

---

# Handwriting Comparison: Explainable AI approach

---

**Yashankit Shikhar**  
50289472  
yshikhar@buffalo.edu  
Computer Science and Engineering  
SUNY, University at Buffalo  
Buffalo, NY

## Abstract

This aim of 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. This project is implemented by combining the concepts of deep learning and probabilistic graphical models. Handwriting patterns which were annotated by the students of Advanced Machine Learning course have been analyzed. The result obtained was used to determine whether two handwriting samples are from the same writer or different writers. The cosine similarity score between the two images as well as each feature was reported.

## 1 Introduction

In this study, we are given 13569 handwritten words written by known writers. We have worked on following different sets of features:

1. Human determined features
2. Deep learning features

**Human Determined Features:** We assume that we have human-described variables (and the values that they can take) for an input image. Each description of an input consists of a set of  $D$  discrete random variables (called features):  $x = [x_1, x_2, \dots, x_D]$ , where values taken by the variables are  $x_i \in x_i^0, x_i^1, \dots, x_i^{d_i-1}$ ,  $i = 1, 2, \dots, D$  and  $x_i^j = j$ ,  $j = 0, 1, \dots, d_i - 1$ . In this approach, we have tried to determine that whether the two images originated from the same source ( $h^0$ ) or originated from different sources ( $h^1$ ). The decision is made by computing two probabilities:  $\frac{p(X_1, X_2)}{h^0}$  and  $\frac{p(X_1, X_2)}{h^1}$ . After calculating these probabilities, we have determined which probability is larger.

**Deep Learning Features:** In this approach, we have learned the representation by processing the scanned images by a network. We have tried two approaches to arrive at the conclusion.

1. Siamese Network using Convolutional Neural Network
2. Autoencoder using Convolutional Neural Network

Following tasks are performed in this project:

1. Data-set Annotation
2. Handwriting comparison using Bayesian Network
3. Handwriting comparison using Siamese Network and Autoencoders
4. Features comparison using Multi-task learning

## 2 Task 1: Data set Annotation

Annotation in machine learning is a process of labeling the data on images containing specific objects like humans, vehicles, objects to make it recognizable for machines. Image annotation is done by humans manually using the image annotation tools to store the large volume of data stored after annotation. Basically, the annotation is the art of encoding meaning onto raw data.

Data annotation is important in machine learning because in many cases, it makes the work of the machine learning program much easier. This has to do with the difference between supervised and unsupervised machine learning.

With supervised machine learning, the training data is already labeled so the machine can understand more about the desired results. For example, if the purpose of the program is to identify cats in images, the system already has a large number of photos tagged as cat or not. It then uses those examples to contrast new data to make its results.

With unsupervised machine learning, there are no labels, and so the system has to use attributes and other techniques to identify the cats. Engineers can train the program on recognizing visual features of cats like whiskers or tails, but the process is hardly ever as straightforward as it would be in supervised machine learning where those labels play a very important role.

In this annotation task, we have annotated the “AND” images to generate handcrafted features. For doing this annotation, we have used University at Buffalo, CEDAR- Handwriting Truthing Tool where 15 different features were present for "AND" images like letter spacing, size, dimension, word formation, tilt, etc.

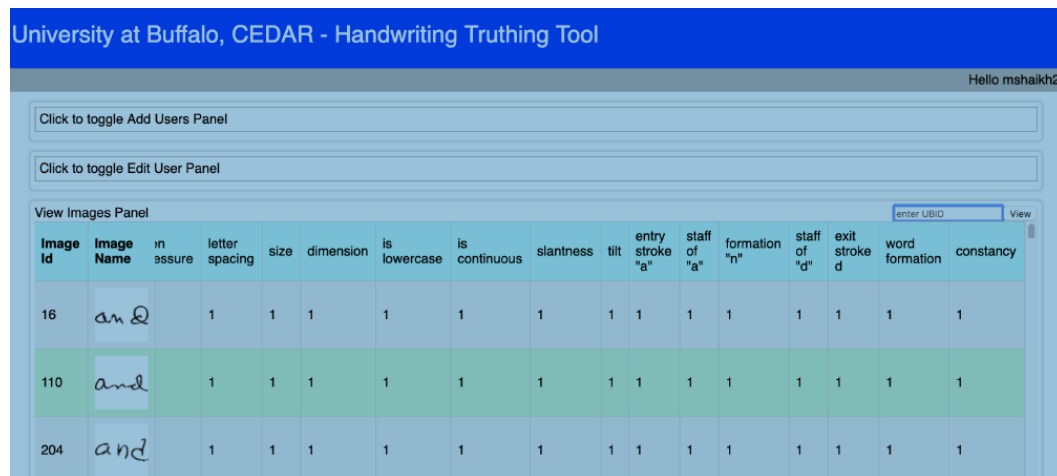


Image Id	Image Name	in assure	letter spacing	size	dimension	is lowercase	is continuous	slantness	tilt	entry stroke "a"	staff of "a"	formation "n"	staff of "d"	exit stroke "d"	word formation	constancy
16	and		1	1	1	1	1	1	1	1	1	1	1	1	1	1
110	and		1	1	1	1	1	1	1	1	1	1	1	1	1	1
204	and		1	1	1	1	1	1	1	1	1	1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

Figure 1: Annotation Tool

## 3 Task 2: Handwriting comparison using Bayesian Network

**Bayesian Network:** A Bayesian network is a directed acyclic graph in which each edge corresponds to a conditional dependency, and each node corresponds to a unique random variable. Formally, if an edge (A, B) exists in the graph connecting random variables A and B, it means that  $P(B|A)$  is a factor in the joint probability distribution, so we must know  $P(B|A)$  for all values of B and A in order to conduct inference.

The advantage of using Bayesian Network for Handwriting Comparison:

1. Bayesian networks are able to handle incomplete or noisy data which is very frequently in image analysis.
2. Bayesian networks are able to ascertain causal relationships through conditional independencies, allowing the modeling of relationships between variables.

- Bayesian networks are able to incorporate existing knowledge, or pre-known data into its learning, allowing more accurate results by using what we already know.

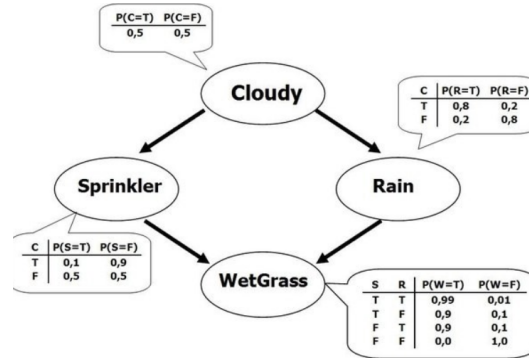


Figure 2: Bayesian Network Example

Determining Inference using Bayesian Network: Inference over a Bayesian network can come in two forms:

- The first is simply evaluating the joint probability of a particular assignment of values for each variable (or a subset) in the network.
- The second is to find  $P(x|e)$ , or, to find the probability of some assignment of a subset of the variables (x) given assignments of other variables.

To reach the inference we tried two approaches to build the Bayesian Network and query the inference. They have been described below:

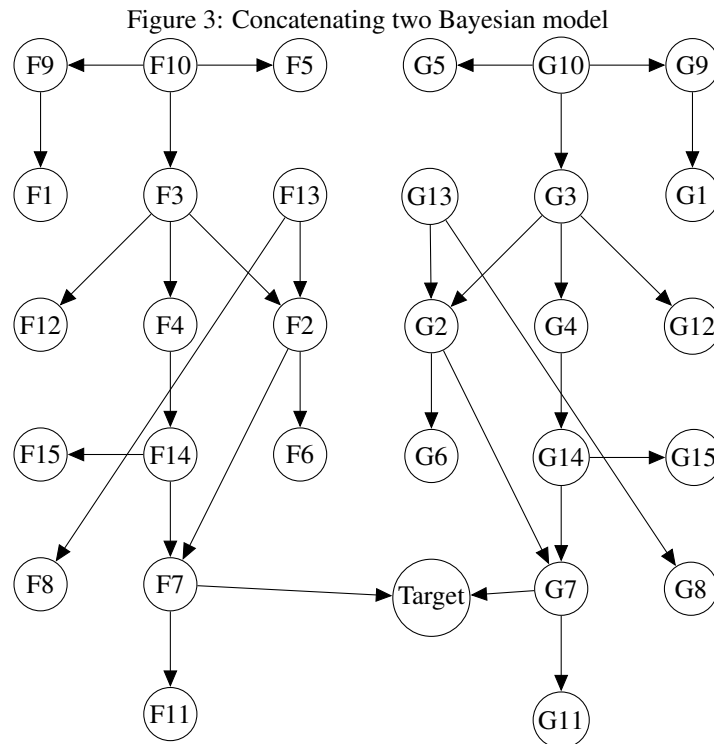


Figure 3: Concatenating two Bayesian model

- In the first approach, we considered the 'and' pattern data-set and performed a hill climb search on it to obtain the best Bayesian network possible in terms of K2 Score with a

g1	phi(g1)	g4	phi(g4)	g7	phi(g7)	g10	phi(g10)	g13	phi(g13)
g1_0	0.4195	g4_0	0.3048	g7_0	0.5910	g10_0	0.1935	g13_0	0.2603
g1_1	0.5805	g4_1	0.5067	g7_1	0.2538	g10_1	0.5512	g13_1	0.4233
g1_2	0.1885	g4_2	0.1885	g7_2	0.0636	g10_2	0.0749	g13_2	0.1287
g2	phi(g2)	g5	phi(g5)	g8	phi(g8)	g11	phi(g11)	g14	phi(g14)
g2_0	0.2379	g5_0	0.0154	g8_0	0.7901	g11_0	0.2332	g14_0	0.5925
g2_1	0.4907	g5_1	0.9846	g8_1	0.2099	g11_1	0.7668	g14_1	0.4075
g2_2	0.2714	g5_2	0.6596	g8_2	0.0628	g11_2	0.4904	g14_2	0.5662
g3	phi(g3)	g6	phi(g6)	g9	phi(g9)	g12	phi(g12)	g15	phi(g15)
g3_0	0.2322	g6_0	0.3404	g9_0	0.9372	g12_0	0.1086	g15_0	0.4338
g3_1	0.5149	g6_1	0.6596	g9_1	0.0628	g12_1	0.4904	g15_1	0.5662
g3_2	0.2529	g6_2	0.3404	g9_2	0.9372	g12_2	0.4010	g15_2	0.4338

Figure 4: Probability of G network when F Network values and target:1 is queried

maximum of one parent for each node. The following Bayesian Network was obtained. It has been shown as a list of edges:

```
[('f3', 'f4'), ('f4', 'f14'), ('f7', 'f11'),
 ('f10', 'f3'), ('f10', 'f5'), ('f10', 'f9'),
 ('f13', 'f8'), ('f13', 'f2'), ('f14', 'f15'),
 ('f9', 'f1'), ('f2', 'f6'), ('f2', 'f7'),
 ('f3', 'f12'), ('f3', 'f2')]
```

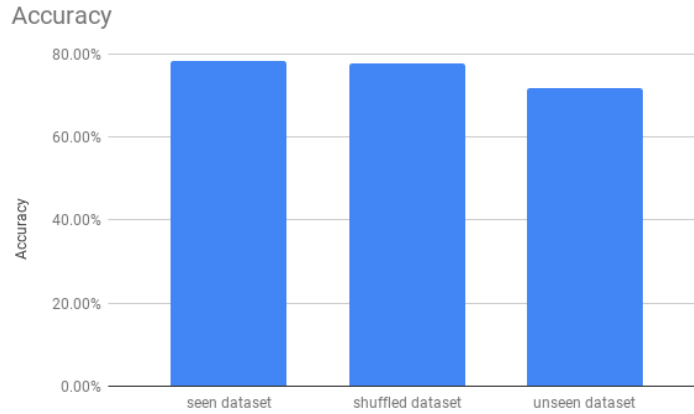


Figure 5: Accuracy from feature subtraction method

Using these results obtained we tried to build two networks and connect them using a verification node which will take the value of 1 or 0. If the sample is from the same writer then it is labeled 1 otherwise 0. The combined Bayesian Network is shown in figure 3. After the construction, we fit the model with training data. F1, F2,..., F15 takes a feature from one sample and G1, G2,..., G15 takes a feature from another sample while the verification node takes the value of 1 or 0. After this, we query values of F Bayesian network and verification

node and get values of G Bayesian network and vice versa. The output obtained is shown in Figure 4. This took around 20 minutes to train or fit the data.

2. In the second approach, we constructed a Bayesian Model by subtracting the two image features and taking the absolute value to create a 15 feature vector. We train the Bayesian Network using this data and query validation data in a similar format without the target variable. The Bayesian network predicts Target values and we obtain accuracy of 78.2% on seen data-set, 78% on shuffled data-set and 72% on unseen data-set. This method takes relatively less time to fit data and infer i.e around 10 minutes.

#### 4 Task 3: Handwriting comparison using Siamese Network and Autoencoders

**Siamese Network:** Siamese neural network is a class of neural network architectures that contain two or more identical subnetworks. Identical here means they have the same configuration with the same parameters and weights. Parameter updating is mirrored across both subnetworks.

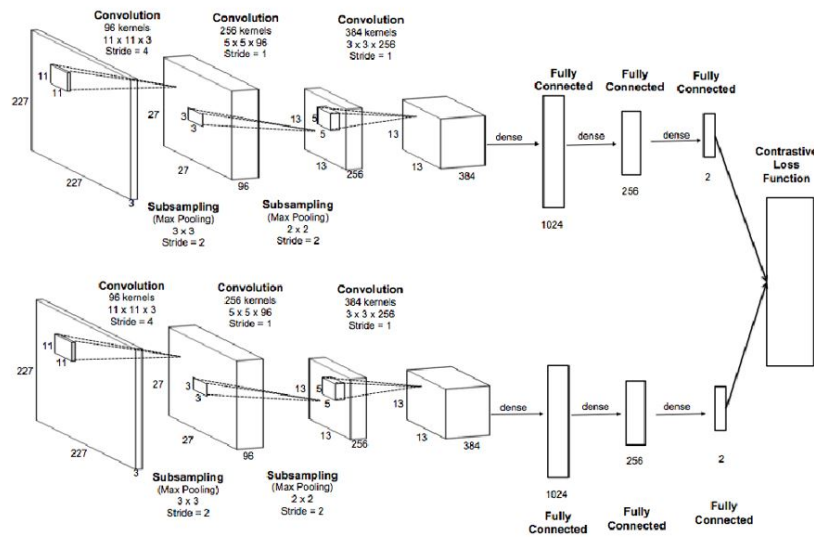


Figure 6: Siamese Network Example

Siamese Neural Networks are popular among tasks that involve finding similarity or a relationship between two comparable things.

Reasons for using Siamese Networks for finding the similarities between two comparable things are:

1. Sharing weights across subnetworks means fewer parameters to train for, which in turn means less data required and less tendency to overfit.
2. Each subnetwork essentially produces a representation of its input. If your inputs are of the same kind, like matching two sentences or matching two pictures, it makes sense to use similar model to process similar inputs. This way you have representation vectors with the same semantics, making them easier to compare.

We created the two siamese network using the following code and merged the output using several dense layers as shown in Figure 7. The validation accuracy and loss obtained from training the network were not encouraging and hence we decided to try the autoencoder approach.

```
from keras.models import Model
from keras.layers import Input, Conv2D, BatchNormalization,
MaxPool2D, Activation, Flatten, Dense, Dropout
img_in = Input(shape = train_data1.shape[1:])
n_layer = img_in
```

```

for i in range(2):
    n_layer = Conv2D(8*2**i, kernel_size = (3,3),
        activation = 'linear')(n_layer)
    n_layer = BatchNormalization()(n_layer)
    n_layer = Activation('relu')(n_layer)
    n_layer = Conv2D(16*2**i, kernel_size = (3,3),
        activation = 'linear')(n_layer)
    n_layer = BatchNormalization()(n_layer)
    n_layer = Activation('relu')(n_layer)
    n_layer = MaxPool2D((2,2))(n_layer)
n_layer = Flatten()(n_layer)
n_layer = Dense(32, activation = 'relu')(n_layer)
n_layer = Dropout(0.5)(n_layer)
n_layer = BatchNormalization()(n_layer)
n_layer = Activation('sigmoid')(n_layer)
feature_model = Model(inputs = [img_in], outputs = [n_layer])

```

Layer (type)	Output Shape	Param #	Connected to
ImageA_Input (InputLayer)	(None, 64, 64, 1)	0	
ImageB_Input (InputLayer)	(None, 64, 64, 1)	0	
FeatureGenerationModel (Model)	(None, 32)	181712	ImageA_Input[0][0] ImageB_Input[0][0]
merge_features (Concatenate)	(None, 64)	0	FeatureGenerationModel[1][0] FeatureGenerationModel[2][0]
dense_22 (Dense)	(None, 16)	1040	merge_features[0][0]
batch_normalization_41 (BatchNo	(None, 16)	64	dense_22[0][0]
activation_41 (Activation)	(None, 16)	0	batch_normalization_41[0][0]
dense_23 (Dense)	(None, 4)	68	activation_41[0][0]
batch_normalization_42 (BatchNo	(None, 4)	16	dense_23[0][0]
activation_42 (Activation)	(None, 4)	0	batch_normalization_42[0][0]
dense_24 (Dense)	(None, 1)	5	activation_42[0][0]
=====			
Total params: 182,905			
Trainable params: 182,657			
Non-trainable params: 248			

Figure 7: Siamese Network Neural Network layer

**Autoencoders:** Autoencoder is a neural network, as well as an unsupervised learning (feature learning) algorithm.

1. It applies backpropagation, by setting the target value the same as the input.
2. It tries to predict  $x$  from  $x$ , without the need for labels.
3. It tries to learn an approximation of an “identity” function.
4. It represents original input from compressed, noisy, or corrupted data.
5. It consists of a narrow hidden layer between the Encoder & Decoder.

**Convolutional Autoencoder:** In this, we replace fully connected layers by convolutional layers. These, along with pooling layers, convert the input from wide and thin to narrow and thick. This helps the network extract visual features from the images, and therefore obtain a much more accurate latent space representation. The reconstruction process uses upsampling and convolutions.

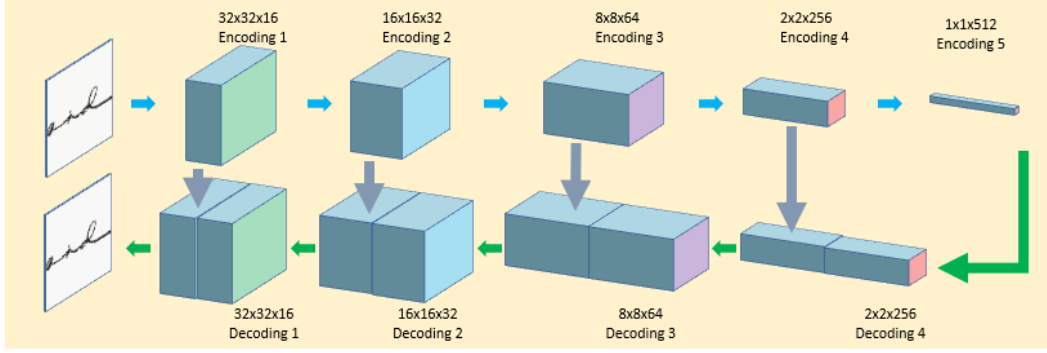


Image 2: Translation Invariant AutoEncoder Architecture

Figure 8: AutoEncoders Architecture

We first trained the encoder and decoder layers on the and image dataset and ran it for 125 epochs to train the encoder. The training and validation loss is shown in Figure 11. Then we built another encoder layer and assigned it the weights from the previous encoder layer of Autoencoder. We then took the feature vector generated from the images and found cosine similarity between two images. Then we set a threshold of 0.5 and predicted that the image pair with cosine similarity higher than 0.5 were labeled from the same writer while the rest from different writers. Using this method accuracy of 86% was achieved on the seen dataset.

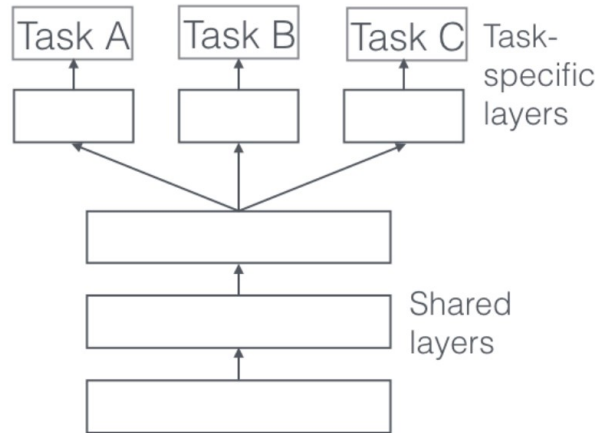


Figure 9: Multi-task learning example

The encoder code is shown below:

```
def encoder(input_img):
    x = Conv2D(16, (3, 3), activation='relu', padding='same')(input_img)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(256, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(512, (3, 3), activation='relu', padding='same')(x)
```

```

encoded = MaxPooling2D((2, 2), padding='same')(x)
return encoded
encoder = Model(input_img1, encoder(input_img1))
for l1,l2 in zip(encoder.layers[:12],autoencoder.layers[0:12]):
    l1.set_weights(l2.get_weights())

```

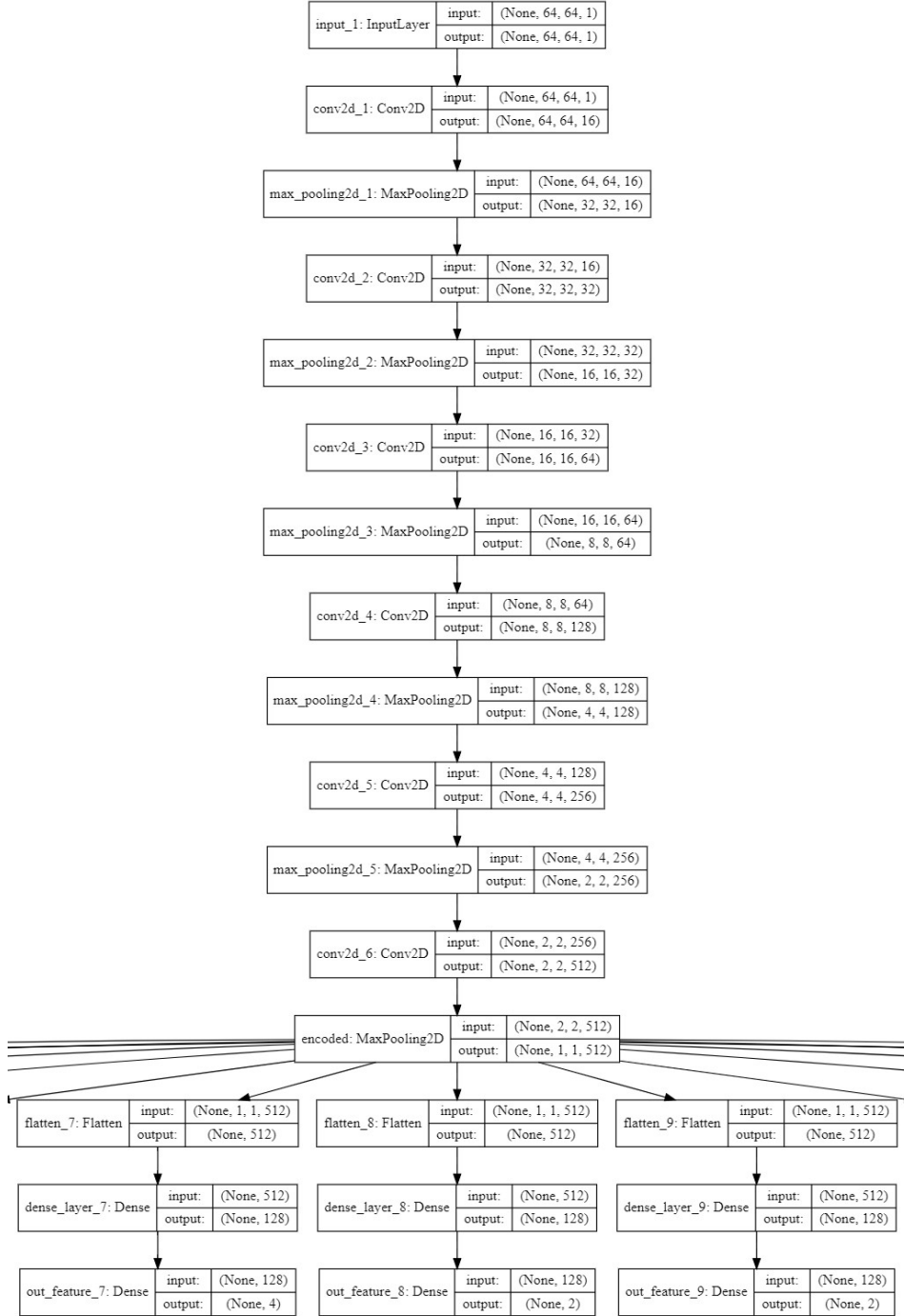


Figure 10: Snippet from Multi-task learning architecture



Autoencoder Training\_loss and Autoencoder Validation\_loss

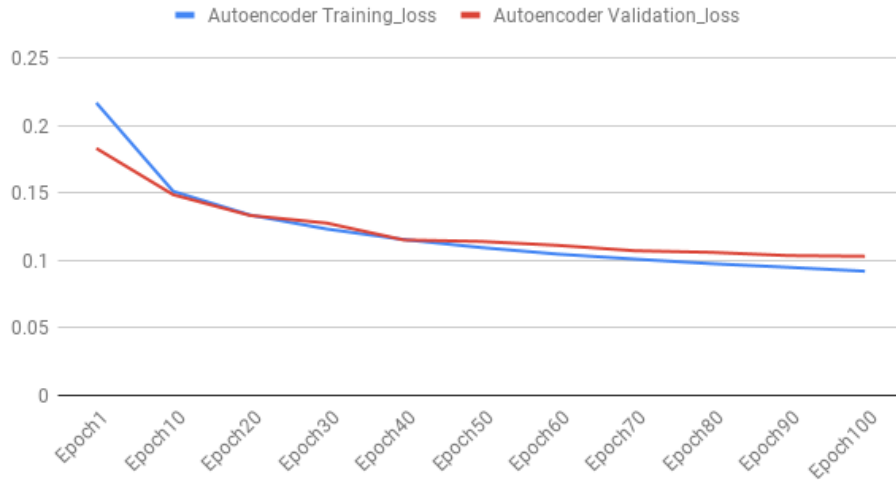


Figure 11: Training and Validation loss for Autoencoders

## 5 Task 4: Features comparison using Explainable AI

In Machine Learning (ML), we typically care about optimizing for a particular metric, whether this is a score on a certain benchmark or a business KPI. In order to do this, we generally train a single model or an ensemble of models to perform our desired task. We then fine-tune and tweak these models until their performance no longer increases. While we can generally achieve acceptable performance this way, by being laser-focused on our single task, we ignore information that might help us do even better on the metric we care about. Specifically, this information comes from the training signals of related tasks. By sharing representations between related tasks, we can enable our model to generalize better on our original task. This approach is called Multi-Task Learning (MTL). The example model is shown in Figure 9.

In this method, we borrow the encoder layer obtained from the performing Task 3 and its weights. Then we attach 15 different dense layers to the output from the encoder layer and train these 15 layers based on one hot encode vector of all 15 features. Then we input two images to this module and compute the similarity between each feature of the image. The output has been shown below when two images were compared which were from different writers. The Architecture of the Network is shown in Figure 10.

overall\_similarity predicted by AutoEncoder: 0.3941423296928406  
different writer

```
feature 0 cosine score: 0.9999999403953552
feature 1 cosine score: 0.999992847442627
feature 2 cosine score: 0.9999998211860657
feature 3 cosine score: 0.0007817999576218426
feature 4 cosine score: 1.0
feature 5 cosine score: 0.999987006187439
feature 6 cosine score: 0.9999868273735046
feature 7 cosine score: 0.0027961046434938908
feature 8 cosine score: 0.9999956488609314
feature 9 cosine score: 0.9999947547912598
feature 10 cosine score: 0.9999988079071045
feature 11 cosine score: 0.9999883770942688
feature 12 cosine score: 0.006634489167481661
feature 13 cosine score: 0.9999961853027344
```

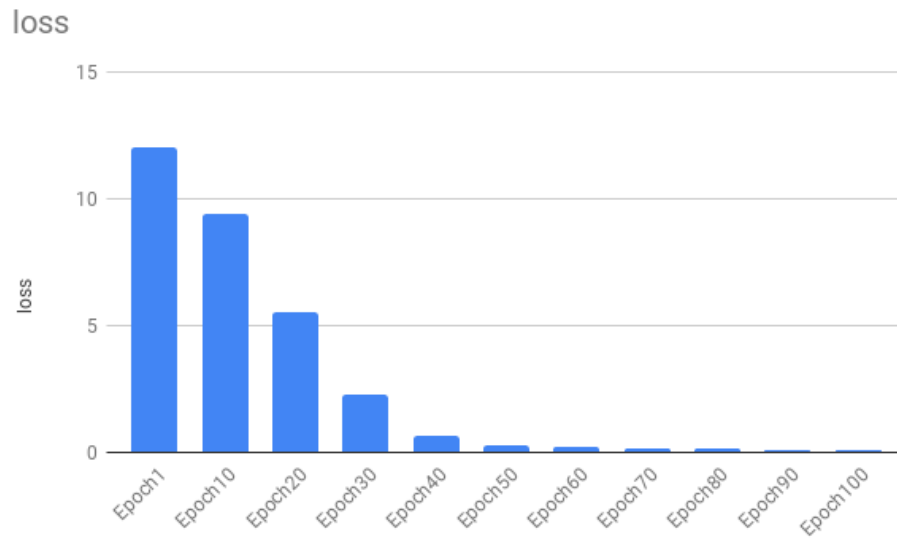


Figure 12: Training loss for Multi-task learning network

feature 14 cosine score: 0.0011881800601258874

Here feature 0 to 14 means 'pen\_pressure', 'letter\_space', 'size', 'dimension', 'lowercase', 'continuous', 'slantness', 'tilt', 'entry stroke of a', 'staff of a', 'formation of n', 'staff of d', 'exit stroke of d', 'word\_formation' and 'constancy' respectively.

## References

- [1] Determining Probabilities of Handwriting Formations using PGMs - Project Description
- [2] Advanced Machine Learning Course Slides
- [3] PGMPY Documentation: <http://pgmpy.org/index.html>