Handwritten Signature Verification Using Deep Learning and comparison of loss functions

J COMPONENT PROJECT REPORT

Submitted by:

Registration Number	Name
16BEC0784	Harshit Parvatiyar
16BEC0766	Bopanna Yaswant Kumar
17BEC0462	Siddharth S Nyamagouda

ECE3999 – Technical Answers for Real World Problems

Slot : TD2

FACULTY

Prof. ILAVARSAN T.

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Fall Semester 2020-2021

Table of Contents

ABSTRACT

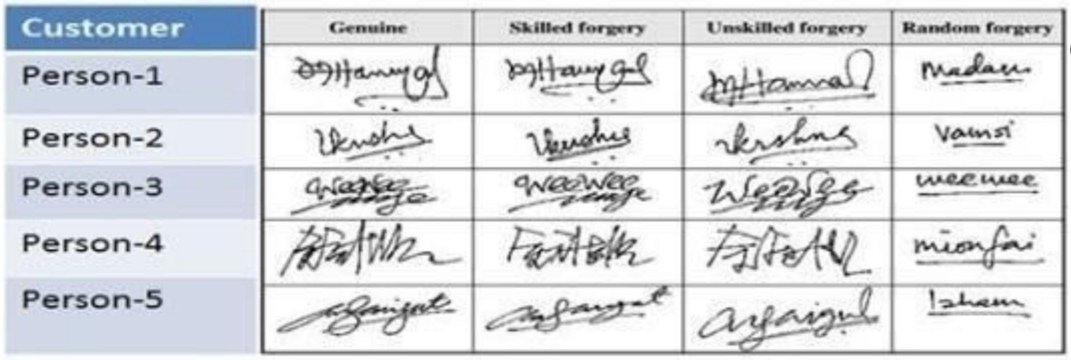
Handwritten signature verification is usually done manually by humans. Barring a few experts, more people are not skilled enough to sniff out smart forgeries. We attempt to change this process by using this emerging technology called deep learning. We use convolution neural networks to solve this, training is done using images from a signature verification competition called SigComp. Instead of normal training, we use triplet loss function to train it, which uses 3 images at a time, forgery, base and positive image to learn to distinguish between them.

What we hope to make towards the end of this project is a signature verification system that has already been trained on previous fake and real signatures and can be used to check the validity of signature easily by normal people. We use three different types of triplet loss functions and test out each and every one of them and see which is superior. All these model have one thing in common, they just need one base image to be able to function, hence take the advantage of one shot learning.

Introduction, problem statement and objectives

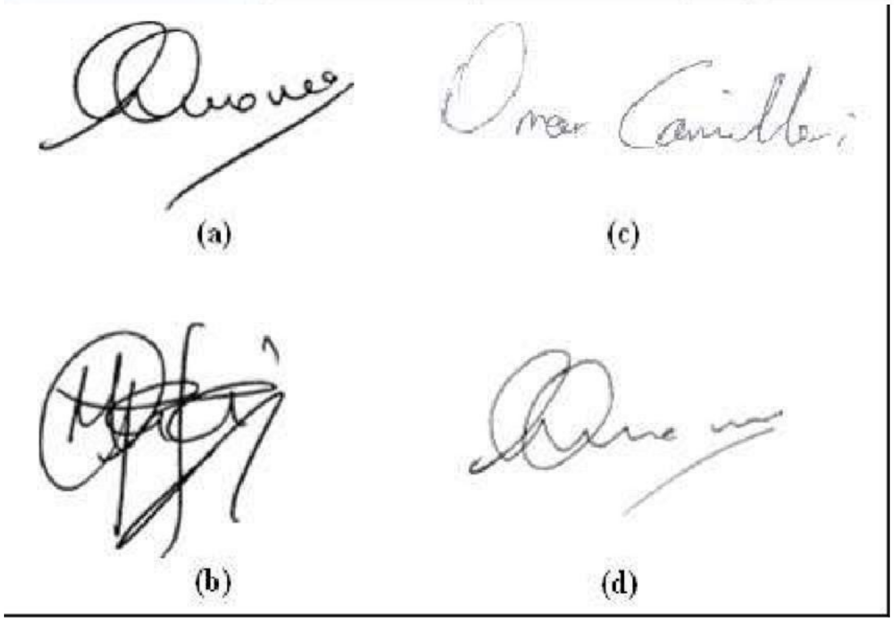
In real life a signature forgery is an event in which the forger mainly focuses on accuracy rather than fluency.
The range of signature forgeries falls into the following three categories:

1. **Random/Blind forgery** — Typically has little or no similarity to the genuine signatures. This type of forgery is created when the forger has no access to the authentic signature.
2. **Unskilled (Trace-over) Forgery:** The signature is traced over, appearing as a faint indentation on the sheet of paper underneath. This indentation can then be used as a guide for a signature.
3. **Skilled forgery** — Produced by a perpetrator that has access to one or more samples of the authentic signature and can imitate it after much practice. Skilled forgery is the most difficult of all forgeries to authenticate.



Customer	Genuine	Skilled forgery	Unskilled forgery	Random forgery
Person-1	[illegible]	[illegible]	[illegible]	Madan
Person-2	[illegible]	[illegible]	[illegible]	Vamsi
Person-3	[illegible]	[illegible]	[illegible]	weewee
Person-4	[illegible]	[illegible]	[illegible]	mionfai
Person-5	[illegible]	[illegible]	[illegible]	Isham

(c) simulated



An effective signature verification system must have the ability to detect all these types of forgeries by means of reliable, customized algorithms.

Manual verification conundrum:

Because of subjective decision and varies heavily depending on human factors such as expertise, fatigue, mood, working conditions etc manual verification is more error-prone and inconsistent, in the case of Skilled forgery(offline method)leads to following instances:

False Rejection: Flagging transactions fraudulent (when they are not) mistakenly declined, creating a negative impact on customer satisfaction, often called as type-I error.

False Acceptance: Genuine signature and skilled forgery that operator accepted as an authentic signature, leading to financial and reputational loss, often called as type-II error.

genuine signatures
False rejection
Genuine
Skilled forgery
False acceptance

The goal of an accurate verification system to minimize both types of error.

Signature traits:

Let's understand signature features for a human document examiner to distinguish frauds from genuine.Following is a non-exhaustive list of static and dynamic characteristics used for signature verification:

· *Shaky handwriting(static)*

· *Pen lifts(dynamic)*

· *Signs of retouching(static and dynamic)*

· *Letter proportions(static)*

· *Signature Shape/dimension(static)*

· *Slant/angulation(static)*

· *Very close similarity between two or more signatures(static)*

· *Speed(dynamic)*

· *Pen pressure(dynamic)*

· *Pressure Change Patterns(dynamic)*

· *Acceleration pattern(dynamic)*

Based on the verification environment and sample collection condition, not all the features are available for analysis

Types of automatic signature verification system

As discussed depending on the feasible(available) signature characteristics extraction and business/functional requirement, broadly two categories of Signature Verification systems exist in the market.

A) Offline Signature Verification: Deployed where there is no scope for monitoring the real-time signature activity of a person. In applications that scrutinize signed paper documents, only a static, two-dimensional image is available for verification. For obvious reasons in this type of verification engine, dynamic characteristics. In order to account for the loss of this important information and produce highly accurate signature comparison results, off-line signature verification systems have to imitate the methodologies and approaches used by human forensic document examiners.

For offline Signature Verification the machine learning tasks can be further categorized in 1)General learning (person-independent)- The verification task is performed by comparing the questioned signature against each known signature in 1:1 basis and 2) Special learning (which is person-dependent) — To verify whether the questioned signature falls within the range of variation among multiple multiple genuine signatures of the same individual.

B) Online Signature Verification: Signing is a reflex action based on a repeated action, rather than deliberately controlling muscles and even accurate forgeries take longer to produce than genuine signatures. As the name suggests in this type of verification system, capture of crucial dynamic features, such as speed, acceleration, and pressure, etc., is feasible. This type of system is more accurate as even for the copy machine or an expert, it is virtually impossible to mimic unique behavior patterns and characteristics of the original signer.

According to a report by PwC in 2015, financial fraud contributes to an estimated 20 billion USD in direct losses annually. The worst effect of financial frauds is on FDI inflows into India. An essential part of forging a document is always forging a signature. To disprove a document, or prove a document is fraud, the most common method is to perform a signature analysis to prove the signature is not from the claimed person. With thousands of cheques and documents involved every day, the task is to solve this issue by automating it. Various methodologies have been proposed and tried.

While, getting a good score, accuracy and f1 score is important what’s more important is the real life viability of the solution. The problem with using traditional

CNN architectures is that when a new member’s signature is added, the whole model has to be retrained to adjust to it. What we propose is a one shot learning to resolve this issue we also present and contrast three different innovative loss functions - triplet loss , lossless triplet loss and contrastive loss.

LITERATURE SURVEY

S. No.	Title of the Paper	Algorithms used	Datset used	Performance measures	Limitations (Gap identified)	Scope for future work
1	Writer-independent Feature Learning for Offline Signature Verification using Deep Convolutional Neural Networks	OTSU algorithm, normalization and inversion, CNNs	GPDS-960 dataset, Brazilian PUC-PR dataset	Equal Error Rate, False Acceptance rates of forgery and the Average Error Rate	Performance is worse compared to systems where multiple feature extractors / classifiers are used	Balancing of False Rejection and False Acceptance rates
2	Dynamic Signature Verification System Based on One Real Signature	Biological neuromuscular model	SUSIG-Visual Subcorp us, SVC-Task1 Subcorp us, MCYT1 00 Subcorp us	EER - Equal error rate	The images obtained are visually checked to limit the variability of the system parameter	Study of the real stroke variability under the sigma-lognormal mode

3	Preprocessing and Feature Selection for Improved Sensor Interoperability in Online Biometric Signature Verification	Gaussian Mixture Models, DTW (Dynamic Time Warping)	Biosecu re DS2, DS3 (HP PDA under mobile scenario)	EER - Equal error rate	Need to take into account systems where X and Y coordinate s are not used	Performance of the system using devices with the same quality for interoperability cases and using newer devices such as tablets and smartphones
4	One-Class Writer-Independent Off-line Signature Verification Using Feature Dissimilarity Thresholding	Contourlet Transform, Dissimilarity Thresholdin g	CEDAR dataset, GPDS dataset	FAR (False Rejection Rate), FRR (False Acceptance Rate) and AER (Average Error Rate),	There is a need of diversifying information during the design step through mixing dataset's writers for a better writer-independent approach	Developing a global system that can be deployed in many institutions
5	Analyzing features learned for Offline Signature Verification using Deep CNNs	AlexNet_N reduced, AlexNet_N, VGG_Nreduced, VGG_N	GPDS dataset	Equal Error Rate	Vulnerable to forgeries that closely resemble genuine signatures, even if line quality is bad i.e the case of slowly-traced forgeries	Investigate the combination of such features with features particularly targeted to discriminate the quality of pen strokes

METHODOLOGY

A one-shot learning solution using a VGG19 pre trained network. A triplet loss function is used
for training the last few layers. The VGG19 networks acts as a feature extractor and gives us NumPy array of 512 numbers. We then compare the features and can differentiate the original signature from the forgeries.

1) Siamese networks

A Siamese neural network is an artificial neural network that uses the same weights while working in tandem on two different input vectors to compute comparable output vectors. One of the output vectors is precomputed, thus forming a baseline against which the other output vector is compared.

Architectural representation of the network:

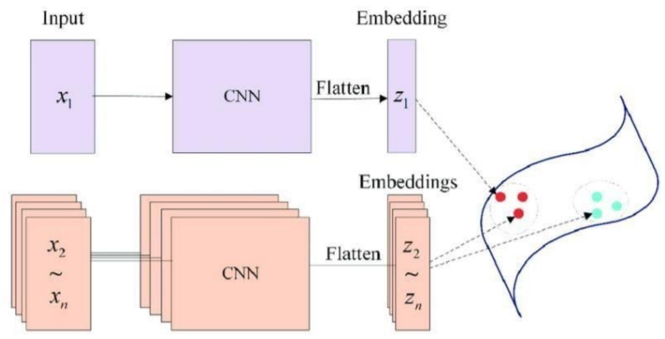


figure 1.1

2) Triplet Loss function

Triplet loss is a loss function for artificial neural networks where a baseline (anchor) input is compared to a positive (truthy) input and a negative (false) input.

$$Loss = \sum_{i=1}^{N} \left[\|f_i^a - f_i^p\|_2^2 - \|f_i^a - f_i^n\|_2^2 + \alpha \right]_+$$

The distance from the baseline (anchor) input to the positive (truthy) input is minimized, and the distance from the baseline (anchor) input to the negative (false) input is maximized

3) Contrastive Loss

The contrastive loss is a distance-based loss as opposed to more conventional error-prediction losses. This loss is used to learn embeddings in which two similar points have a low Euclidean distance and two dissimilar points have a large Euclidean

$$\ell_{i,j} = -\log \frac{\exp(\mathrm{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\mathrm{sim}(\boldsymbol{z}_i, \boldsymbol{z}_k)/\tau)},$$

Distance. How is it different from Triplet loss? For a given triplet of Reference point (A) and a point similar (P) and dissimilar (N) to it: Triplet Loss maximizes the difference between A-N & A-P distance (with margin). But Contrastive loss looks to maximize A-N and minimize A-P separately (again with margins).The contrastive loss function defined by:

$$(1-Y)\frac{1}{2}(D_W)^2 + (Y)\frac{1}{2}\{max(0, m - D_W)\}^2$$

Contrastive loss function where Dw is Euclidean distance between the outputs of the sister Siamese networks. It can be written in an equation form by:

$$\sqrt{\{G_W(X_1) - G_W(X_2)\}^2}$$

4) Lossless triplet loss

loss = K.maximum(basic_loss,0.0)

There is a major issue here, every time your loss gets below 0, you lose information, a ton of information. It tries to bring the Anchor (current record) with the Positive (A record that is in theory similar with the Anchor) as far as possible from the Negative (A record that is different from the Anchor). So as long as the negative value is further than the positive value + alpha there will be no gain for the algorithm to condense the positive and the anchor.

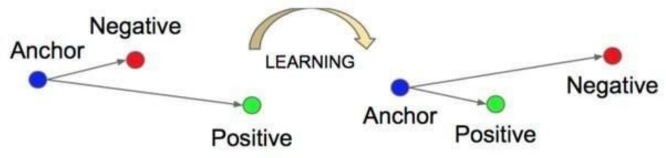


figure 1.3

Figure 1.3 shows that the anchor should be close to the positive and far away from the negative.

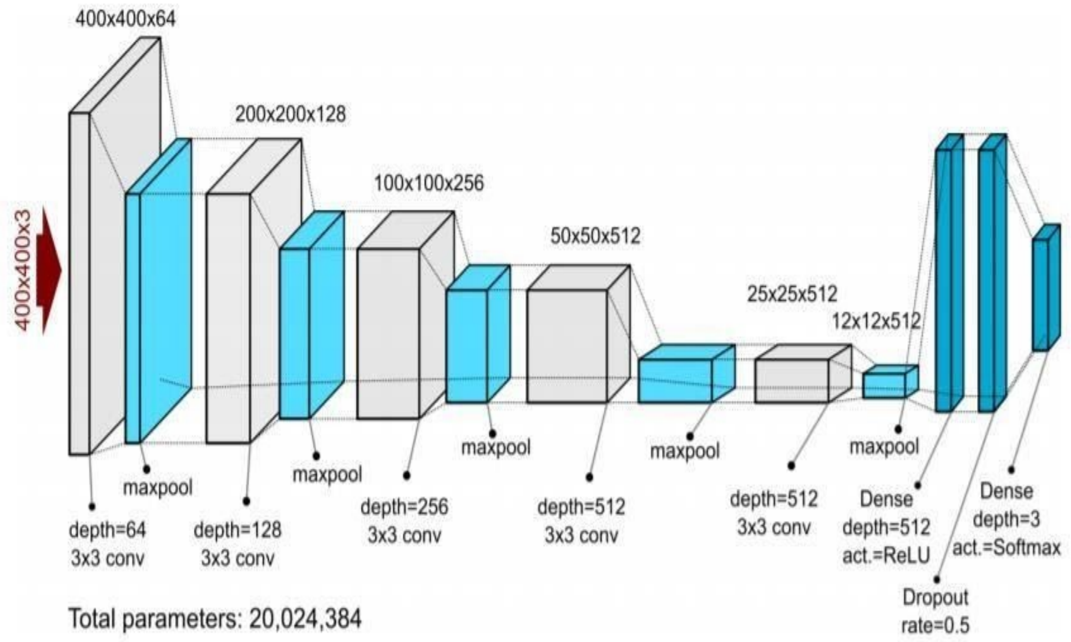
5) Pretrained VGG 19 network

Keras has a lot of pretrained models available. Including the smaller variant of VGG 19, the VGG 16. We choose this model because of it's simplicity and because it has proven to generalize quickly and well.

The pretrained weights have weights that are already adjusted for a particular task, in our case recognizing objects. Now if our task is similar, the weights will prove to be useful. In case of VGG 19, the earlier layers capture the lower level features of a picture and hence they are frozen.

In a deep neural network as the layers progress, they start capturing more and more specific information about the images. So the first layer captures simply lines, the second layer might capture strokes, the 3rd layers will start noticing shapes, etc.

The rest of the unfrozen layers are which we work upon while readjusting the weights when we use the fit_generator function.

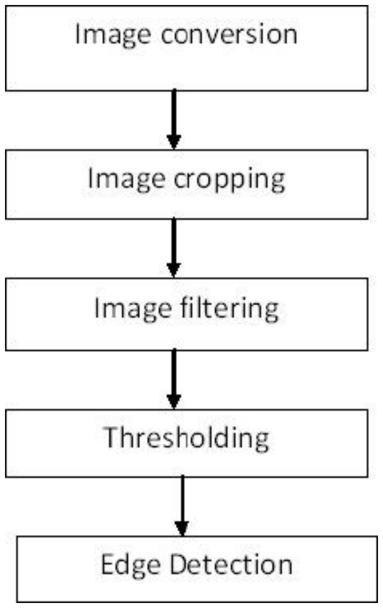


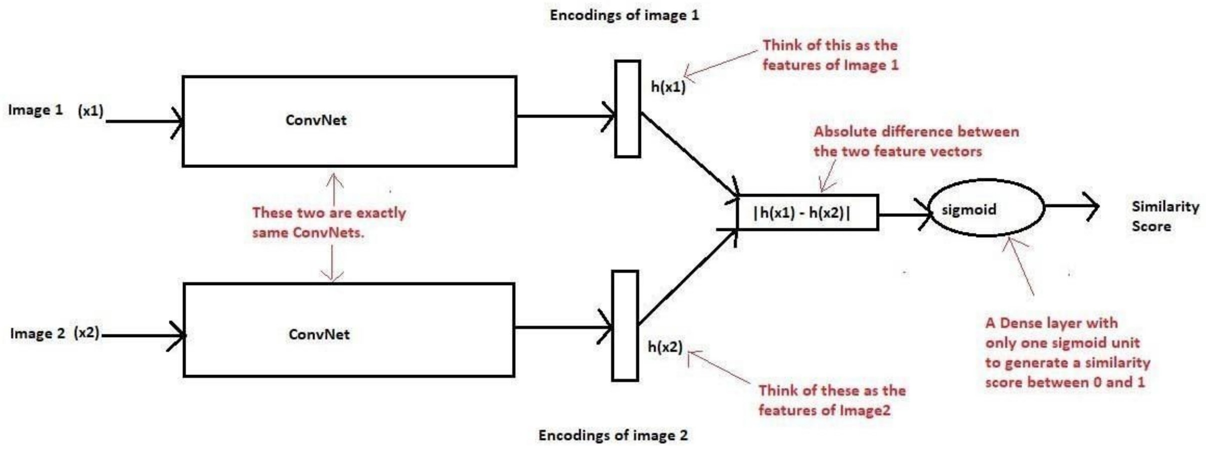
400x400x3
400x400x64
200x200x128
100x100x256
50x50x512
25x25x512
12x12x512
depth=64
3x3 conv
maxpool
depth=128
3x3 conv
maxpool
depth=256
3x3 conv
maxpool
depth=512
3x3 conv
maxpool
depth=512
3x3 conv
maxpool
Dense
depth=512
act.=ReLU
Dropout
rate=0.5
Dense
depth=3
act.=Softmax
Total parameters: 20,024,384

WORKFLOW

For the network to perform well with real-world signatures, the SigComp dataset can be augmented by adding noise and random blur to force the network to generalize well.

In total we plan to have 5 preprocessing steps





Encodings of image 1
Think of this as the features of Image 1
h(x1)
Image 1 (x1)
ConvNet
Absolute difference between the two feature vectors
These two are exactly same ConvNets.
|h(x1) - h(x2)|
sigmoid
Similarity Score
Image 2 (x2)
ConvNet
h(x2)
A Dense layer with only one sigmoid unit to generate a similarity score between 0 and 1
Think of these as the features of Image2
Encodings of image 2

IMPLEMENTATION

Code for defining the triplet loss functions –

```
def triplet_loss(y_true,  y_pred): alpha = 0.5
     anchor,  positive,  negative  =y_pred[0,0:512],  y_pred[0,512:1024],  y_pred[0,1024
:1536]

     positive_distance  =  K.mean(K.square(anchor  -  positive),axis=-1) negative_distance  =
     K.mean(K.square(anchor  -  negative),axis=-1)
     return  K.mean(K.maximum(0.0,  positive_distance  -  negative_distance  +  alpha))
```

```
def  lossless_triplet_loss(y_true,  y_pred,  beta=3,  epsilon=1e-8):
     anchor,  positive,  negative  =y_pred[0,0:512],  y_pred[0,512:1024],  y_pred[0,1024
:1536]

     pos_dist  =  K.mean(K.square(anchor  -  positive),axis=-1) neg_dist  =
     K.mean(K.square(anchor  -  negative),axis=-1)

     N=3
     pos_dist  =  -tf.math.log(-tf.divide((pos_dist),beta)+1+epsilon) neg_dist  =
     -tf.math.log(-tf.divide((N-neg_dist),beta)+1+epsilon) loss  =  neg_dist  +
     pos_dist

     return loss

def contrastive_loss(y_true,  y_pred): margin
     = 1
     square_pred  =  K.square(y_pred)
     margin_square  =  K.square(K.maximum(margin  -  y_pred,  0))
     return  K.mean(y_true  *  square_pred  +  (1  -  y_true)  *  margin_square)
```

Code for pre-processing images-

```
img_width,  img_height,  channels  =  224,  224,  3 dim =

(img_width, img_height)

def to_rgb(img):
     img  =  cv2.resize(img,  dim,  interpolation  =  cv2.INTER_AREA) img_rgb  =
     np.asarray(np.dstack((img,  img,  img)),  dtype=np.uint8) return img_rgb

def returnimages(path,img):
     image=cv2.imread(path+"/"+  img)          #bringing  the  image
     image=cv2.resize(image,  (img_width,  img_height))
```

```
        image=cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        image=to_rgb(image).reshape(1,img_width, img_height,3)/255.0                    #resizing a
nd normalizing
        return image
```

Code for converting a VGG19 model to start accepting triplets and give 3 outputs

By default, our VGG network takes in one image and gives one output. The anchor in and anchor out part of code ensures that I can give it 3 images and it'll give me 3 outputs of 512 length corresponding to those 3 images. We do this so that it fits in well with the loss function we wrote.

```
model1 = applications.vgg19.VGG19(weights='imagenet', include_top=False, pooling=' max')
for layer in model1.layers[:15]:
     layer.trainable = False

anchor_in = Input(shape=(img_width, img_height, channels)) pos_in =
Input(shape=(img_width, img_height, channels)) neg_in =
Input(shape=(img_width, img_height, channels))

anchor_out = model1(anchor_in)
pos_out = model1(pos_in) neg_out =
model1(neg_in)
merged_vector = concatenate([anchor_out, pos_out, neg_out],axis=1)

model_triplet_loss = Model(inputs=[anchor_in, pos_in, neg_in], outputs=merged_vect or)
```

```
model_triplet_loss.compile(optimizer=Adam(lr=0.00001),loss=triplet_loss)
```

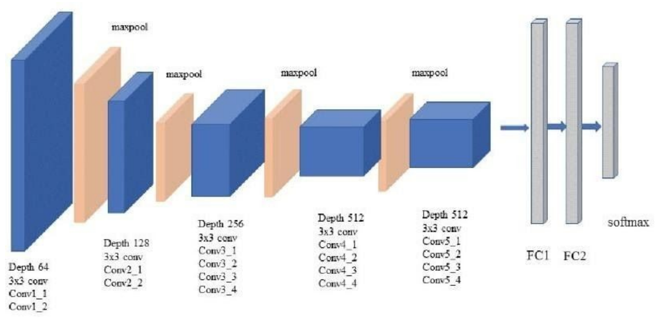
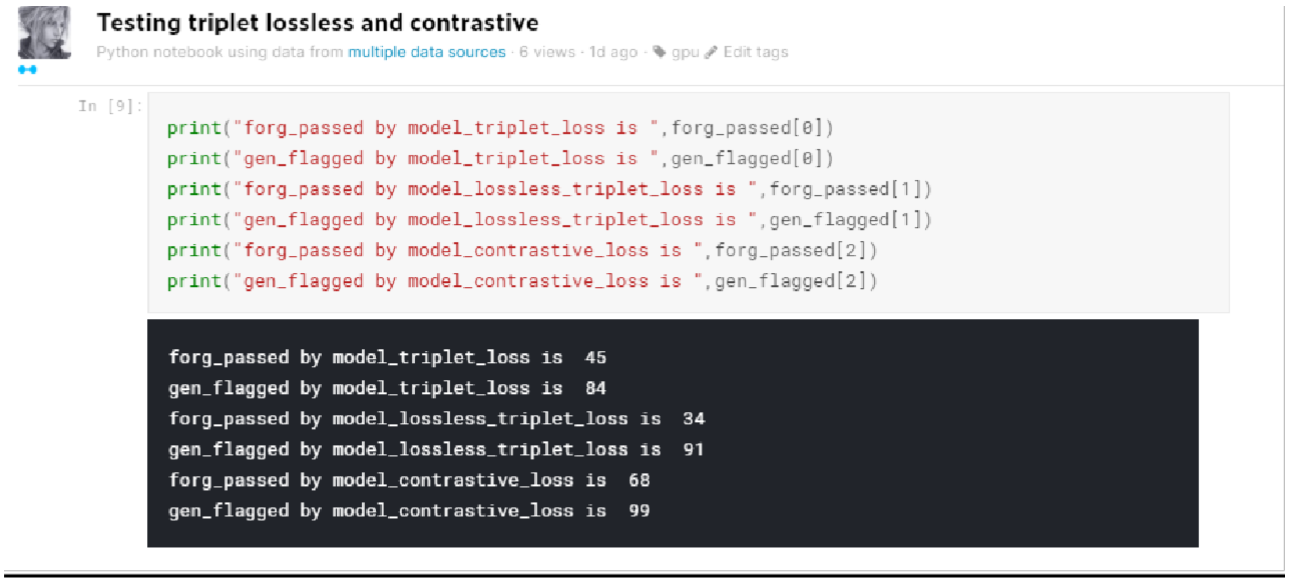


figure 1.4

Figure 1.4 represents the VGG19 architecture.

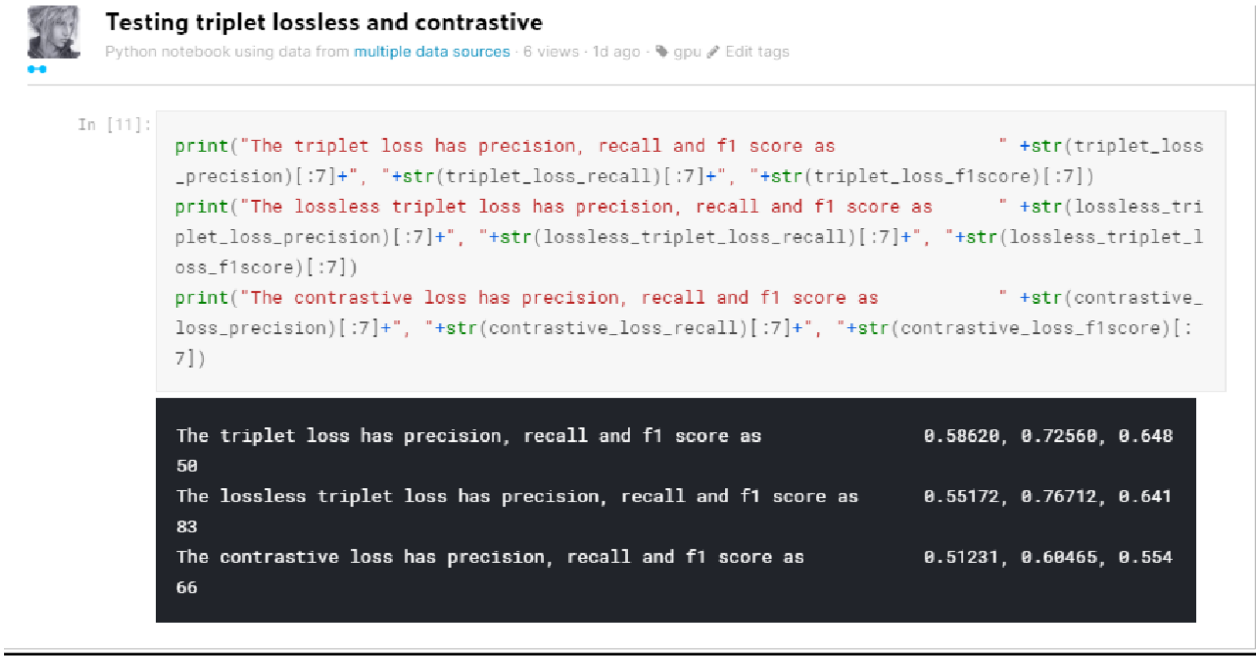
RESULT


Testing triplet lossless and contrastive

Python notebook using data from multiple data sources · 6 views · 1d ago · gpu Edit tags

In [9]:
```
print("forg_passed by model_triplet_loss is ",forg_passed[0])
print("gen_flagged by model_triplet_loss is ",gen_flagged[0])
print("forg_passed by model_lossless_triplet_loss is ",forg_passed[1])
print("gen_flagged by model_lossless_triplet_loss is ",gen_flagged[1])
print("forg_passed by model_contrastive_loss is ",forg_passed[2])
print("gen_flagged by model_contrastive_loss is ",gen_flagged[2])
```

```
forg_passed by model_triplet_loss is  45
gen_flagged by model_triplet_loss is  84
forg_passed by model_lossless_triplet_loss is  34
gen_flagged by model_lossless_triplet_loss is  91
forg_passed by model_contrastive_loss is  68
gen_flagged by model_contrastive_loss is  99
```


Testing triplet lossless and contrastive

Python notebook using data from multiple data sources · 6 views · 1d ago · gpu Edit tags

In [11]:
```
print("The triplet loss has precision, recall and f1 score as           " +str(triplet_loss
_precision)[:7]+", "+str(triplet_loss_recall)[:7]+", "+str(triplet_loss_f1score)[:7])
print("The lossless triplet loss has precision, recall and f1 score as     " +str(lossless_tri
plet_loss_precision)[:7]+", "+str(lossless_triplet_loss_recall)[:7]+", "+str(lossless_triplet_l
oss_f1score)[:7])
print("The contrastive loss has precision, recall and f1 score as          " +str(contrastive_
loss_precision)[:7]+", "+str(contrastive_loss_recall)[:7]+", "+str(contrastive_loss_f1score)[:
7])
```

```
The triplet loss has precision, recall and f1 score as           0.58620, 0.72560, 0.648
50
The lossless triplet loss has precision, recall and f1 score as     0.55172, 0.76712, 0.641
83
The contrastive loss has precision, recall and f1 score as          0.51231, 0.60465, 0.554
66
```

Conclusion, Limitations and Scope for future Work

S. No.	Type of loss function	Precision	Recall	F1 score
1.	Triplet loss	0.58620	0.72560	0.64850
2.	Lossless triplet loss	0.55172	0.76712	0.64183
3.	Contrastive loss	0.51231	0.60465	0.55466

The triplet loss is better with precision and Lossless triplet loss seems to be better with Recall. Overall, the Lossless triplet loss seems to have the higher accuracy.

The contrastive loss isn't working well. While the F1 score seems to prefer triplet loss over lossless triplet loss the difference isn't much. we'd rather go for the lossless triplet loss with higher recall. The higher recall could be due to the positive examples also contributing to the loss function in lossless triplet loss.

The main limitation we encountered was the lack of hardware necessary to perform sufficient testing.

In the future, the hyperparameters and thresholds need to be properly examined. Moreover, a balanced test set and a training set with some more variations and hard triplets need to be found and incorporated

Overall, we've seen that there will be some benefit in recall when we change from triplet loss to lossless triplet loss, even if that calls for a loss in precision. We think this loss is worth the gain in Recall and hence we've seen a better alternative to Triplet loss.

REFERENCES

1. L. G. Hafemann, R. Sabourin and L. S. Oliveira, "Writer-independent feature learning for Offline Signature Verification using Deep Convolutional Neural Networks," 2016 International Joint Conference on Neural Networks (IJCNN), Vancouver, BC, 2016, pp.2576-2583.doi: 10.1109/IJCNN.2016.7727521

2. 2. M. Diaz, A. Fischer, M. A. Ferrer and R. Plamondon, "Dynamic Signature VerificationSystem Based on One Real Signature," in IEEE Transactions on Cybernetics, vol. 48, no. 1,pp. 228-239, Jan. 2018.doi: 10.1109/TCYB.2016.2630419

3. 3. R. Tolosana, R. Vera-Rodriguez, J. Ortega-Garcia and J. Fierrez, "Preprocessing and FeatureSelection for Improved Sensor Interoperability in Online Biometric Signature Verification,"in IEEE Access, vol. 3, pp. 478-489, 2015doi: 10.1109/ACCESS.2015.2431493

4. 4. A.Hamadene and Y. Chibani, "One-Class Writer-Independent Offline Signature Verification Using Feature Dissimilarity Thresholding," in IEEE Transactions on iInformation Forensics and Security, vol. 11, no. 6, pp. 1226-1238, June 2016. doi: 10.1109/TIFS.2016.2521611

5. L. G. Hafemann, R. Sabourin and L. S. Oliveira, "Analyzing features learned for OfflineSignature Verification using Deep CNNs," 2016 23rd International Conference on Pattern Recognition (ICPR), Cancun, 2016, pp. 2989-2994.doi: 10.1109/ICPR.2016.7900092

6. ICDAR 2009 Signature Verification Competition data at - http://www.iapr-tc11.org/mediawiki/index.php/ICDAR_2009_Signature_Verification_Competition_(SigComp2009)