Explainable AI

Abhishek Bhave
UBitName: abhave PersonNumber: 50289049
Department of Computer Science
University of Buffalo
Buffalo, NY 14214
abhave@buffalo.edu

Abstract

This project is to develop a machine learning system that learns explainable features for a task domain while it is learning to answer a variety of queries in that domain. It involves combining deep learning and probabilistic graphical models.

We are given M samples of an input, e.g., a photograph of a person, scanned images of a handwritten word written by a known writer. Our goal is to work with three different sets of features: human determined features, deep learning features, explainable deep learning features.

1.1 Introduction

Neural networks are increasingly being used to build programs that can predict and classify in a myriad of different settings. However, we don't really know how they work. Even if we tune the hyperparameters, we get highly accurate and efficient model with very little loss, we still cannot explain why the model predicted a certain value, or why the model is giving out these sets of results. For this purpose, we use the concept of explainable AI, which is basically explaining the result along with the predictions.

Explainable AI (XAI), Interpretable AI, or Transparent AI refer to techniques in artificial intelligence (AI) which can be trusted and easily understood by humans. It contrasts with the concept of the "black box" in machine learning where even their designers cannot explain why the AI arrived at a specific decision. XAI can be used to implement a social right to explanation. Some claim that transparency rarely comes for free and that there are often tradeoffs between how "smart" an AI is and how transparent it is; these tradeoffs are expected to grow larger as AI systems increase in internal complexity.

Why the need for Explainable Models?

Neural Networks are not infallible. Besides the problems of overfitting and underfitting that we've developed many tools (like Dropout or increasing the data size) to counteract, neural networks operate in an opaque way.

We don't really know why they make the choices they do. As models become more complex, the task of producing an interpretable version of the model becomes more difficult.

In decision trees or neural networks, the complexity of the method itself removes the ability to carry through the original factors that drive the algorithm's predicted outcome. Of course, regression models provide an explanation but as previously stated, they aren't machine learning and suffer from lower precision relative to their machine learning counterparts.

So, Explainable AI (XAI) is basically any machine learning technology that can accurately explain a prediction at the individual level.

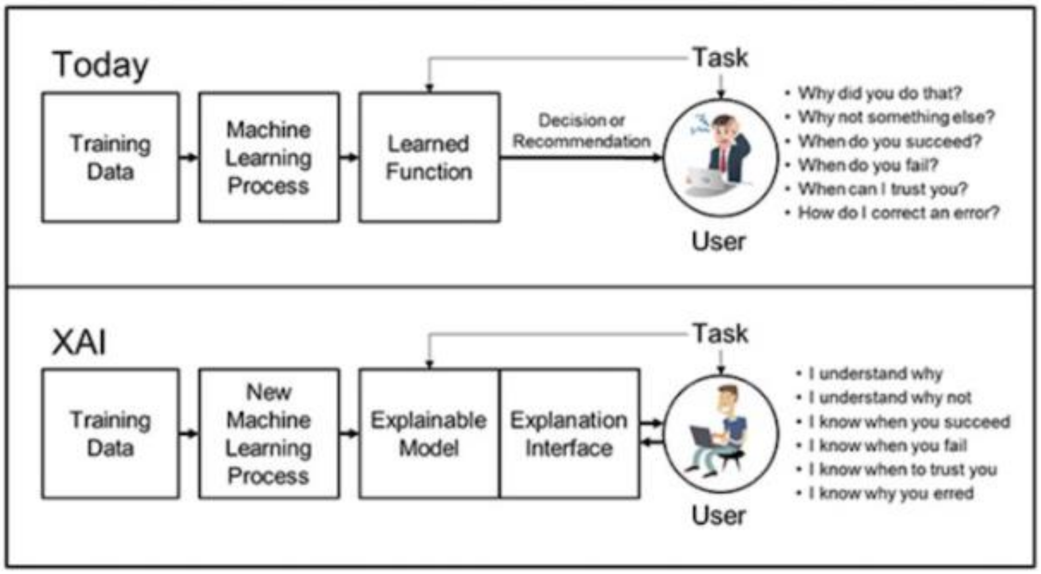


Figure 1 Need of Explainable AI (XAI)

1. Human determined features

We assume that we have human-described variables (and the values that they can take) for an input image.

2. Features learnt using deep learning

Here the raw input (scanned images) are processed by a network to learn a representation, e.g., by means of a several convolutional network and pooling layers. The training could be performed by either by unsupervised learning (e.g., an autoencoder) or by supervised learning

3. Explainable features learnt using deep learning

These are the representation learnt using deep learning is as similar as possible to the human explainable features described in the first part. Ideally the representation learnt by the deep learner is the same as the human explainable features. This can be learnt using supervised learning where the desired outputs are the explainable features.

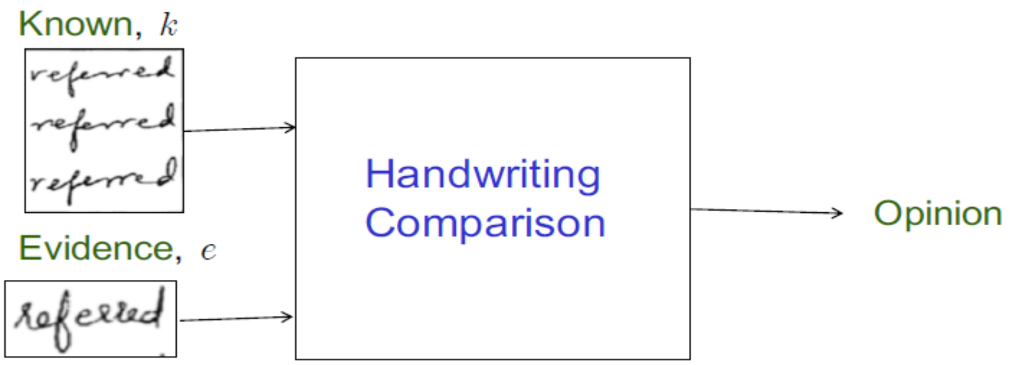


Figure 2 Handwriting Comparison Task

1.2 Dataset

In this project we are going to be working on the AND dataset and we are going to create the Bayesian model on the AND dataset.

Data containing handwritten 'AND' images. The details of the AND dataset is as follows:

• Every writer wrote three different pages namely a,b,c.

• Number of writers: 1552

• Number of images: 15260

• Filename Format: XXXXY_numZ.png [where XXXX is writer number, Y is page number, Z is occurrence of 'AND' in Y]

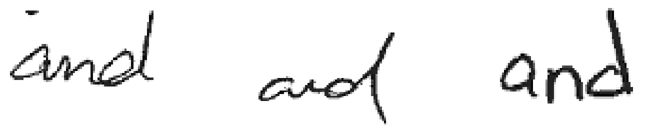


Figure 3 Various images depicting different structures of AND

For every image of AND in our dataset, we have 15 features corresponding to it, and for every feature we have a feature class associated to it. The 15 features are defined as follows:

Pen Pressure – Pressure applied when writing AND letter.

Letter Spacing – Letter spacing is the distance or the spacing between two letters in the AND word

Size – Size is dependent on dimension and letter spacing.

Dimension – We can classify dimension into high-dimension when the height of the characters are bigger than the width and low-dimension, otherwise low or medium.

Is LowerCase – If sample image is in lowercase or not.

Is Continuous – Is Continuous is dependent on IsLowerCase

Slantness – Can have the classes slight right, very right,left and no slant.

Tilt – Tilt feature is dependent on Slantness.

Entry stroke "A" – Entry stroke of "A" can be nostroke or downstroke.

Staff of "A" – Can have classes like retraced, loopy, teneted or no staff.

Formation "N" – Generally depends on word formation which is depenedent on constancy and size.

Staff of "D" - Staff of "D" is dependent on Staff of "A" and can have retraced, loopy and no staff classes.

Exit stroke "D" – Exit stroke "D" is dependent on Entry stroke "A".

Word formation -Word formation is whether the word has been formed correctly or not and is dependent on constancy .

Constancy – Constancy is dependent on Size which is dependent on dimension and letter spacing.

These 15 features were used in the image and processing was done on only these 15 features.We also had to generate these features in the Task 1 i.e. the Data Annotation task wherein we labelled images for 15 features.

2 Task 1 - Data Annotation

In the first task of project 2 we had to do the annotation task which basically involved annotating the “AND” images to generate handcrafted features.
We had to annotate 150 images over all the 15 features according to our perception and what we saw in the image.

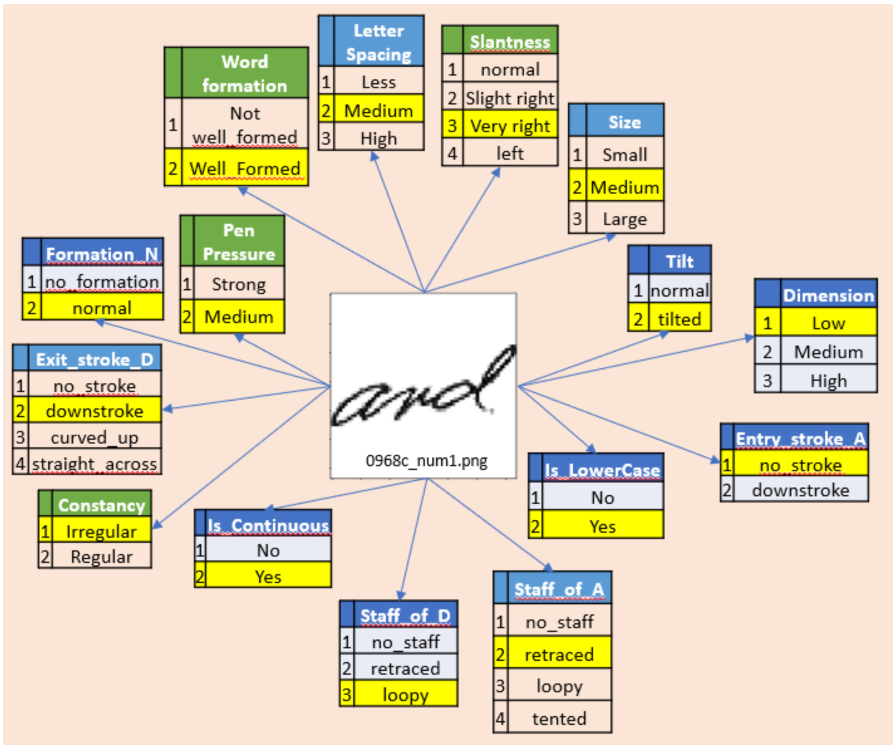


Figure 4 Example of Data Annotation Task

An example of the data annotation task is illustrated in the figure 4. Here for a sample image ‘0968c_num1.png’ we have labeled all the 15 features. For instance, since the word ‘AND’ is in lowercase we have classified Is_Lowercase as Yes and similarly for all the rest of the features.

The relevance of the data annotation task with the project of Explainable AI is that we have to create the dataset as the dataset for 15 features for all the images does not exist and creating the dataset will help us in the predictions as well as serve as the input.

Moreover, the task of data annotation helped us in getting to know the data better so that we are able to create models using our intuition and create graphical models based on the features as we have done in Task 2.

3 Task 2 - Bayesian inference

In this task we are asked to build multiple Bayesian models for task of verification. Then we will query these models using the inference to get the accuracy and the time to infer as the metrics for the model.

Bayesian inference is a method of statistical inference in which Bayes' theorem is used to update the probability for a hypothesis as more evidence or information becomes available. Bayesian inference is an important technique in statistics, and especially in mathematical statistics. Bayesian inference derives the posterior probability as a consequence of two antecedents: a prior probability and a "likelihood function" derived from a statistical model for the observed data.

In this task the first part what I did was I read the 15 features csv file and ran the Hill Climb Search algorithm on the data to generate a probabilistic model. Hill Climbing is heuristic search used for mathematical optimization problems.

Basically, what it does is, given a large set of inputs and a good heuristic function, it tries to find a sufficiently good solution to the problem.

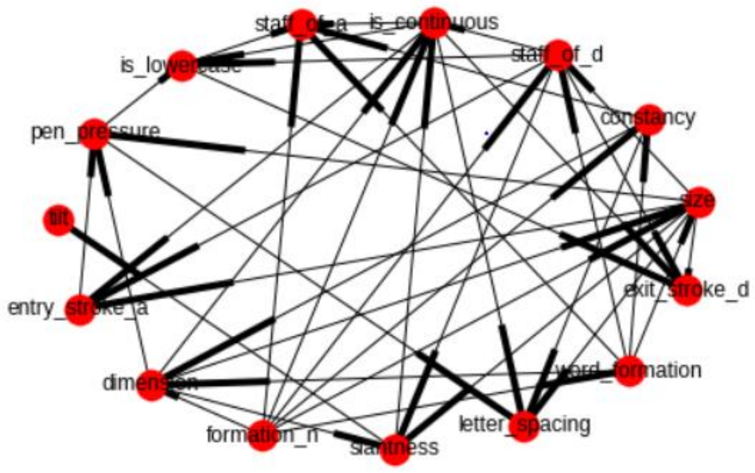


Figure 5 Best Model and its K2 score

K2 score / No of edges	K2 score	No of edges
Model 1	-139940.602501	42
Model 2	-151557.755695	13
Model 3	-140370.616562	40
Model 4	-140263.626330	37

Feature vs K2 score	9 features	15 features
Model K2 score	-9462.704892371388	-139940.6025016286

Figure 6 K2 score Comparison

Next, I created 3 more models using my intuition by removing or adding edges between different nodes/features. In these models one thing to note is that no loop or cycle should be formed between various nodes. Then I generated the K2 score for all the models that were created and compared the K2 score and took the best model. Figure 5 depicts the best model which is graphically displayed using the networkx library.

The above table labelled as figure 6 is a comparison of the models returned by Hill Climb Search for the 9 features and 15 features. Since both the datasets were available for training, I have calculated the best models for both the datasets and calculated the K2 scores for both the datasets. The table above also shows the K2 score of various models and respective number of edges.

For the task of verification, I have out the inference part of the task using the following approach. Here I have considered my best model of 15 features.

Now using the edges of this model, I have created a separate model of 15 features with feature names as ‘sizej’ and ‘constancyj’ instead of ‘size’ and ‘constancy’. Now I have created a final model and added all of the edges of the two above models and then added the verification node as an edge to 2 features namely constancy and staff_of_d. Verification Node edges are as follows: (constancy','label'), ('constancyj','label'), ('staff_of_d','label'), ('staff_of_dj','label')

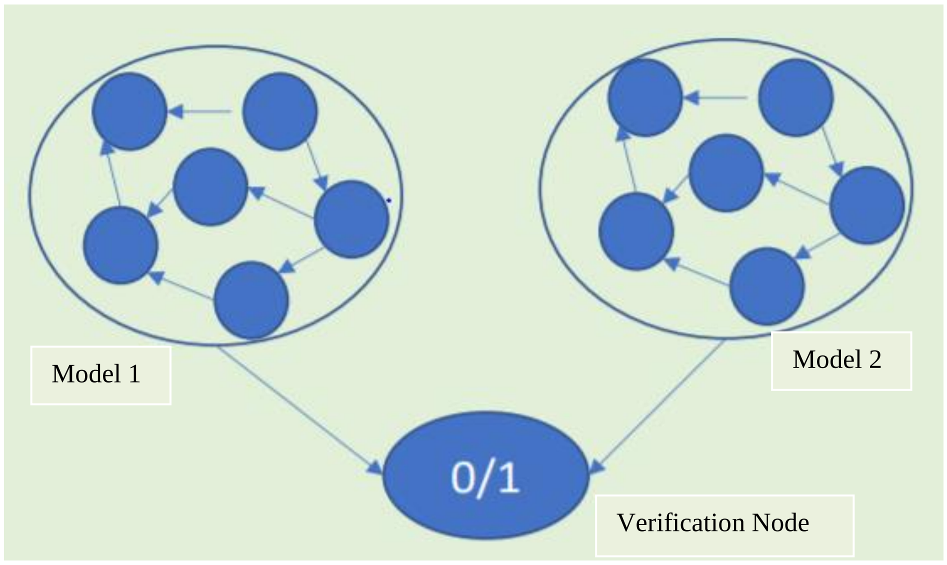


Figure 7 PGM for verification

The metrics for all the models for the task of verification is as follows:

Data used Validation Set	Accuracy (in %)	Time to Train (in seconds)	Time to infer (in seconds)
Seen Dataset	87.2035	0.357197	264.092
Unseen Dataset	78.0224	0.421700	2153.919
Shuffled Dataset	68.664	0.37879	1569.9456

Figure 8 Table for metrics for all the 3 datasets

4 Task 3 – Deep Learning Inference

The requirement for this task was to build a "Siamese network" OR a "classification network" for the task of verification. For this purpose, we first need to do data preprocessing.

For this deep learning model which takes a whole image as input and we then the model trains on the image using the autoencoder and encoder and gets back the image using the decoder. For this purpose, I have used all the images in the training dataset to train the model and all the images in the validation dataset to predict the output.

In that aspect of data preprocessing, we first read all the images in the seen, unseen and shuffled training and validation dataset separately. Then by manually checking the first 4 characters of the filename we check if the pairs are same or not and generate the list of labels.

Now that the preprocessing is done, we create an autoencoder which takes an 64x64 image as input and trains on all the images and then predict on the autoencoder.

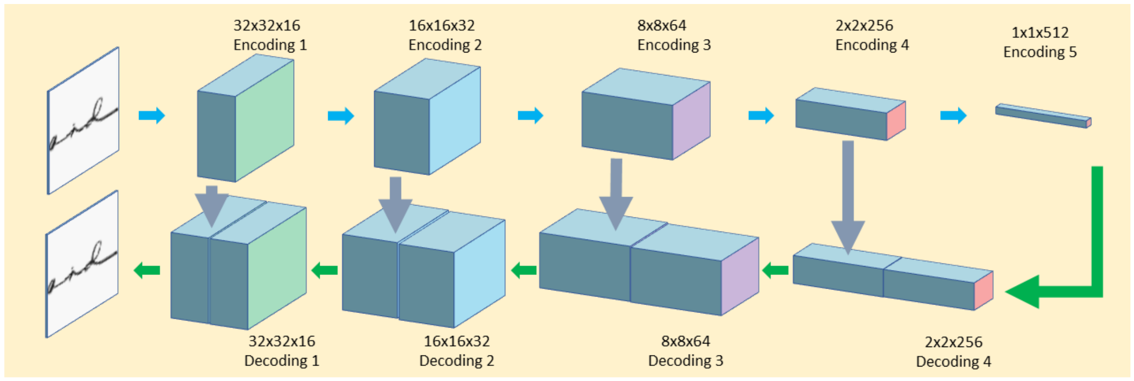


Figure 9 Working of autoencoder

Autoencoders are neural networks trained to reconstruct their original input. An Autoencoder is a form of feature extraction algorithm. We are using a autoencoders which is stacked. Autoencoders are lossy, which means that the decompressed outputs will be degraded compared to the original inputs. Autoencoders are data-specific, which means that they will only be able to compress data similar to what they have been trained on. For example, an autoencoder trained on pictures of faces would do a rather poor job of compressing pictures of trees, because the features it would learn would be face-specific.

Similarly, in our case we the features that the autoencoder learns are specific to handwritten digit images, so our autoencoder will give good output on those types of input.

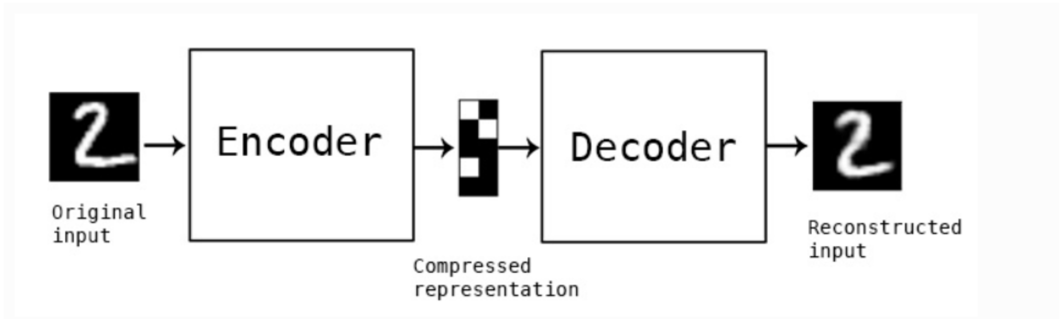


Figure 10 Example of autoencoder, encoder and decoder

The above figure i.e. figure 10 is an example of a simple autoencoder where we send a 64x64 image to it as the input. The autoencoder consists of an encoder and a decoder. The encoder takes the 64x64 image as the input and then compresses it into a 1x512 vector which is the compressed representation of the input image.

Now the decoder takes this intermediate vector as the input and tries to reconstruct the original image. This is how training and prediction on the autoencoder takes place.

Now to compare the predicted images we use the concept of cosine similarity.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

Figure 11 Cosine Similarity

Cosine similarity is a metric used to determine how similar the documents are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. We keep a threshold and if the cosine similarity of two images is greater than the threshold we label those two images as 1 or same. In this way we have generated the list of predicted labels.

Now we compare these predicted labels to the original labels and then we calculate the accuracy. The accuracy and the graphs for loss are plotted below.

Data used	Training Loss	Validation Accuracy (in %)	Validation Loss	Training Accuracy (in %)
Seen Dataset	0.1161	86.2	0.1143	88.7
Unseen Dataset	0.1161	83.6	0.1067	88.3
Shuffled Dataset	0.1169	85	0.1372	87.2

Figure 12 Table for metrics for all the 3 datasets

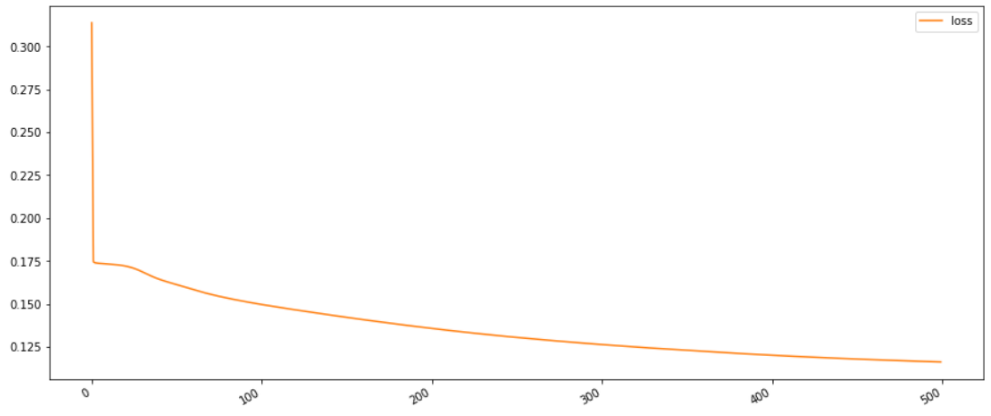


Figure 13 Training loss for Seen dataset

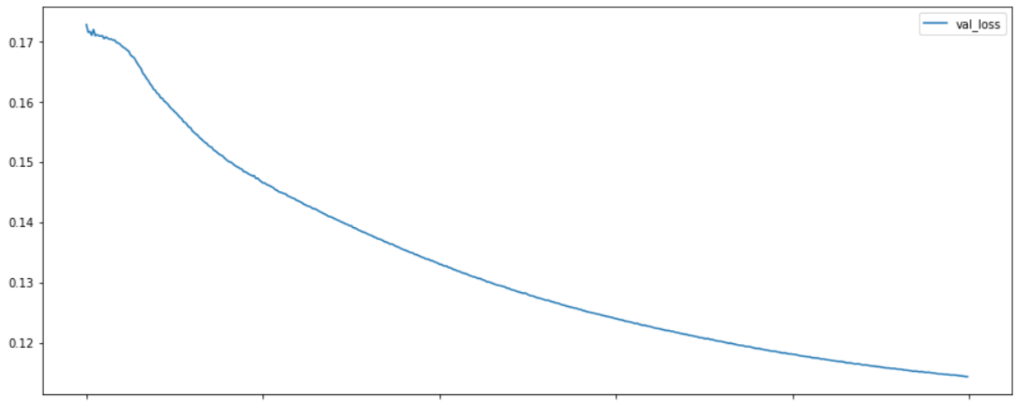


Figure 14 Validation loss for Seen dataset

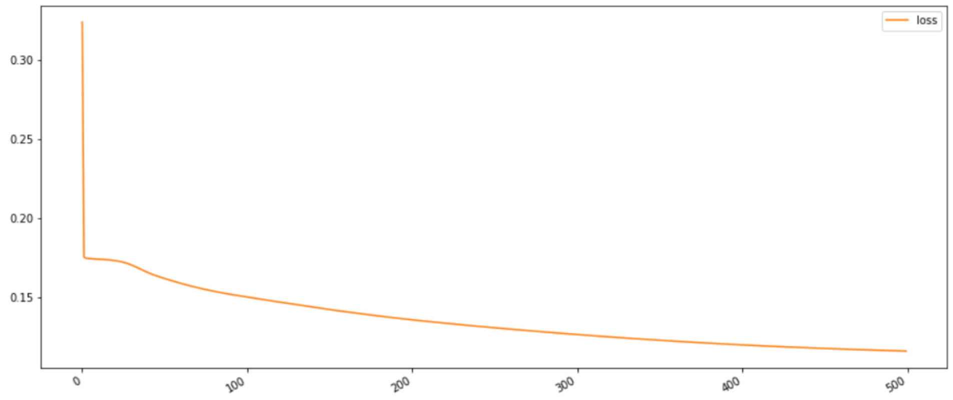


Figure 15 Training loss for Unseen dataset

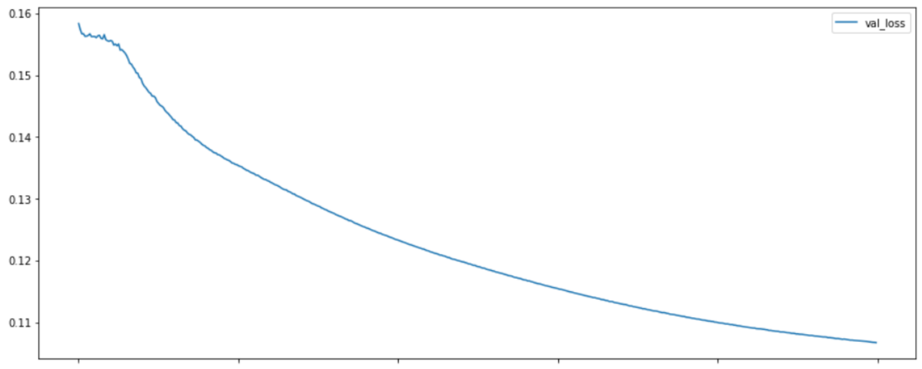
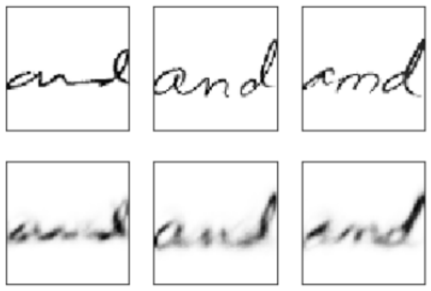


Figure 16 Validation loss for Unseen dataset



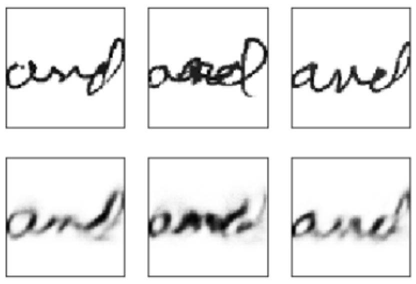


Figure 17 and Figure 18 Predicted images for seen and unseen respectively

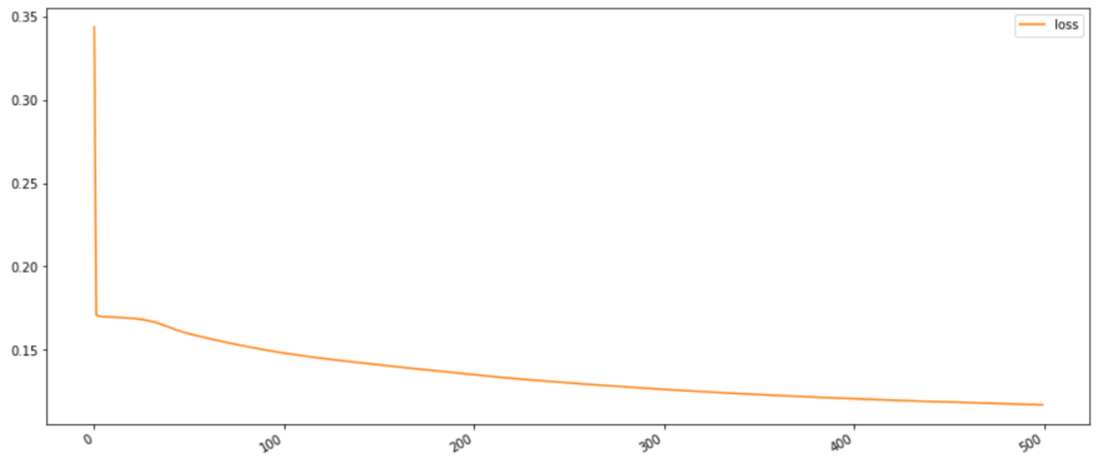


Figure 19 Training loss for Shuffled dataset

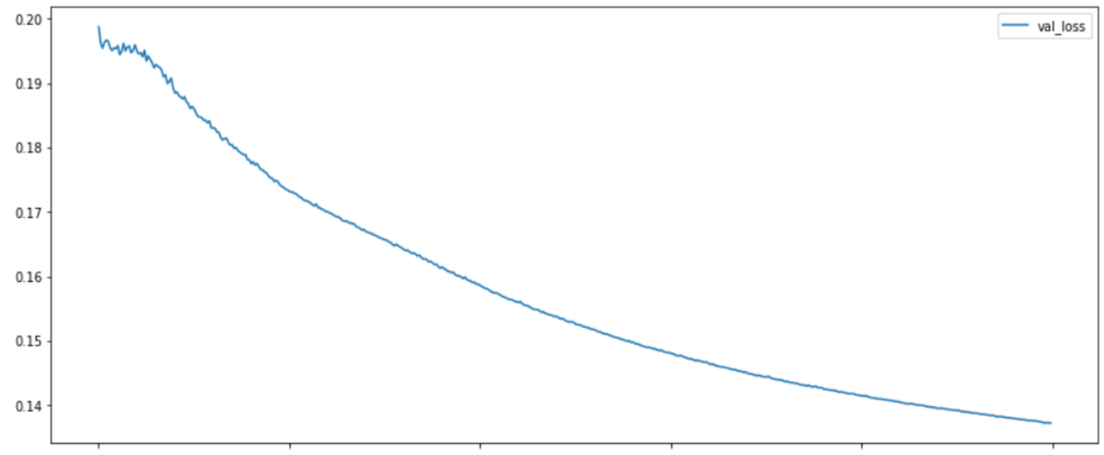


Figure 20 Validation loss for Shuffled dataset

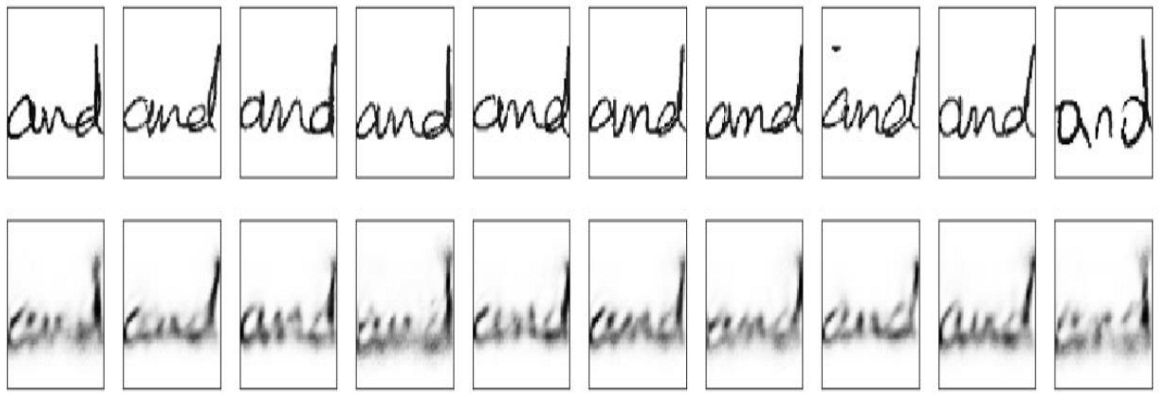


Figure 21 Predicted images for Shuffled dataset

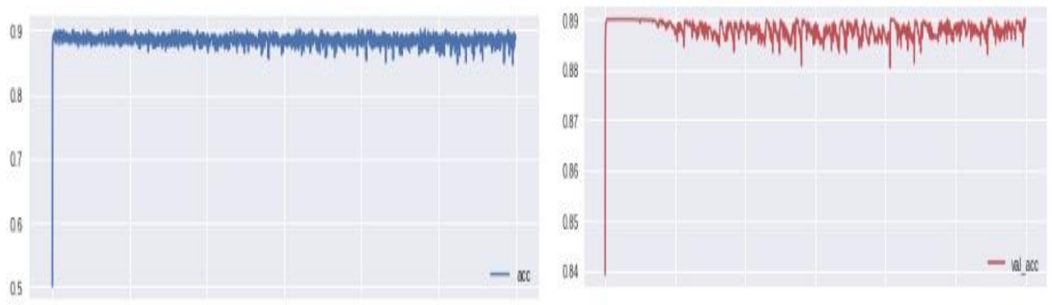


Figure 22 Training and Validation Accuracy for seen dataset

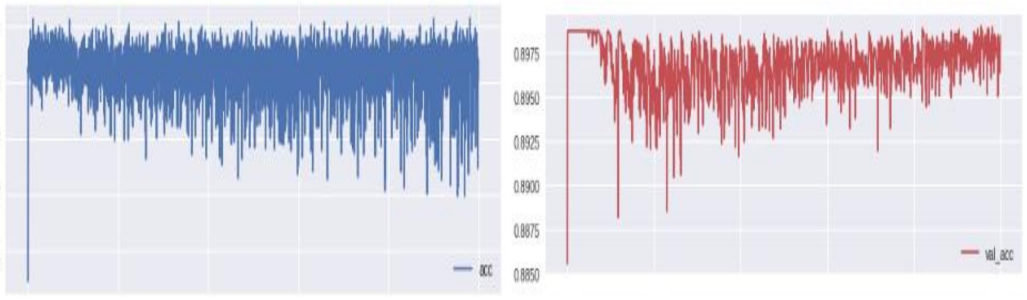


Figure 23 Training and Validation Accuracy for unseen dataset

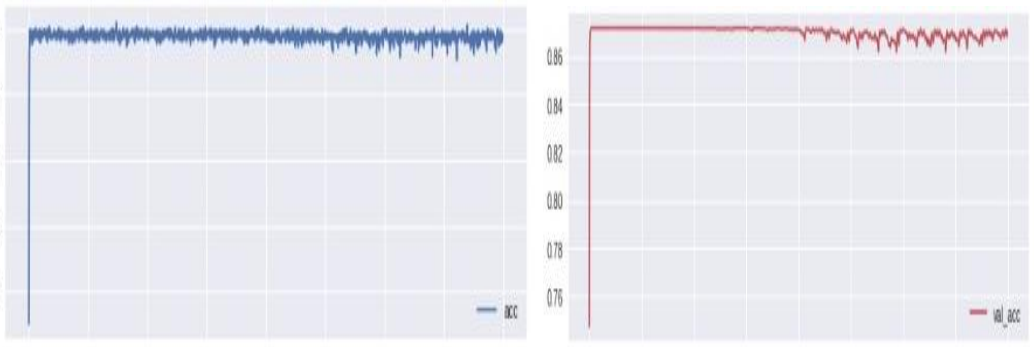


Figure 24 Training and Validation Accuracy for shuffled dataset

5 Task 4 – Explainable AI

The requirement for this task was to build a Multitask Learning (MTL) model to learn the mapping between “AND” images and handcrafted features. The task is to classify the image to identify a feature value and then explain why the model gave such a result.

For the purpose of explainable AI, we use a multi task model. Multi-task learning (MTL) is a subfield of machine learning in which multiple learning tasks are solved at the same time, while exploiting commonalities and differences across tasks. This can result in improved learning efficiency and prediction accuracy for the task-specific models, when compared to training the models separately.

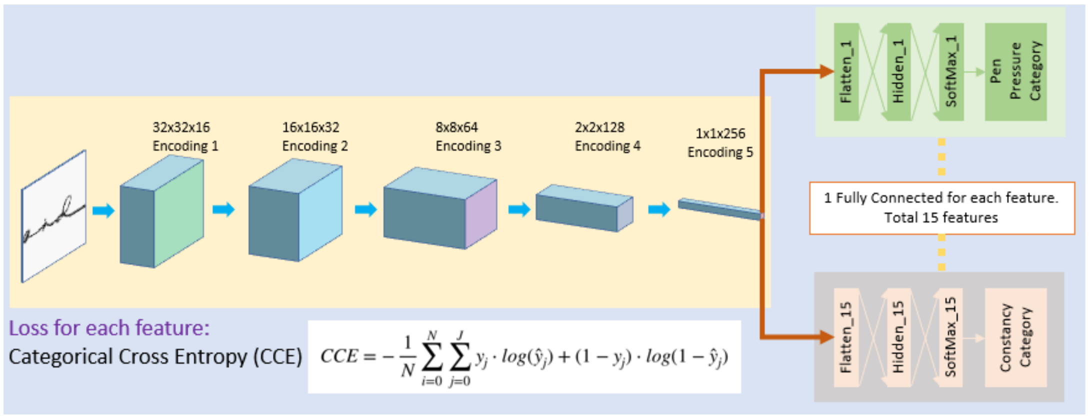


Figure 25 MTL model using frozen layers

In our model we use a variation of fine-tuning where part of the convolutional layers is frozen to prevent overfitting, and only top fully connected layers are finetuned. This is a compromise between finetuning and feature extraction. Basically, in a pretrained model you are freezing the earlier layers by making the weights unchangeable. This is so it can retain the already learned basic representations. The later layers are not frozen so that they can learn the representations specific to your task.

So basically, we create a set of frozen layers so that we prevent overfitting and generalize the model. Also, by freezing a subset of all the layers we also save on computation time as backpropagation is not needed for all the frozen layers and fitting the model is much faster as compared to all convolutional layers.

Now to introduce the concept of exploitability, we have used SoftMax as the final layer for the 15 features and then compare the two images using cosine similarity to obtain feature wise explanation.

Multi-task learning works because regularization induced by requiring an algorithm to perform well on a related task can be superior to regularization that prevents overfitting by penalizing all complexity uniformly. So basically, by sharing representations between related tasks, we can enable our model to generalize better on our original task.

Data used	Training Loss	Validation Accuracy (in %)	Validation Loss
Seen Dataset	5.8	76.5	6.31
Unseen Dataset	6.5	81.3	6.3
Shuffled Dataset	7.3	75.2	6.3

Figure 26 Table for metrics for all the 3 datasets

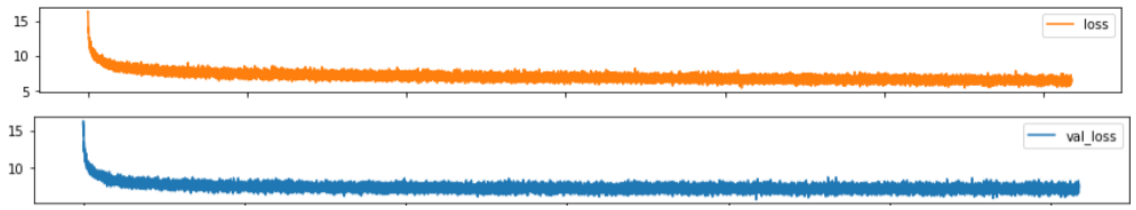


Figure 27 Training and Validation curve for seen data

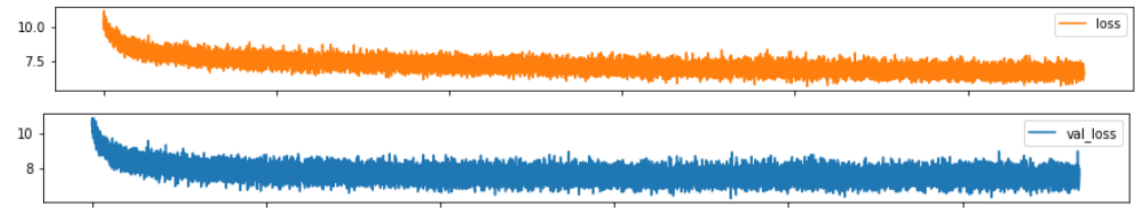


Figure 28 Training and Validation curve for unseen data

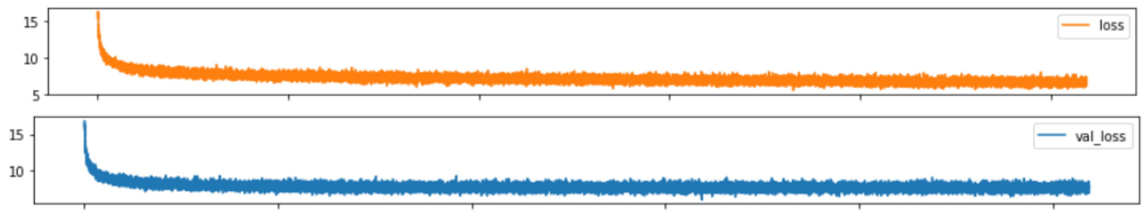


Figure 29 Training and Validation curve for shuffled data

References

[1]-https://medium.freecodecamp.org/an-introduction-to-explainable-ai-and-why-we-need-it-a326417dd000

[2]- https://simmachines.com/explainable-ai/

[3]- https://en.wikipedia.org/wiki/Explainable_artificial_intelligence

[4]- https://en.wikipedia.org/wiki/Bayesian_inference

[5]- https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/

[6]- https://blog.keras.io/building-autoencoders-in-keras.html

[7]- https://www.darpa.mil/program/explainable-artificial-intelligence

[8]- https://en.wikipedia.org/wiki/Multi-task_learning

[9]- https://arxiv.org/pdf/1606.09282.pdf