LABORATORY ORIENTED PROJECT

FINAL REPORT

Finger Spelling to Text Translation Using Deep Learning

Prof. In Charge: Dr.Surekha Bhanot

SAGNIK MAJUMDER | 2014A8TS0464P

Birla Institute of Technology and Science, Pilani - PIlani Campus

Contents

A. INTRODUCTION

‘Machine Translation’(MT) is in great need nowadays and there is a lot of work on it all around the globe. MT deals with the task of translating text from one input language into proper semantic text of another language i.e. meaningful text in an output language. MT is generally associated only with text to text translation keeping its purpose true to the actual meaning of translation i.e. translation from English to German etc. But, in this work, I give a new dimension to translation. I introduce here translation from a gesture or sign based language into a textual language. The scope of this work today is quite promising keeping in mind the fact that not much has been done in this particular domain.

The prospects of this work are many -
1. Sign or gesture based language, popularly known as sign language, is a relatively slow language in comparison to other verbal or textual language because of the time taken to produce signs. So, a good text generation model can be used to complete a sentence or generate further text from some starting primer in some sign language.This can speed up communication and can immensely help both the speaker.

2. When the speaker is using sign language, the listener may not always know it and may need the help of a translator for each sentence the speaker speaks. I can do away with the translator if I can use such a text generation model which can generate text and convey the meaning the speaker is trying to put through. The text generation can be used to complete the words from just a few initial characters. This can be rendered useful in a lot of basic conversation i.e. in case of the most common words that the verbally challenged speaker might use in his/her day to day communication.

There are many other uses of such a system which I will temporarily skip for the reason of brevity.

The fundamental tools behind the building of such a model are Convolutional Neural Networks (CNN) and Sequential Neural Networks (SNN). The CNNs are used to convert a particular sign or gesture into its corresponding character in some other verbal/textual language. The CNNs are basically used to generate the primer or the introductory text that will be used for further text generation. The SNNs take this primer text as the input and generate further text to complete the word or the sentence using Recurrent Learning. So, the model, in fullness, employs a CNN to convert images into

characters, to be used as the text primer, and the SNN will use the primer text to generate further text to complete a word or a sentence.

A Brief Overview of Deep Learning

Deep Learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks. It refers to artificial neural networks that are composed of many layers. It is just another name for artificial neural networks (ANN for short) but in a more refined and easier avatar. They have been existing for more than 40 years. In 2000's, NVIDIA accelerated ANN by bringing their chips for scientific computing and from then onwards, use of neural networks has picked up.

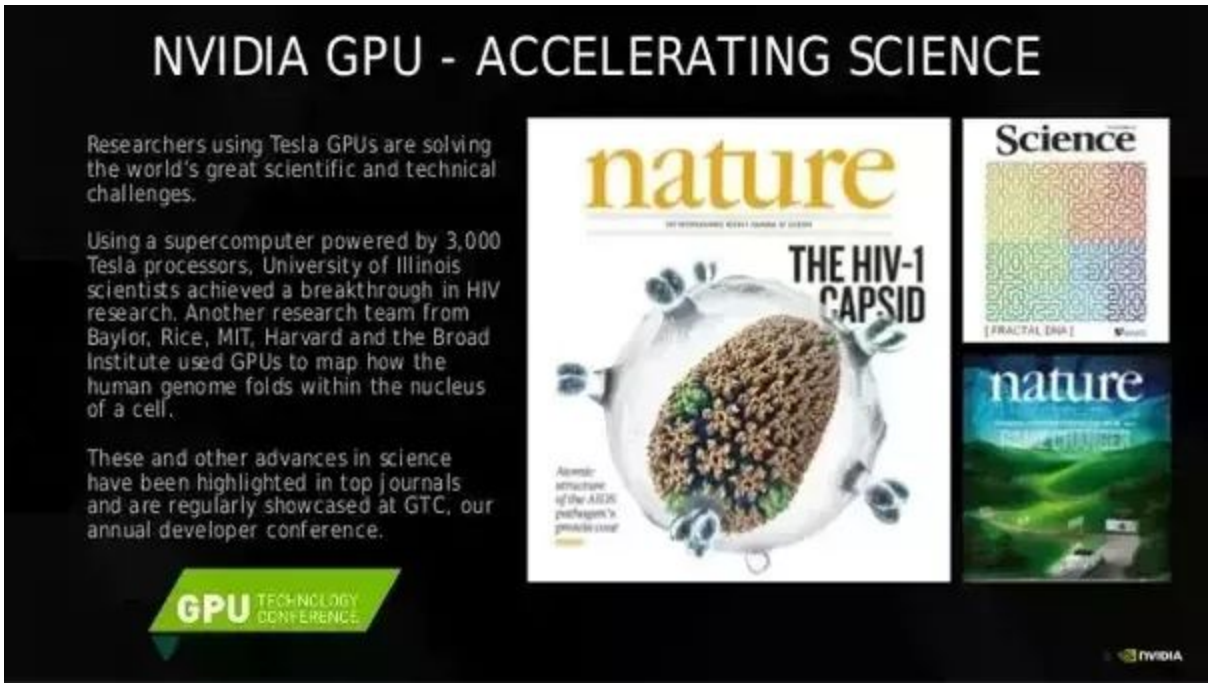


Figure 1: GPUs Have Helped Deep Learning Advance

It's a growing trend in ML due to some favorable results in applications where the target function is very complex and the datasets are large. For example in Hinton et al. (2012), Hinton and his students managed to beat the status quo prediction systems on five well known datasets: Reuters, TIMIT, MNIST, CIFAR and ImageNet. This covers speech, text and image classification - and these are quite mature datasets, so a win on any of these gets some attention. A win on all of them gets a lot of attention.

In machine learning and cognitive science, artificial neural networks (ANNs) are a family of models inspired by biological neural networks (the central nervous systems of animals, in particular the brain) and are used to estimate or approximate functions that

can depend on a large number of inputs and are generally unknown. Artificial neural networks are generally presented as systems of interconnected "neurons" which exchange messages between each other. The connections have numeric weights that can be tuned based on experience, making neural nets adaptive to inputs and capable of learning.

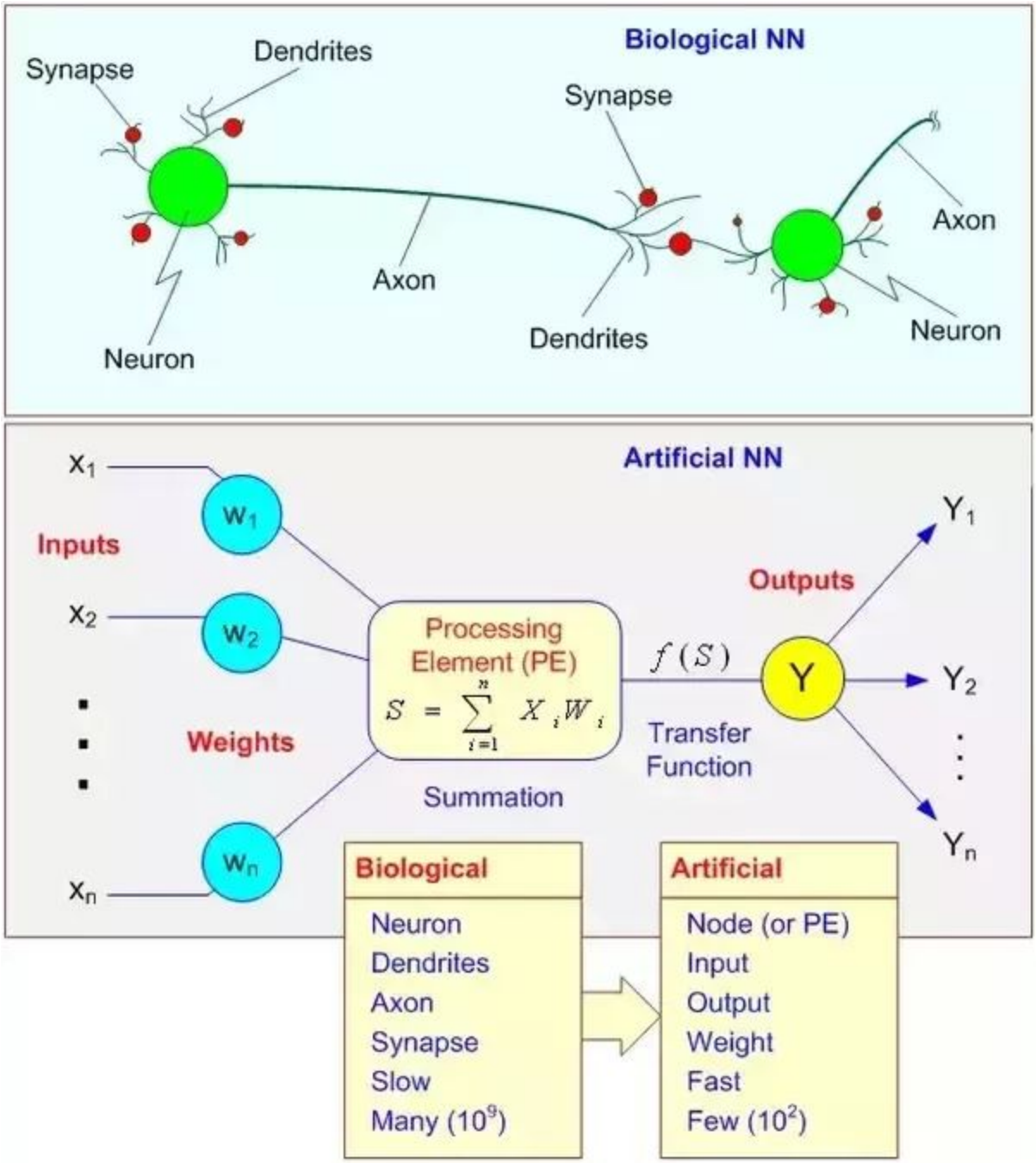


Figure 2: A Comparison Between Biological Neurons and an ANN in Deep Learning

Why 'Deep Learning' is called deep? It is because of the structure of ANNs. Earlier 40 years back, neural networks were only 2 layers deep as it was not computationally

feasible to build larger networks. Now it is common to have neural networks with 10+ layers and even 100+ layer ANNs are being tried upon.

One can essentially stack layers of neurons on top of each other. The lowest layer takes the raw data like images, text, sound, etc. and then each neuron stores some information about the data they encounter. Each neuron in the layer sends information up to the next layers of neurons which learn a more abstract version of the data below it. So the higher one goes up, the more abstract features the layers learn. One can see in the picture below has 5 layers in which 3 are hidden layers.

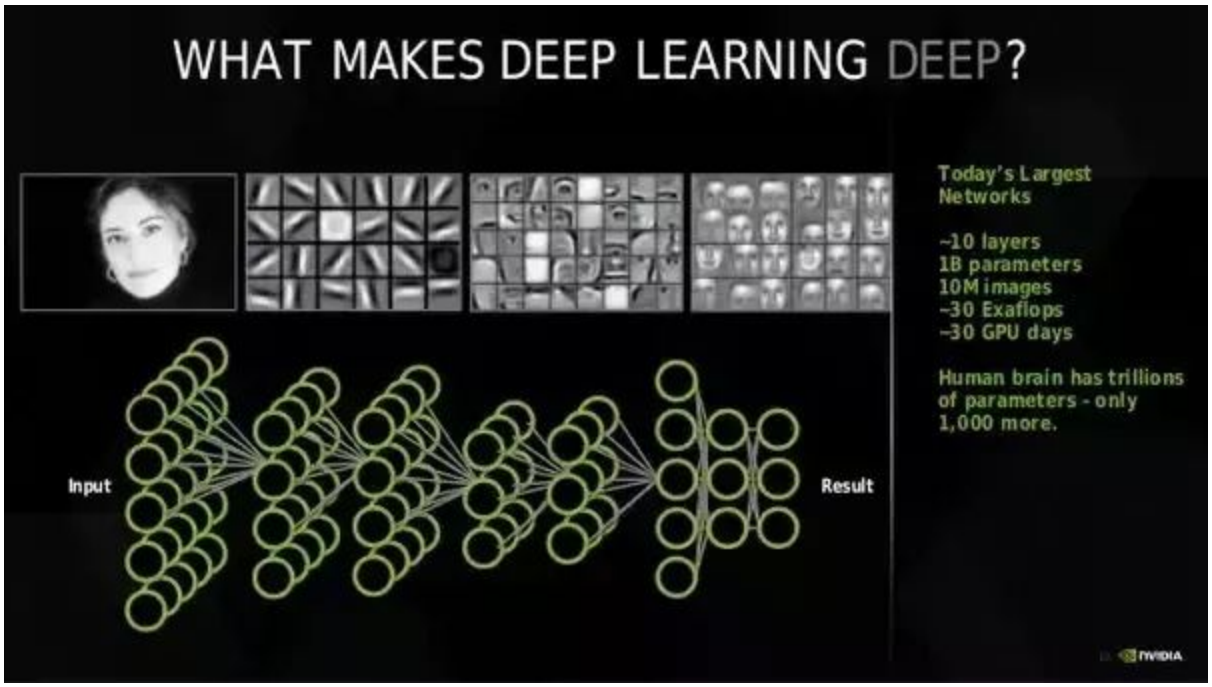


Figure 3: A Representative Deep Neural Network

Why feature engineering may turn obsolete?

Today as a data scientist, one may be learning feature engineering as a part of machine learning skill. In that, one need to transform one's data to the computer in a form that it can understand. For that one may use R or Python or spreadsheet software to translate the data. One may be converting one's data into a large spreadsheet of numbers containing rows and columns of instances and features. Then one feeds this data into the machine learning algorithm and it tries to learn from this data. As engineering the features is a time consuming task, we need to extract only the relevant features that improve our model. But as one is generally unaware of the usefulness of these features until one trains and tests the model, one is caught into the vicious cycle of developing new features, rebuilding the model, measure results, and repeat until one is satisfied with the results. This is very time consuming task and takes lot of your time.

How deep learning may save one's time?

In deep learning, ANNs are automatically extracting features instead of manual extraction in feature engineering. One can take the example of image as input. Instead of a user taking an image and hand compute features like distribution of colors, image histograms, distinct color count, etc., the user just have to feed the raw images in ANN. ANNs have already proved their worth in handling images, but now they are being applied to all kinds of other datasets like raw text, numbers etc. This helps the data scientist to concentrate more on building deep learning algorithms.

Big Data is required for deep learning

Soon, feature engineering may turn obsolete but deep learning algorithm will require massive data for feeding into our models. Fortunately, we now have big data sources not available two decades back – facebook, twitter, Wikipedia, project Guttenberg etc. However, the bottleneck remains in cleaning and processing these data into required format for powering the machine learning models. More and more big data will be made available for public consumption in near future.

Where can one get help in deep learning?

Open sourcing is predominant in deep learning, Tensorflow, Torch, Keras, Big Sur hardware, DIGITS and Caffe are some of the massive deep learning projects. In academic research, there are lots of papers with algorithm source code along with their findings. arXiv.org (a Cornell University Library) has open access to over 1 million papers in deep learning.

Does one need a computer expertise to learn deep learning?

One does not need a rigorous computer expertise to learn deep learning. The user's domain knowledge of the discipline will help him in building deep learning models. If one has learnt any data science language like 'R' or 'Python', spreadsheets like 'Excel' or any other basic programming and studied 'STEM', then one is proficient to delve into deep learning.

What are deep learning advantages?

One advantage that has already been explained is that the user doesn't have to figure out the features ahead of time. Another advantage is that one can use the same neural net approach for many different problems like Support Vector Machines, Linear Classifier, Regression, Bayesian, Decision Trees, Clustering and Association Rules. It is fault tolerant and scales well. Many perceptual tasks have been performed with help of CNNs. Some case studies are given below:

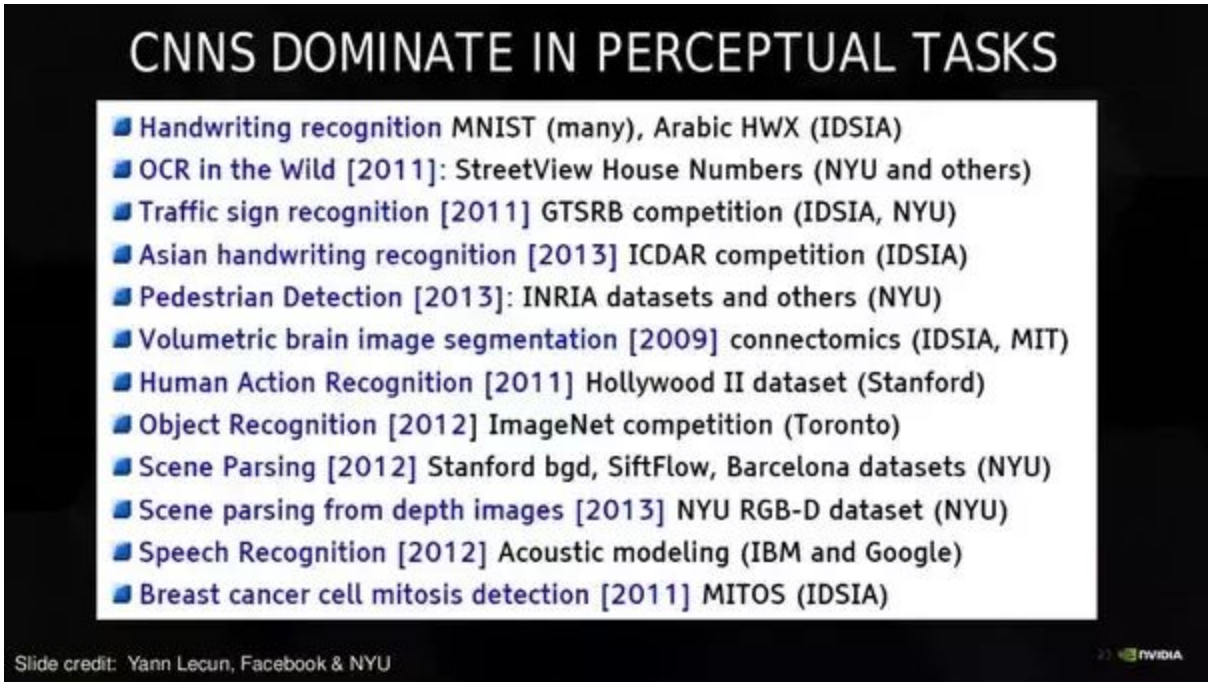


Figure 4: A Few Case Studies Showing the Vast Prospects of Deep Learning

How classical ML using feature engineering can be compared with deep learning using CNN? Well it can be seen with the help of these pictures.

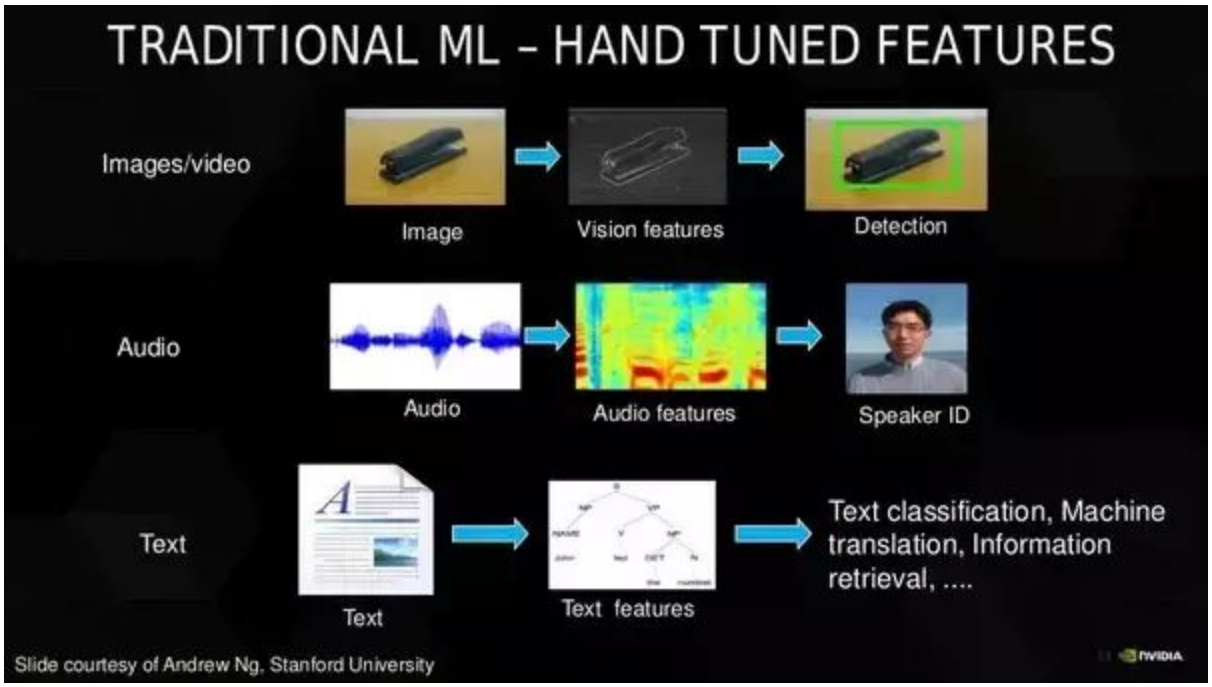


Figure 5 : Traditional ML

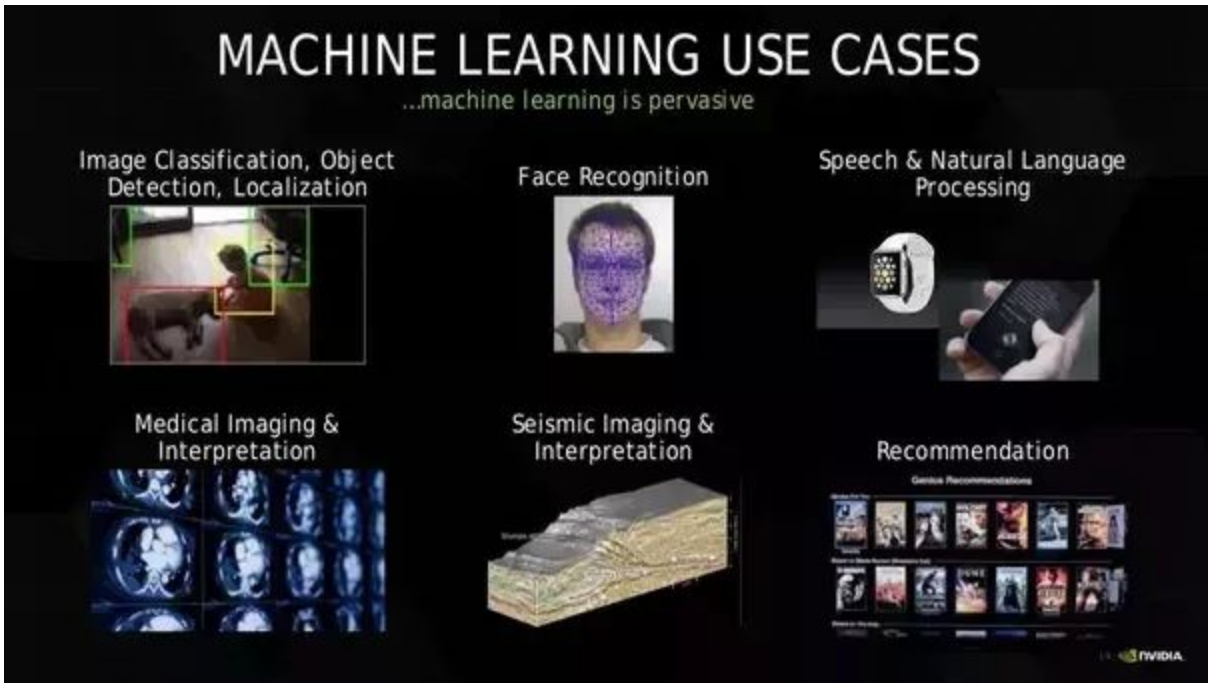


Figure 6 : Deep Learning in ML

From these pictures, one can see the wide applicability of deep learning in all aspects of life.

B. TOOLS USED

1. CONVOLUTIONAL NEURAL NETWORKS -

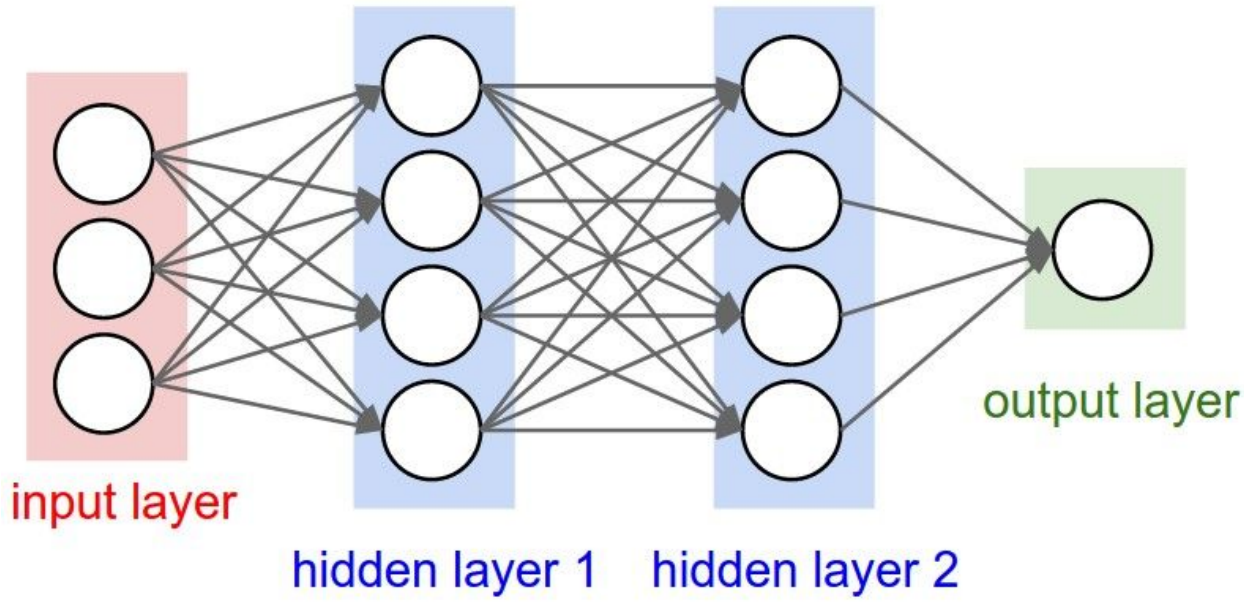
Convolutional Neural Networks are very similar to ordinary Neural Networks: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other.

For eg.in CIFAR-10, images are only of size 32x32x3 (32 wide, 32 high, 3 color channels), so a single fully-connected neuron in a first hidden layer of a regular Neural Network would have 32*32*3 = 3072 weights. This amount still seems manageable, but clearly this fully-connected structure does not scale to larger images. For example, an image of more respectable size, e.g. 200x200x3, would lead to neurons that have 200*200*3 = 120,000 weights. Moreover, one would almost certainly want to have

several such neurons, so the parameters would add up quickly Clearly, this full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

3D volumes of neurons. Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: **width, height, depth**. (It is to be noted that the word *depth* here refers to the third dimension of an activation volume, not to the depth of a full Neural Network, which can refer to the total number of layers in a network.) For example, the input images in CIFAR-10 are an input volume of activations, and the volume has dimensions 32x32x3 (width, height, depth respectively). The neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output layer would for CIFAR-10 have dimensions 1x1x10, because by the end of the ConvNet architecture it will reduce the full image into a single vector of class scores, arranged along the depth dimension.

Here is a visualization:



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds

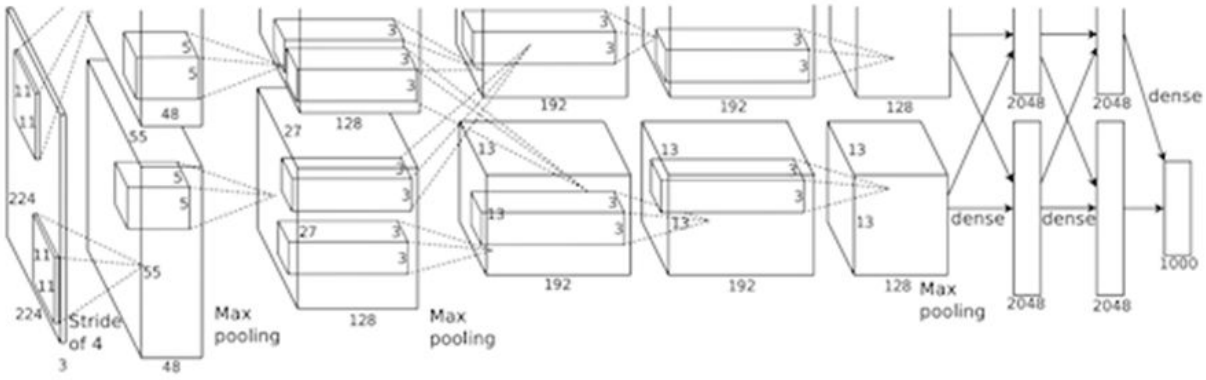
the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

A ConvNet is made up of Layers. Every Layer has a simple API: It transforms an input 3D volume to an output 3D volume with some differentiable function that may or may not have parameters.

AlexNet -

The one that started it all (Though some may say that Yann LeCun's paper in 1998 was the real pioneering publication). This paper, titled "ImageNet Classification with Deep Convolutional Networks", has been cited a total of 6,184 times and is widely regarded as one of the most influential publications in the field. Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton created a "large, deep convolutional neural network" that was used to win the 2012 ILSVRC (ImageNet Large-Scale Visual Recognition Challenge). For those that aren't familiar, this competition can be thought of as the annual Olympics of computer vision, where teams from across the world compete to see who has the best computer vision model for tasks such as classification, localization, detection, and more. 2012 marked the first year where a CNN was used to achieve a top 5 test error rate of 15.4% (Top 5 error is the rate at which, given an image, the model does not output the correct label with its top 5 predictions). The next best entry achieved an error of 26.2%, which was an astounding improvement that pretty much shocked the computer vision community. Safe to say, CNNs became household names in the competition from then on out.

In the paper, the group discussed the architecture of the network (which was called AlexNet). They used a relatively simple layout, compared to modern architectures. The network was made up of 5 conv layers, max-pooling layers, dropout layers, and 3 fully connected layers. The network they designed was used for classification with 1000 possible categories.



AlexNet architecture (May look weird because there are two different "streams". This is because the training process was so computationally expensive that they had to split the training onto 2 GPUs)

Main Points

- Trained the network on ImageNet data, which contained over 15 million annotated images from a total of over 22,000 categories.
- Used ReLU for the nonlinearity functions (Found to decrease training time as ReLUs are several times faster than the conventional tanh function).
- Used data augmentation techniques that consisted of image translations, horizontal reflections, and patch extractions.
- Implemented dropout layers in order to combat the problem of overfitting to the training data.
- Trained the model using batch stochastic gradient descent, with specific values for momentum and weight decay.
- Trained on two GTX 580 GPUs for five to six days.

Why It's Important

The neural network developed by Krizhevsky, Sutskever, and Hinton in 2012 was the coming out party for CNNs in the computer vision community. This was the first time a model performed so well on a historically difficult ImageNet dataset. Utilizing techniques that are still used today, such as data augmentation and dropout, this paper really illustrated the benefits of CNNs and backed them up with record breaking performance in the competition.

2. SEQUENTIAL NEURAL NETWORKS -

There are many types of SNNs. SNNs are used for classification purposes in cases of sequential data where the data at a particular moment is dependent on the data from

the past i.e. temporal data. Examples of temporal data are videos or frame sequence, text from a book or a speech among other. SNNs deal with such data by storing its behavior in response to past data or the history of the network. The way to store the history varies from one type of SNN to the other and that's what differentiates one type of SNN from another. A few popular types of SNNs are Recurrent Neural Networks (RNN), Long Short Term Memory (LSTM) and Gated Recurrent Network (GRU).

RNN -

A recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed cycle. This allows it to exhibit dynamic temporal behavior. Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition. The governing equations of RNNs are

$$h_t = W f(h_{t-1}) + W^{(hx)} x_t$$

$$\hat{y} = W^{(S)} f(h_t)$$

Here, 'W' is the weight matrices and 'h' is the state of the RNN. 'h' is a function of 't' and it is called the RNN state at time 't'. 'y' is the output of the the RNN at the particular time instant 't'.

A drawback of normal RNN is that due to the long term time dependence of the RNN state on past states, the gradients may die out during learning due to multiple steps of backpropagation. This is called the problem of 'vanishing gradients'. So, a better replacement for RNNs are LSTMs and GRUs.

LSTM -

LSTMs Ire invented as a solution to the 'vanishing gradients' of the RNNs. It exploits short term dependencies in place of long term dependencies which is the root cause of the vanishing gradients. The LSTMs use a memory cell denoted by 'c' which represents

the short term memory state of the LSTM at time ‘t’. The governing equation of the LSTMs are

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

$$\tilde{C}_t = tanh(W_c x_t + U_c h_{t-1} + b_c)$$

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

$$C_t = i_t * \tilde{C}_t + f_t * C_{t-1}$$

I have used LSTMs for the purpose of text generation given a primer or starting text. LSTMs are still a pretty popular choice for a lot of sequential operations because of its simplicity and convenience of training in comparison to the other better alternatives.

I have skipped describing the GRUs since they are not relevant for this particular task as of now though I plan to try with GRUs also once I am finished with modelling the whole thing

C. DATASETS -

1. Datasets for Text Generation -

a) The Penn Treebank, in its eight years of operation (1989-1996), produced approximately 7 million words of part-of-speech tagged text, 3 million words of skeletally parsed text, over 2 million words of text parsed for predicate argument structure, and 1.6 million words of transcribed spoken text annotated for speech disfluencies. The material annotated includes such wide-ranging genres as IBM computer manuals, nursing notes, Wall Street Journal articles, and transcribed telephone conversations, among others. This paper describes the design of the three annotation schemes used by the Treebank: POS tagging, syntactic bracketing, and disfluency annotation and the methodology employed in production. All available Penn Treebank materials are distributed by the Linguistic Data Consortium http://www.ldc.upenn.edu. The whole dataset is split up into training, validation and testing where the validation set can be used to check the BLEU score and set the hyperparameters.

b) The is a full extract from a GitHub repository which I used for my initial trials with my model. I split it into training, validation and testing for the purpose. The splitting was done absolutely randomly.

2. Datasets for Image to Character Conversion

I used the American Sign Language (ASL) finger-spelling dataset for my work. FInger-spelling refers to the system of signs or gestures where each gesture represents a single character. The datasets contain a set of RGB and depth images for each letter in the alphabet, organized by subject, for estimating generalization.

There are two sub-datasets to this dataset which are graded on the basis of difficulty of classification. They are as follows

a) The first dataset comprises 24 static signs (excluding letters j and z because they involve motion). This was captured in 5 different sessions, with similar lighting and background.

b) The second dataset (depth only) is captured from 9 different persons in two very different environments and lighting.

D. WORK PIPELINE

1. Language Model using Common Bag of Words Model -

There are an estimated 13 million tokens for the English language but are they all completely unrelated? Feline to cat, hotel to motel? I think not. Thus, I want to encode word tokens each into some vector that represents a point in some sort of "word" space. This is paramount for a number of reasons but the most intuitive reason is that perhaps there actually exists some N-dimensional space (such that N much less than 13 million) that is sufficient to encode all semantics of my language. Each dimension would encode some meaning that I transfer using speech. For instance, semantic dimensions might indicate tense (past vs. present vs. future), count (singular vs. plural), and gender (masculine vs. feminine).

So let’s dive into my first word vector and arguably the most simple, the one-hot vector: Represent every word as an R(dimensions : |V|×1) vector with all 0s and one 1 at the index of that word in the sorted english language. In this notation, |V| is the size of my vocabulary. Word vectors in this type of encoding would appear as the following:

$$w^{aardvark} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, w^{a} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, w^{at} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \cdots w^{zebra} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

I represent each word as a completely independent entity. As I previously discussed, this word representation does not give us directly any notion of similarity. For instance,

$$(w^{hotel})^T w^{motel} = (w^{hotel})^T w^{cat} = 0$$

So maybe I can try to reduce the size of this space from R(dimensions : |V|) to something smaller and thus find a subspace that encodes the relationships betIen words. Common Bag of Words (CBOW) model is an example of one such model and is quite commonly used for text generation in Natural Language Processing (NLP).

One approach is to treat {"The", "cat", 'over", "the', "puddle"} as a context and from these words, be able to predict or generate the center word "jumped". This type of model I call a Continuous Bag of Words (CBOW) Model.
The setup is largely the same but I essentially swap my x and y i.e. x in the CBOW are now y and vice-versa. The input one hot vector (center word) I will represent with an x (since there is only one). And the output vectors as y(j) . I define V and U the same as in CBOW. I breakdown the way this model works in these 6 steps:

1. I generate my one hot input vector x
2. I get my embedded word vectors for the context vc = Vx
3. Since there is no averaging, just set vˆ = vc ?
4. Generate 2m score vectors, uc−m, . . . , uc−1, uc+1, . . . , uc+m using u = Uvc
5. Turn each of the scores into probabilities, y = softmax(u)
6. I desire my probability vector generated to match the true probabilities which is y (c−m) , . . . , y (c−1) , y (c+1) , . . . , y (c+m) , the one hot vectors of the actual output.

As in CBOW, I need to generate an objective function for us to evaluate the model. A key difference here is that I invoke a Naive Bayes assumption to break out the probabilities. If you have not seen this before, then simply put, it is a strong (naive) conditional independence assumption. In other words, given the center word, all output words are completely independent.

$$
\begin{aligned}
\text{minimize } J &= -\log P(w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m} | w_c) \\
&= -\log \prod_{j=0, j \neq m}^{2m} P(w_{c-m+j} | w_c) \\
&= -\log \prod_{j=0, j \neq m}^{2m} P(u_{c-m+j} | v_c) \\
&= -\log \prod_{j=0, j \neq m}^{2m} \frac{\exp(u_{c-m+j}^T v_c)}{\sum_{k=1}^{|V|} \exp(u_k^T v_c)} \\
&= -\sum_{j=0, j \neq m}^{2m} u_{c-m+j}^T v_c + 2m \log \sum_{k=1}^{|V|} \exp(u_k^T v_c)
\end{aligned}
$$

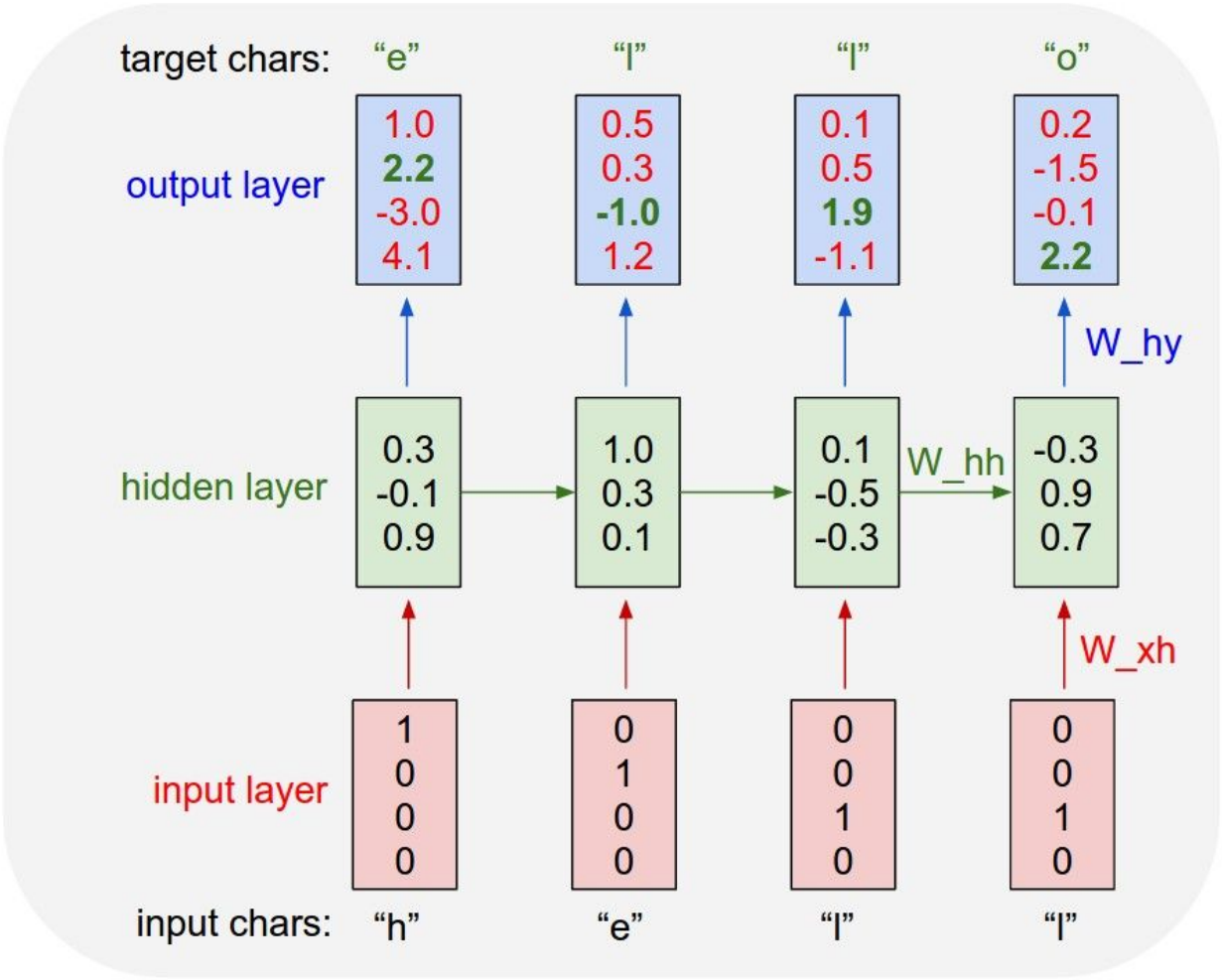
With this objective function, I can compute the gradients with respect to the unknown parameters and at each iteration update them via Stochastic Gradient Descent.

2. Text Generation using Character LSTM

I'll train RNN character-level language models. That is, I'll give the RNN a huge chunk of text and ask it to model the probability distribution of the next character in the sequence given a sequence of previous characters. This will then allow us to generate new text one character at a time.

As a working example, suppose I only had a vocabulary of fmy possible letters "helo", and wanted to train an RNN on the training sequence "hello". This training sequence is in fact a smyce of 4 separate training examples: 1. The probability of "e" should be likely given the context of "h", 2. "l" should be likely in the context of "he", 3. "l" should also be likely given the context of "hel", and finally 4. "o" should be likely given the context of "hell".

Concretely, I will encode each character into a vector using 1-of-k encoding (i.e. all zero except for a single one at the index of the character in the vocabulary), and feed them into the RNN one at a time with the step-function. I will then observe a sequence of 4-dimensional output vectors (one dimension per character), which I interpret as the confidence the RNN currently assigns to each character coming next in the sequence. Here's a diagram:



An example RNN with 4-dimensional input and output layers, and a hidden layer of 3 units (neurons). This diagram shows the activations in the forward pass when the RNN is fed the characters "hell" as input. The output layer contains confidences the RNN assigns for the next character (vocabulary is "h,e,l,o"); I want the green numbers to be high and red numbers to be low.

For example, I see that in the first time step when the RNN saw the character “h” it assigned confidence of 1.0 to the next letter being “h”, 2.2 to letter “e”, -3.0 to “l”, and 4.1 to “o”. Since in my training data (the string “hello”) the next correct character is “e”, I would like to increase its confidence (green) and decrease the confidence of all other letters (red). Similarly, I have a desired target character at every one of the 4 time steps that I’d like the network to assign a greater confidence to. Since the RNN consists entirely of differentiable operations I can run the backpropagation algorithm (this is just a recursive application of the chain rule from calculus) to figure out in what direction I should adjust every one of its weights to increase the scores of the correct targets

(green bold numbers). I can then perform a *parameter update*, which nudges every light a tiny amount in this gradient direction. If I lre to feed the same inputs to the RNN after the parameter update I would find that the scores of the correct characters (e.g. “e” in the first time step) would be slightly higher (e.g. 2.3 instead of 2.2), and the scores of incorrect characters would be slightly loIr. I then repeat this process over and over many times until the network converges and its predictions are eventually consistent with the training data in that correct characters are always predicted next.

A more technical explanation is that I use the standard Softmax classifier (also commonly referred to as the cross-entropy loss) on every output vector simultaneously. The RNN is trained with mini-batch Stochastic Gradient Descent and I like to use RMSProp or Adam (per-parameter adaptive learning rate methods) to stabilize the updates.

Notice also that the first time the character “l” is input, the target is “l”, but the second time the target is “o”. The RNN therefore cannot rely on the input alone and must use its recurrent connection to keep track of the context to achieve this task.

At test time, I feed a character into the RNN and get a distribution over what characters are likely to come next. I sample from this distribution, and feed it right back in to get the next letter. This process is repeated and text sampling is done. I have written the character-level LSTM in PyTorch.

3. Character Label Generation from ASL Images using CNN

The ASL data set images were fed resized to 32x32 images and were fed into a modified AlexNet, the simplest of standard baseline CNN architectures, which has 24 output neurons same in number as the number of classes in the ASL database. I used transfer learning to train an AlexNet, already trained on the ImageNet dataset, on the ASL dataset. I tried two versions of transfer learning: training only the last fully connected layer and training both fully connected layers at the end. I obtained the best results by training the last three fully connected layers since they contain the majority of the trainable parameters. I will describe transfer learning here in detail.

Transfer Learning -

Chen et al argue that a good training data must have width and depth, i.e. the dataset must contain a large number of subject classes for training and each individual class must have a good number of images in order for the network to train effectively. The ASL dataset from the Exter university suffers severely from the problem of small depth.

Chen et al have resorted to the creation of a new dataset namely, Wide and Deep Reference Net (WDRef) in order to cope with this problem and later hope to transfer the learning acquired by the training of this model to the ASL Dataset. Essentially, the model trained on WDRef learns a high dimensional representation of faces towards the end of the network, which ASL did not allow due to its lack of depth. Training on ASL would hence be more convenient for the network, given that it has already learnt how to represent images from the previous ImageNet training.

I switched off training of all the other layers apart from the last few layers on which I am applying transfer learning by stopping the flow of gradients into those layers. This also speeds up training to a large extent.

E. FINAL RESULTS

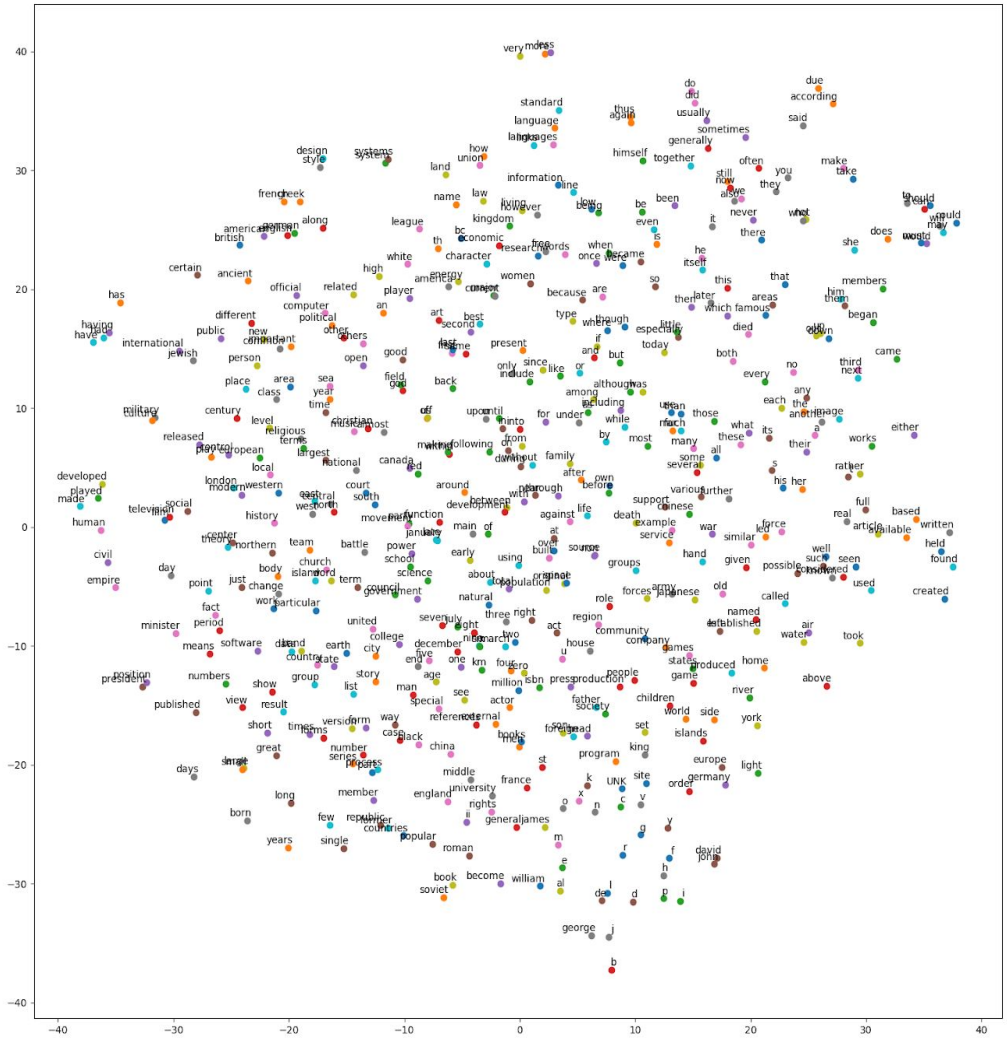


Fig. 1 - t-SNE plot of language model using CBOW

t-distributed stochastic neighbor embedding (t-SNE) is a machine learning algorithm for dimensionality reduction developed by Geoffrey Hinton and Laurens van der Maaten. It is a nonlinear dimensionality reduction technique that is particularly well-suited for embedding high-dimensional data into a space of two or three dimensions, which can then be visualized in a scatter plot. Specifically, it models each

high-dimensional object by a two- or three-dimensional point in such a way that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points.

The t-SNE algorithm comprises two main stages. First, t-SNE constructs a probability distribution over pairs of high-dimensional objects in such a way that similar objects have a high probability of being picked, whilst dissimilar points have an extremely small probability of being picked. Second, t-SNE defines a similar probability distribution over the points in the low-dimensional map, and it minimizes the Kullback–Leibler divergence between the two distributions with respect to the locations of the points in the map. Note that whilst the original algorithm uses the Euclidean distance between objects as the base of its similarity metric, this should be changed as appropriate.

t-SNE has been used in a wide range of applications, including computer security research, music analysis, cancer research, bioinformatics, and biomedical signal processing. It is often used to visualize high-level representations learned by an artificial neural network.

Below are the graphs for training accuracy and validation accuracy of training and validation errors for my LSTM text generation model on the Shakespearian extract dataset.

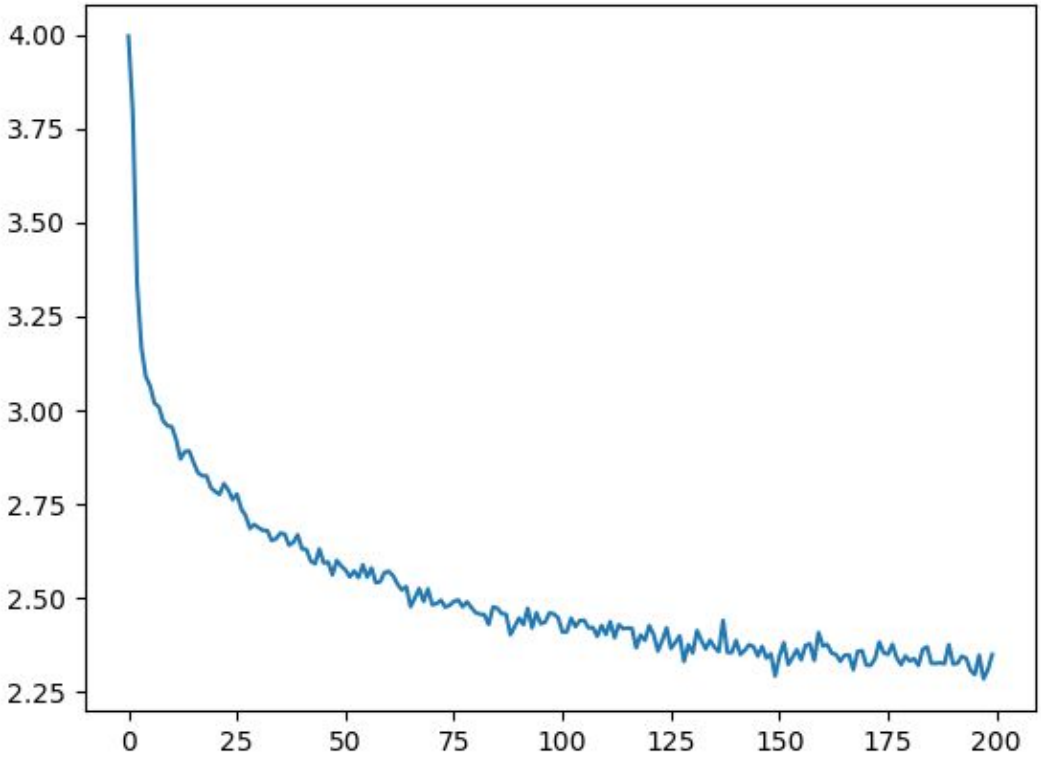


Fig. 2 - Training Error with Epochs for SNN

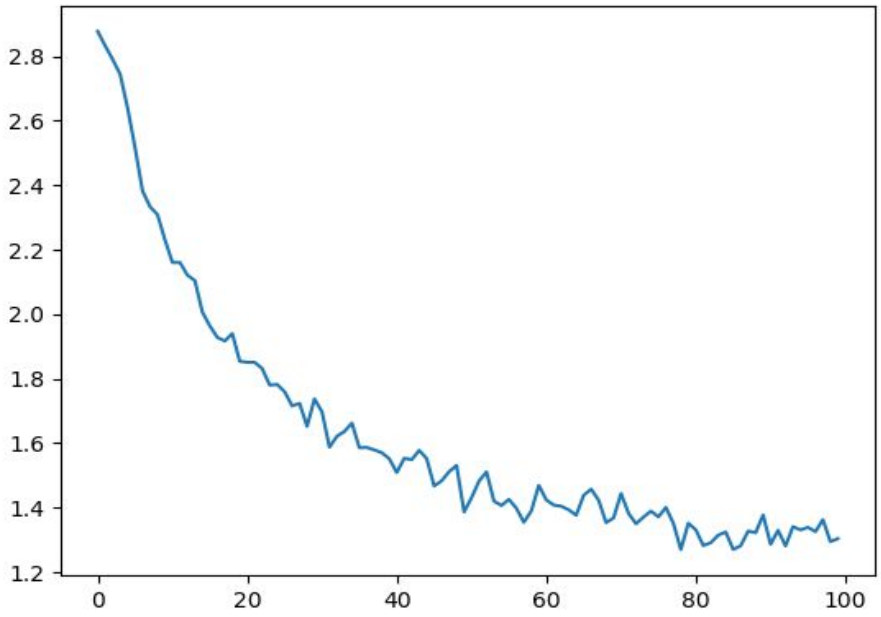


Fig. 2 - Validation Error with Epochs for SNN

Below are the graphs for training accuracy and validation accuracy of training and validation errors for my fine tuned AlexNet model with the last FC layer and the last two FC layers trained on the Exter University ASL dataset.

Training for ASL Dataset with 2 FC layer fine-tuned

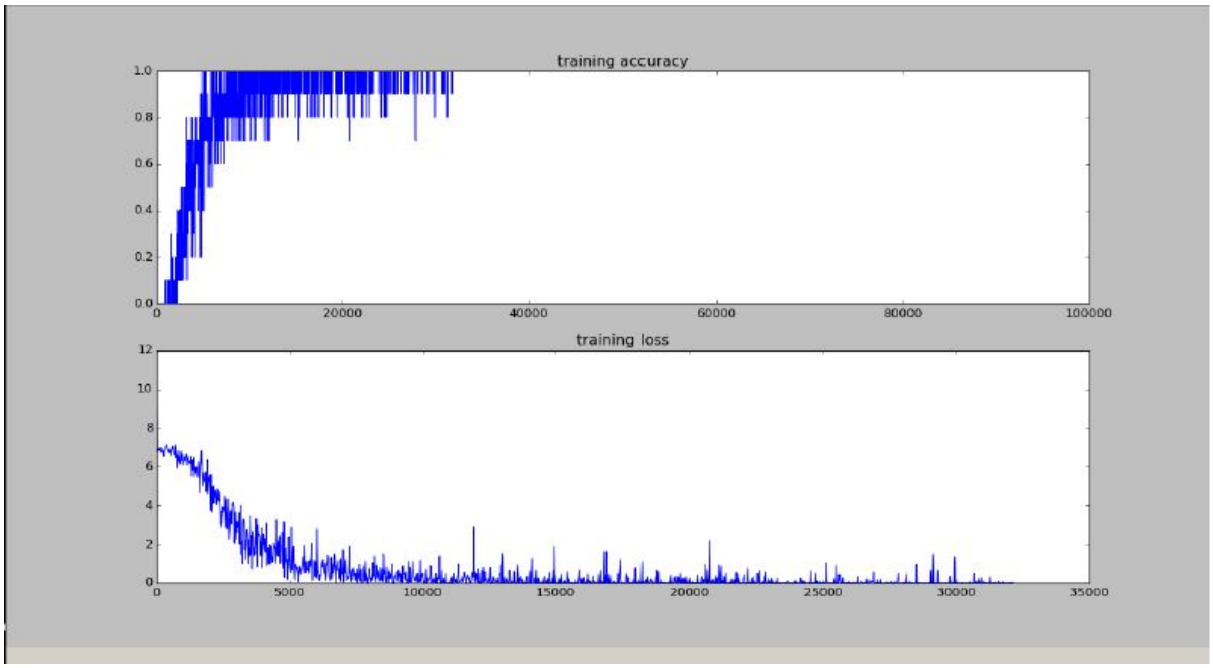


Fig. 4: Training Error and Accuracy on ASL Dataset with last 2 FC layers fine-tuned

Validation for ASL Dataset with 2 FC layer fine-tuned

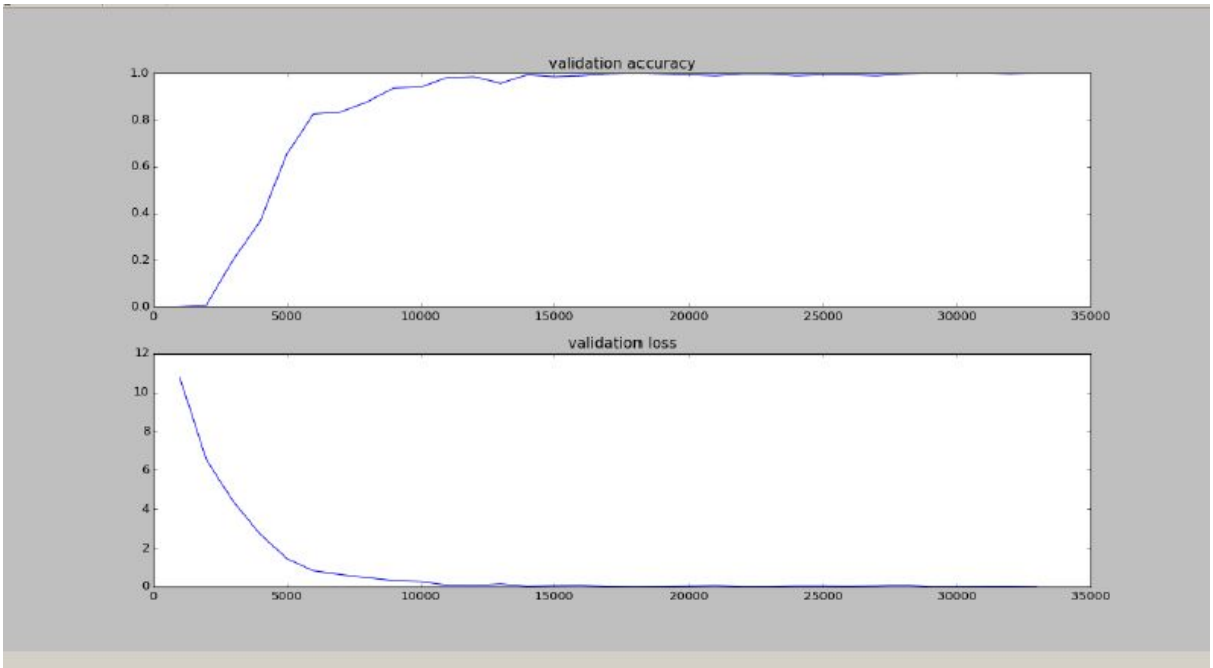


Fig. 5: Validation Error and Accuracy on ASL Dataset with last 2 FC layers fine-tuned

Training for ASL Dataset with 1 FC layer fine-tuned

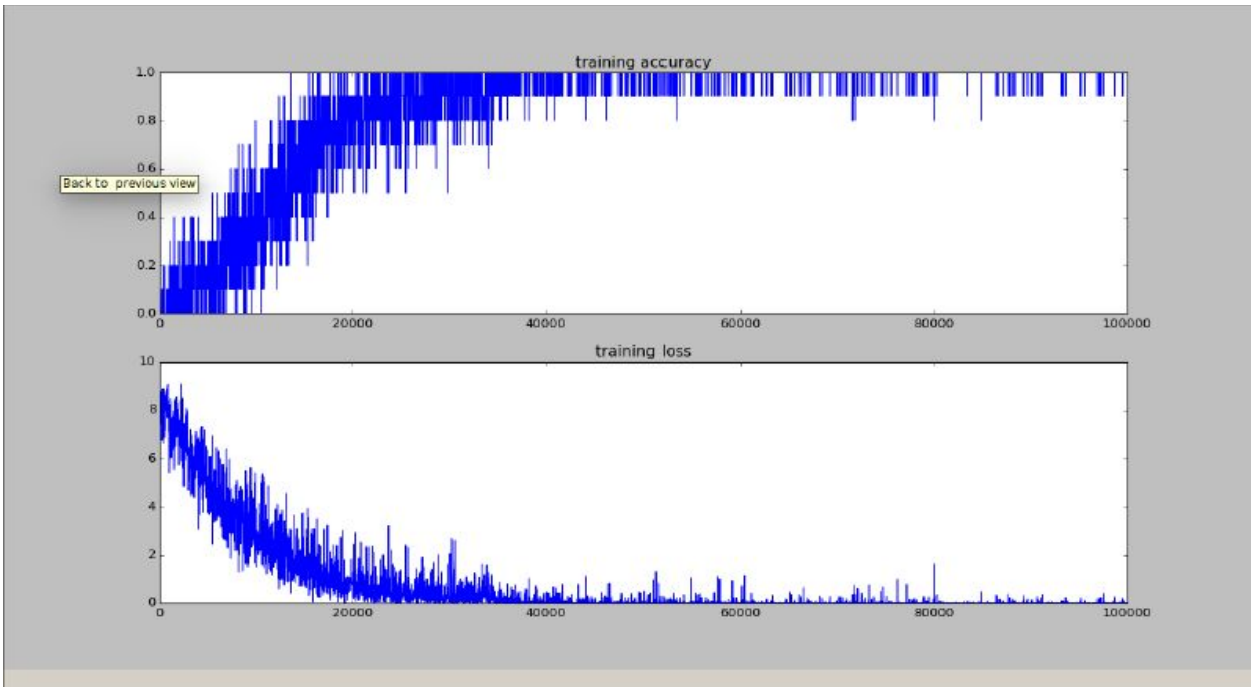


Fig. 6: Training Error and Accuracy on ASL Dataset with last 1 FC layers fine-tuned

Validation for ASL Dataset with 1 FC layer fine-tuned

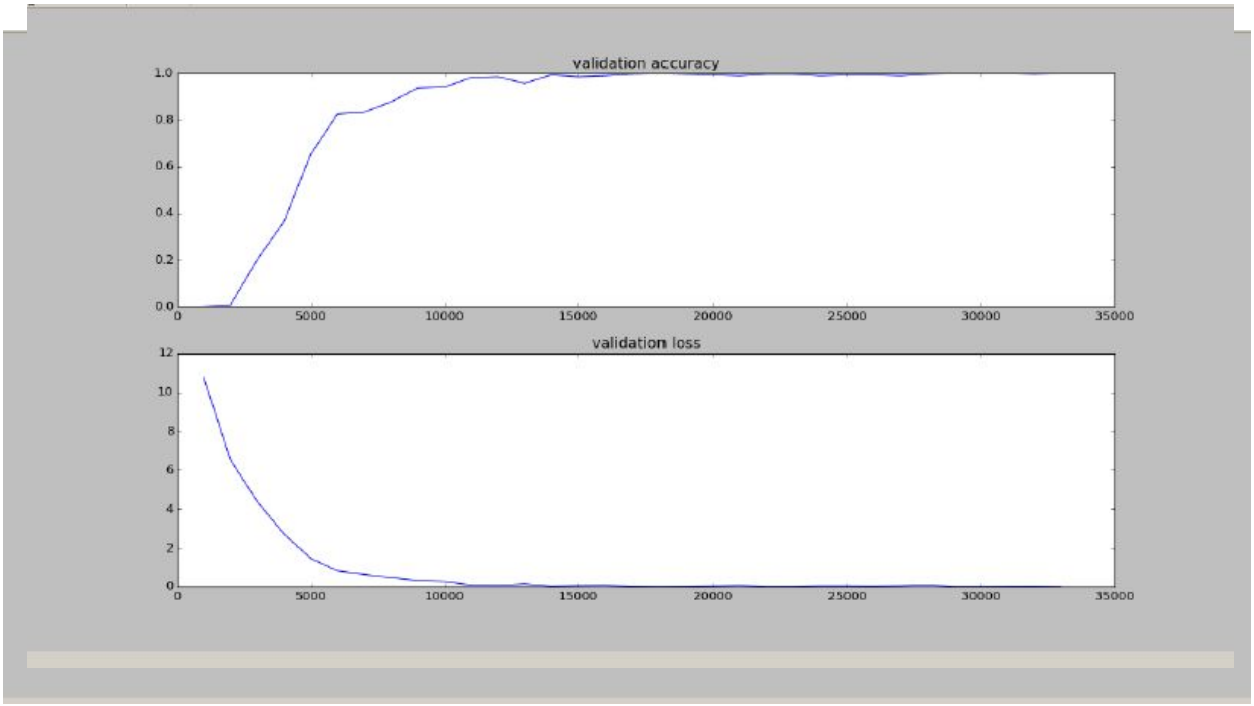
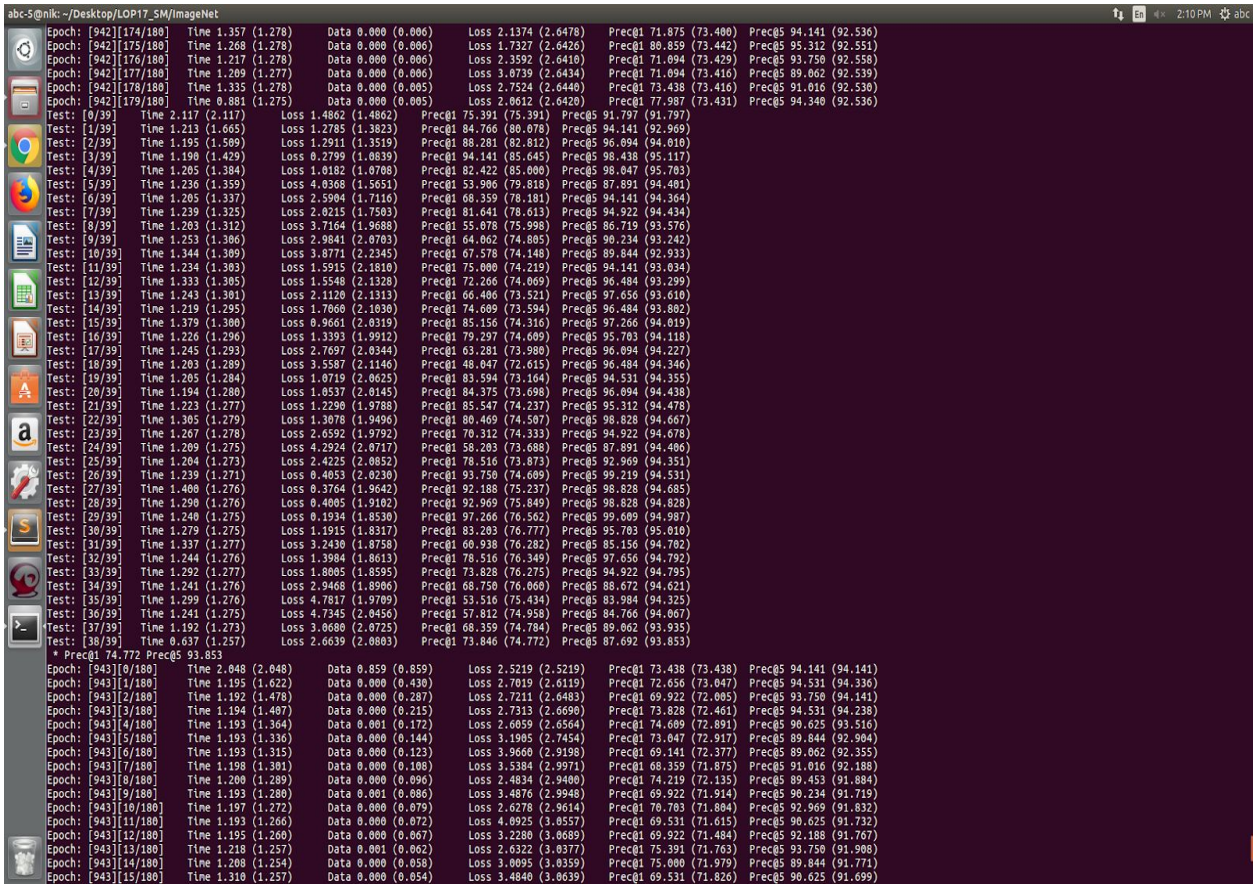


Fig. 7: Validation Error and Accuracy on ASL Dataset with last 1 FC layers fine-tuned

The final accuracy values for validation are as follows -

- Top-5 accuracy in percentage - 74.772
- Top-1 accuracy in percentage - 93.853

Please refer to the figure below.



Also, a user-friendly interfacing code was written to input variable number of individual ASL images to create the primer text for the sentence completion. Kindly refer to the videos.