Generate textual descriptions of STEM images to aid Visually-impaired

Objective:
Classify infographics of science, technology, engineering, and mathematics and come up with descriptions to make them accessible to individuals with visual and print-reading disabilities.

How we planned to do it:

- Classify the images into their respective types - (Bar chart, Area graph etc.)
- Retrieve text from the charts
- Describe the chart using classification and the retrieved text information.

Pipeline:

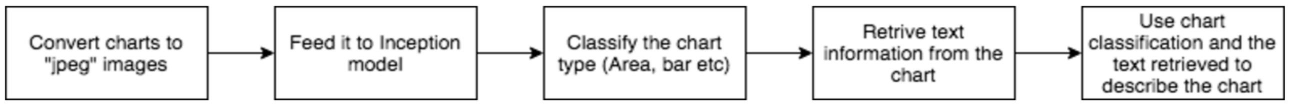
Convert charts to "jpeg" images
Feed it to Inception model
Classify the chart type (Area, bar etc)
Retrive text information from the chart
Use chart classification and the text retrieved to describe the chart

Figure 1: Pipeline of tasks

Background: In the last few years the field of machine learning has made tremendous progress on addressing these difficult problems. Deep learning in particular has caused enormous breakthroughs in the fields of image recognition, speech recognition and text recognition. This has been possible with the exponential data growth (abundant training data) and also relatively powerful computing systems available to train the networks to make them scalable. In particular, we have found that deep convolutional neural network can achieve reasonable performance on hard visual recognition tasks – matching or exceeding human performance in some domains.

Due to the known efficiency of CNN for image classification, we have decided to use a CNN model for classifying the charts. " An analysis of single layer networks in Unsupervised learning" by Andrew Y.Ng talks about how learning features from unlabelled data (a sort of pre-training the network) can improve the performance of classification tasks rather than a fully supervised learning model.

Thus we have tried to develop a model that best captures the features of images through a CNN , then followed by few fully connected layers that act as a classifier based on the features generated through convolution. This can be very useful in the case where we have a lot of unlabeled data and limited labeled data. Unlabeled data can be used to pre-train the network and labeled data can be used in training the final classifier.

Tensorflow:
We decided to use "Tensorflow" python library for our project to build and train our model.

TensorFlow is an open source software library for numerical computation using data flow graphs. It also has GPU support, which would be an important factor in training our model if our model training takes too much time when run on our CPUs.

Model Architecture:

1. Preprocess images to a standard 100*100 pixels. We have used Imagemagick to perform the resizing - padding, resolution increase/decrease. Input image dimension is thus 100*100*3 (height * width * no of channels - RGB).

2. **Convolution layer 1 :** generates 32 filter responses using "SAME" Padding.
   Input image size - [100,100,3]
   Filters - 32
   Kernel size - [5,5,3]
   Output - [100,100,32] (height, width, no of filter responses)

3. **Max pooling layer:** To reduce dimensionality of output - 2*2 maxpooling
   Input - [100,100,32] response from previous layer
   Output - [50,50,32] response

4. **Convolution layer 2 :** generates 64 filter responses using "SAME" Padding.
   Input imgsize - [50,50,32]
   Filters - 64
   Kernel size - [5,5,32]
   Output - [50,50,64] (height, width, no of feature maps)

5. **Max pooling layer:** To reduce dimensionality of output - 2*2 maxpooling
   Input - [50,50,64] response from previous layer
   Output - [25,25,64] response

6. **Fully connected layer 1:** (batchsize, 25*25*64) —- (batchsize, 1024)
   We flatten out the input from previous layer into a single dimension
   Input size - [no of images in a batch, 25*25*64]
   No of hidden nodes - 1024
   Output size - [no of images in batch, 1024]

7. **Fully connected layer2:** (batchsize, 1024) — (batchsize, no of categories)
   Input size - [no of images in batch, 1024]
   No of nodes (no of categories) - 10
   Output size - [no of images in batch, 10]

Unsupervised training (feature learning):
We use **Autoencoders** which is one of the popular algorithms for feature learning via unsupervised learning. Autoencoders primarily used for dimensionality reduction have proved to be a good algorithm for feature learning in the past.

The convolution layers in our model were pretrained with the autoencoder. Encoding was our model's convolution layers. Decoding was done on the encoded output with 2 fully connected dense layers to generate back the image input.

Encoder - Convolution layer 1, maxpooling, Convolution layer 2, Max pooling
Decoder - 2 Fully connected layers to generate the image input.

Now we have a good initial feature learning. The CNN layers have good filters learnt that represent the image rather than a random initialization of the filter weights.

Supervised learning:
Now we have to train our CNN model with a set of 2000 labeled images to output the label.

Implemented Model - Inception model):
Initially as per our plan, we wrote a CNN model from the scratch. It also took a lot of time to build the auto encoder and debug it. We havent yet trained the network on all the images. We figured that coming up with model with more than 80% accuracy might take much more time than intended. So we decided to use an existing model that which has proven to be good at image classification. We will continue working on training the CNN we built with the data set.

We wanted to try out inception model for this classification task since Inception model is a much more advanced implementation of CNN with multiple convolutions, pooling performed at each layer and the output feature maps stacked to form inputs to subsequent layers.

Thus we decide to train our data on an existing model "Inception". Inception mode was originally trained for the ImageNet Large Visual Recognition Challenge using the data from 2012. This is a standard task in computer vision, where models try to classify entire images into 1000 classes, like "Zebra", "Dalmatian", and "Dishwasher". The top-5 error rate of Inception model on image net model is 3.43%.

Inception architecture:

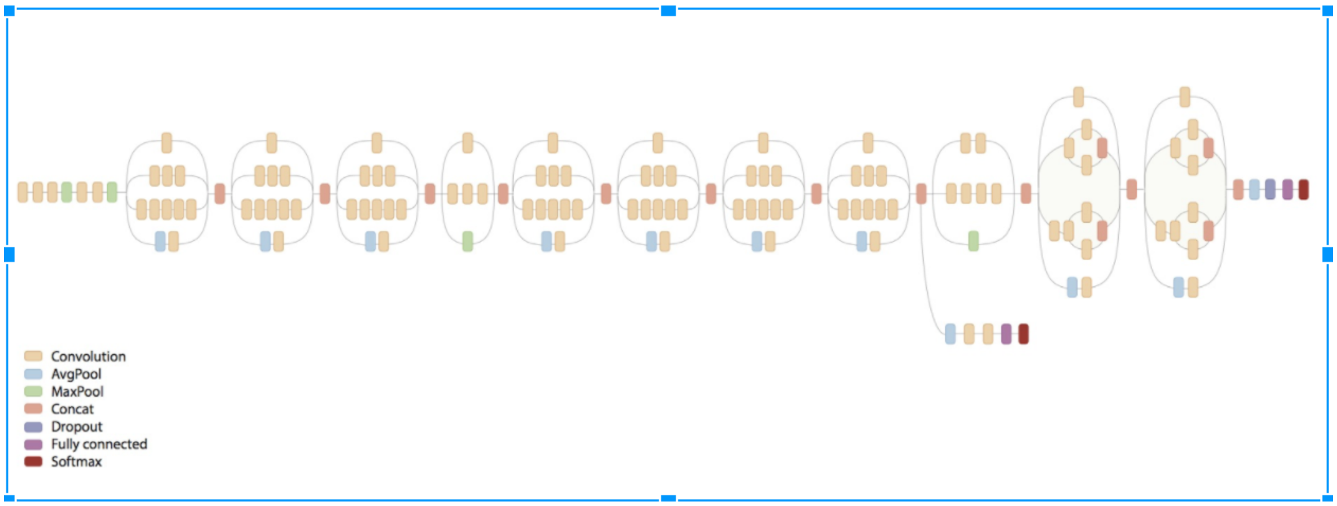

Figure 2: Inception architecture

Progress:

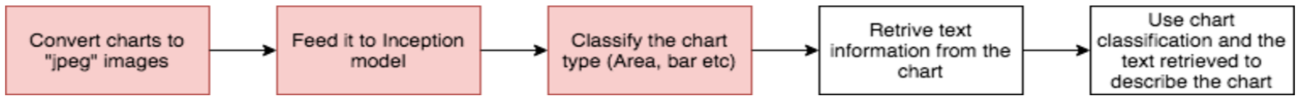
Convert charts to "jpeg" images
Feed it to Inception model
Classify the chart type (Area, bar etc)
Retrive text information from the chart
Use chart classification and the text retrieved to describe the chart

Figure 3: Progress so far

We were able to setup the Tensorflow development environment. We downloaded around 2000 images from the Stanford ReVision Image Corpus. The problem we face during this phase was that images were obtained were in various formats "png, gif, jpeg". For our ease and uniformity we converted all the images obtained to "jpeg" format.

The Inception model is pretrained on the "Imagenet" dataset. Though we can retrain the entire model, we retrained the last layer of the inception model using our data. This process involved restoring all weights from the pre-trained Inception except for the final classification layer; this was randomly initialized instead and our images were used to learn the weights. We wanted checkout the accuracy for training by doing that.

By retraining the last layer Inception model with our data we were able to achieve a validation accuracy of 84% on our dataset.

Comparison with existing work:
The existing implementation ReVision to classify infographics used a SVM classifier and had an accuracy of 96%. However, we were not able to produce the same level of accuracy when using Inception since the CNN was trained on the ImageNet model which has a lot of diverse categories of images (1000 categories). We hope to achieve a good accuracy once we train the CNN we have built.

Remaining work:
We have to train the CNN we built with the image corpus and find the accuracy of classification. After the classification of the images, we have to extract the textual information out of the chart images. This might be the most difficult part of the project, as there is no specific layout in which we can expect the text in charts to be. The text can be scattered around chart. We plan to solve this problem for few categories as doing the same for all 10 chart categories might not be feasible within the given time period.
After extraction of these information from the charts we can build a description for the charts.

References:

1. Project repository on Github - https://github.com/kirthan18/CS766—Project

2. http://www.ijser.org/researchpaper/Image-Classification-Using-Convolutional-Neural-Networks.pdf

3. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

4. Rethinking the Inception Architecture for Computer Vision -https://arxiv.org/abs/1512.00567

5. Tensorflow - https://www.tensorflow.org/tutorials/image_retraining

6. ImageMagick - http://www.imagemagick.org/script/index.php