Mugla Journal of Science and Technology

A TENSORFLOW BASED METHOD FOR LOCAL DERIVATIVE PATTERN

Devrim AKGUN*, Software Engineering Department, Sakarya University, Turkey, dakgun@sakarya.edu.tr
(https://orcid.org/0000-0002-0770-599X)

Received: 24.11.2020, Accepted: 20.02.2021 Research Article
*Corresponding author DOI: 10.22531/muglajsci.830691

Abstract

Python programming language provides a very convenient environment of implementing machine learning applications. However, programmers usually faced with a poor performance compared to compiled functions when they write script based programs that demands intense computations. TensorFlow framework provides acceleration by enabling the utilization of various computing resources such as multicore CPU and GPU unit as well as including various compiled algorithms for developing machine learning applications. In this way, algorithms developed using existing TensorFlow operations can shorten computation times by using these resources indirectly without requiring parallel programming or GPU programming. In this study, Local Derivative Pattern (LDP) analysis which is one of the efficient feature extraction approaches for machine learning models was realized using a TensorFlow based algorithm. Independent pixel based operations in LDP algorithm which requires intense computations, enable developing an efficient tensor based algorithm. The performance of the TensorFlow based algorithm has been measured by comparing it with the Python script version of the same algorithm. The results obtained for various sizes and numbers of sample images show that TensorFlow operations provide significant acceleration rates for the LDP algorithm.

Keywords: TensorFlow, Local Derivative Pattern, Feature extraction

YEREL TÜREV ÖRÜNTÜ İÇİN TENSORFLOW TABANLI BİR METOD

Özet

Python programlama dili makine öğrenmesi uygulamalarının gerçekleştirilmesi için oldukça uygun bir ortam sağlar. Ancak programcılar yoğun işlem gerektiren script tabanlı fonksiyonlar yazdığı zaman, genelde derlenmiş kodlara göre düşük performans problemi ile karşılaşır. TensorFlow çerçevesi, makine öğrenmesi uygulamalarının geliştirilmesi için çeşitli derlenmiş algoritmalara sahip olmakla birlikte çok çekirdekli CPU ve GPU birimleri gibi hesaplama kaynaklarını da kullanarak hızlandırma sağlar. Bu sayede mevcut TensorFlow operasyonları kullanılarak geliştirilen algoritmalar, paralel programlama veya GPU programlama gerektirmeden dolaylı yoldan bu kaynakları kullanarak hesaplama sürelerini kısaltabilir. Bu çalışmada, makine öğrenmesi uygulamalarında etkili özellik çıkarma yöntemlerinden biri olarak kullanılan Yerel Türev Örüntü (LDP) analizi TensorFlow tabanlı bir algoritma ile gerçekleştirilmiştir. Yoğun hesaplama yükü gerektiren LDP algoritmasında bağımsız piksel tabanlı işlemler, verimli bir tensör işlemleri tabanlı algoritma geliştirilmesine olanak tanımaktadır. TensorFlow ile gerçekleştirilen algoritmanın başarımı, aynı algoritmanın Python script ile gerçekleştirilen sürümü ile karşılaştırılarak ölçülmüştür. Çeşitli boyutlarda ve sayılarda örnek görüntüler için elde edilen sonuçlar, TensorFlow işlemlerinin LDP algoritması için önemli hızlandırma oranları sağladığını göstermektedir.

Anahtar Kelimeler: TensorFlow, Yerel Türev Örüntü, Özellik çıkarımı

Cite

Akgun, D., (2021). "A TensorFlow Based Method For Local Derivative Pattern, Mugla Journal of Science and Technology, 7(1), 59-64.

1. Introduction

Feature extraction methods which involves most of the machine learning applications helps reduce the dimensionality of dataset while improving the learning process. LDP is of one the efficient feature extraction methods used in computer vision applications. It was first introduced in a face identification and verification study and shown to be successful on various datasets [1]. LDP checks local variations using derivative and forms a binary number according to directions. Local variations can be measured in various angles enabling detailed information about the input data. In literature there are various computer vision studies utilizing LDP for example; infrared image based fingerprint recognition [2], image annotation [3], person re-identification [4], content based medical image retrieval [5], image registration [6], facial expression recognition [7] and various face recognition approaches [8]–[10].

Because of the pixel based multiplication and comparison operations in LDP algorithm, the computational demand of the algorithm increases as the size of the image is increased. Running the algorithm using Python may result in longer durations when compared to low level programming languages. Languages like C/C++ are closer to hardware and they are optimized during compile time to reduce running time of the program. However, Python is a scripting language where an interpreter processes input lines and convert it to machine code which adds some overhead to the execution of the program. For all that, machine learning and scientific applications are widely developed using Python because of the ease scripting languages provide. Due to its popularity one can find various resources on machine learning and numerical algorithms. TensorFlow is one of the major open source libraries in machine learning and deep learning [11], [12]. It provides various tools and makes it practical to train and test in Python.

TensorFlow also supports GPU (Graphics Processing Unit) utilization which provides significant accelerations for intense computations. Users can define their custom operations in TensorFlow as well as constructing custom layers [13]–[15]. In addition to coding from the scratch, custom operations can be defined by combining existing TensorFlow operations.

In the presented study, a custom operation for LDP has been implemented using TensorFlow operations. Proposed algorithm can be used with deep learning models as intermediate layer or preprocessing function. In order to test the efficiency of the algorithm images in various dimensions such as 448×448, 224×224, 112×112, 56×56 and 28×28 were used. Also various batch sizes for each image sizes were used in experimental evaluations. Organization of the rest of the study is as follows; information about the basics of LDP was given in following section, the proposed design was explained in Section 3, numerical evaluations using TensorFlow implementation were realized in Section 4. A brief conclusions of the study were given in the last section.

2. Local Derivative Transform

LDP works similar to Local Binary Patterns (LBP) [16], but LBP only use first order derivatives with respect to neighbors within selected mask. LDP also incorporates second order derivatives based on the first order derivatives which extracts more detailed features from the input image. Also the approach enables computing n^{th} order derivatives based on the $(n\text{-}1)^{th}$ order derivatives.

Let the directional first order derivatives of input image $I(P)$ for the directions such as α=0°, α=45°, α=90° and α=135° be $I'_0(p_0)$, $I'_{45}(p_0)$, $I'_{90}(p_0)$ and $I'_{135}(p_0)$ respectively . First order derivatives at the p_0 location for all directions can be computed as follows,

$$\left.\begin{aligned} I'_0(p_0) &= I(p_0) - I(p_4) \\ I'_{45}(p_0) &= I(p_0) - I(p_3) \\ I'_{90}(p_0) &= I(p_0) - I(p_2) \\ I'_{135}(p_0) &= I(p_0) - I(p_1) \end{aligned}\right\} \quad (1)$$

Where $p0$ describes the center pixel and $p1$, $p2$,...,$p8$ pixels within 3×3 neighborhood as shown by Figure 1.

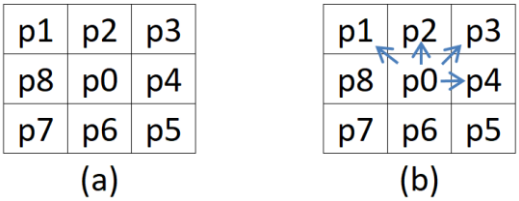


Figure 1. a) Example neighbors of P_0 pixel and b) directional derivatives for P_0.

The second-order derivatives are written based on the first order derivatives. For example, second order derivatives for p_0 can be written for α=0° as below,

$$LDP^2_0(p_0) = \begin{cases} f(I'_0(p_0)I'_0(p_1)) \\ f(I'_0(p_0)I'_0(p_2)) \\ \quad\cdots \\ f(I'_0(p_0)I'_0(p_8)) \end{cases} \quad (2)$$

In the same way, the equations are obtained for α=45° as below,

$$LDP^2_{45}(p_{45}) = \begin{cases} f(I'_{45}(p_0)I'_0(p_1)) \\ f(I'_{45}(p_0)I'_0(p_2)) \\ \quad\cdots \\ f(I'_{45}(p_0)I'_0(p_8)) \end{cases} \quad (3)$$

In Eq.2, $f(.)$ contains a comparison function that produces a binary value as given by Eq. 4. It checks the derivative directions for different neighbor pixels.

$$f\left(I'_\alpha(P_0)I'_\alpha(P_i)\right) = \begin{cases} 0, & I'_\alpha(P_0)I'_\alpha(P_i) > 0 \\ 1, & I'_\alpha(P_0)I'_\alpha(P_i) \le 0 \end{cases} \quad (4)$$

Computations given by Eq. 2 and Eq. 3 are repeated for other directions as follows,

$$LDP^2(P) = \begin{cases} LDP^2_0(P) \\ LDP^2_{45}(P) \\ LDP^2_{90}(P) \\ LDP^2_{135}(P) \end{cases} \quad (5)$$

3. TensorFlow Implementation

TensorFlow framework provides most of the tools for developing, training and testing machine learning models. TensorFlow has APIs for several languages such as Python, Java and C++. Also it has the flexibility of defining custom functions either from scratch or as a

combination of existing functions. Because TensorFlow supports GPU computation, in the latter case GPU support is enabled for custom function without writing GPU level code. This is possible by using basic functions that have GPU support provided by TensorFlow library such as *add*(), *multiply*() and *matmul*().

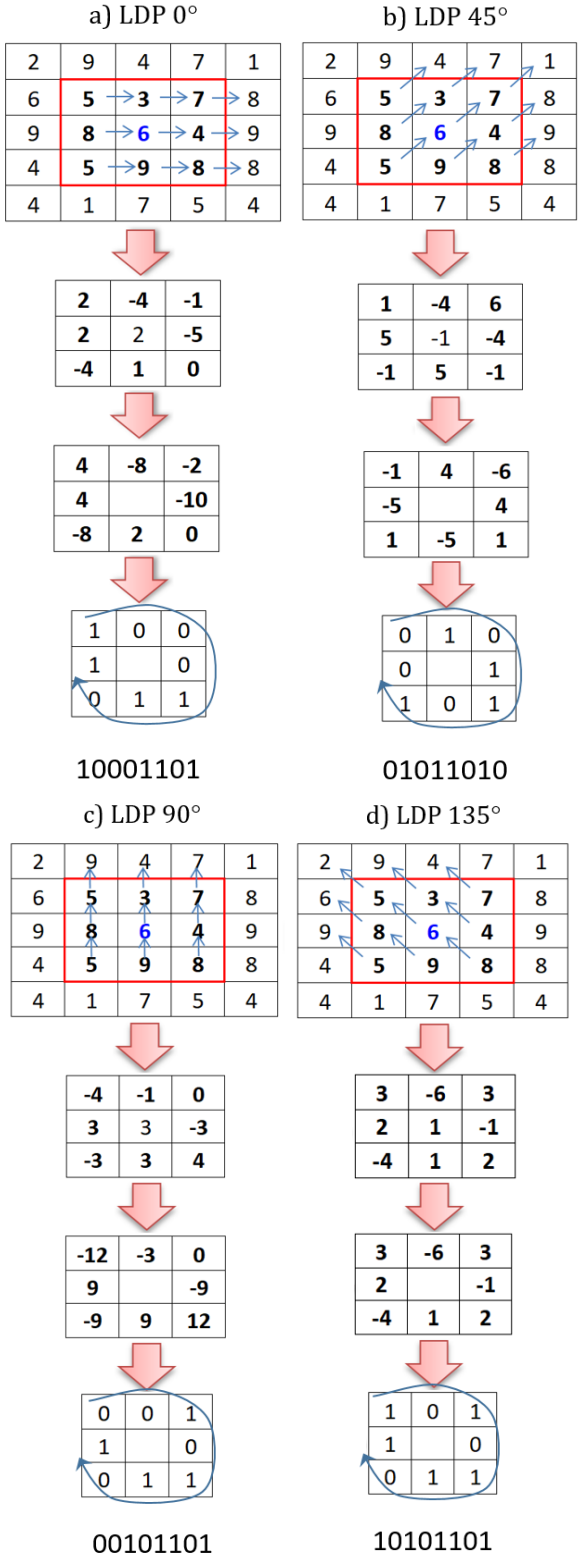


Figure 2. Example computations for first order LDP where α=0°, α=45°, α=90° and α=135°.

Example second order computations using Eq. 2 for α=0°, α=45°, α=90° and α=135° are shown by Figure 2. Operations for four angles are similar except the directions of the derivatives which are computed initially. A Python realization of the algorithm for α=0° is given by Code snippet 1. At first, Eq. 1 is applied to a pixel which is determined by for-loops and its neighborhood of 3×3 dimensions. Then Eq.2 and Eq.3 are applied to obtain second order derivatives. Extracted feature is computed by summing the resulting values after converting each digit to a decimal value.

```
1.  #Allocate memory for output
2.  L=np.zeros((rows,cols))
3.  #Traverse the pixels of image
4.  for i in range(2,nRows-2):
5.    for j in range(2,nCols-2):
6.      #Compute first order derivatives for α=0° (Eq. 1)
7.      d1= I[i-1,j-1]-I[i-1,j+0]
8.      d2= I[i-1,j+0]-I[i-1,j+1]
9.      d3= I[i-1,j+1]-I[i-1,j+2]
10.     d4= I[i+0,j-1]-I[i+0,j+0]
11.     d5= I[i+0,j+1]-I[i+0,j+2]
12.     d6= I[i+1,j-1]-I[i+1,j+0]
13.     d7= I[i+1,j+0]-I[i+1,j+1]
14.     d8= I[i+1,j+1]-I[i+1,j+2]
15.     #Select the pixel at the center
16.     dc = I[i,j]-I[i,j+1]
17.     #Compute second order derivatives (Eq. 2)
18.     p1=(d1*dc)>=0)*128
19.     p2=(d2*dc)>=0)*64
20.     p3=(d3*dc)>=0)*32
21.     p4=(d4*dc)>=0)*1
22.     p5=(d5*dc)>=0)*16
23.     p6=(d6*dc)>=0)*2
24.     p7=(d7*dc)>=0)*4
25.     p8=(d8*dc)>=0)*8
26.     #Compute extracted feature
27.     s=p1+p2+p3+p4+p5+p6+p7+p8
28.     L[i,j]=s
```

Code snippet 1. Python implementation of LBP transform for α=0°.

LDP algorithm can be realized by existing TensorFlow operations since the computation of output pixels for LDP transform involves independent matrix operations. It requires various comparison, multiplication, addition and subtraction for each pixel operations and these are independent for each 3×3 mask. Therefore, the two for-loop can be eliminated in TensorFlow implementation by using tensors which are N-dimensional arrays. Code snippet 2 shows a fragment of the program using TensorFlow operations. In this code *y0*, for example is a matrix that defines the pixels at center for all masks. Similarly *y4* defines the right neighbor of *y0* in matrix form. Initially the first order derivatives are determined for the center pixels by using *tf.subtract*(). Before doing the comparison given by Eq.3, false and true conditions are defined. Initially, false condition is defined by *tf.zeros*() and for the following pixels it is used as previous results which in the false condition doesn't change the result. Similarly true condition is also defined using the values according to

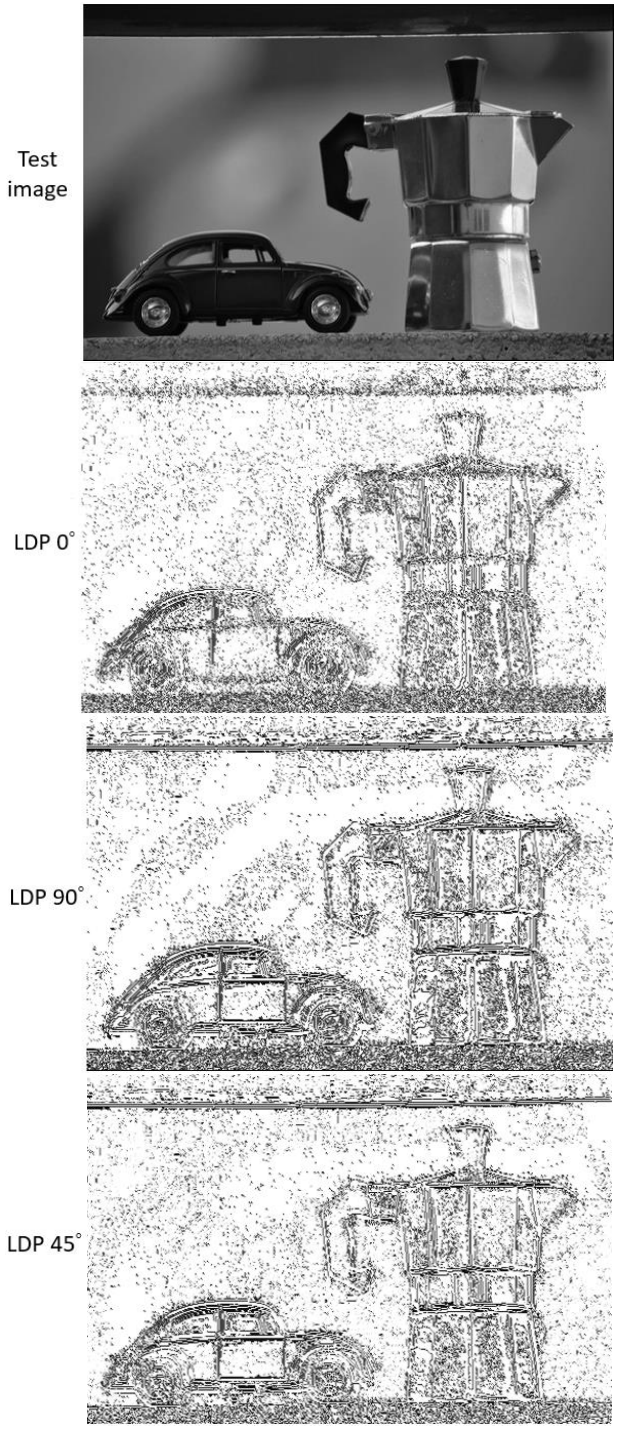
weight of the binary digit which is 128 for $p1$. Then, first order derivative is computed for the selected neighbor and multiplied with $p0$ like in the Code Snippet. Following the comparison with *tf.greater_equal*(), one of the conditions are selected with *tf.where*(). After $p1$, computations for $p2$ are realized in the example code fragment. In this case, false condition is selected as the previous results. Other operations are the same as the previous one except the weight of the binary digit. Hence, similar operations are repeated for the other neighbor of the center pixel to obtain LDP transform for α=0°. Also this procedure is repeated for α=45°, α=90° and α=135° for computing all four directions. Then the obtained 8 byte results are concatenated to obtain a 32 bit feature vector and this vector is used to extract features in various applications [17], [18].

```
#compute first order derivative for center pixel
dc=tf.subtract(y0,y4)
#computation of p1 in code snippet 1
cond_false=tf.zeros(p0.shape,tf.float32)
cond_true =tf.add(cond_false,
           tf.constant(128.0/255.0))
d0 =tf.subtract(y1,y2)
d0 =tf.multiply(d0,dc)
g  =tf.greater_equal(dc,0)
z  =tf.where(g,cond_true,cond_false)
# computation of p2 in Code Snippet 1
cond_false=z
cond_true =tf.add(z,
            tf.constant(64.0/255.0))
d1 =tf.subtract(y2,y3)
d1 =tf.multiply(d1,dc)
g  =tf.greater_equal(d1,0)
z  =tf.where(g,cond_true,cond_false)
# computation of p3 in Code Snippet 1
cond_false=z
cond_true =tf.add(z,
              tf.constant(32.0/255.0))
d2 =tf.subtract(y3,y4)
d2 =tf.multiply(d2,dc)
g  =tf.greater_equal(d2,0)
z  =tf.where(g,cond_true,cond_false)
```

Code snippet 2. A fragment of TensorFlow implementation of LDP transform for α=0°.

4. Experimental Results

Running times for the example cases were measured using time command from time library in Python as shown by Code Snippet 3. In that code, the features are obtained separately for four directions. In practice, these features are used together to train a machine learning model. The average of repeated measurements for 30 times to obtain example running times. Various dimensions of images and batch of images were used in tests. Experiments were carried on a hardware that contains AMD FX2700 processor and GTX1080 Nvidia GPU. Operating system was Ubuntu 18.04 where Python 3.7.9 and TensorFlow 2.3.1 were installed. Example images for obtaining performance evaluations are selected randomly from ImageNet [19] dataset. An example image and its LDP visualizations for α=0°, α=90°α=45° and α=135° were given by Figure 3. In the experiments, selected images were rescaled to various dimensions such as 28×28, 56×56, 112×112, 224×224 and 448×448. Since the content of the image doesn't have effect on the computation times similar experiments can be repeated using another dataset.



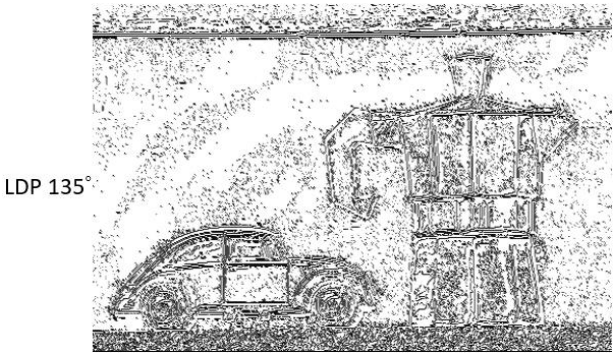


Figure 3. Example image and its LDP transformed images for α=0°, α=90°α=45° and α=135°.

```
#get starting time
start_time = time.time()
# Compute LDP for a given batch of images
ldp_feat_0  =ldp0(  batch_of_images).numpy()
ldp_feat_45 =ldp45( batch_of_images).numpy()
ldp_feat_90 =ldp90( batch_of_images).numpy()
ldp_feat_135=ldp135(batch_of_images).numpy()
# get finishing time and determine the elapsed time
elapsed_time  = time.time() - start_time
```

Code snippet 3. TensorFlow implementation of LBP transform.

Table 1. LDP algorithm running times for Python implementation (seconds)

Batch	28×28	56×56	112×112	224×224	448×448
1	0.1282	0.5699	2.3753	9.7110	39.435
2	0.2571	1.1439	4.7235	19.420	78.813
4	0.5076	2.2595	9.5201	39.020	159.70
8	1.0208	4.5776	19.061	78.212	319.73
16	2.0228	9.1329	38.089	156.49	629.95
32	4.0286	17.991	76.081	311.15	1271.32

Table 2. LDP algorithm running times for TensorFlow implementation (seconds)

Batch	28×28	56×56	112×112	224×224	448×448
1	0.0080	0.0083	0.0096	0.0163	0.0235
2	0.0082	0.0085	0.0118	0.0209	0.0503
4	0.0085	0.0089	0.0165	0.0252	0.0929
8	0.0086	0.0117	0.0187	0.0498	0.2068
16	0.0093	0.0168	0.0242	0.1116	0.3580
32	0.0119	0.0196	0.0496	0.1926	0.5958
64	0.0169	0.0227	0.0984	0.3420	2.6339
128	0.0199	0.0411	0.1999	0.7094	5.2664
256	0.0246	0.0892	0.3590	1.9699	10.484

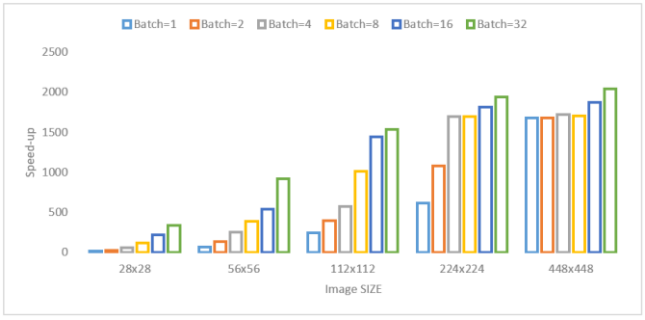


Figure 4. Speed-up evaluations

Running durations were first obtained for Python script for all directions. Code Snippet 1 which was given for α=0° was realized for the other directions which are α=45°, α=90° and α=135°. According to results given by Table 1 running times increases significantly as the images dimensions and batch size are increased. Running times are approximately related to each other linearly. For example, while running duration for 112×112 is 2.375 for the batch size 1, it increases to 4.724 when the batch size is increased to 2. The same is not true for the TensorFlow measurements where GPU and multicore CPU can be utilized. As shown by Table 2 where the results for TensorFlow implementation were given, while some measurements have linear relation when considering the batch size or image size, some measurements don't have such relation. On the other hand, when the numerical results given by Table 1 and Table 2 are compared, it is seen that the running durations are considerably reduced. The speed-up is significantly increased as the image size and batch size are increased as shown by Figure 4.

5. Conclusion

LDP algorithm that extracts features from images depending on derivative operations in four directions is very useful for machine learning applications. These kinds of algorithms are usually costly to implement with Python script as the experimental results imply. TensorFlow enables users writing custom functions as the combinations of basic TensorFlow operations and these functions can be used as custom deep learning layer or preprocessing function. Since operations in TensorFlow compiled with optimization and they support multicore CPU and GPU device, running durations of custom functions are also reduced significantly when compared with the Python scripts. In order to better utilize operations, the algorithm should be expressed in terms of tensors which are n-dimensional arrays. In the case of LDP, since the pixels based computations are independent, the algorithm can be expressed in terms of basic operations. According to experimental running durations, custom function for LDP provides significant accelerations over Python script implementation. Also the results indicate that acceleration with TensorFlow increases as the tested image dimensions and batch size are increased which is

mainly due to the efficiency of the GPU utilization for small images deteriorates. Although the results will vary according to experimental hardware specifications and software versions, they give good insight into the TensorFlow acceleration for various dimensions of images and batch sizes.

6. References

[1] Zhang , B., Gao, Y., Zhao, S. and Liu, J., "Local derivative pattern versus local binary pattern: Face recognition with high-order local pattern descriptor," IEEE Trans. Image Process., vol. 19, no. 2, pp. 533–544, 2010.

[2] Lee, E. C., H. Jung and Kim, D., "New finger biometric method using near infrared imaging," Sensors, vol. 11, no. 3, pp. 2319–2333, 2011.

[3] Srivastava,G. and Srivastava, R., "Annotation of images using local binary pattern and local derivative pattern after salient object detection using minimum directional contrast and gradient vector flow," Signal, Image Video Process., pp. 1–9, 2020.

[4] Imani, Z. and Soltanizadeh, H., "Local Binary Pattern, Local Derivative Pattern and Skeleton Features for RGB-D Person Re-identification," Natl. Acad. Sci. Lett., vol. 42, no. 3, pp. 233–238, 2019.

[5] Darapureddy, N., N. and Karatapu, Battula, T. K., "Optimal weighted hybrid pattern for content based medical image retrieval using modified spider monkey optimization," Int. J. Imaging Syst. Technol., p. e22475, 2020.

[6] Jiang, D., Shi, Y., Chen, X., Wang, M. and Song, Z., "Fast and robust multimodal image registration using a local derivative pattern:," Med. Phys., vol. 44, no. 2, pp. 497–509, 2017.

[7] Kalam, A., Hasan, M., Enamul Haque, M., Ibrahim, M., Jashem, M. and Jabid, T., "Facial expression recognition using local composition pattern," in ACM International Conference Proceeding Series, 2019, pp. 63–67.

[8] Soltanpour, S. and Wu, Q. M. J., "Weighted Extreme Sparse Classifier and Local Derivative Pattern for 3D Face Recognition," IEEE Trans. Image Process., vol. 28, no. 6, pp. 3020–3033, 2019.

[9] Kwon, O. S., "Illuminant-invariant face recognition using high-order local derivative pattern," J. Imaging Sci. Technol., vol. 62, no. 1, p. 10501, 2018.

[10] Liang, J., Zhou, J. and Gao, Y. "3D local derivative pattern for hyperspectral face recognition," in 2015 11th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition, FG 2015, 2015, vol. 1, pp. 1–6.

[11] Abadi, M. et al., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," arXiv Prepr. arXiv1603.04467, 2016.

[12] Abadi, M. et al., "TensorFlow: A system for large-scale machine learning," in Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, 2016, pp. 265–283.

[13] Chien, S. W. D., Markidis, S., V. Olshevsky, Bulatov, Y., E. Laure and Vetter, J., "TensorFlow Doing HPC," in 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2019, pp. 509–518.

[14] Agrawal, A. et al., "TensorFlow Eager: A Multi-Stage, Python-Embedded DSL for Machine Learning," arXiv Prepr. arXiv1903.01855, 2019.

[15] Andrade-Loarca, H. and Kutyniok, G., "tfShearlab: The TensorFlow Digital Shearlet Transform for Deep Learning," arXiv Prepr. arXiv2006.04591, 2020.

[16] Pietikäinen, M., Hadid, A., Zhao, G. and Ahonen, T., "Local Binary Patterns for Still Images," Springer, pp. 13–47, 2011.

[17] Zhao, Y., Ding, Y. and Zhao, X. Y., "Image quality assessment based on complementary local feature extraction and quantification," Electron. Lett., vol. 52, no. 22, pp. 1849–1851, 2016.

[18] Gomathy Nayagam, M. and Ramar, K., "Reliable object recognition system for cloud video data based on LDP features," Comput. Commun., vol. 149, pp. 343–349, 2020.

[19] Deng, J., Dong, W., Socher, R., Li, L.J., Li, Kai and Fei-Fei, Li, "ImageNet: A large-scale hierarchical image database," IEEE conference on computer vision and pattern recognition, 2010, pp. 248–255.