Video Description using Deep Learning

Summer Undergraduate Research Award

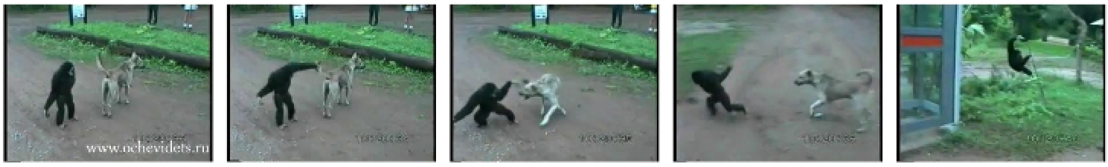
Suyash Agrawal
2015CS10262
Computer Science
CGPA: 9.87
Mob: 9717060183
cs1150262@iitd.ac.in

Madhur Singhal
2015CS10235
Computer Science
CGPA: 8.69
Mob: 9540972599
cs1150235@iitd.ac.in

Supervisor:
Subhashis Banerjee
Professor
Department of CSE
suban@cse.iitd.ac.in
IIT Delhi

1 Introduction

Video Description is the process of discovering knowledge, structures, patterns and events of interest in video data and generating a description of them in natural human language. Video Description is an open problem in computer vision and currently the predominant source of video description is manual human work.
Video Description has wide variety of applications. It can help visually impaired people "see" the world by describing the scene around them. It has also use in automated surveillance by analyzing the videos in real time and reporting malicious or unusual activities. It can also be used to efficiently index large video databases based upon their content for ease of accessibility.



Description: A monkey pulls a dog's tail and is chased by the dog.

Figure 1: Sample Video Description

In this project we created an end to end Neural Network pipeline which generates an english sentence from a given input video clip. This pipeline is inspired by the recent success of Convolutional Neural Networks[10] in image analysis and that of Sequence to Sequence Recurrent Neural Networks in Machine Translation[13]. Our network can be thought of as translating between a video clip and an english sentence, wherein individual frames of the video are "words" in the video language. Since our network is end to end we have a combined loss function for the whole network based on the sentence returned by the network and the desired sentence. Finally we use an Adam Optimizer and backpropagation through the network to optimize the weights. In practice the Convolutional Neural Networks are too big to be trained using the limited video clip datasets we have and thus we use a CNN that has already been trained on the separate task of image classification for which there is a huge dataset called ImageNet.

2 Past Work

Much of the prior work in this field is focused on generating natural language descriptions from images. Recently the use of an end to end deep neural network architecture which takes an image as input and generates an english description has been shown

to give excellent results. Vinyals et al.[3] gave a end to end pipeline for image description task which produced excellent results. Wu et al.[13] gave a end-to-end learning approach for automated language translation. Subhashini et al.[2] used a two layered LSTM[7] structure for video translation on MSVD dataset.
We extend this approach to a Video Description network which takes in a sequence of frames as input and generates an english description of the actions happening in the video. For this we use an encoder-decoder framework widely used in Machine Translation[13], which maps a variable length input to a variable length output.

3 Data

The data used is a crucial factor in the effectiveness of a complex neural network like ours. We primarily used the following two datasets for training and validation purposes.

1. **Microsoft Research - Video to Text (MSR-VTT)** [12]: The dataset contains 41.2 hours and 200K clip-sentence pairs in total, covering the most comprehensive categories and diverse visual content, and representing the largest dataset in terms of sentence and vocabulary.

2. **Montreal Video Annotation Dataset (M-VAD)**: The M-VAD movie description corpus is another recent collection of about 49,000 short video clips from 92 movies. It is similar to MPII-MD, but only contains AD data and only provides automatic alignment.

Both of these datasets consist of short Youtube video clips (average length near 20 seconds) and english sentences (average length near 12 words) describing the clips. Most of the video clips also have a synchronized audio clip which we utilize in some of our experiments. Empirically we found out that these datasets were big enough for our network to learn proper English language sentence structure and grammar automatically, and the sentences we return are almost always grammatically correct. The datasets also identify a wide host of objects and actions which make the network applicable in general situations.
The main problem we faced with these datasets was that they were not properly sampled and thus there was an inherent bias towards the types of videos seen on Youtube as opposed to those encountered in the real world. For example, the datasets had a large amount of videos of video games which often involve rapidly changing frames and our network began to associate things like car crashes to video game clips.

4 Methods

4.1 Overall Approach

Our approach to video description was motivated by the recent successful approach of using Encoder-Decoder system in Image Description. Our pipeline consisted of a CNN network that extracted features from the individual video frames, one LSTM network which encoded these features into a fixed length feature vector[5] and another LSTM network which decoded the feature vector into a natural language description.

4.2 Word Embeddings

To generate sentences we need a vocabulary, we use a vocabulary which consists of about 13000 words as well as punctuation marks. We also add two special tokens ⟨Start⟩ and ⟨End⟩ to the vocabulary to represent the beginning and end of a sentence. Since a simple one-hot representation of a word is very sparse and unstructured, we use a fully connected layer (called the "embedding layer" to convert a word to a dense vector space. We experimented with different embedding methods, including random initialization of a trainable embedding layer and using pretrained GloVe[14] embeddings. Empirically we found out that using pretrained embeddings gives almost identical results to randomly initializing the embedding layer and training it within our pipeline.

At the output stage we need another matrix to decode the outputs to words and we used another fully connected layer which gives an output of length vocabulary size with each element representing the probability of that word being generated. Note that while it may seem reasonable to use the same weights in the embedding and decoding layers as first sight, such an approach gives poor results in our experiments since the vector space of embedding outputs and the vector space of decoding inputs are different and unrelated to each other.

4.3 Video Feature Extraction

We used Convolutional neural networks for extracting features from video frames. In recent times CNNs have shown amazing results in object detection in images and thus they form a natural choice in extracting features from images. Also, since videos are a sequences of images, we down-sample video frames at a fixed frame rate and then individually extract features from each video frame and concatenate them to get the feature vector of the video.

We used Inception V4[16] as our choice of CNN for video feature extraction as it has shown the best results in ImageNet challenge and is thus more likely to give better feature extraction from image frames. We also experimented with audio features in the video. Specifically we looked at the MFCC features[15] of the audio and we processed these features along with image features in our encoder network to get better representation of the video. But we later abandoned this approach as it did not show any significant improvement in the results and also added audio dependency to our network which would have made this un-usable in the case of video surveillance and other areas where audio is not available.

4.4 Video Feature Encoding

We used a LSTM based network for encoding feature vectors of individual video frames into a single feature vector for the whole clip. This approach is largely inspired by similar approach in machine translation which also use LSTMs for encoding decoding between languages. Here we used the feature vector of video frames as inputs to LSTM units and the final hidden state of the LSTM network as the video encoding.
We experimented with various types of encoding networks like:

- Single layer LSTM network
- Multi layered LSTM network (2-5 layers)
- Residual multi layered LSTM network
- Encoder network with attention model

Apart from the above, we also experimented with dropout in between layers and also experimented with audio features concatenated with video features.

4.5 Decoding to Natural Language

The previous stage gives us a fixed length feature vector which represents the whole video clip. We use this feature vector as the initial hidden state of our decoder LSTM network. After this we feed words (embedded in a vector space as described above) one by one into the LSTM, which gives the next word. There is a special ⟨Start⟩ token which represents the beginning of a sentence and is always the first one to be given as input. The output word is deduced from the output of the LSTM network using a fully connected layer with softmax activation, which results in a vector of length equal to the size of our vocabulary and where each element can be interpreted as the probability of that word coming next in the sentence.
Initially we used a greedy approach to sentence formation by taking the word with maximum probability at the output of each step, adding it to our sentence and feeding

it as input in the next step. This approach often resulted in convoluted and inaccurate sentences, since some degree of look-ahead is needed for proper sentence structure. Finally we used Beam Search Decoder[3] to provide us with a certain amount of look-ahead, and observed our sentences and results both improve dramatically. A Beam Search is a compromise between the totally greedy approach which can be short-sighted and the breadth-first search approach which is not computationally tractable. Instead of choosing the best sentence at the end of each step, here we keep a 'beam' of sentences (three sentences in the beam were used in practice). At each step, for each sentence in the beam we consider the top three predictions for the next word and make nine sentences out of those, then we filter down from these nine sentences to three sentences on the basis of the combined probability for the original sentence and the added word.

4.6 Training Phase

In training phase, we had video-caption pairs in our dataset. Using this we first encoded the video into sequence of feature vectors and after encoding these frames using encoder network we pass the hidden state into the decoder LSTM. In this we first pass the start token ⟨Start⟩ with the hidden state corresponding to the last encoder LSTM unit. The decoder LSTM unit gave the probability of each word in our vocabulary being the next word in a vector of the size of our vocab. At each step we fed the next from correct description into the LSTM unit and recorded its output. Our loss is the sum of the negative log likelihood of the correct word at each step as follows:

$$L(V, S) = -\sum_{t=1}^{N} \log p_t(S_t)$$

Where,
V = Input Video
S = Correct caption of video V. It is a sequence of words $(S_0, S_1, \ldots, S_N)$
$L(V, S)$ = Loss corresponding to video caption pair (V,S)
$p_t(S_t)$ = Probability of word S_t in the output of LSTM unit at time t.

The above loss is minimized w.r.t. all the parameters of the Encoder LSTM unit, Decoder LSTM unit, the top layer of the image embedder CNN and word embeddings. The parameters we used for our training are as follows:

Number of frames per video = 100
Image dimensions for input = 299 x 299
Maximum caption length = 20

CNN last layer dimension = 1x1536
Image embedding size = 2000
Word embedding size = 1000
Hidden LSTM size = 1000
Number of LSTM layer = 2
Adam learning rate = 10^{-4}
Initialzation of weights = Random in range $(-0.1, 0.1)$
Dropout between LSTM layers = 0.5

4.7 Inference Phase

We used multiple approaches for inference phase for generating a sentence given a video.
The first one was **Sampling** where we just sample the first word according to p1, then provide the corresponding embedding as input and sample p2, continuing like this until we sample the special end-of-sentence token ⟨End⟩ or some maximum length.
The second one is **Beam Search**: iteratively consider the set of the k best sentences up to time t as candidates to generate sentences of size t + 1, and keep only the resulting best k of them. We used the Beam Search approach in the following experiments, with a beam of size upto 10. Using a beam size of 1 (i.e., greedy search) did degrade our results.

4.8 Dealing with Over-fitting

The main problem we had to deal with throughout the project was overfitting. This is a phenomenon where the Neural Network becomes attuned to the training data itself, rather than learning patterns that can generalize easily to new inputs. This is akin to a student memorizing answers to questions from a guide book and not being able to answer new questions easily. A characteristic indication of overfitting is that the training loss is much less than the validation loss. The main factor influencing overfitting is the *complexity* of the model which is basically the number of trainable parameters in the model and the training method employed. A neural network which is too complex for the problem it is designed to solve will utilize the remaining degrees of freedom to provide lower loss in response to specific training data samples. Also if the network is trained too long or with a poorly designed optimizer it may become specific to training data.

To deal with overfitting we experimented a lot and found the following approaches to be very useful.

1. **Adjusting Complexity :** Reducing our LSTM encoder decoder network from a four layer network to a two layer network was very helpful in reducing overfitting. We also reduced the number of parameters is the fully connected layers by reducing the word embedding size.

2. **Dropout :** Dropout layers take a vector as input and return a vector where each element is either zero with a certain probability or the original value. Dropout helps with overfitting because even when the same input is fed multiple times into the network the parameters influencing the output may be different each time. We used a Dropout Wrapper on our LSTM cells which makes a certain fraction (we used 0.4) of the LSTM outputs zero and found that it reduced overfitting.

3. **Proper Training :** The choice of optimizer is important and thus we decided to use Adam Optimizer which maintains a separate learning rate for each parameter of the model and keeps reducing the rate with time. This keeps the parameters from oscillating wildly and helps with overfitting. We also took care to train the model only as far as it reduced the validation loss and stop when the validation loss started to increase which indicates overfitting.

With all these implemented, overfitting was a much smaller problem then before and our training loss was mostly near our validation loss.

4.9 Other experiments

We tried a lot of different approaches to improve the model, some of the promising ones are the following.

1. **Attention Mechanism:** In any video clip it can be observed that not all frames are equally important with respect to the goal of generating a description. The earlier approach weighs all frames equally since everyone has a chance of modifying the hidden state of the encoder LSTM equally. An attention mechanism allows us to weigh some frames of the video more than others and to learn how to generate the weights from the video itself. We didn't observe extensive improvement in the dataset results with this modification. One reason could be that our datasets only consisted of short video clips and was already downsampled wrt time and thus applying attention may have lowered weightage of some important frames.

2. **Audio Features:** In the encoding process, instead of just taking feature vectors derived from video frames we also tried to concatenate feature vectors obtained from the audio near that particular frame. For this purpose we used

MFCC (Mel-frequency cepstral coefficients) which is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum[15]. Here too we observed not much improvement in the results, even when we weighed the audio features equal to the video features. This indicates that even though the audio features are helping with learning but the information contained in the two is complimentary ie one can be substituted for the other. Knowing this and the fact that audio feeds are rare in many potential applications we decided to remove this feature from the final project.

3. **Changing Loss Function:** The loss function we use is a tad simplistic since it depends a lot on word positioning. So for example if the desired output is "A man catches a ball" and the network outputs "A ball is caught by a man" our loss function will give a very high loss even though the two sentences seem to be same. To improve this we thought of using n-grams to compare sentences like in the BLEU algorithm[13]. Unfortunately even a simple implementation of this was not able to run well using the GPU and thus was basically useless since it would stretch our training time from 10 hours to a week at least.

5 Results

The evaluation is done according to different metrics popular in the community. Below we have listed some of our models and their performance and the state of the art models and their performance. As can be seen our model performs near the state of the art and even beats them in one metric.

Model	Bleu@1	Bleu@3	Bleu@4	Meteor	CIDEr	ROUGEL
Our Final Model	0.782	0.498	0.381	0.278	0.457	0.59
Using GloVe Embedding	0.785	0.499	0.380	0.276	0.445	0.590
Using Attention Model	0.761	0.469	0.354	0.260	0.397	0.575
Without Beam Search	0.774	0.476	0.358	0.274	0.421	0.579
VideoLAB	-	-	0.391	0.277	0.441	0.606
v2t_navigator	-	-	0.408	0.282	0.448	0.609

We also did some manual testing by feeding random youtube videos and found that our model gave sensible and reasonable results.

6 Summary

In our SURA, we have successfully created an end to end pipeline that, given a video, automatically generates a reasonable description in plain English. It is based on a convolution neural network that encodes the video into a compact representation, followed by a recurrent neural network that encodes this representation into a fixed length vector and ending with a recurrent neural network the decodes this representation into the corresponding sentence. The model is trained to maximize the likelihood of the description given the video as input. We also experimented with various techniques like attention models, residual networks and found mixed results. We further found that the results strongly depended on the amount of training data available and that as the size of the available datasets for video description increases, so will the performance of this model. Furthermore, it will be interesting to use this model in areas like automated video surveillance once the dataset for the same become publicly available.
Our code for this project can be found at: https://github.com/ozym4nd145/SURA

7 Uses and applications

- Assisting visually impaired people to get description of their surroundings, thus enabling them to "see".
- Very useful for automated surveillance and theft detection by being able to analyze large amounts of data which is unfeasible to be done by humans.
- Allowing content based video retrieval by describing the contents of video in textual format which is indexable by web crawlers.
- This can also be used to detect catastrophic events through security cameras like fire breakout, murder etc.
- This project can also be applied in helping robotic vision as this project basically allows one to understand what is happening in the video and thus robots will be able to get a "true" sense of their surroundings.

We tried to explore the use of this technique in the area of automated video surveillance. But the major setback we faced was the lack of training data available for doing such a task. We could not find any reasonably sized dataset for training our system in video surveillance and thus could not further experiment with it. We did try our model in a normal room environment and it was successfully able to give appropriate description like: " Person sitting in a chair", "People moving in the area", "Person showing the working of a computer" etc.

8 Appendix

8.1 Deep Learning

Deep Learning is a branch of machine learning in which multiple parameter based models are used in series. In a deep network, there are many layers between the input and output, allowing the algorithm to be executed in multiple processing steps, composed of **multiple linear and non-linear transformations**. At each layer, the signal is transformed by a processing unit, like an artificial neuron, whose parameters are **'learned'** through training. Deep Learning has been shown to excel in tasks where the goal is to find **intuitive** patterns in the data.[8] In particular, in the field of Computer Vision, deep networks are increasingly used to extract **feature descriptions and inter-relationships between features** from images.[10]

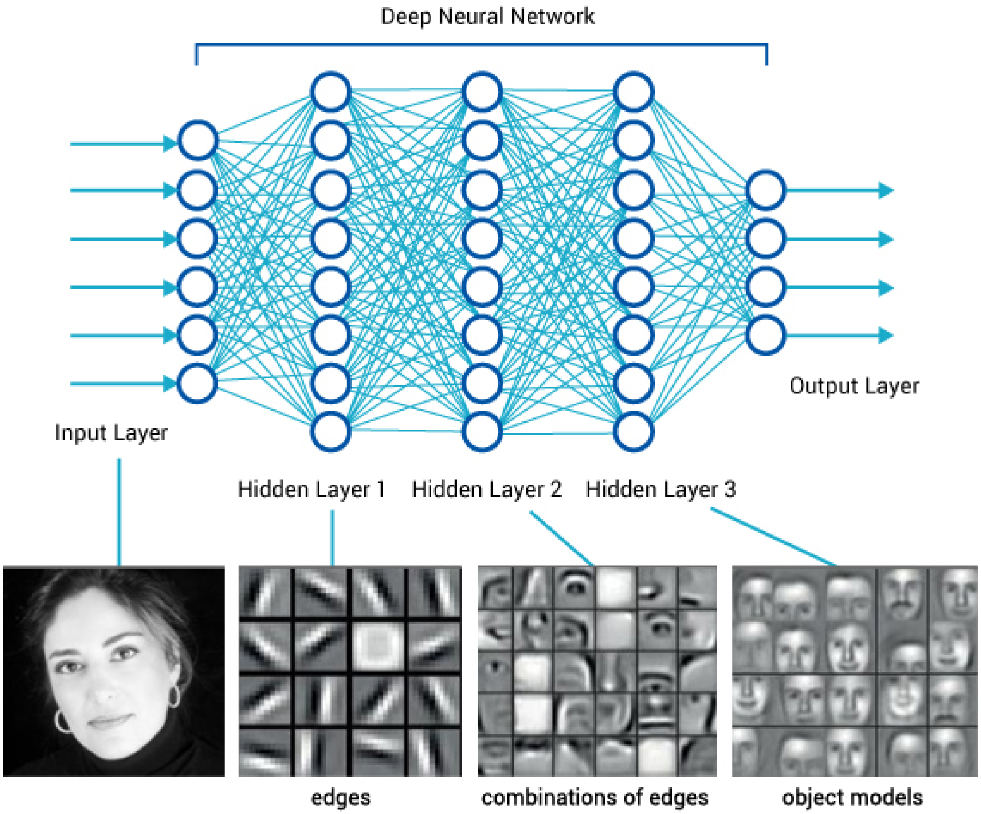


Figure 2: Illustration of Deep Learning as applied to Vision

8.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN, or ConvNet) are a type of feed-forward artificial neural network in which the connectivity pattern between the neurons is inspired by the organization of the animal visual cortex. Individual cortical neurons respond to stimuli in a restricted region of space known as the **receptive field**. The receptive fields of different neurons partially overlap such that they tile the visual field. The response of an individual neuron to stimuli within its receptive field can be approximated mathematically by a **convolution operation**. A Convolutional Neural Network consists of the following layers.[3]

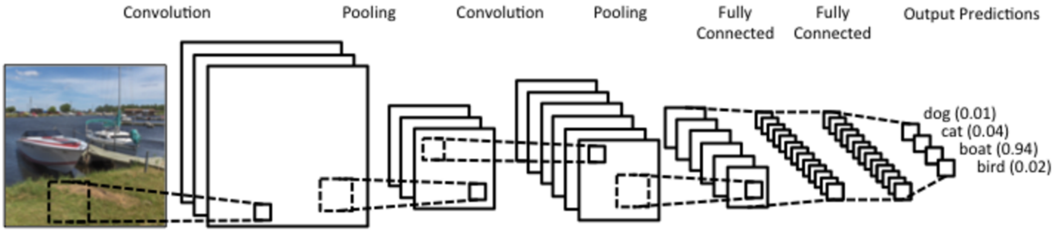


Figure 3: A Typical Convolutional Neural Network

8.2.1 Convolutional Layer

The convolution layer is the core building block of a CNN. The layer's parameters consist of a set of **learnable filters** (or kernels), which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter.[10] As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.

8.2.2 Max Pooling Layer

It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture. Its function is to **progressively reduce the spatial size** of the representation to reduce the amount of parameters and computation in the network, and hence to also control over-fitting. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the max operation.[10]

8.2.3 Fully-Connected Layer

Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have **full connections to all activations** in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.[11] Thus output of the fully connected layer is a vector with elements representing the 'probability' (not in a strictly statistical sense) of the image containing specific objects or actions.

8.3 Long Short Term Memory Networks

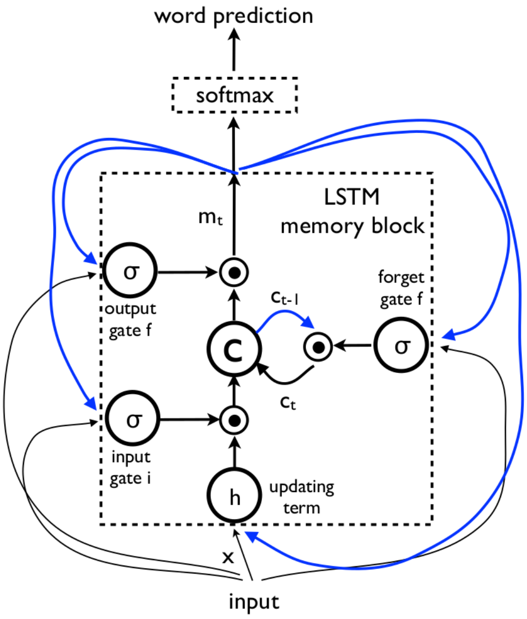


Figure 4: A single LSTM unit

Long Short Term Memory Networks are a type of Recurrent Neural Networks. These networks are based upon recursion, so that variable length inputs can be handled easily and sequential information can be processed with better results. LSTM's are specifically used for making RNNs learn long term patterns since traditional RNNs tend to **favour short term temporal dynamics**. It can be difficult to train traditional RNNs to learn long-term dynamics, likely due in part to the **vanishing and exploding gradients problem** that can result from propagating the gradients down through the many layers of the recurrent network, each corresponding to a particular

time step[7]. LSTMs provide a solution by incorporating memory units that explicitly allow the network to learn when to “forget” previous hidden states and when to update hidden states given new information[4].
Below we list the main equations governing the behaviour of the LSTM networks.

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \tag{1}$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \tag{2}$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \tag{3}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \tag{4}$$

$$h_t = o_t \circ \sigma_h(c_t) \tag{5}$$

The symbol meanings are:
$\mathbf{x_t}$: Input vector
$\mathbf{h_t}$: Output vector
$\mathbf{c_t}$: Cell state vector
W, **U** and **b**: Parameter matrices and vector
$\mathbf{f_t}$: Forget gate vector. Weight of remembering old information.
$\mathbf{i_t}$: Input gate vector. Weight of acquiring new information.
$\mathbf{o_t}$: Output gate vector. Output candidate
$\sigma_\mathbf{g}$, $\sigma_\mathbf{c}$ and $\sigma_\mathbf{h}$: Activation functions

8.4 Training Deep Neural Networks

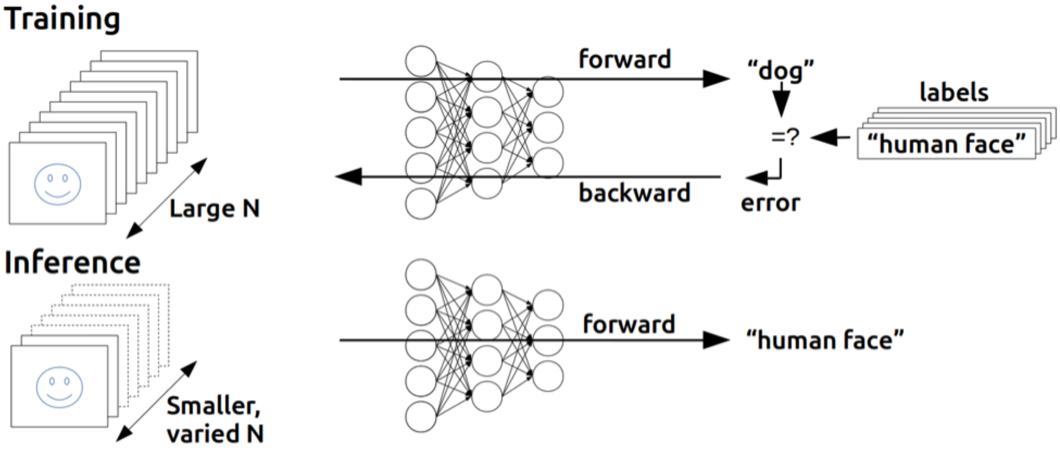


Figure 5: Training and Inference Processes

A Deep Neural Network is at it's core a parameter based function. All of these parameters are **trained** automatically from inputs and expected output tuples (training data). The training process revolves around minimizing a particular cost function using methods like **Stochastic gradient descent**. The input is given to the network in a feed forward fashion and the parameters are modified from the last layer to the first **(Backpropagation)**. Neural Networks, by design, require huge amounts of training data and take a large time to get trained. For some perspective, most current state of the art image classifiers have > 100 million parameters and are trained on more than 1.2 million images.

8.5 Finetuning

Fine-tuning a network is a procedure based on the concept of **transfer learning**. We start training a CNN to learn features for a broad domain with a classification function targeted at minimizing error in that domain. Then, we replace the classification function and **optimize the network** again to minimize error in another, more specific domain. Under this setting, we are transferring the features and the parameters of the network from the broad domain to the special one.[9] In our project we will need to use the pre-trained image classification models to actually decode individual frames of the video, thus we are planning to **fine-tune those models with respect to the output of our LSTMs**.

References

[1] Subhashini Venugopalan *Natural Language Video Description using Deep Recurrent Neural Networks*, 2015.

[2] Venugopalan, Subhashini and Rohrbach, Marcus and Donahue, Jeff and Mooney, Raymond and Darrell, Trevor and Saenko, Kate *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015

[3] Oriol Vinyals and Alexander Toshev and Samy Bengio and Dumitru Erhan *Show and Tell: A Neural Image Caption Generator* 2014.

[4] Wojciech Zaremba, Ilya Sutskever *Learning to Execute* ICLR 2015

[5] Li Yao, Atousa Torabi, Kyunghyun Cho, Nicolas Ballas, Christopher Pal, Hugo Larochelle, Aaron Courville *Describing Videos by Exploiting Temporal Structure* ICCV 2015

[6] Aishwarya Agrawal, Jiasen Lu , Stanislaw Antol, Margaret Mitchell, C. Lawrence Zitnick, Dhruv Batra, Devi Parikh *VQA: Visual Question Answering* 2016

[7] Jeff Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko, Trevor Darrell *Long-term Recurrent Convolutional Networks for Visual Recognition and Description* 2016

[8] Bengio, Yoshua; LeCun, Yann; Hinton, Geoffrey *Deep Learning* 2015

[9] Angie K. Reyes, Juan C. Caicedo and Jorge E. Camargo *Fine-tuning Deep Convolutional Networks for Plant Recognition* LifeCLEF 2015

[10] Andrej Karpathy *CS231n Convolutional Neural Networks for Visual Recognition* http://cs231n.github.io/convolutional-networks/

[11] Santanu Chaudhury, Anupama Mallik, Hiranmay Ghosh *Multimedia Ontology: Representation and Applications*

[12] Jun Xu, Tao Mei, Ting Yao, Yong Rui *MSR-VTT: A Large Video Description Dataset for Bridging Video and Language* CVPR 2016

[13] Wu et al *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* CoRR, vol. abs/1609.08144 (2016)

[14] Jeffrey Pennington, Richard Socher, Christopher D. Manning *GloVe: Global Vectors for Word Representation* 2014

[15] Hinton, Geoffrey, et al. *Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups.* IEEE Signal Processing Magazine 29.6 (2012): 82-97.

[16] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Alex Alemi *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning* abs/1602.07261