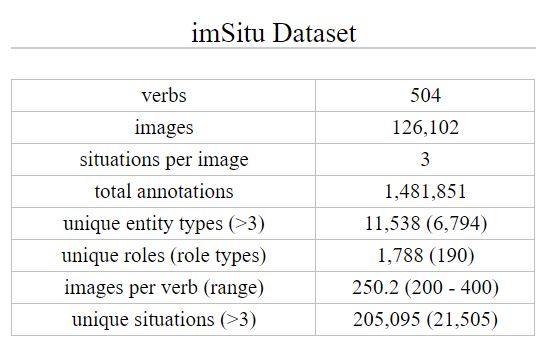
Link to dataset:<http://imsitu.org/>





Task 1: Visualize the data and make the model for verb prediction.

**FUSION NETWORK:** For predicting actions, we use the feature fusion network of [16] which obtained state-of-the-art performance on the HICO dataset [4]. This network (called Fusion in the following) combines local features from detected human boxes and global features from the whole image to make predictions that are then pooled. It defaults to the full image in case no human is detected in the image.

[16]:A. Mallya and S. Lazebnik. Learning models for actions and person-object interactions with transfer to question answering. In ECCV, 2016

Architecture for fusion network network:<https://arxiv.org/pdf/1604.04808.pdf>

> Used weighted cross-entropy for biased data

>Cite CVPR '16 paper for using Imsitu dataset:

@inproceedings{yatskar2016,

title={Situation Recognition: Visual Semantic Role Labeling for Image Understanding},

author={Yatskar, Mark and Zettlemoyer, Luke and Farhadi, Ali},

booktitle={Conference on Computer Vision and Pattern Recognition},

year={2016}

}

> It is checked that every image has size (256, 256, 3).. . The bounding box annotations are not present, so we are not detecting objects and making fusion network without person box

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

import json

imsitu = json.load(open("imsitu\_space.json"))

nouns = imsitu["nouns"]

verbs = imsitu["verbs"]

verbs["clinging"]

# {u'abstract': u'an AGENT clings to the CLUNGTO at a PLACE',

# u'def': u'stick to',

# u'framenet': u'Retaining',

# u'order': [u'agent', u'clungto', u'place'],

# u'roles': {

# u'agent': {u'def': u'The entity doing the cling action',u'framenet': u'agent'},

# u'clungto': {u'def': u'The entity the AGENT is clinging to',u'framenet': u'theme'},

# u'place': {u'def': u'The location where the cling event is happening',u'framenet': u'place'}

# }

# }

nouns["n02129165"]

#{u'def': u'large gregarious predatory feline of Africa and India having a tawny coat with a shaggy mane in the male',

# u'gloss': [u'lion', u'king of beasts', u'Panthera leo']}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

import json

train = json.load(open("train.json"))

train['clinging\_250.jpg']

#{u'frames': [{u'agent': u'n01882714', u'clungto': u'n05563770', u'place': u''},

# {u'agent': u'n01882714', u'clungto': u'n05563770', u'place': u''},

# {u'agent': u'n01882714', u'clungto': u'n00007846', u'place': u''}],

# u'verb': u'clinging'}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

> See how to make loss and accuracy curves in PyTorch

> pre-processing and pre-processing2 is for making the labels and X from the whole image but we are taking only 50 verbs for our ease as it is taking very much time for even the prepeocessing. We are saving the new data to data matrices for fifty verbs folder. Pres-processing3 is for making dictionary\_fifty\_verbs.npy, X\_name.npy, Y\_verbs.npy. Let us make pre-processing4 for making X.

> Normalization: The images are in 8-bit , So deviding by 255. Should be sufficient

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Keras Embedding Layer**

Keras offers an [Embedding](https://keras.io/layers/embeddings/#embedding) layer that can be used for neural networks on text data.

keras.layers.Embedding(input\_dim, output\_dim, embeddings\_initializer='uniform', embeddings\_regularizer=**None**, activity\_regularizer=**None**, embeddings\_constraint=**None**, mask\_zero=**False**, input\_length=**None**)

It requires that the input data be integer encoded, so that each word is represented by a unique integer. This data preparation step can be performed using the [Tokenizer API](https://keras.io/preprocessing/text/#tokenizer) also provided with Keras.

The Embedding layer is initialized with random weights and will learn an embedding for all of the words in the training dataset.

It is a flexible layer that can be used in a variety of ways, such as:

* It can be used alone to learn a word embedding that can be saved and used in another model later.
* It can be used as part of a deep learning model where the embedding is learned along with the model itself.
* It can be used to load a pre-trained word embedding model, a type of transfer learning.

The Embedding layer is defined as the first hidden layer of a network. It must specify 3 arguments:

It must specify 3 arguments:

* **input\_dim**: This is the size of the vocabulary in the text data. For example, if your data is integer encoded to values between 0-10, then the size of the vocabulary would be 11 words.
* **output\_dim**: This is the size of the vector space in which words will be embedded. It defines the size of the output vectors from this layer for each word. For example, it could be 32 or 100 or even larger. Test different values for your problem.
* **input\_length**: This is the length of input sequences, as you would define for any input layer of a Keras model. For example, if all of your input documents are comprised of 1000 words, this would be 1000

> The set of nouns N is derived from WordNet.

>the RNN tries to predict the noun entity associated with the first semantic role in the arbitrarily selected but fixed ordering, and so on, until a noun entity is predicted for each semantic role for that verb

>**return\_sequences**: Boolean. Whether to return the last output in the output sequence, or the full sequence.

>**TimeDistributed:**

Consider a batch of 32 samples, where each sample is a sequence of 10 vectors of 16 dimensions. The batch input shape of the layer is then (32, 10, 16), and the input\_shape, not including the samples dimension, is (10, 16).

You can then use TimeDistributed to apply a Dense layer to each of the 10 timesteps, independently:

> We use embedding layer as the input the lstm should be a 3D input (batch\_size , no. of elements, vector corresponding to those elements). So, in order to convert our vectors([48],[54],...) into vector form, we use embedding layers

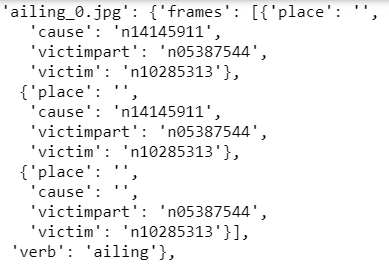
> Pre-processing5 is for pre-processing of data for prediction of nouns

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

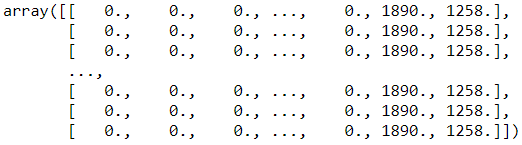
The .npy files and the predictions looks like this:-

**The data-set and the .npy files.**

1. **list\_of\_actions:** Contains the list of the actions
2. **actions\_to\_ind:** Can access using actions\_to\_ind.item(). It contains 39 actions and their corresponding indices corresponding to all the images that we have. Eg. goal , item, tool
3. **ind\_to\_actions:** It contains the index to actions dictionary for the actions
4. **dictionary\_fifty\_verbs:** Contains the dictionary of the random fifty verbs that we selected from the data-set
5. **ind\_to\_verbs:** Contains the index to verbs transformation
6. **verbs\_to\_ind:** Contains the verbs to index transformation
7. **list\_of\_nouns:** It consists of dictionary of all the 2310 nouns corresponding to all the actions in all the images in our data-set
8. **nouns\_to\_ind:** Contains the nouns to index transformation
9. **Ind\_to\_nouns:** Contains the index to nouns transformation
10. **X\_name:** Contains the names of all the images in a sequential order that is used in making X and corresponding Y
11. **X\_noun:** Contains the three different sets of possible nouns corresponding to the images we have. As we are working on only one

****

1. **X\_noun\_one\_noun:** Contains only one set of nouns with their action sets as we are working with oly one
2. **X\_noun\_one\_noun\_every\_action:** Contains only one set of nouns with same action set for every image as in this research paper, we are taking a fixed sequence of actions and then predicting the corresponding nouns.
3. **Y\_nouns:**  It is the uncategorical output for the second architecture to predict nouns in our project. It has a shape of (13379, 39)



1. **Y\_nouns\_categorical:** It is same as above except the categorical version for each noun to corresponding action. It is the end output of our corresponding second network.
2. **X:** Corresponding to X\_name, it is the matrix of all the images having shape (13379, 256, 256, 3)
3. **Y\_verbs:** Corresponding to the X or X\_name, it is the un\_categorical matrix of verbs, which act as output in case of fusion network and one of the inputs in case of other network

Now let us see the predictions:

1. The fusion network was trained for approximately 20 epochs with learning rate as 1e-4 for the first 8 epochs and 1e-5 for the last two epochs and gave an accuracy of around 80% for the test set
2. The other network for prediction of nouns went good and gave a test accuracy of 99% and was trained in 2 epochs with learning rate as 1e-4