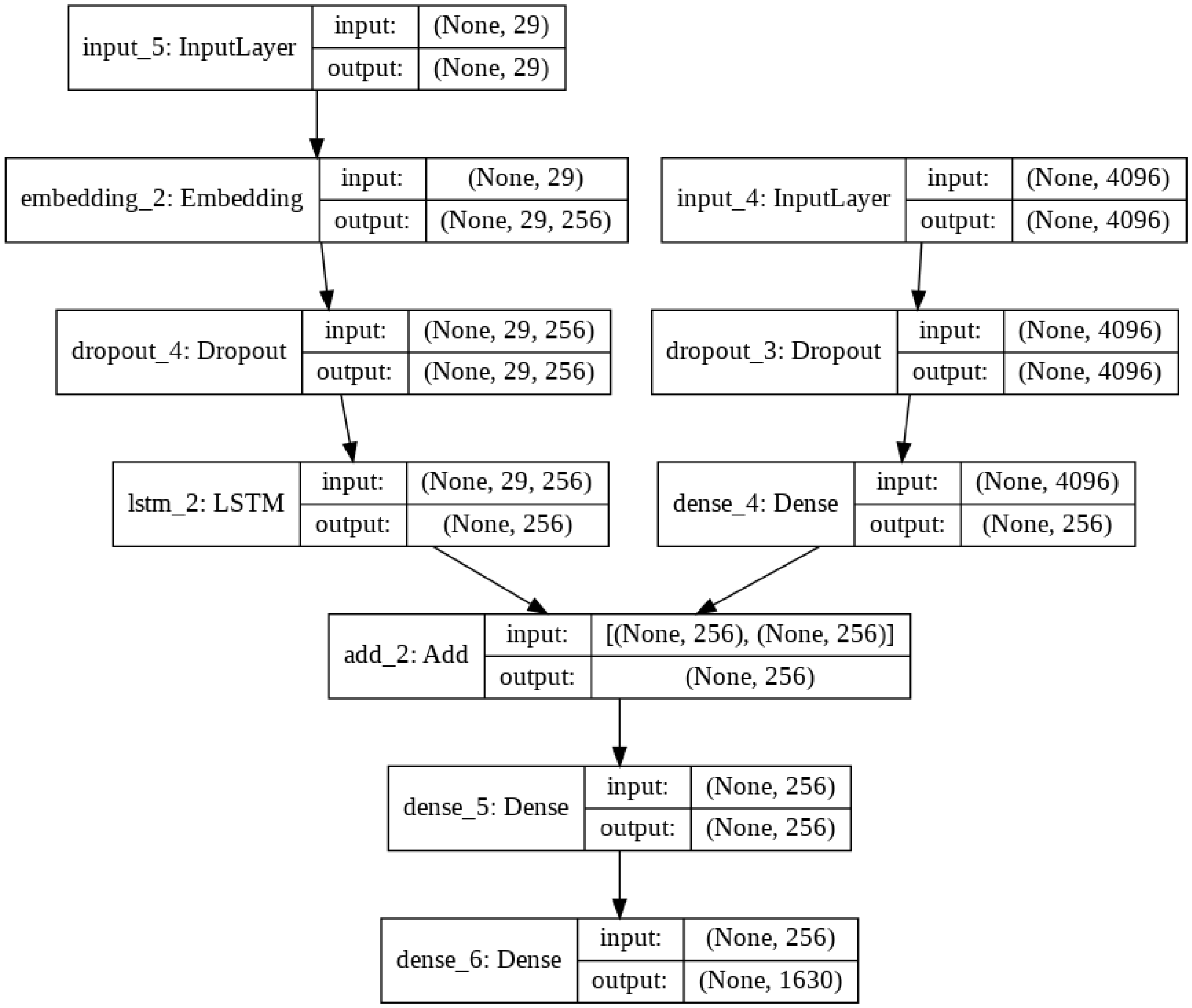
KERAS network

Training the image ,text and combiners separately which means

We will get image features from the VGG16 and save those results to Machine's Disk

We will then train the text features using captions from the dataset and serializes to the Disk

we then load the two models and train the Fully connected Decoder which produces the output



Keras layers produced in python script

Trail 3 : kerasv0.1

Challenges:

===========

Even though we trained the model separately,sometimes we are facing memory issue according to the load of the machine

After some point of time we identified this is happening due to Loading entire dataset (all 8k images) at once into the System memory(RAM) results in system crash

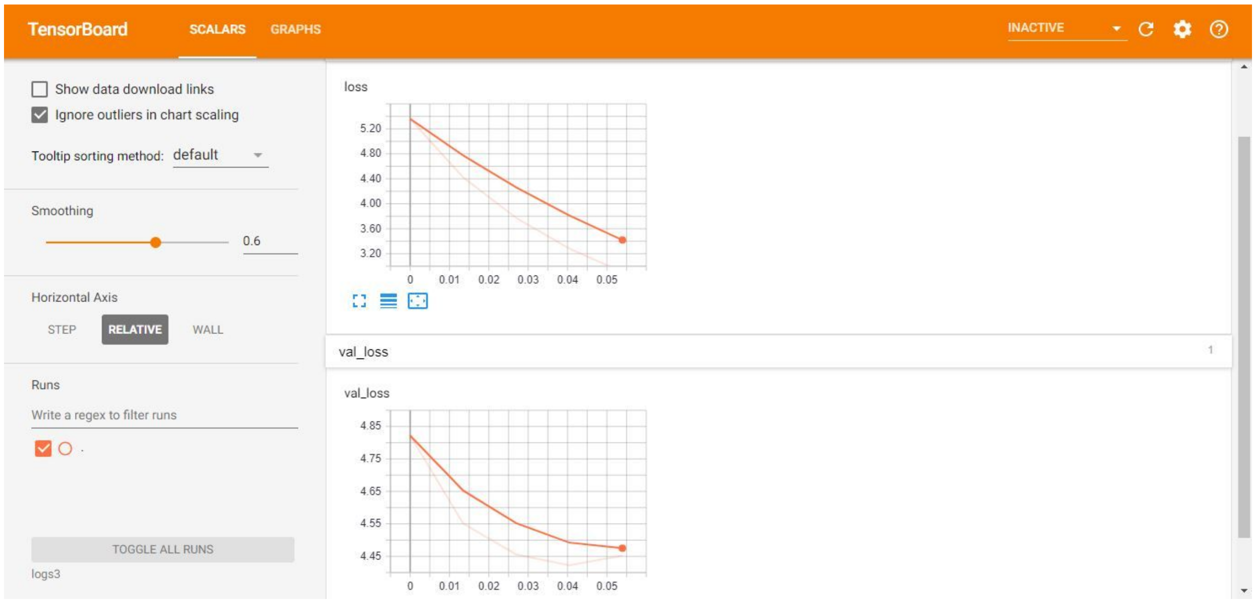
Workaround :

Instead of the loading the entire dataset ,we used python iterator/generator which is kind of lazy loading which loads the bacth of ,say 32 images and captions pairs ,trains the model and loads the other set of pairs(image and captions)

We used keras handy function to train the model in batch process which is Fit_generator which takes input directly from the python generator

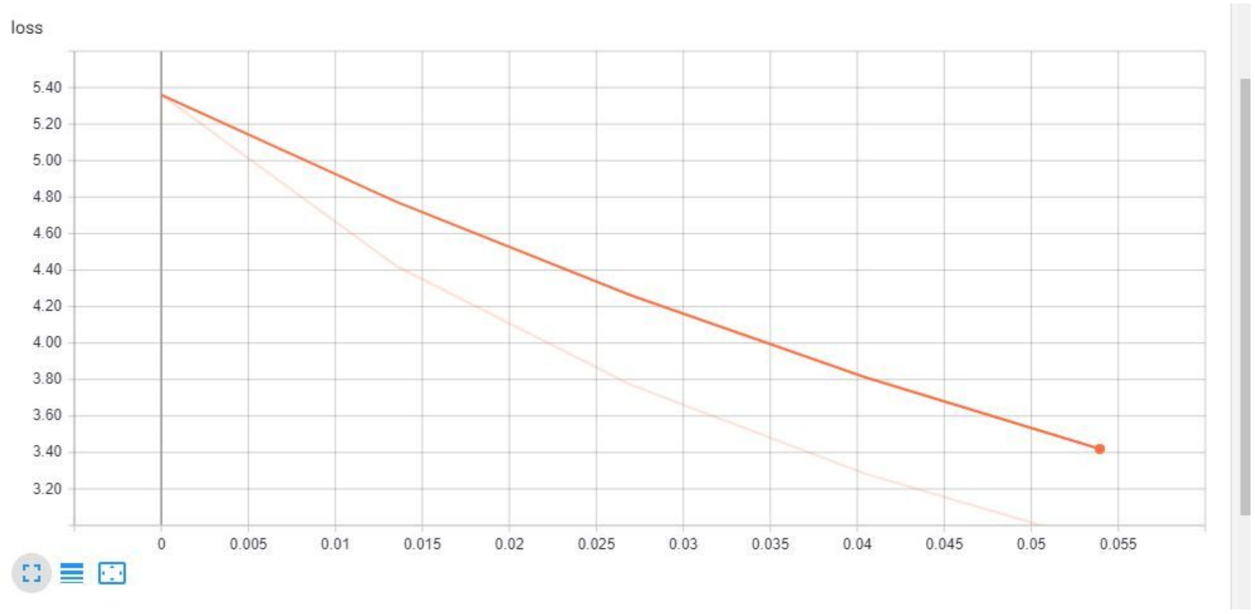
Tensorboard visualizations:

We log our training and testing process in tensorboard and visualized the same as below

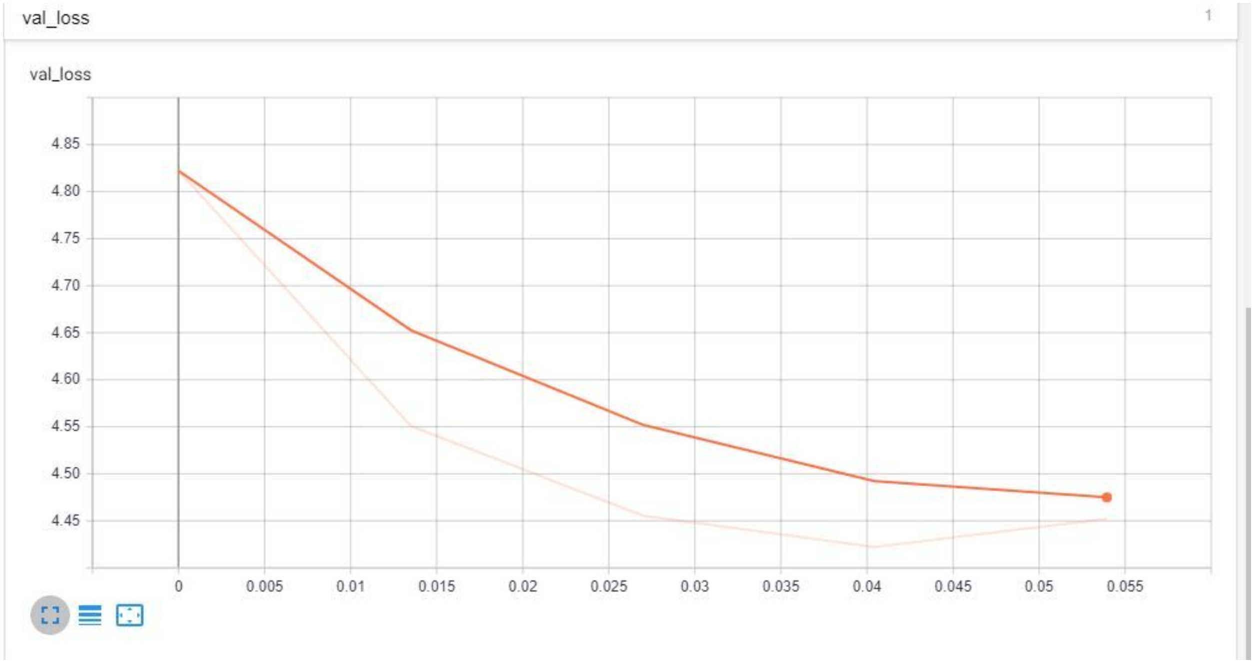


Show data download links
✓ Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0.6
Horizontal Axis STEP [✓ RELATIVE] WALL
Runs
Write a regex to filter runs
logs3

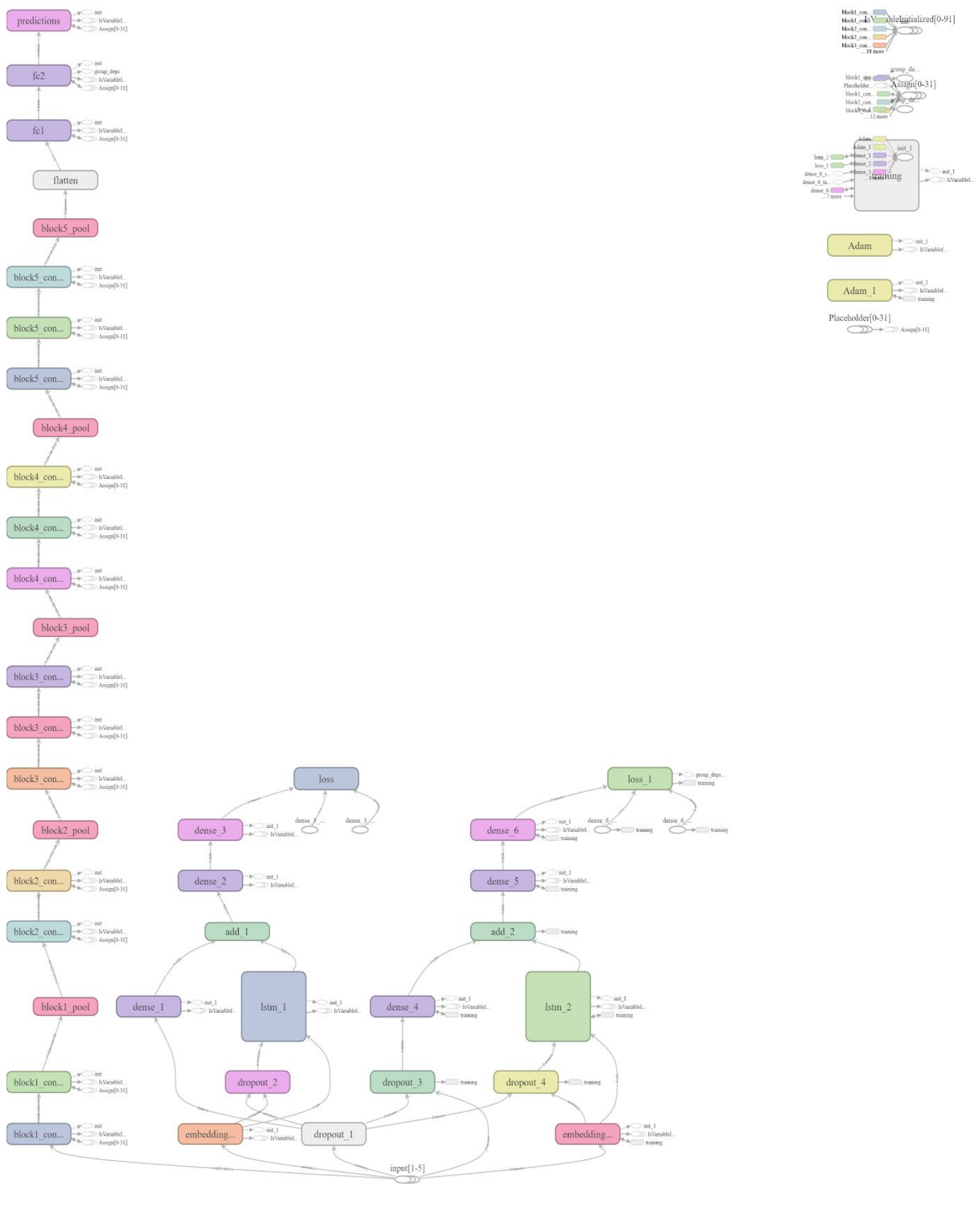
Tensorboard Experiments



Training loss (loss decreases at the training time)



Validation loss (for every epoch of training we validate our model with Validation dataset which is also decreasing as shown in the figure)

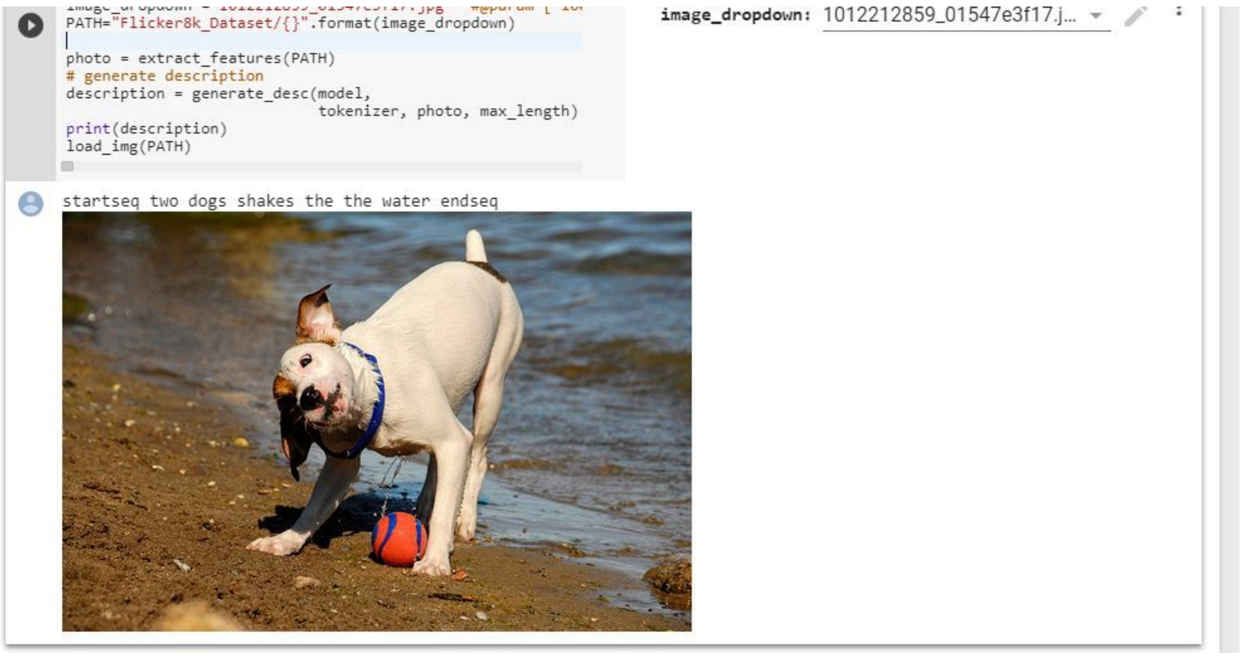


Tensorboard _neural architecture Visualization

Wrong and correct prediction:

After training the model ,we are getting both the correct and wrong results from the network

We trained different models ,sometimes the trained able to predict the caption correcly ,sometimes model predict the same word again and again

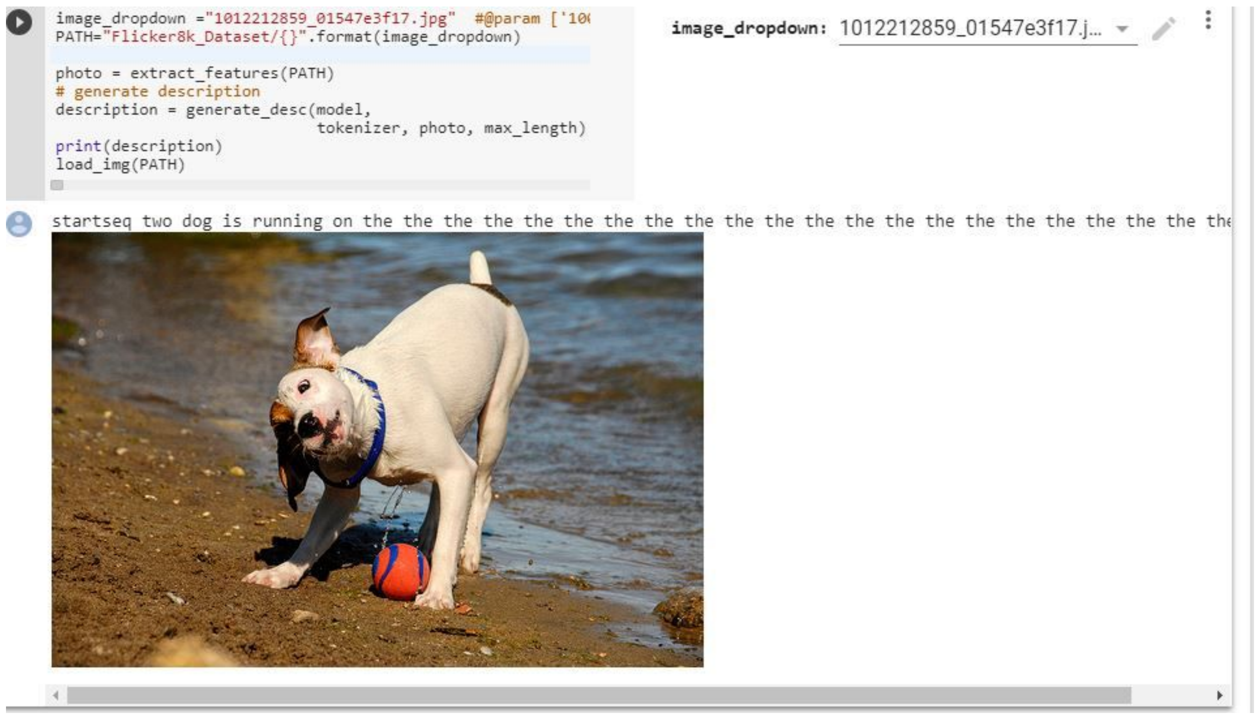


```
PATH="Flicker8k_Dataset/{}".format(image_dropdown)

photo = extract_features(PATH)
# generate description
description = generate_desc(model,
                           tokenizer, photo, max_length)
print(description)
load_img(PATH)
```

```
startseq two dogs shakes the the water endseq
```

Correct caption generation



```
image_dropdown ="1012212859_01547e3f17.jpg"  #@param ['10
PATH="Flicker8k_Dataset/{}".format(image_dropdown)

photo = extract_features(PATH)
# generate description
description = generate_desc(model,
                            tokenizer, photo, max_length)
print(description)
load_img(PATH)
```

image_dropdown: 1012212859_01547e3f17.j...

```
startseq two dog is running on the the the the the the the the the the the the the the the the the the the the th
```

Wrong caption generation (predicting the word “the “ again and again)

Image Explanations

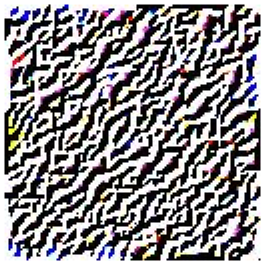
To have the deeper understanding of the network , I tried to understand whats happening in the each hidden layers of the network

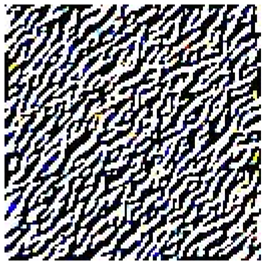
We are using Image and Text encoder networks which are completely black box
There are some frameworks /techniques to understand the interior of the neural network

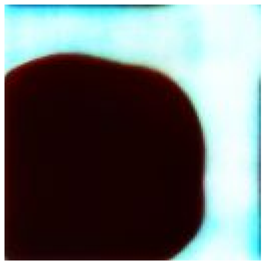
Explanation Trail 1:

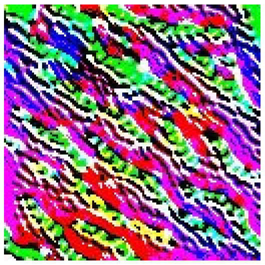
We tried exploring the different techniques such as Gradient explainer from SHAP, LIME explainer etc but one thing which worked for us was the framework called lucid which is realeased by the google (we read this amazing blog post from https://distill.pub/2018/building-blocks/) and tried to use the framework for our Image network

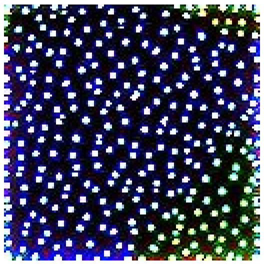
We explored layer by layer for the VGG16 ,respective layer weights visualizations was attached with this report

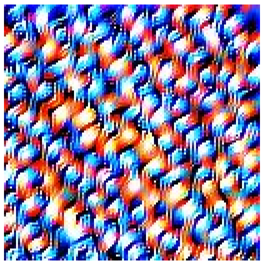

conv1_1/conv1_1

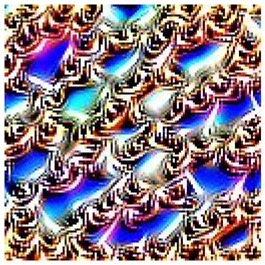

conv1_2/conv1_2

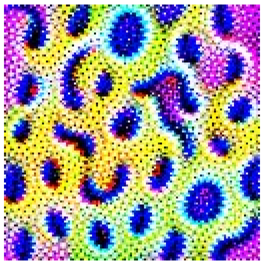

conv1_2/conv1_2

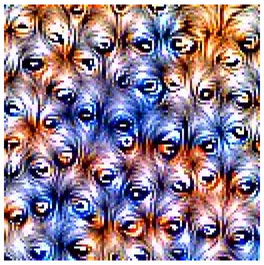

conv2_2/conv2_2

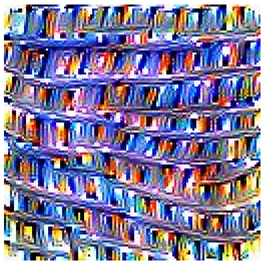

conv3_1/conv3_1


conv3_2/conv3_2

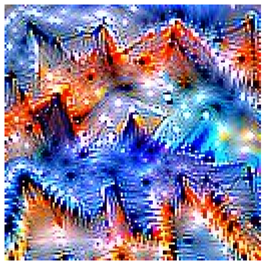

conv3_3/conv3_3

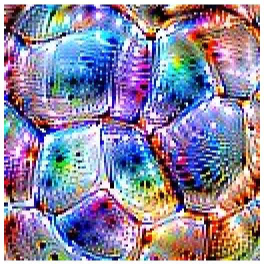

conv4_1/conv4_1


conv4_2/conv4_2


conv4_3/conv4_3

conv5_1/conv5_1


conv5_2/conv5_2


conv5_3/conv5_3

Visualization of the hidden conv layer weights

conv3_1/conv3_1

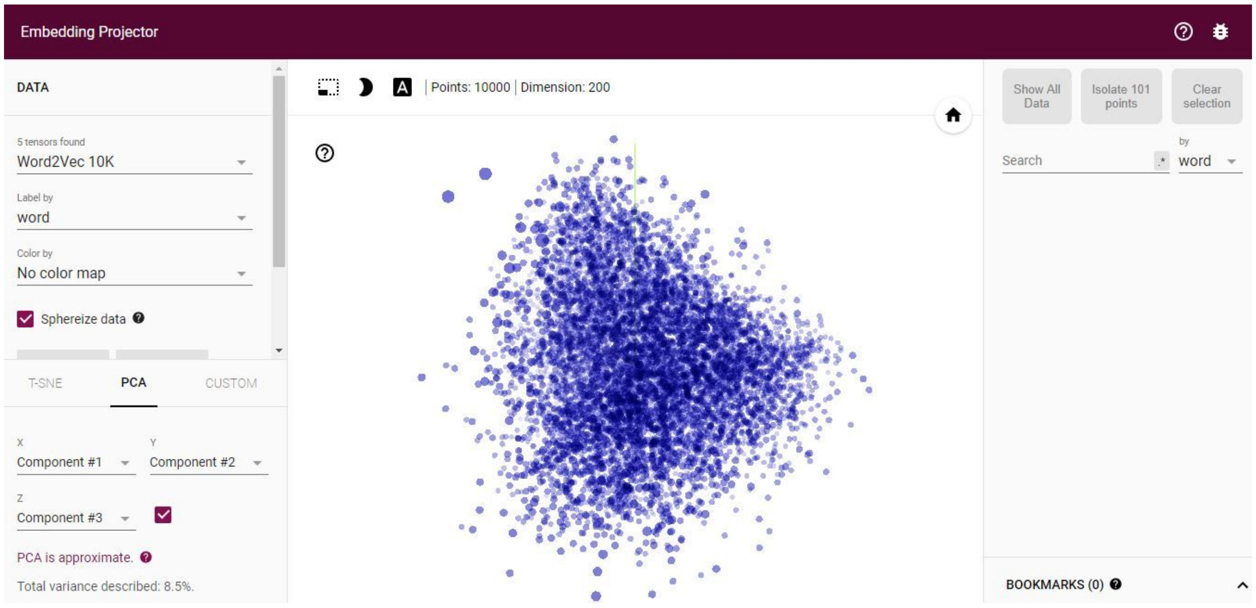
conv3_1/conv3_1

When the photo was passed into the respective layers the image pixel values gets multiplied by the weights in the neural layer which helps in identifying the pattern and triggering the correct output

Text Explanation

We try to identify the text encoder also , we read out the word embedings visualizations from the tensorflow website (https://www.tensorflow.org/tutorials/representation/word2vec)

Word Embedings url :
https://projector.tensorflow.org



5 tensors found: Word2Vec 10K

Label by: word

Color by: No color map

Sphereize data

X: Component #1 | Y: Component #2 | Z: Component #3

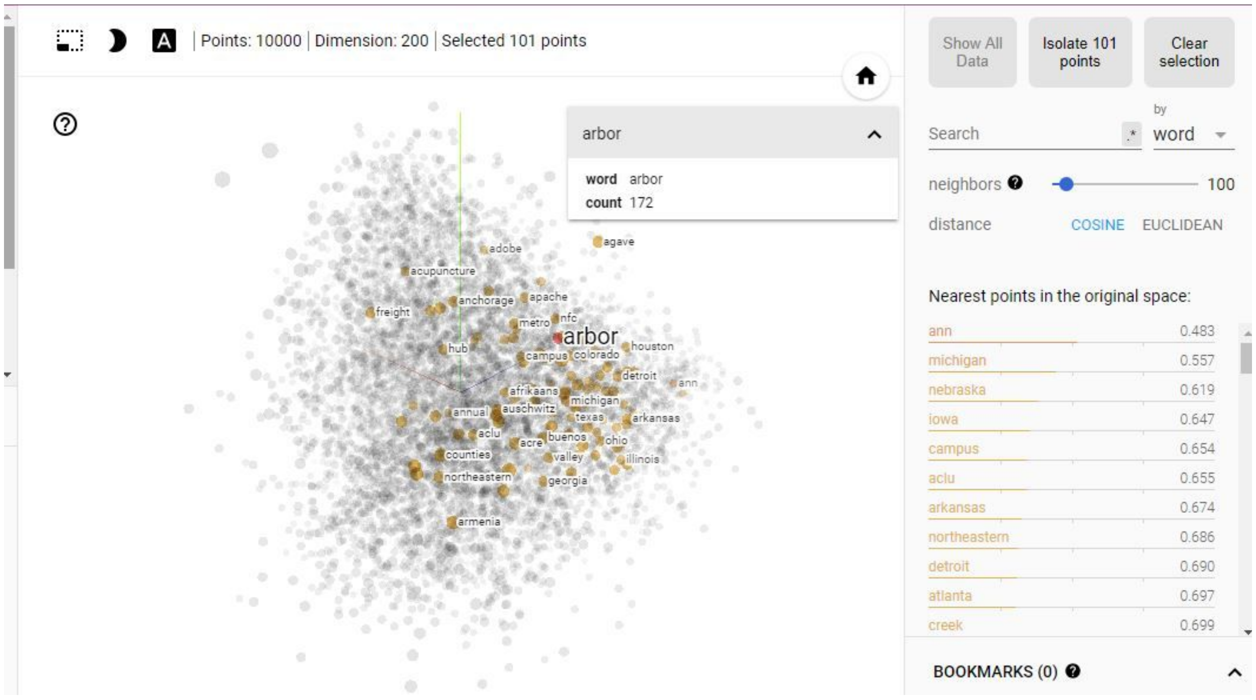
PCA is approximate.

Total variance described: 8.5%.

Points: 10000 | Dimension: 200

by: word

Word vec visualization (courtesy of Google-tensorflow)



Points: 10000 | Dimension: 200 | Selected 101 points

arbor

word arbor
count 172

by: word

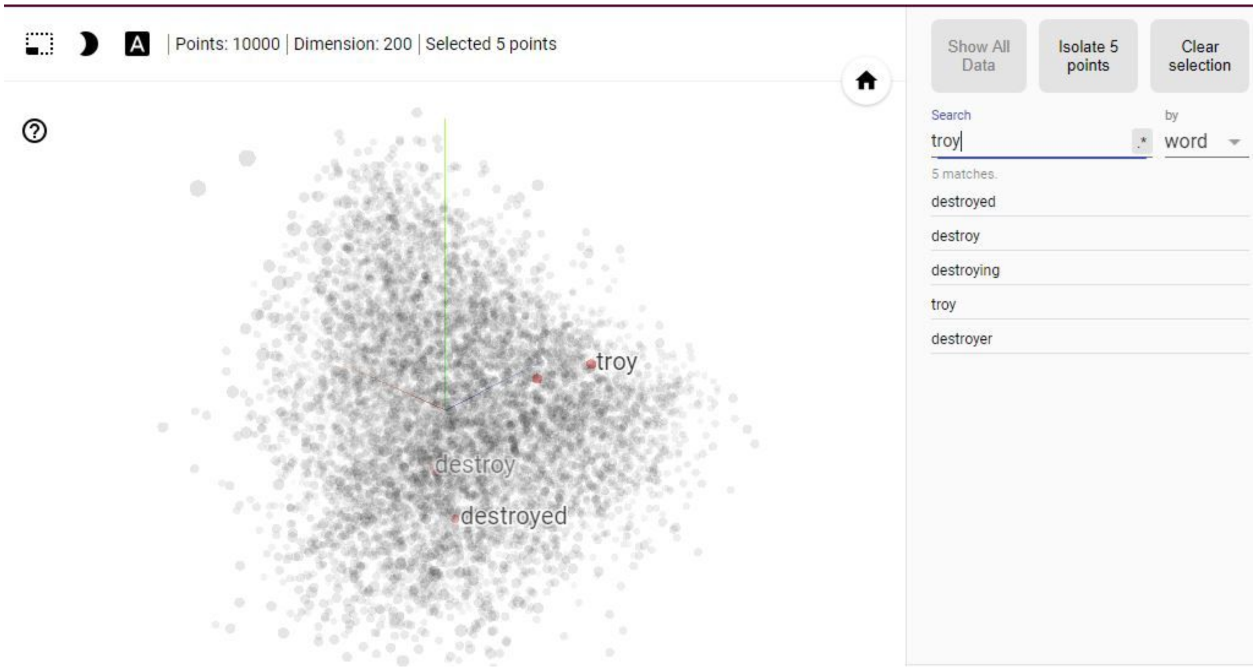
neighbors 100

distance: COSINE EUCLIDEAN

Nearest points in the original space:

ann	0.483
michigan	0.557
nebraska	0.619
iowa	0.647
campus	0.654
aclu	0.655
arkansas	0.674
northeastern	0.686
detroit	0.690
atlanta	0.697
creek	0.699

Nearest neighbors in the word embeding clusters


Points: 10000 | Dimension: 200 | Selected 5 points

Search: troy by word

5 matches.

destroyed

destroy

destroying

troy

destroyer

Search for similar word -troy