Verifikasi Kemiripan Wajah Menggunakan *Deep Learning* Dengan Arsitektur Jaringan *Siamese*

Hairul Imam[1], Jian Budiarto[2], Kartarina[3]

[1,3] *Jurusan Ilmu Komputer Universitas Bumigora*
Jln. Ismail Marzuki, Kota Mataram, Nusa Tenggara Barat. 83127, INDONESIA

[2] *1310520075@stmikbumigora.ac.id*

INFORMASI ARTIKEL
Received:
Received in revised:
Accepted:

KEYWORD
Siamese, Triplet Loss,* Verifikasi Wajah, *Face Embedding, Dimensionality Reduction.

INTISARI

Verifikasi wajah adalah masalah yang cukup populer dalam bidang *computer vision.* Banyak pendekatan yang telah dilakukan untuk menyelesaikan masalah tersebut baik menggunakan model matematika murni dengan mempelajari pola geometri pada wajah secara manual maupun cara otomatis menggunakan pendekatan pembelajaran mesin.

Penelitian ini mencoba memecahkan masalah tersebut dengan pendekatan *deep learning*, dimana model dilatih menggunakan *triplet loss* yang didefinisikan pada paper *FaceNet.* Rancangan model yang digunakan adalah *Siamese* dengan menerapkan ResNet-50 yang telah dimodifikasi untuk mempelajari fitur yang ada pada gambar sehingga mampu mereduksi dimensi gambar yang tinggi menjadi vektor baris yang rendah berdimensi 1x128 yang disebut sebagai *embedding*.

Setelah model berhasil mempelajari *embedding* yang baik pada gambar maka masalah verifikasi wajah bisa diselesaikan dengan membandingkan jarak *embedding* antar gambar dimana jarak yang dekat dapat diartikan sebagai wajah yang mirip (*genuine*) dan jarak yang jauh dapat diartikan sebagai wajah yang berbeda (*impostor*).

Pada penelitian ini, model berhasil dilatih pada dataset VGG Face v2 (*Visual Geometry Group*) dengan nilai akurasi 92% pada dataset LFW (*Labeled Faces in the Wild*) sebagai data *testing* dan mendapatkan nilai AUC (*Area Under the Curve*) 97%. Nilai AUC yang tinggi dapat diartikan bahwa model dapat memverifikasi dengan baik gambar wajah orang yang sama sebagai *genuine* dan gambar wajah orang yang berbeda sebagai *impostor.*

I. INTRODUCTION

Face verification is a problem of matching two pair of images and decide whether the two pair of images are similar (*genuine*) or dissimilar (*impostor*). Those decisions were made based by comparing the features between the two images. Hence the features were an integral part of deciding whether the pair of images are genuine or impostor those makes the main challenge of face verification problem is to find the best method that best extracts features on images. Various methods were taken to solve the challenge (see section II) such as using mathematical approach by extracting the face geometric pattern manually, with that approach an extra expertise on certain field will be required hence this approach is somewhat hard to be applied.

Another approach that also possible is by using *deep learning* which this method requires no extra expertise on certain field (prior knowledge) since this method is able to extract the feature pattern by learning an enormous amount of data that fed through them, hence this research uses deep learning approach to tackle the face verification problem.

The network architecture that this research uses Siamese network which the body of the network uses modified ResNet-50, then the network will be trained using *triplet loss* function. Triplet loss function is used with the objective to reducing the image with high dimensions to only represented by a row vector with lower dimension (*embedding*).

After the model optimally being trained then face verification problem can be conceived as comparing the embedding of two pair of images which by a simple distance function such as Manhattan distance or Euclidean distance we can get how far away or how close the distance of the two embeddings which means the closer the distance the more genuine the images or else otherwise. The techniques that were taken to be able train the optimal embedding will be explained on section III.

II. RELATED WORK

There were several methods to solve this problem were used before such as one used by Tri Mulyono et al [1] where on his work the approach he is using was by combining the *eigenface* with ANN (artificial neural network). Where the features were extracted using PCA (*Principal Component Analysis*) then the features that successfully extracted then being fed to the dense network (multilayer perceptron). This case uses ANN to only classify the features that has been extracted first with PCA, there's no feature extraction that done by ANN, the feature extractions were only done by PCA that makes this method depend highly on the ability of PCA to extract best feature of given images. He successfully train the ANN on 120 images with 10 individuals and successfully recognize faces with 84,6% on average.

Another similar method were also used by Dimas Achmad Akbar Kusuma [2] where on extracting features he uses *Discrete Cosine Transform* then the feature that successfully extracted then fed to the ANN (multilayer perceptron). The amount of dataset he is using was 100 with 10 different individuals with different poses each and successfully got an accuracy of 90% to 100% on evaluation process with the same person he is using during the training process.

Doing the manual feature extraction such as the two method that explained above is considered as a classical learning, where features were extracted first using certain algorithm then after the features successfully extracted then features fed to the multilayer perceptron that are being trained using some loss function with the objective to solve a given problem. This approach has a good result on a relatively small dataset, it's not yet tested is the same method also able to generalize on a larger dataset.

III. METHOD

This work will use the triplet loss function to train the network to be able to achieve a good embedding as defined on the Facenet paper by Florian Schroff et al [3]. Where the main component of triplet loss is by providing three different type of images also known as triplet pairs those are the anchor, the positive and the negative triplets. Anchor is the reference image, positive is the same individual image as the anchor and negative as the different individual as the anchor.

Image 1. The **triplet pair examples**. Images are considered as a valid triplet if satisfies the triplet criteria where anchor and positive triplets are the same person and negative is different person from the anchor.

The objective of this loss function is to have a fairly small distance between the anchor and the positive triplets and fairly large distance between anchor and negative triplets as can be seen on the equation (1) below.

$$\left\|x_i^a - x_i^p\right\|_2^2 + \alpha < \|x_i^a - x_i^n\|_2^2 , \forall \left(x_i^a, x_i^p, x_i^n\right) \in \mathrm{T} \tag{1}$$

Where α is the margin that assures there's a slightly small difference between positive, negative with the anchor triplet hence this loss function needs to be minimize as shown on the equation (2) below.

$$L = \sum_{i}^{N} \left[\left\|f(x_i^a) - f(x_i^p)\right\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha\right]_+ \tag{2}$$

Intuitively the above equation can be red that the distance between the anchor and the positive triplets need to be as small as it's difference is smaller than the difference between negative and anchor triplets. The α on the equation (2) above assures that the difference between positive triplet must be smaller than the difference between anchor and negative but not surpass the margin value so that the loss is maintained on positive scale. This objective is illustrated on the image 2[1] given below.

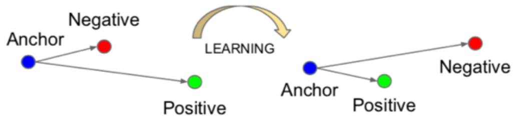


Image 2. **Triplet loss objective illustration**. The model needs to be trained so that the distance between positive and anchor less than the difference between negative and anchor.

Considering the equation (2) above, not all triplet pairs can be used to train the network. On the section below will be explained the type of triplets that can be considered to use for training.

[1] Sumber gambar paper FaceNet, referensi nomor 3.

A. Different Type of Triplets

To be able to train the network optimally it needs a good pair of triplets, three different triplets that can be used are:

1) Easy Triplet

Easy triplet is the kind of the triplet that has already satisfies the constraint function on equation 1. This type of triplets is unnecessary to the training process since will have 0 loss value and can causes the convergence process longer. Mathematically this type of triplet is expressed in the equation (3) below.

$$d(f(x^a), f(x^p)) + \alpha < d(f(x^a), f(x^n)) \tag{3}$$

Where d is a distance function, so the definition of easy triplet according above equation is that the distance between anchor and positive embedding plus some margin is less than the distance between anchor and negative embedding.

2) Hard Triplet

Hard triplet is the type of triplet where the distance between anchor and negative embedding is less than the distance between anchor and positive embedding, as can be seen on the equation (4) below.

$$d(f(x^a), f(x^n)) < d(f(x^a), f(x^p)) \tag{4}$$

3) Semi-hard Triplet

Semi-hard triplet is the type of triplet where the distance between anchor and negative embedding is in between distance of anchor and positive and anchor and positive with some margin so that this also can be interpreted as the difference between positive and negative is not greater than the margin value. This definition is expressed in the mathematical equation (5) below.

$$d(f(x^a), f(x^p)) < d(f(x^a), f(x^n)) < d(f(x^a), f(x^p)) + \alpha \tag{5}$$

So, after knowing the type of triplets above now our time to decide which type of triplets will be used to train the network. On this work the type of triplets we are using is semi-hard triplet.

B. Network Architecture

On the bigger picture the network will be constructed as shown in the image 3 below.

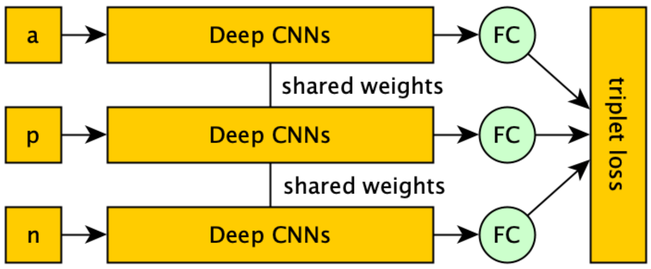


Image 3. **The network architecture big picture**. Each pair of triplets will be fed forward through the identical network block (shared weights). The features will be learned on the Deep CNNs block then this feature will fed through the Fully Connected layer (dense layer). The FC layer will be trained using triplet loss so that it can produce a good embedding.

1) Siamese

The main idea of Siamese architecture is to forward an input through two (twine) or even three (triplet) identical network architecture that shares the same parameters (weights and bias) so that on the training process the inputs will be processed with the same/identical network. On the image 3 above there are 3 networks that shares the same weight corresponding to the category of inputs that going to be passed through. The Deep CNNs block above is going to be modified Resnet-50 as explained below.

2) Modified ResNet-50

The complete block of ResNet-50 can be seen on the image 4 below.

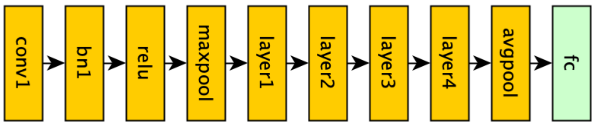


Image 4. **The complete ResNet-50 block**. The complete ResNet-50 block is constructed from conv1, bn1, relu, maxpool, layer1, layer2, layer3, layer4, avgpool and fc layer.

The modification of ResNet-50 that used on this work is based on the work by Hermans et al [4] and Rajeev Ranjan et al [5]. The modification here is the deletion of avgpool layer and the addition of L2-Normalize and alpha scale multiplication on the FC layer. The L2-Normalize and the alpha scale multiplication used here can be seen on the equation (6) and (7) below.

$$y = \frac{x}{\|x\|_2} \tag{6}$$

$$z = y \cdot \alpha \tag{7}$$

On the L2-Normalize equation (6) each value on inputs will be normalized with L2 norm of an input vector then the result will be multiplied by some alpha scale as seen on the equation (7) above. The value of *alpha scale* can be trained on the training process the same way as other parameters, but on the paper by Rajeev Ranjan stated that using a fixed alpha value performs better. On this work the alpha value that is used is 10.

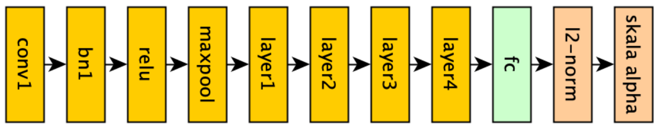


Image 5. **Modified block layer ResNet-50.** layer ResNet-50 with dropped avgpool layer and the addition of l2-norm and alpha scale on the FC layer.

The detail of inputs and outputs of each ResNet-50 blocks are presented on the table 1 below.

TABEL I. DETAILS OF INPUTS AND OUTPUTS OF RESNET BLOCKS

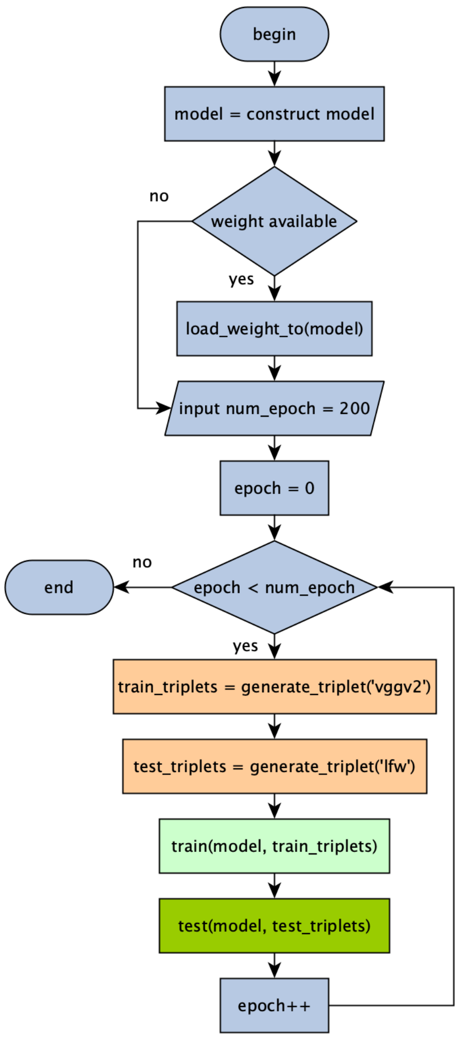
Layer	Input	Proses	Output
Conv1	1x3x224x224	Conv2D	1x64x112x112
BN1	1x64x112x112	BatchNorm	1x64x112x112
ReLU	1x64x112x112	ReLU	1x64x112x112
MaxPool	1x64x112x112	MaxPool	1x64x56x56
Layer1	1x64x56x56	Bottleneck1	1x256x56x56
	1x256x56x56	Bottleneck2	1x256x56x56
	1x256x56x56	Bottleneck2	1x256x56x56
Layer2	1x256x56x56	Bottleneck3	1x512x28x28
	1x512x28x28	Bottleneck4	1x512x28x28
	1x512x28x28	Bottleneck4	1x512x28x28
	1x512x28x28	Bottleneck4	1x512x28x28
Layer3	1x512x28x28	Bottleneck5	1x1024x14x14
	1x1024x14x14	Bottleneck6	1x1024x14x14
	1x1024x14x14	Bottleneck6	1x1024x14x14
	1x1024x14x14	Bottleneck6	1x1024x14x14
	1x1024x14x14	Bottleneck6	1x1024x14x14
Layer4	1x1024x14x14	Bottleneck7	1x2048x7x7
	1x2048x7x7	Bottleneck8	1x2048x7x7
	1x2048x7x7	Bottleneck8	1x2048x7x7
FC	1x100,352	Linear	1x128

Each input to the network will be passed through the block layers as shown on the table 1 above. Conv1 is the first layer of ResNet, this layer needs an input with 3 colour channel with the size of 224×224 each. The leading number 1 on each dimension of the tensor on the table 1 above is representing the size of data that being fed through the network (*batch size*).

The conv1 to layer4 what is called Deep CNNs block on the image 3 above, the outputs of this Deep CNNs block is a tensor with dimension of $1 \times 2048 \times 7 \times 7$ for each input. This extracted feature from Deep CNNs then will be passed through the FC layer as the features that this FC layer will be trained on to produce a good embedding. The dimension of a feature that will be passed through the FC layer is 100,352 this result is comes from after flatting the $1 \times 2048 \times 7 \times 7$ dimensions. The output of this FC layer is a row vector with dimension of 1×128 which will be the resulting embedding dimension so that all images that is high dimension is enough represented as 1×128 row vector.

C. *Training and Testing Process*

The bigger picture of the training and testing process can be seen on the image 6 below.



begin
model = construct model
weight available
no
yes
load_weight_to(model)
input num_epoch = 200
epoch = 0
epoch < num_epoch
no
end
yes
train_triplets = generate_triplet('vggv2')
test_triplets = generate_triplet('lfw')
train(model, train_triplets)
test(model, test_triplets)
epoch++

Image 6. **Training and testing bigger picture**. First model will be constructed, if there's already available model from the previous training it's going to be loaded first. The number of training and testing process will be the number of epoch wanted as good as it be. On each process will be generated an amount of valid triplets with datasets corresponding on each process.

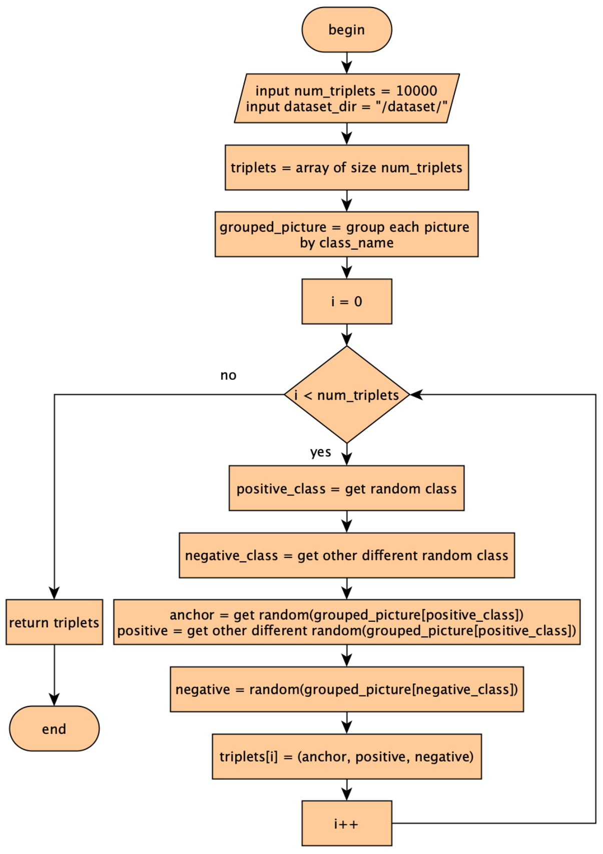
1) *Dataset*

The dataset that will be used on this work is a dataset from VGG (Visual Geometry Group) Face v2 as the training dataset and Labelled Face in the Wild (LFW) as the testing dataset. These two datasets are the most popular datasets that is use widely on face related problems caused by the variety and the data size.

VGG Face v2 itself has 9000 persons with variety on ethnics, accent, professions, genders and ages and has 3,3 million images. This dataset is distributed freely under Creative Common Attribution-ShareAlike 4.0 license.

LFW dataset is provided by University of Massachusetts. This dataset is widely used as the testing dataset to test how good the deep learning model generalizes a new dataset. This dataset containing 13,233 images with 5749 different persons where 1680 of them are has more than one images and the rest has only 1 image each.

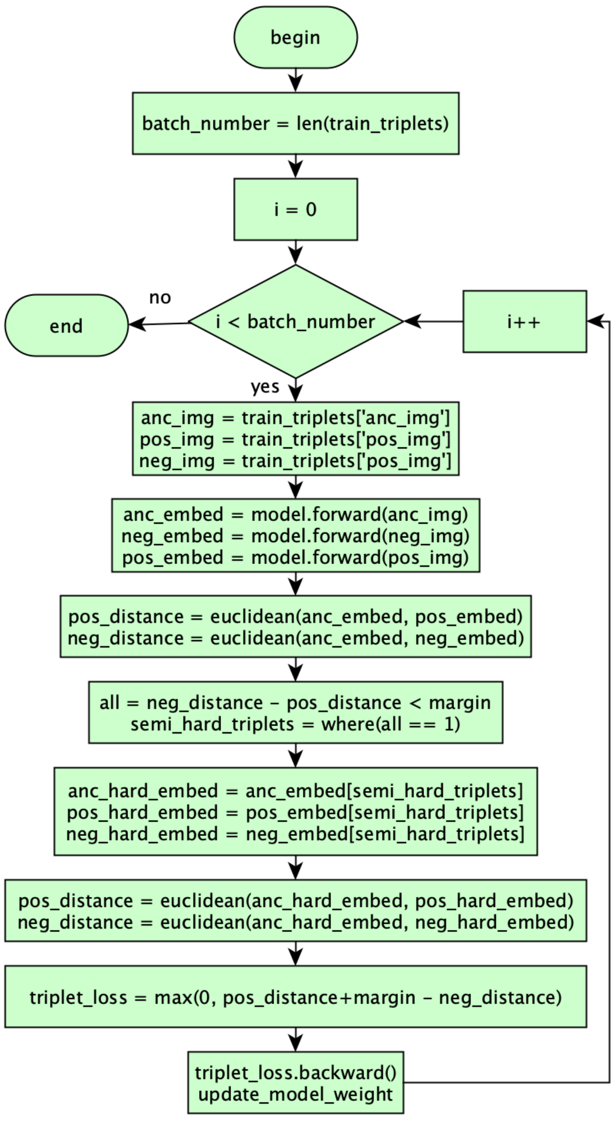
2) *Generate Triplet*



begin
input num_triplets = 10000
input dataset_dir = "/dataset/"
triplets = array of size num_triplets
grouped_picture = group each picture by class_name
i = 0
i < num_triplets
no
yes
positive_class = get random class
negative_class = get other different random class
anchor = get random(grouped_picture[positive_class])
positive = get other different random(grouped_picture[positive_class])
negative = random(grouped_picture[negative_class])
triplets[i] = (anchor, positive, negative)
i++
return triplets
end

Image 7. **The creation of random triplets flowchart**. First define the number of random triplets that will be constructed. Then do the loop on each loop take two images of the same class (person) as the anchor and positive and take one of other class as negative.

3) *Training Process*



begin
batch_number = len(train_triplets)
i = 0
i < batch_number
no
end
i++
yes
anc_img = train_triplets['anc_img']
pos_img = train_triplets['pos_img']
neg_img = train_triplets['pos_img']
anc_embed = model.forward(anc_img)
neg_embed = model.forward(neg_img)
pos_embed = model.forward(pos_img)
pos_distance = euclidean(anc_embed, pos_embed)
neg_distance = euclidean(anc_embed, neg_embed)
all = neg_distance - pos_distance < margin
semi_hard_triplets = where(all == 1)
anc_hard_embed = anc_embed[semi_hard_triplets]
pos_hard_embed = pos_embed[semi_hard_triplets]
neg_hard_embed = neg_embed[semi_hard_triplets]
pos_distance = euclidean(anc_hard_embed, pos_hard_embed)
neg_distance = euclidean(anc_hard_embed, neg_hard_embed)
triplet_loss = max(0, pos_distance+margin - neg_distance)
triplet_loss.backward()
update_model_weight

Image 8. **Training flowchart**. The number of training iteration is the number of random triplets after divided by batch number.

The important part of the training process is when choosing the good triplet to train the network with. On the image 8 above the triplet selection process done on the fourth block on the loop scope of the flowchart. The distance value of positive embeddings and the distance value of negative embedding will be used to determine the semi-hard triplets, where there must be a pair of triplets that the positive is greater than the negatives, all those images then used to calculate the gradient and do the backpropagation to update the weight with.

4) Testing Process

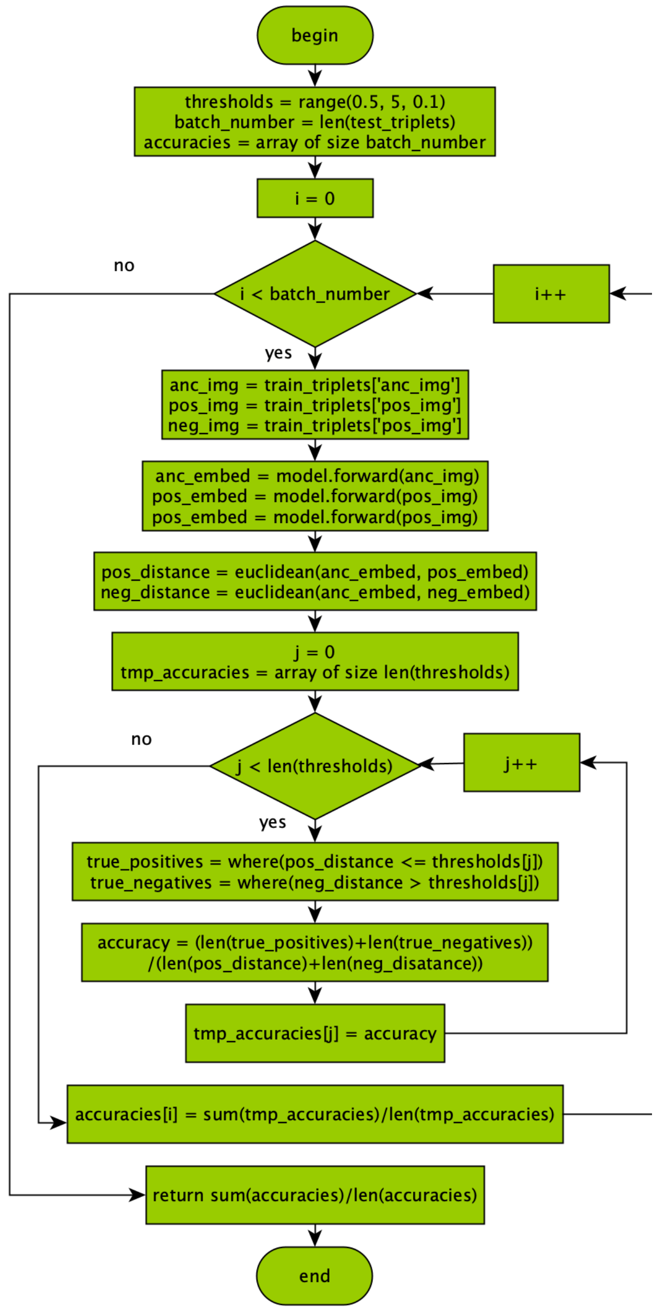

begin
thresholds = range(0.5, 5, 0.1)
batch_number = len(test_triplets)
accuracies = array of size batch_number
i = 0
no
i < batch_number
i++
yes
anc_img = train_triplets['anc_img']
pos_img = train_triplets['pos_img']
neg_img = train_triplets['pos_img']
anc_embed = model.forward(anc_img)
pos_embed = model.forward(pos_img)
pos_embed = model.forward(pos_img)
pos_distance = euclidean(anc_embed, pos_embed)
neg_distance = euclidean(anc_embed, neg_embed)
j = 0
tmp_accuracies = array of size len(thresholds)
no
j < len(thresholds)
j++
yes
true_positives = where(pos_distance <= thresholds[j])
true_negatives = where(neg_distance > thresholds[j])
accuracy = (len(true_positives)+len(true_negatives))
/(len(pos_distance)+len(neg_disatance))
tmp_accuracies[j] = accuracy
accuracies[i] = sum(tmp_accuracies)/len(tmp_accuracies)
return sum(accuracies)/len(accuracies)
end

Image 9. **Testing flowchart**. The accumulation accuracy is given by the average accuracy on each iteration.

On the testing process, the accuracy is calculated by the number of true positives and true negatives based on the predefined threshold values. The threshold that's going to be used here is between 0.5 to 5.0 with 0.1 steps.

IV. Results

For the testing purpose the hardware and software setup used is described on table below, where the hardware below are provided by GCP (Google Cloud Platform) service.

HARDWARE	4 x 12GB NVIDIA Tesla K80 GPU
	8 Core vCPU Intel Broadwell
	52GB RAM
	256GB Boot Disk
	200GB Dataset Disk
SOFTWARE	Debian 4.9.130-2 (2018-10-27) x86_64 GNU/Linux
	Python v3.7.1
	Pytorch v1.0
	CUDA v10.0

The training and testing is divided into two steps, first the training and testing on the FC layer and second training and testing on the Deep CNN layer.

1) Training and Testing FC Layer

On this process the inputs to the FC layer is the features that learned from the pretrained ResNet-50. The feature extraction layer (Deep CNN) will be retrained to gain the more understanding on the training dataset that is used on this work. The result of training this layer is given by the image 10 and 11 below.

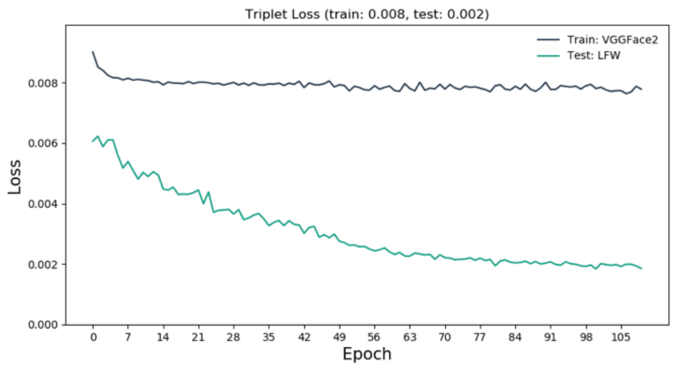


Image 10. **loss each epoch FC layer**. Model successfully trained on FC layer and gained loss 0.008 for training set and 0.002 for testing set.

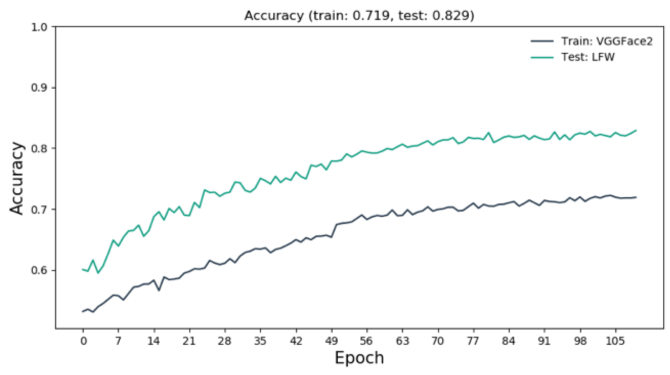


Image 11. **Accuracy each epoch FC layer**. Model successfully trained on FC layer and gained 71% accuracy on training set and 82% accuracy on testing set.

2) *Training and Testing Deep CNN Layer*

On this process the Deep CNN layer will be retrained to be able to learn a new pattern on the training dataset that is used here. The results are given on the image below.

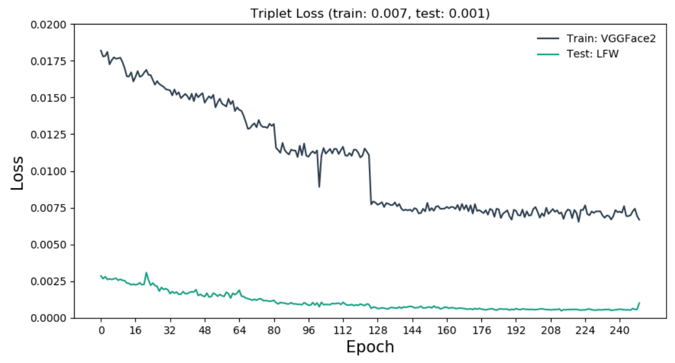
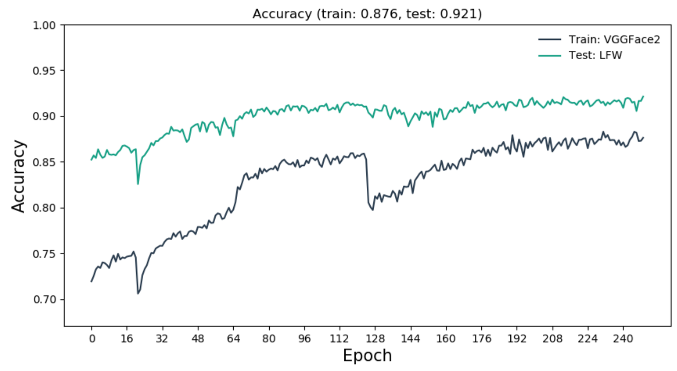


Image 12. **Loss each epoch Deep CNN layer**. Model successfully trained on FC layer and gained loss 0.001 for training set and 0.001 for testing set.



Gambar 13. **Accuracy result each epoch Deep CNN layer**. Model successfully trained on FC layer and gained 87% accuracy on training set and 92% accuracy on testing set.

As can be seen on the image 13 above model has successfully gained 87% accuracy on the training dataset and gained 92% accuracy on the testing dataset. The accuracy value that is greater on testing set than training set indicating that the model has learn and not overfit hence able to generalize the dataset even for new dataset that the model has never been see before. The ROC (Receiver Operating Characteristic) on the last training and testing process can be seen on the image 14 below.

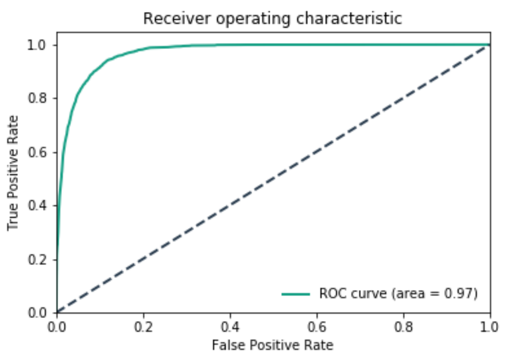


Image 14. **ROC chart and the AUC value**. Model successfully gained AUC value of 97.

The AUC value of 97% can be interpreted that the model is able to verify correctly the genuine faces and the impostor faces 97% of the time.

3) *Face Verification Testing*

After the model successfully gained 92% accuracy on the testing dataset. This section will try to do the face verification on 30 pair of persons randomly with different threshold as explained on the sub section Testing Process. Then the number of true positives and true negatives of each threshold will be count. The result of this section is given on the image 15 below.

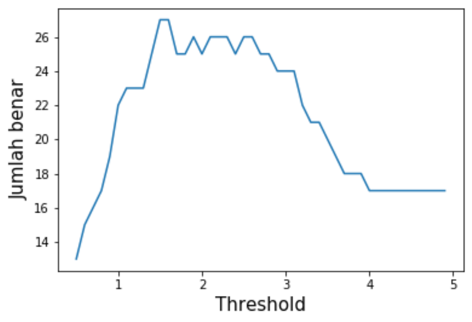


Image 15. **Correct numbers each threshold**. This chart shows that the highest correct number is provided on the threshold 1,5 and 1.6.

As shown on the cart above the highest number of correct verifications is given on the threshold 1.5 and 1.6 with the number of correct is 27 out of 30 trials. The complete list of correct verification number on each threshold is provided on the table 2 below.

TABEL 2. CORRECT VERIFICATION NUMBER EACH THRESHOLDS

Nilai Threshold	Jumlah Benar
0,5	13
0,6	15
0,7	16

0,8	17
0,9	19
1,0	22
1,1 – 1,3	23
1,4	25
1,5 – 1,6	27
1,7 – 1,8	25
1,9	26
2,0	25
2,1 – 2,3	26
2,4	25
2,5 – 2,6	26
2,7 – 2,8	25
2,9 – 3,1	24
3,2	22
3,3 – 3,4	21
3,5	20
3,6	19
3,7 – 3,9	18
4,0 – 4,9	17

V. CONCLUSION

Based on the research results above we can conclude a few things that:

1. Siamese architecture is successfully learned the face feature from VGGv2 using modified ResNet-50 so that it could produce a good embedding
2. Siamese architecture successfully learned a good embedding by using modified ResNet-50 with 92% accuracy on testing dataset.
3. The model successfully learned the features on training dataset and has been successfully tested on testing dataset and gained AUC value of 97%.

REFERENCE

[1] T. Mulyono, K. Adi, and R. Gernowo, “Sistem Pengenalan Wajah Dengan Metode Eigenface Dan Jaringan Syaraf Tiruan (Jst),” *Berk. Fis.*, vol. 15, no. 1, pp. 15–20, 2012.

[2] D. A. A. Kusuma, F. Ardilla, and B. S. B. Dewantara, “Verifikasi Citra Wajah Menggunakan Metode Discrete Cosine Transform Untuk Aplikasi Login,” *Ind. Electron. Semin.*, vol. 8, no. 5, p. 55, 2011.

[3] B. Huang, “FaceNet: A Unified Embedding for Face Recognition and Clustering,” pp. 1–2, 2015.

[4] A. Hermans, L. Beyer, and B. Leibe, “In Defense of the Triplet Loss for Person Re-Identification,” 2017.

[5] R. Ranjan, C. D. Castillo, and R. Chellappa, “L 2 -constrained Softmax Loss for Discriminative Face Verification,” 2017.