Explainable AI : Writer Verification

Parth Shah
SEAS, University at Buffalo
Buffalo, NY, 41214
Person Number: 50291125
parthnay@buffalo.edu

Abstract

The project aims 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.

1 Introduction

The given problem is an inference problem. We use PGM to recognize the patterns in the trigram'*and*' samples as seen by 3 datasets viz - Seen, Unseen and Shuffled Datasets. We use AutoEncoders and Neural Networks to then create explainable features from these samples as seen from other tasks. We report the similarity in the original sample and the predicted features for the same image as seen by our Network Model. The project is divided into 4 tasks

- ➔ Task 1: Dataset annotation
- ➔ Task 2: Bayesian Inference of Handcrafted Features
- ➔ Task 3: Deep Learning using "AND" images
- ➔ Task 4: Explainable AI – Deep Learning + PGM

2 Datasets and Data Preprocessing

We were provided with a 15-Feature.csv that described '*and*' image samples with 15 features and their values as annotated from task1. A total of 13570 image samples were given with 15 features describing each. The csv columns were *imagename, pen_pressure, letter_spacing, size, dimension, is_lowercase, is_continuous, slantness, tilt, entry_stroke_a, staff_of_a,formation_n, staff_of_d, exit_stroke_d, word_formation and constancy*. Every image was described into these 15 features.

Value	Features														
	Pen Pressure	Letter Spacing	Size	Dimension	Is LowerCase	Is Continuous	Slantness	Tilt	Entry stroke "A"	Staff of "A"	Formation "N"	Staff of "D"	Exit stroke "D"	Word formation	Constancy
1	Strong	Less	Small	Low	No	No	normal	normal	no stroke	no staff	no formation	no staff	no stroke	Not well formed	Irregular
2	Medium	Medium	Medium	Medium	Yes	Yes	slight right	tilted	downstroke	retraced	normal	retraced	downstroke	Well Formed	Regular
3		High	Large	High			very right			loopy		loopy	curved up		
4							left			tented			straight across		

Along with the feature dataset, we were provided with actual images for other tasks. The images were divided into 3 categories.

1. Seen Dataset
2. UnseenDataset
3. Shuffled Dataset

Each of these contained images divided in TrainingSet and ValidationSet, with a csv describing similar and different writers in each. The images would be used for Task 3 and 4 while the CSV can be used to determine inferences for Task 2.

3 Task 1: Dataset annotation

We were tasked with manually annotating the '*and*' images 150 per student as Task 1 for the project. The annotations were done on UB Ceder Website which provided a well made GUI for the annotation task. The images were to be annotated as given by choosing options from a drop down list. The page can be seen as below

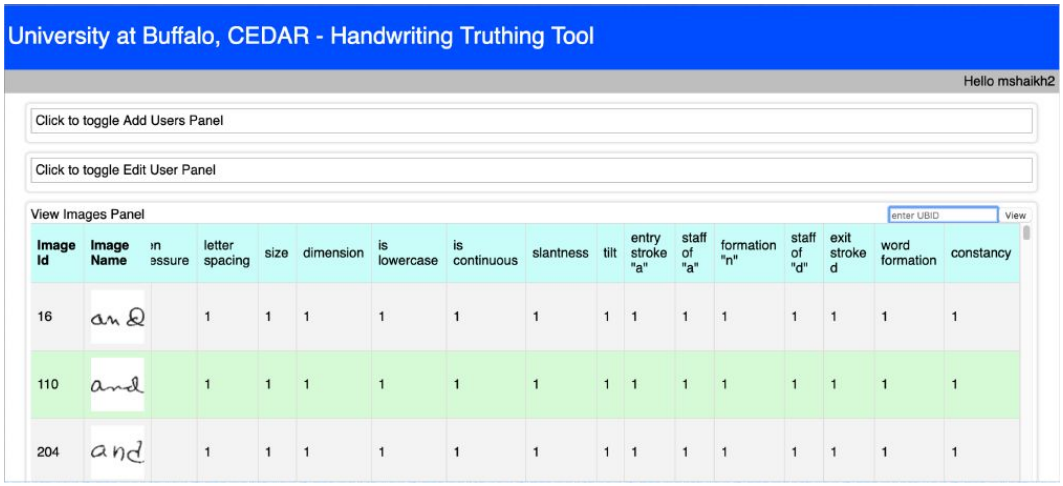


Image Id	Image Name	pressure	letter spacing	size	dimension	is lowercase	is continuous	slantness	tilt	entry stroke "a"	staff of "a"	formation "n"	staff of "d"	exit stroke d	word formation	constancy
16			1	1	1	1	1	1	1	1	1	1	1	1	1	1
110			1	1	1	1	1	1	1	1	1	1	1	1	1	1
204			1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 1. CEDER Handwriting Truthing Tool,

Project2Description.pptx.

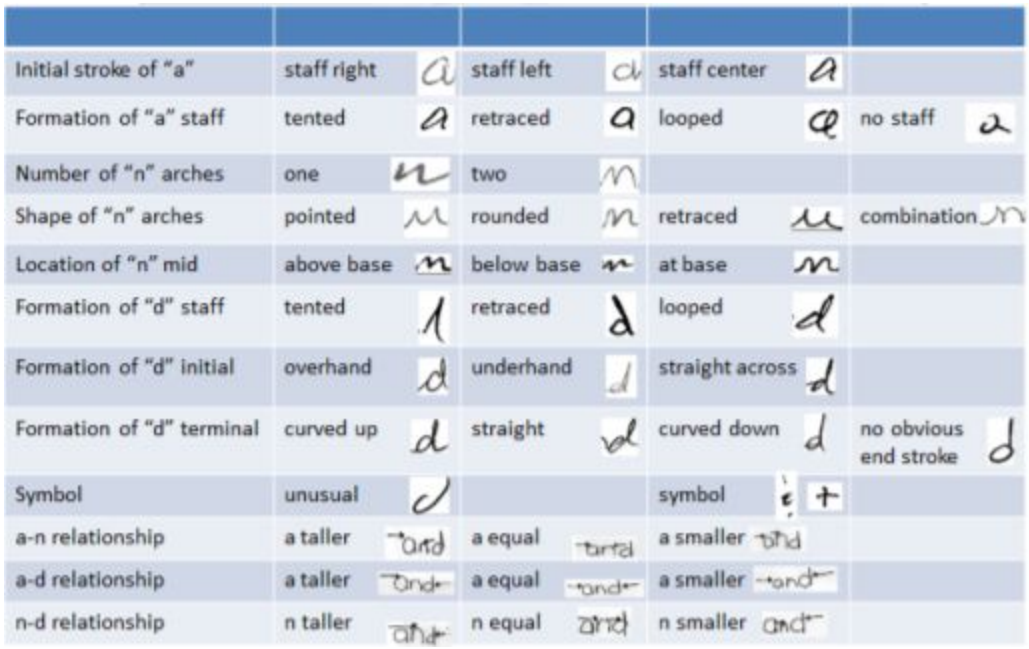


Figure 2. Annotation Labels

Project2 Reference Material IPTES2017

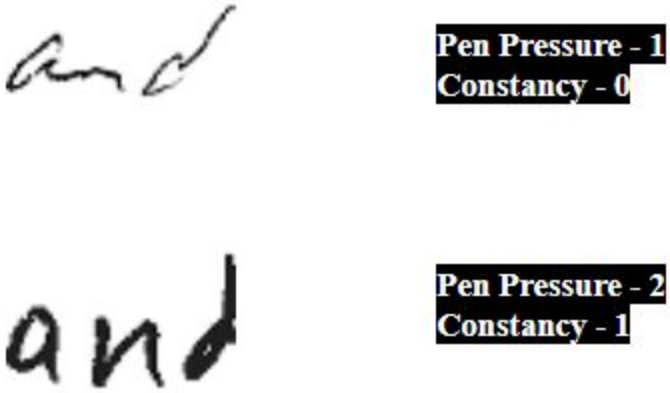


Figure 3. Example Features

The image annotations were based on these example features

4 Task 2: Bayesian Inference of Handcrafted Features

The main aim of this task was to make Bayesian inferences from the Handcrafted Features (15features.csv). We created a PGM model that would take in two image features and then infer if the images were by the same writer or different.

Dataset Creation

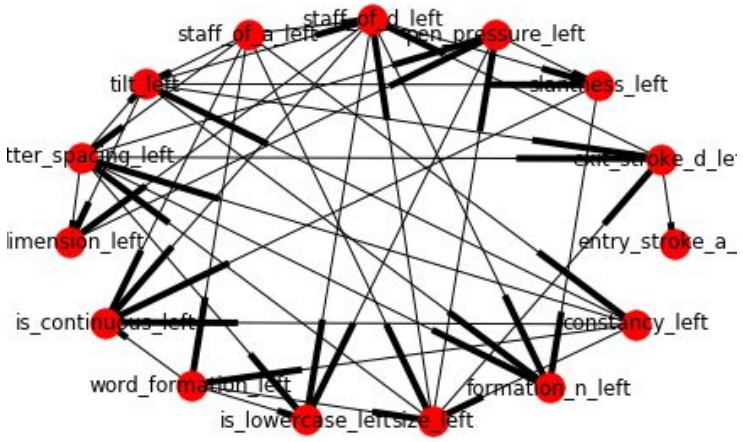
Using the writer pairs given in the seen dataset and the 15features.csv we get pairs of similar writers and their handcrafted features from the 15features data provided. We make a new dataset with pairs of the same writers along with their features and pairs of different writers with their features. The dataset has a column label that is set to 1 if the writers are the same and 0 if they are different. The dataset thus created had 33 columns, one for each image name, 15 for each image feature, one for the label. We generated 111754 data for model fitting.

Bayesian Model Creation

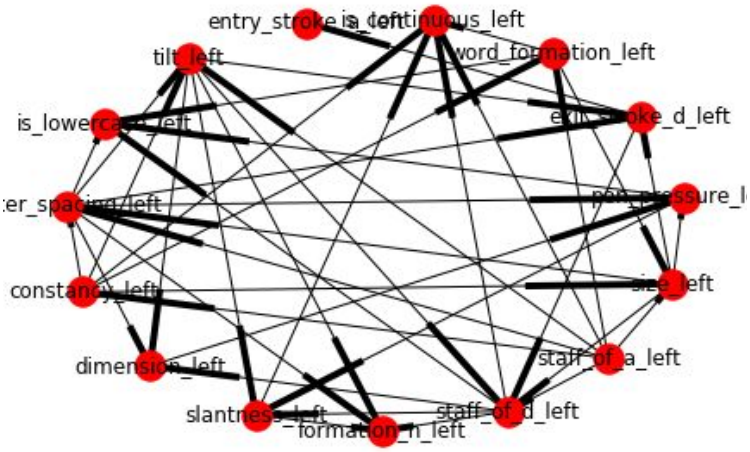
From heuristics and common intuitions we created Bayesian models and tested the models against each other using K2Score as a metric. We also used Hillclimbing to generate the best model and tested it against the handcrafted models.

```
Time To HillClimb : 3.8445698817571006 mins
```

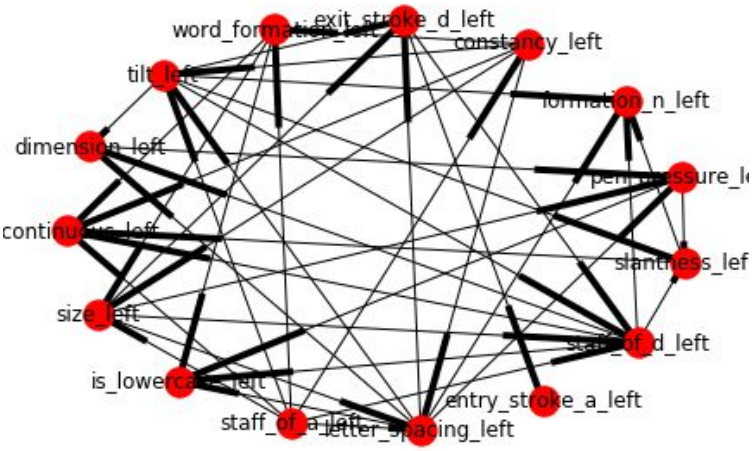
We then took the model with the best K2Score to further construct the model. The Models were generated manually depending upon the independencies in the pairs. The Models thus created were



Model1



Model2



Model3

Model Copying and Final Model Creation

Given the Bayesian Models, we used K2 score as a metric to compare two models. We generated the dataUsing this data we generate a K2Score for each model and the model with the highest K2Score is selected as the best model.

```
The K2 Scores for Models are :
Bayesian Model1 K2 Score is: -143889.5334535705
Bayesian Model3 K2 Score is: -148041.4200154499
Bayesian Model3 K2 Score is: -139940.6025016286
Thus the Best Model is Model3, We will Take this as the bayesian_model from now on.
```

We make a copy of this model and change the node names with the suffixes *_left* for original and *_right* for the copy. We then create a new Bayesian Model that has the edges from both this model. We add a verification node *label*.

We calculate the best nodes to connect to this node by calculating the in-degree for all the nodes, the in degrees can be seen as:

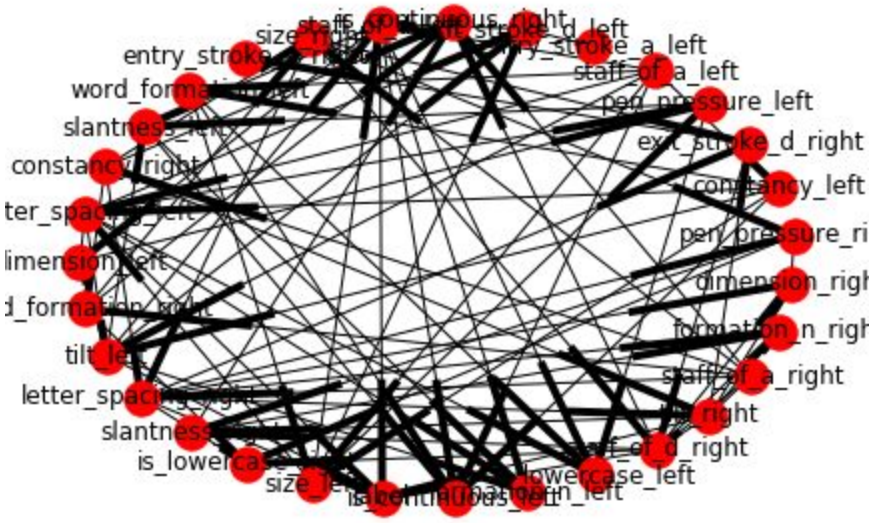
```
pen_pressure_left : 5
slantness_left : 5
is_lowercase_left : 4
is_continuous_left : 5
formation_n_left : 4
letter_spacing_left : 9
exit_stroke_d_left : 5
tilt_left : 8
dimension_left : 4
staff_of_d_left : 9
entry_stroke_a_left : 1
size_left : 7
staff_of_a_left : 7
word_formation_left : 5
constancy_left : 6
pen_pressure_right : 5
slantness_right : 5
is_lowercase_right : 4
is_continuous_right : 5
formation_n_right : 4
letter_spacing_right : 9
exit_stroke_d_right : 5
tilt_right : 8
dimension_right : 4
staff_of_d_right : 9
entry_stroke_a_right : 1
size_right : 7
staff_of_a_right : 7
word_formation_right : 5
constancy_right : 6
```

Thus we connect the nodes with the highest in degrees to the verification node for both the images. We add edges as follows.

```
##Adding Nodes with highest degree

finalModel.add_node('label')
finalModel.add_edge('letter_spacing_left','label')
finalModel.add_edge('letter_spacing_right','label')
finalModel.add_edge('tilt_right','label')
finalModel.add_edge('tilt_left','label')
finalModel.add_edge('staff_of_d_left','label')
finalModel.add_edge('staff_of_d_right','label')
```

The final model that we get after adding the edges can be shown as follows



The Number of edges for the final model is 90.
We use this model to make further inferences.

Inference
Given the Bayesian Models, we used K2 score as a metric to compare two models. We use the dataset created above to fit this model on the data.

```
80% Training Generated..
20% test Generated..
Time To Train : 2.48036527633667 seconds
```

With the model fit on the dataset, we can infer values from the same to find the accuracy of our model. We use MAP Query to query for the labels given two images in the dataset.
The model returns the predicted values for the label which are then compared to the actual values to calculate accuracy. We use a sample of 5000 training and validation data to calculate the accuracies.

```
Time To infer Training Data : 2302.00517868995667 seconds
Accuracy for Training is : 87.0

Time To infer Testing Data : 2597.4149696826935 seconds
Accuracy for Testing is : 85.2
```

Thus we find an accuracy of **87%** and **85.2%** for Training and Testing respectively.
The times taken are also reported above.

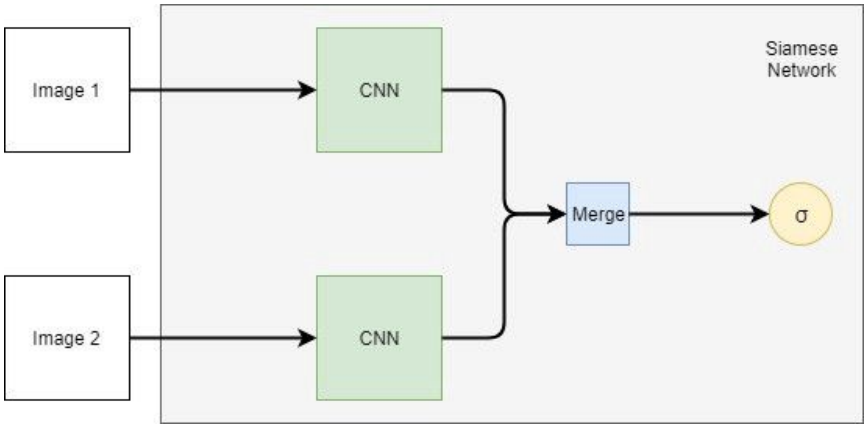
Metric	**BestModel**
Number Of Edges	90
Time to HillClimb	230.64 Seconds
Time To Fit/Train	2.48 Seconds
Time to Infer Training Data(5000)	2302 Seconds
Accuracy for Training Data	87%
Time to Infer Testing Data(5000)	2597 Seconds
Accuracy for Testing Data	85.2%

5 Task 3: Deep Learning using "AND" images

For Task 3 We are required to process the raw input (scanned images) are 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 (e.g., learning whether a pair of samples is by the same writer).
We follow two methods to do the same.

1. **Siamese Networks**

Siamese network is an artificial neural network that use the same weights while working in tandem on two different input vectors to compute comparable output vectors. Often one of the output vectors are precomputed, thus forming a baseline the other output vectors are compared against.[1]



We created a Siamese Network that would take two image samples as input and predict the label as 1 if they are from the same writer and 0 if from different. For this task we were given 3 Dataset viz -: Seen, Unseen and Shuffled each containing a Training Set and a Validation Set.
Our Siamese Network takes input from a datagen that yields image batches in triplets (left, right, label) to train the NN. The datagen is written as

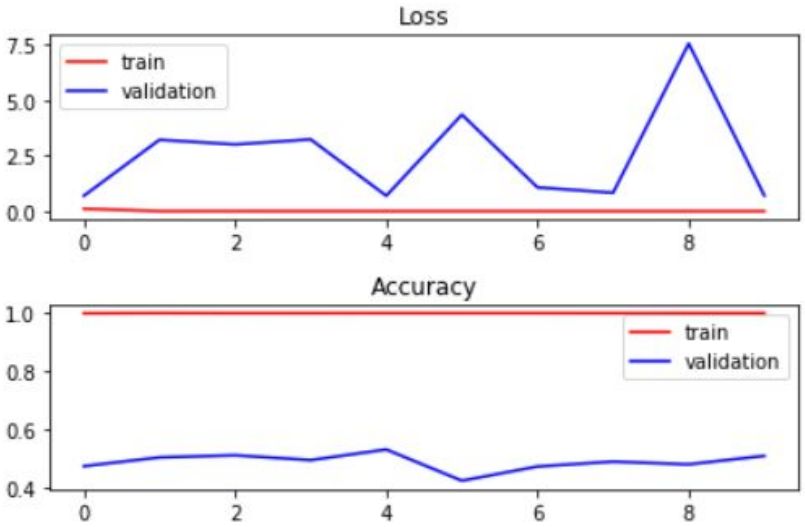
```
def datagen(image_triples, batch_size):
    while True:
        # loop once per epoch
        indices = np.random.randint(0,len(image_triples),batch_size)
        shuffled_triples = [image_triples[ix] for ix in indices]
        num_batches = len(shuffled_triples) // batch_size
        for bid in range(num_batches):
            # loop once per batch
            images_left, images_right, labels = [], [], []
            batch = shuffled_triples[bid * batch_size : (bid + 1) * batch_size]
            for i in range(batch_size):
                lhs, rhs, label = batch[i]
                images_left.append(load_image(lhs))
                images_right.append(load_image(rhs))
                labels.append(label)
            Xlhs = np.array(images_left)
            Xrhs = np.array(images_right)
            Y = np_utils.to_categorical(np.array(labels), num_classes=2)
            yield ([Xlhs, Xrhs], Y)
```

The datagen is used into the fit_generator function for the model as
similarity_model.fit_generator(training_datagen, steps_per_epoch=len(training_data)/batchSize,
epochs=10,
validation_data=validation_datagen,
validation_steps=len(validation_data)/batchSize)
The model produces results according to the dataset provided.

Seen Dataset Results

```
Epoch 1/10
1788/1787 [==============================] - 141s 79ms/step - loss: 0.1090 - acc: 0.9989 - val_loss: 0.6946 - val_acc: 0.4740
Epoch 2/10
1788/1787 [==============================] - 118s 66ms/step - loss: 0.0040 - acc: 0.9998 - val_loss: 3.2207 - val_acc: 0.5042
Epoch 3/10
1788/1787 [==============================] - 119s 67ms/step - loss: 0.0046 - acc: 0.9993 - val_loss: 3.0108 - val_acc: 0.5115
Epoch 4/10
1788/1787 [==============================] - 121s 68ms/step - loss: 0.0036 - acc: 0.9994 - val_loss: 3.2376 - val_acc: 0.4948
Epoch 5/10
1788/1787 [==============================] - 122s 68ms/step - loss: 0.0029 - acc: 0.9996 - val_loss: 0.6948 - val_acc: 0.5312
Epoch 6/10
1788/1787 [==============================] - 120s 67ms/step - loss: 0.0030 - acc: 0.9995 - val_loss: 4.3552 - val_acc: 0.4240
Epoch 7/10
1788/1787 [==============================] - 119s 66ms/step - loss: 0.0027 - acc: 0.9995 - val_loss: 1.0729 - val_acc: 0.4729
Epoch 8/10
1788/1787 [==============================] - 118s 66ms/step - loss: 0.0027 - acc: 0.9994 - val_loss: 0.8309 - val_acc: 0.4896
Epoch 9/10
1788/1787 [==============================] - 119s 67ms/step - loss: 0.0020 - acc: 0.9996 - val_loss: 7.5579 - val_acc: 0.4802
Epoch 10/10
1788/1787 [==============================] - 119s 66ms/step - loss: 0.0016 - acc: 0.9996 - val_loss: 0.7055 - val_acc: 0.5094
```

Epochs for SeenDataset



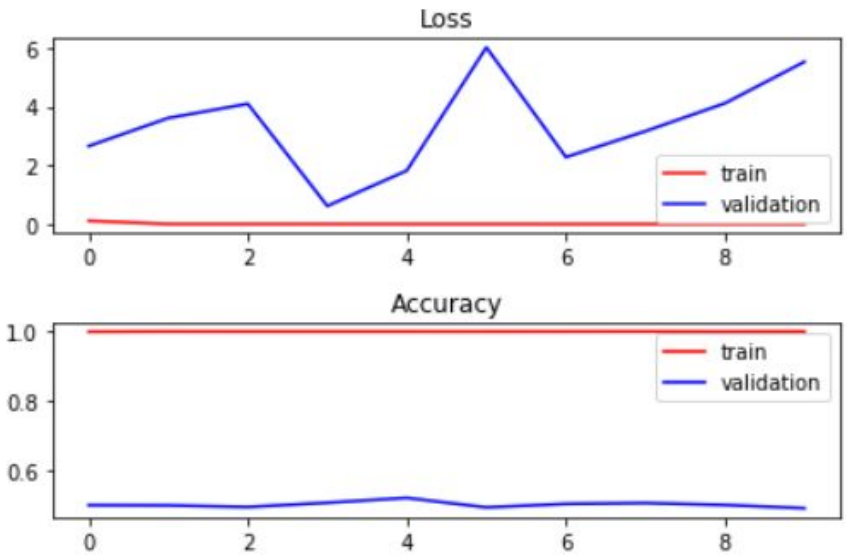
Stats For SeenDataset

Thus we see a **Training loss: 0.0016, Training Accuracy : 0.9996, Validation Loss: 0.7055 - Validation Accuracy : 0.5094**

Unseen Dataset Results

```
Epoch 1/10
2031/2030 [==============================] - 98s 48ms/step - loss: 0.1108 - acc: 0.9991 - val_loss: 2.6577 - val_acc: 0.5011
Epoch 2/10
2031/2030 [==============================] - 96s 47ms/step - loss: 0.0039 - acc: 0.9996 - val_loss: 3.6124 - val_acc: 0.5005
Epoch 3/10
2031/2030 [==============================] - 97s 48ms/step - loss: 0.0032 - acc: 0.9995 - val_loss: 4.0963 - val_acc: 0.4958
Epoch 4/10
2031/2030 [==============================] - 96s 47ms/step - loss: 0.0026 - acc: 0.9996 - val_loss: 0.6177 - val_acc: 0.5081
Epoch 5/10
2031/2030 [==============================] - 98s 48ms/step - loss: 0.0023 - acc: 0.9995 - val_loss: 1.8234 - val_acc: 0.5219
Epoch 6/10
2031/2030 [==============================] - 99s 49ms/step - loss: 0.0021 - acc: 0.9995 - val_loss: 6.0179 - val_acc: 0.4947
Epoch 7/10
2031/2030 [==============================] - 97s 48ms/step - loss: 0.0014 - acc: 0.9996 - val_loss: 2.2824 - val_acc: 0.5049
Epoch 8/10
2031/2030 [==============================] - 98s 48ms/step - loss: 0.0011 - acc: 0.9997 - val_loss: 3.1644 - val_acc: 0.5074
Epoch 9/10
2031/2030 [==============================] - 97s 48ms/step - loss: 0.0021 - acc: 0.9995 - val_loss: 4.1156 - val_acc: 0.5017
Epoch 10/10
2031/2030 [==============================] - 98s 48ms/step - loss: 0.0011 - acc: 0.9997 - val_loss: 5.5289 - val_acc: 0.4923
```

Epochs for UnseenDataset



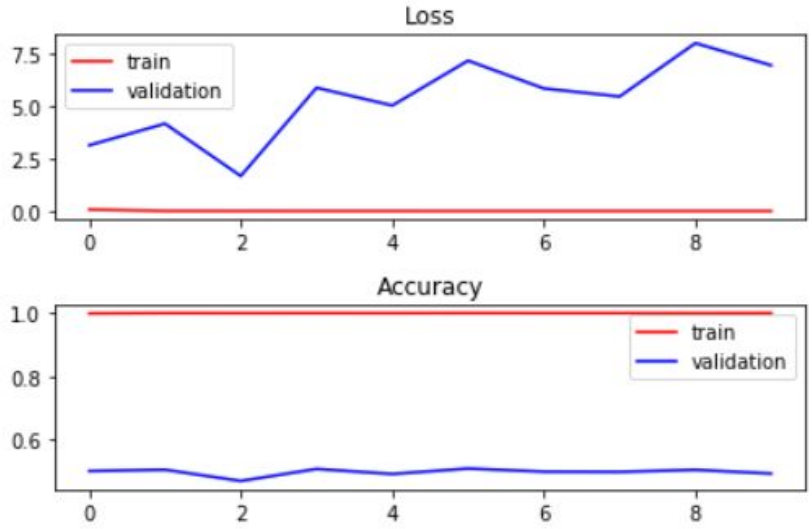
Stats For UnseenDataset

Thus we see a **Training loss: 0.0011, Training Accuracy: 0.9997, Validation Loss: 5.5289 - Validation Accuracy: 0.4923**

Shuffled Dataset Results

```
Epoch 1/10
1599/1598 [==============================] - 367s 229ms/step - loss: 0.0698 - acc: 0.9986 - val_loss: 3.1221 - val_acc: 0.5011
Epoch 2/10
1599/1598 [==============================] - 440s 275ms/step - loss: 0.0050 - acc: 0.9995 - val_loss: 4.1478 - val_acc: 0.5053
Epoch 3/10
1599/1598 [==============================] - 433s 271ms/step - loss: 0.0033 - acc: 0.9995 - val_loss: 1.6631 - val_acc: 0.4699
Epoch 4/10
1599/1598 [==============================] - 439s 275ms/step - loss: 0.0025 - acc: 0.9995 - val_loss: 5.8517 - val_acc: 0.5077
Epoch 5/10
1599/1598 [==============================] - 438s 274ms/step - loss: 0.0024 - acc: 0.9995 - val_loss: 5.0093 - val_acc: 0.4919
Epoch 6/10
1599/1598 [==============================] - 440s 275ms/step - loss: 0.0015 - acc: 0.9996 - val_loss: 7.1398 - val_acc: 0.5090
Epoch 7/10
1599/1598 [==============================] - 439s 274ms/step - loss: 0.0021 - acc: 0.9995 - val_loss: 5.8178 - val_acc: 0.4992
Epoch 8/10
1599/1598 [==============================] - 436s 273ms/step - loss: 0.0016 - acc: 0.9995 - val_loss: 5.4373 - val_acc: 0.4985
Epoch 9/10
1599/1598 [==============================] - 438s 274ms/step - loss: 0.0024 - acc: 0.9994 - val_loss: 7.9695 - val_acc: 0.5049
Epoch 10/10
1599/1598 [==============================] - 437s 273ms/step - loss: 0.0016 - acc: 0.9997 - val_loss: 6.9157 - val_acc: 0.4932
```

Epochs for ShuffledDataset



Stats for ShuffledDataset

Thus we see a Training loss: 0.0016, Training Accuracy: 0.9997, Validation Loss: 6.9157 - Validation Accuracy: 0.4932

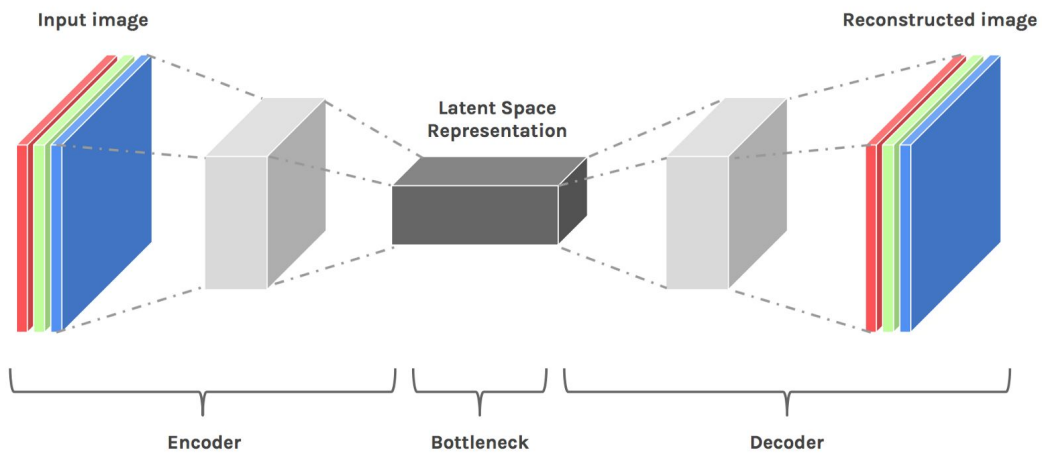
The results can be shown in tabular form as

	Seen	**Unseen**	**Shuffled**
Training Loss	0.0016	0.0011	0.0016
Training Accuracy	0.9996	0.9997	0.9997
Validation Loss	0.7055	5.5289	6.9157
Validation Accuracy	0.5094	0.4923	0.4932

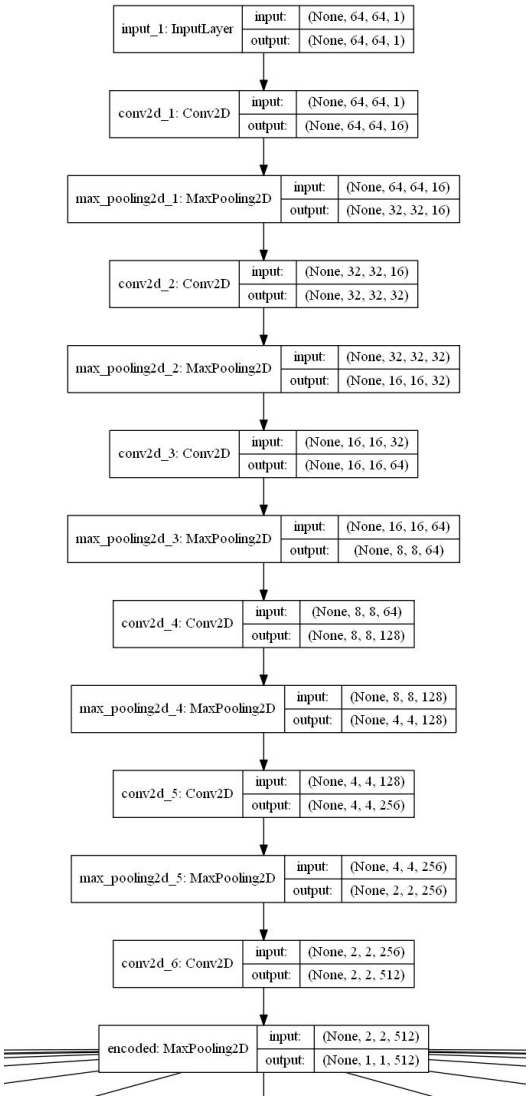
Reference - Neural Network Codes provided in UBLearns by TA's.

2. AutoEncoders

An autoencoder is a type of artificial neural network used to learn efficient data codings in an unsupervised manner. The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore signal "noise." Along with the reduction side, a reconstructing side is learnt, where the autoencoder tries to generate from the reduced encoding a representation as close as possible to its original input, hence its name.[2] The AutoEncoder model can be shown as below.



We create a similar datagen as before just that here one image is passed to the Encoder and we predict the image after it is reconstructed by the Decoder.
The created AutoEncoder can be seen as



The datagen used was provided by the TA's and can be seen below

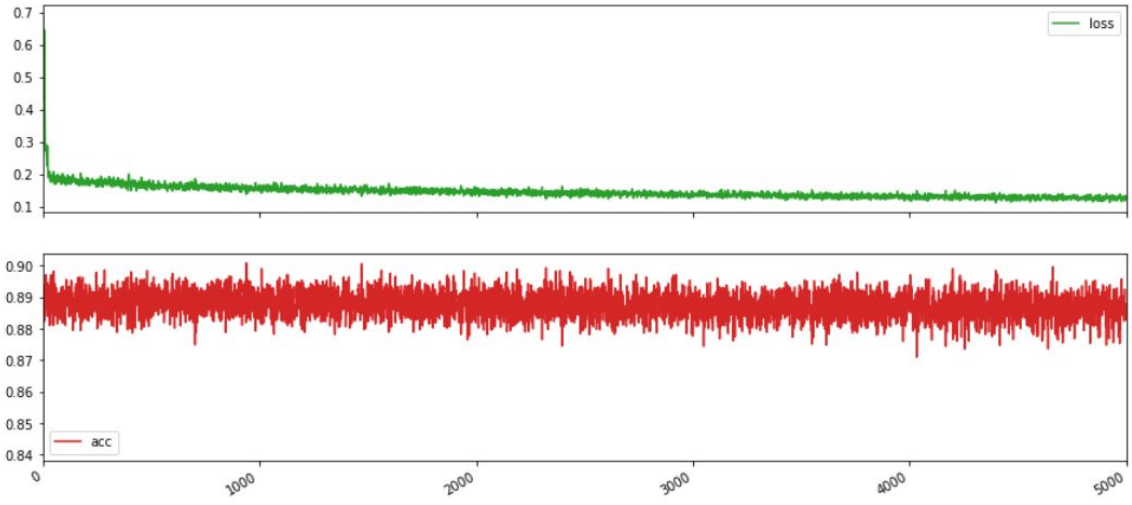
```
def datagen(batch_size,image_path,list_of_writers, hshift = (-64,64) ,vis=False):
    counter = 0
    while True:
        counter = 0
        w_ids,ids = [],[]
        writer_indexes = np.random.randint(0,len(list_of_writers),batch_size)
        x,y,writers = [],[],[]
        for writer_index in writer_indexes:
            writer = list_of_writers[writer_index]
            loaded_image = cv2.imread(os.path.join(image_path,writer),0)
            rand = np.random.randint(hshift[0],hshift[1])
            loaded_image_shifted=np.roll(axis=0,a=loaded_image,shift=rand)
            x.append(255.0-loaded_image_shifted.reshape((64,64,1)))
            y.append(255.0-loaded_image.reshape((64,64,1)))
            w_ids.append(writer[:4])
            writers.append(writer)
            ids.append(writer_index)
            counter+=1
        if counter <= batch_size:
            if vis== True:
                yield np.array(x)/255.0,[np.array(w_ids),np.array(writers)]
            else:
                yield np.array(x)/255.0,np.array(y)/255.0
```

The Encoder outputs a 512 feature vector as a description of the ImageInput. The Decodes takes these 512 features and tries to reconstruct the image back. We see the output images as predicted and actual as seen in the notebook.
We use cosine similarity as a metric to get statistics on the model. Cosine Similarity measures the cosine of the angle between two vectors projected in a multi-dimensional space.[3]
The AutoEncoder as before was tested on the 3 Datasets provided as

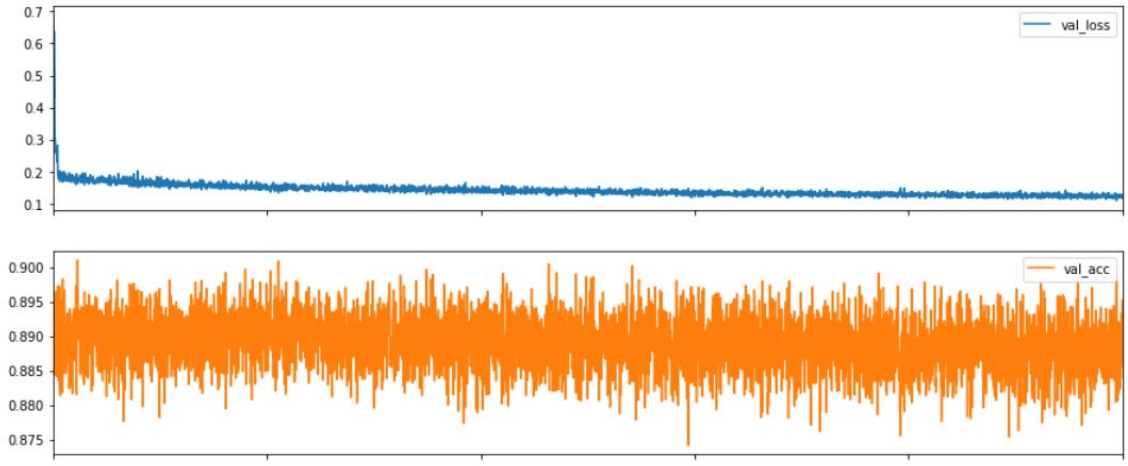
Seen Dataset Results

```
Epoch 05000: val_loss did not improve from 0.11078
```

Final Epoch for AE



Accuracy and Loss for AutoEncoder



Loss and Accuracy for ValidationSet

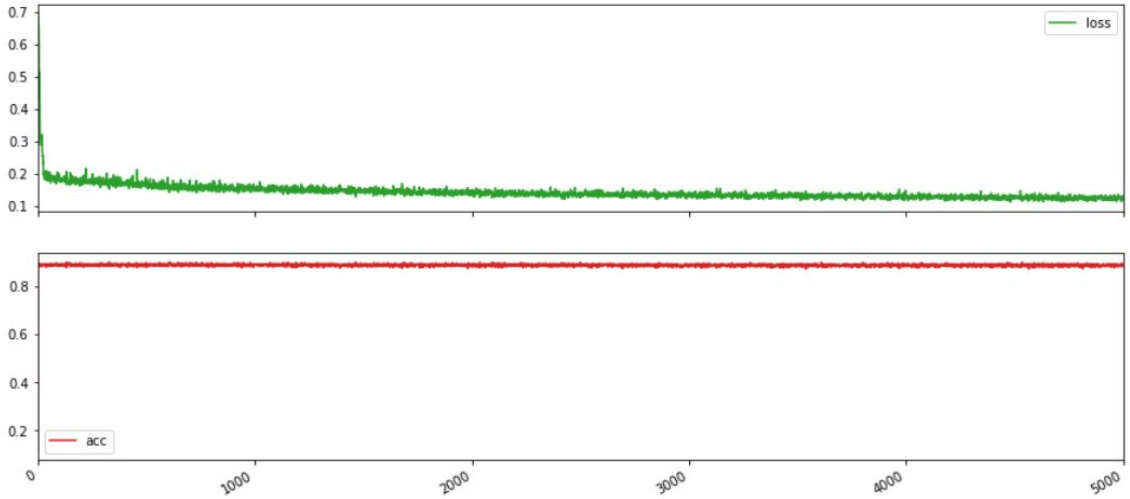
```
precision: 0.021011370912477113
recall: 0.8449617077730206
f1: 0.04100313112691306
Intra Writer Accuracy (Type 1): 0.8506304151648305
Inter Writer Accuracy (Type 2):: 0.9076678465503849
acc: 0.9096004210507757
```

Statistics for AE

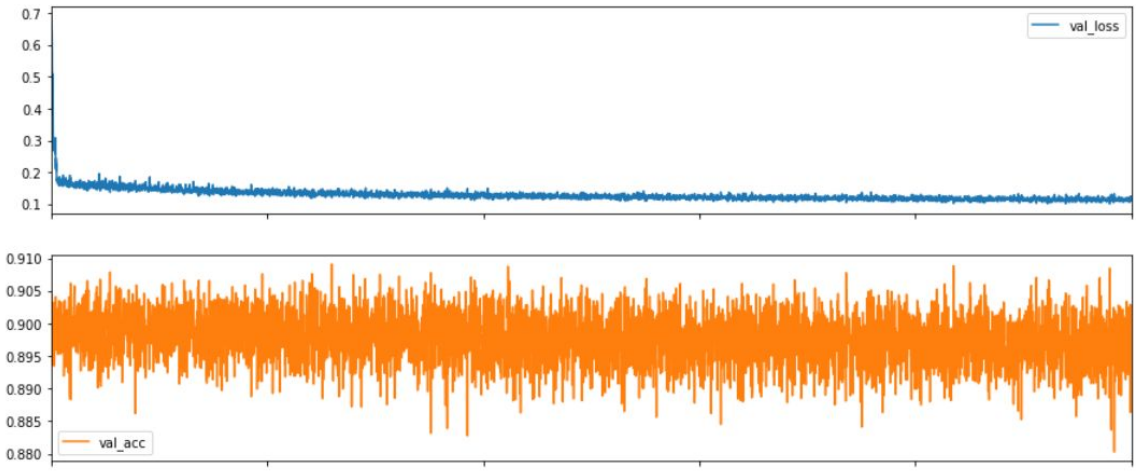
Unseen Dataset Results

```
Epoch 05000: val_loss did not improve from 0.10148
```

Final Epoch for AE



Accuracy and Loss for AutoEncoder



Loss and Accuracy for ValidationSet

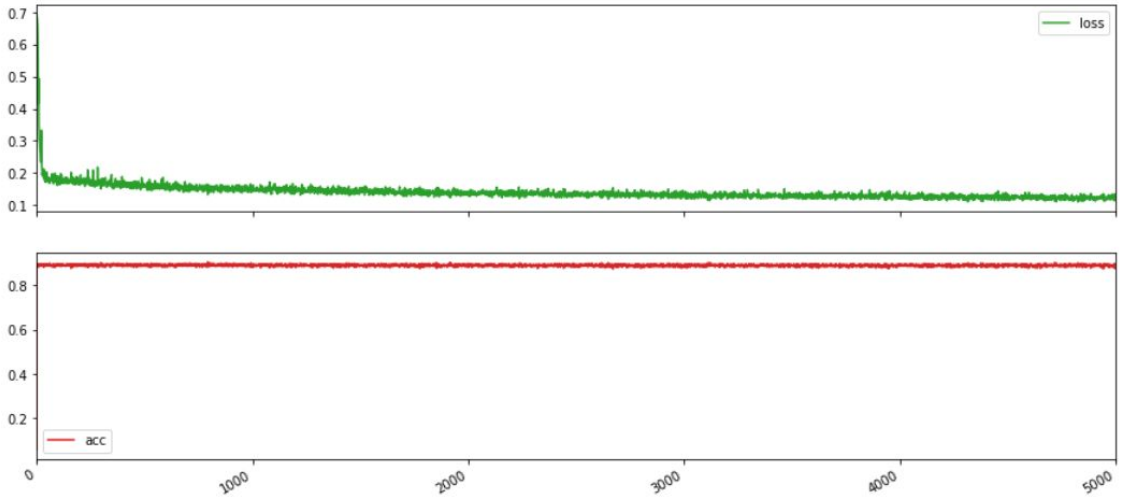
```
precision: 0.0789992143384096
recall: 0.6591873170034992
f1: 0.14108975965859413
Intra Writer Accuracy (Type 1): 0.6684225164609051
Inter Writer Accuracy (Type 2):: 0.9036066788917296
acc: 0.9109228527514014
```

Statistics for AE

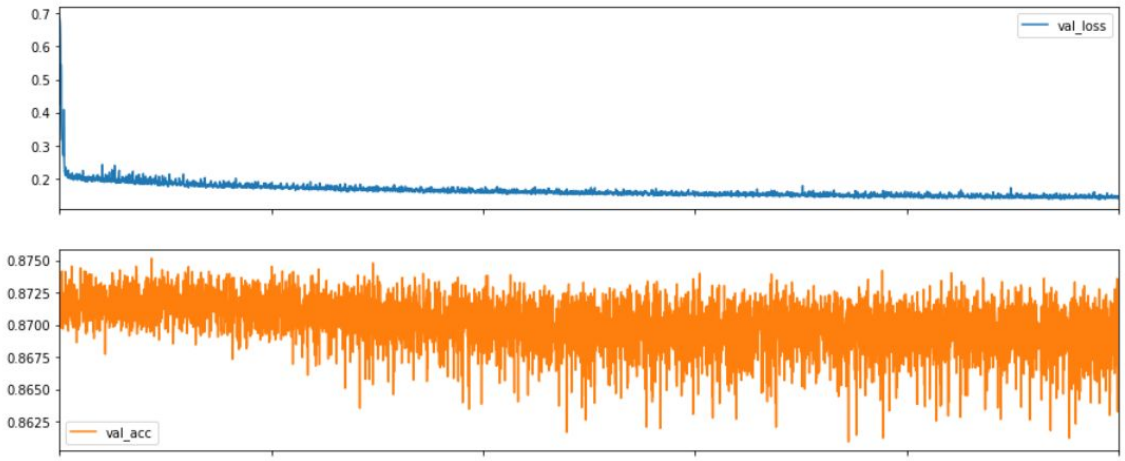
Shuffled Dataset Results

```
Epoch 05000: val_loss did not improve from 0.13765
```

Final Epoch for AE



Accuracy and Loss for AutoEncoder



Loss and Accuracy for ValidationSet

```
precision: 0.017091622808250936
recall: 0.7571914580970787
f1: 0.03342868032268721
Intra Writer Accuracy (Type 1): 0.7871230246631343
Inter Writer Accuracy (Type 2):: 0.9102633801486341
acc: 0.9117887643278089
```

Statistics for AE

The Comparison for the three dataset can be shown in tabular form as

	Seen	**Unseen**	**Shuffled**
Validation Loss	0.1107	0.1014	0.1376
IntraWriter Accuracy	0.8506	0.6684	0.7871
InterWriter Accuracy	0.9076	0.9036	0.9102
Accuracy	0.9096	0.9109	0.9117

Reference - AutoEncoder provided in UBLearns by TA's.

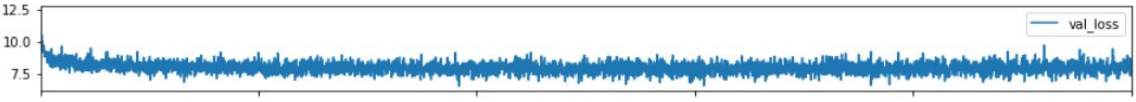
6 Task 4: Explainable AI – Deep Learning + PGM

We were provided with a task to obtain Explainable features different from the latent ones obtained in task3. For this task we are expected to produce back the 15 explainable features as created in task1 and used in task2. We combine approaches of Task2 and Task3 to do the same. We build a Multitask Learning (MTL) model to learn the mapping between "AND" images and handcrafted features. The features learnt by deep learnig are as similar as possible to the actual features but cannot be explained or proven to be the same. We use the same AutoEncoder as used in task3 to create the latent variable mapping of the input images. We directly load weights from the AutoEncoder from task3 as the same can be used here.

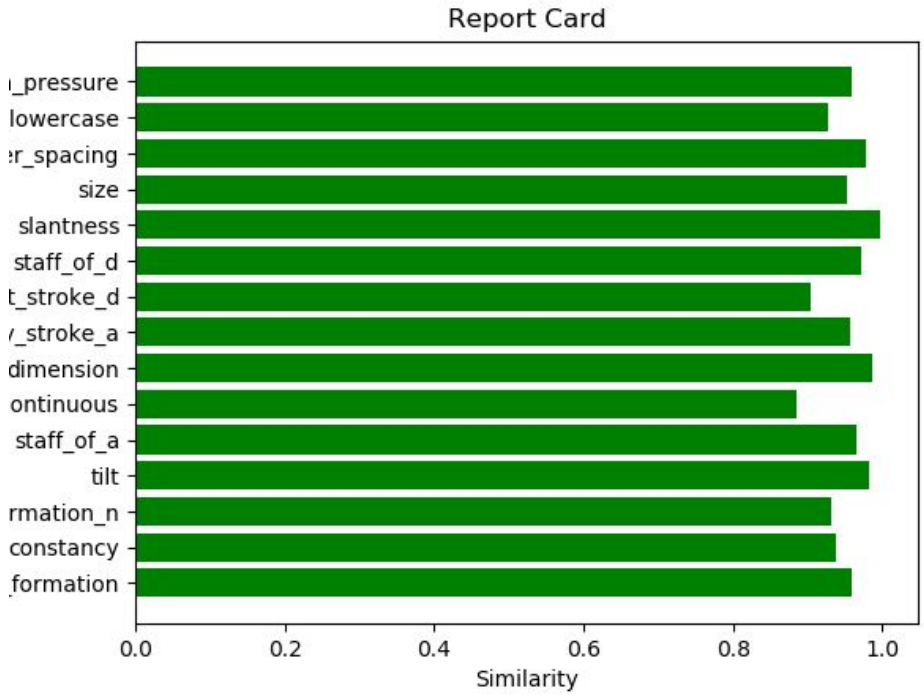
We send this latent mapping to an array of 15 Neural Networks which each map them into one of the 15 Features of the Image. These 15 neural networks have been trained to judge each feature separately, they have their own loss and the overall loss of the model is calculated by combining these separate independent losses. The results for each dataset can be shown as below

Seen Dataset Results

Cumulative Loss

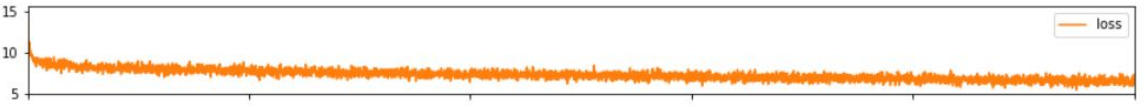


Validation Loss

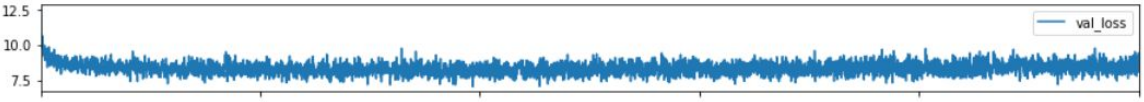


Barplot For Similarity

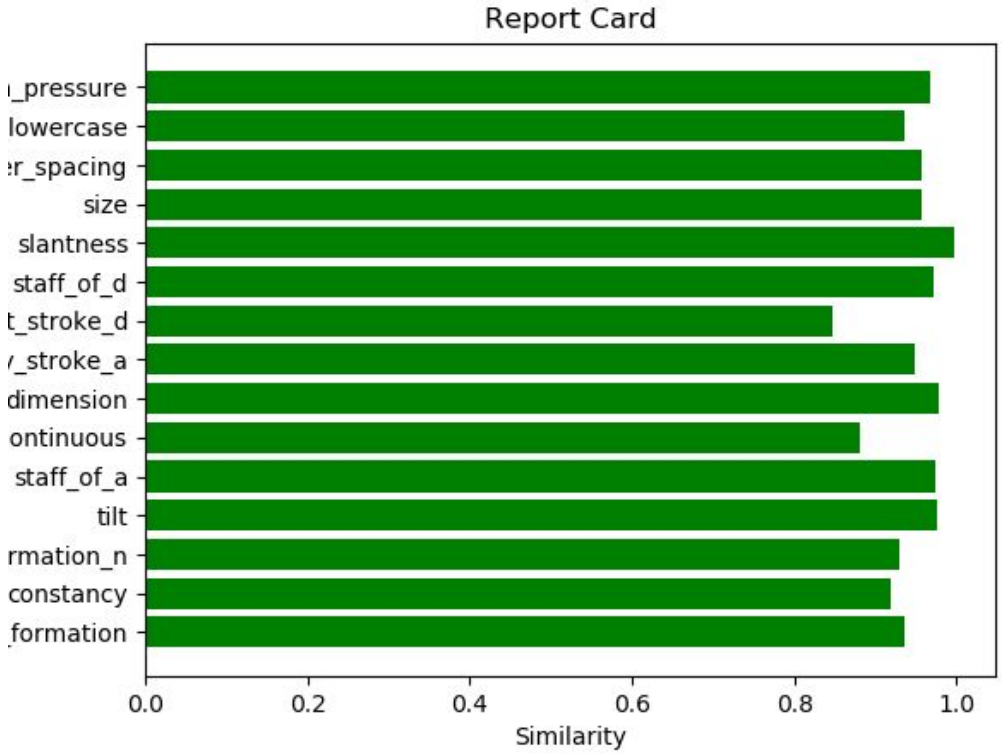
Unseen Dataset Results



Cumulative Loss

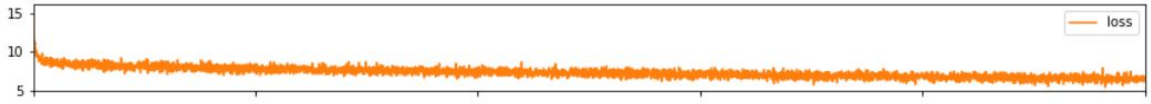


Validation Loss

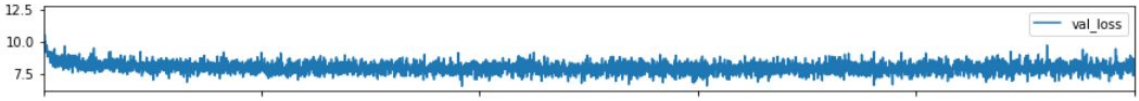


Barplot For Similarity

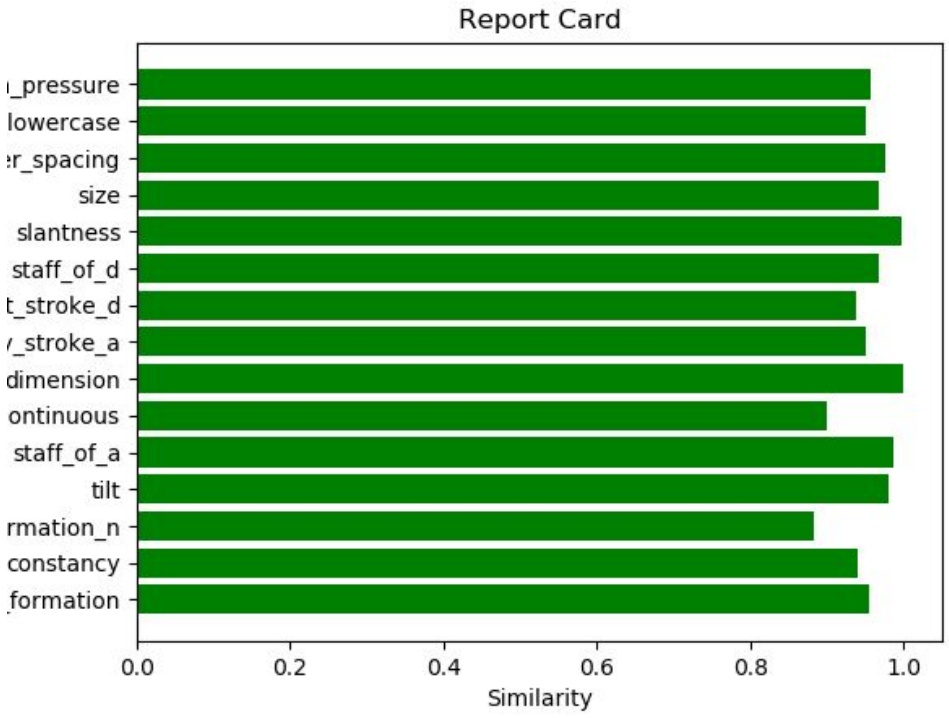
Shuffled Dataset Results



Cumulative Loss



Validation Loss



Barplot For Similarity

7 Conclusion

We used pgmpy to create Bayesian models to predict handcrafted features. Deep Learning to get inferences using direct Images and use Deep Learning + Neural Networks to get Explainable Inferences from images. This strategy for achieving that goal is to develop new or modified machine-learning techniques that will produce more explainable models. As it is equally important for the result to be explainable as it is for it to be correct. Thus explainable AI is the way to go for a stand-off between explainability and precision.

Acknowledgements

This report was prepared in accordance with the project2.pdf, project2Tasks.pdf, Task4.pptx and the pdf of the book provided, pgmpy documentation, cross entropy and PGM definitions, Siamese, AutoEncoder available on Wikipedia.

References

[1] Siamese Networks - https://en.wikipedia.org/wiki/Siamese_network

[2] AutoEncoder- https://en.wikipedia.org/wiki/Autoencoder

[3] pgmpy - http://pgmpy.org/index.html

[4] https://www.machinelearningplus.com/nlp/cosine-similarity/

[5] Probabilistic Graphical Models: Principles and Techniques Book by Daphne Koller and Nir Friedman