International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2020 IJSRCSEIT | Volume 6 | Issue 2 | ISSN : 2456-3307
DOI : https://doi.org/10.32628/CSEIT206245

Training the Image Classifier with and without Data Augmentation

Dr. R. Nithya*[1]

*[1]Department of Computer Science, Dr. N.G.P. Arts and Science College, Coimbatore, Tamil Nadu, India

ABSTRACT

The main objective of this paper is to train the image classifier using Convolutional Neural Networks with tensorflow architecture. The proposed paper focus on systematic approaches in classifying the sample set of images using Convolutional Neural Networks. The CNN model with activation function thus classifies the dataset into two categories exactly like human. Thus, the paper highlights the importance of augmentation by comparing their accuracies.

Keywords: Image Classification, Artificial Neural Networks, Deep Learning, Convolutional Neural Networks, Tensorflow, Data Augmentation

I. INTRODUCTION

Image classification identifies the unique features of the images by assigning different pixel to different classes of images. This computer vision classifies images based on user specified features. Stock Photography and Video Websites, Visual Search for Improved Product Discoverability, Satellite Remote Sensing, Interactive Marketing and Creative Campaigns Automobile, Gaming, Healthcare and Reality Industry are wide range of image classification application. Artificial Neural Network's computational model is inspired by human brain that deals with both classification and prediction problems. Deep Learning is one of the major sub-domains of machine learning technique based on artificial neural networks. ANN is an artificial human nervous system, which receives the input data, process and transmits the information at the end of computation. It also deals with the segregation of images based on visual content as it includes Input layer, Hidden layer and Output layer. The input received will be sent to hidden layer for processing. Initially all the input nodes are set with the values and step by step the activation values will be calculated for each of the hidden layer and it is called as Feed-Forward method. Back propagation is to adjust the weights systematically to produce values as close as possible to the expected values from the training data. Deep learning can then be knows as neural networks with a large number of arguments and layers in one of four fundamental network architectures namely Unsupervised Pre-trained Networks, Recursive Neural Networks (RNN), Generative Adversarial Networks (GAN) and Convolutional Neural Networks (CNN). The pre-trained Network uses some kind of supervised learning to extract features that can then by useful for classification. It handles more than one hidden layer contradiction to simple neural network. Recursive Neural Network use memory that captures information about the calculation earlier. RNN is recurrent because they perform the same task for every element of a sequence and that depends on previous computation. Generative Adversarial Networks generates a new set of images with an existing image. The generated image looks at least

superficially authentic to human observers, having very realistic characteristics of existing images. The discriminator decides whether the given image is real or fake. Convolutional Neural Network (CNN) derived from Neural Network focus particularly on hidden or convolution layer. It is not just for performing a normal function but also as an activation function. Convolution handles two signals one is of input signal or raw image and the other is to filter signal or recognizes the defined features in images. Thus, it produces an output signal classifying the images into two categories. This model works by following feed forward mechanism. The four major layers of this CNN Networks are Convolution, Pooling, Flatten and Full connection.

II. LOADING DATASET

Os stands for Operating system Python's standard utility module, which provides a portable way of using operating system dependent functionality. Cv2 package is nothing but OpenCV-Python binds library function designed to solve computer vision problems. Numpy library supports for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on array of images as shown in Fig. 1.

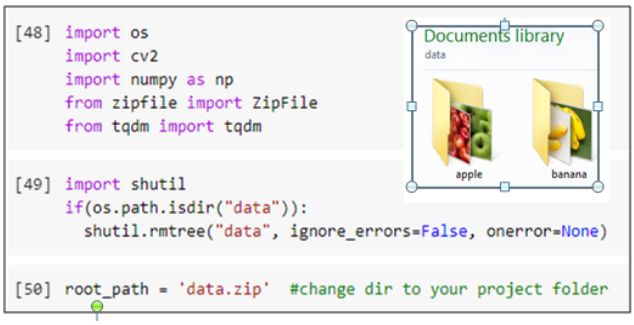

Figure 1. Compressed images – 'data.zip'

Zipfile submodule is imported to uncompress the feature images and labels. Display the image size with the name of the folder. The two folders including *apple* and *banana* to be classified are now compressed as '*data.zip*' as, they occupies huge space of memory if uncompressed as shown in Fig 2 and 3.

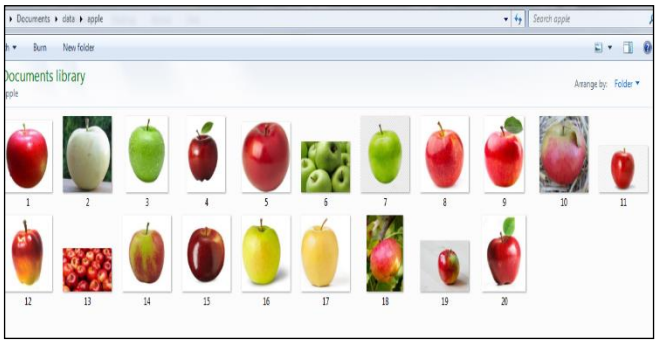

Figure 2. Uncompressed Apple dataset

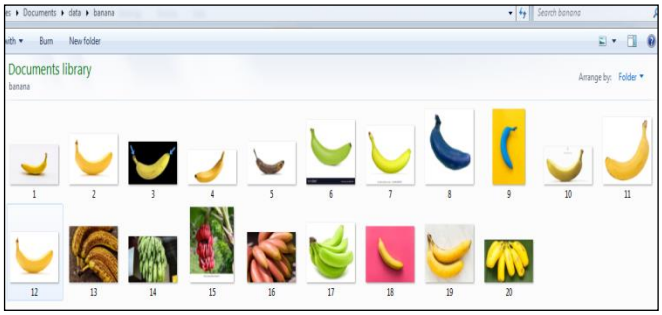

Figure 3. Uncompressed Banana Dataset

III. UNCOMPRESS & SHUFFLE

As classification commences, the shuffled training images are to be sent into input layer. Open the directory and select the path where project related images are located. cv2. imread() method can be used to load an image from the specified file. If the image cannot be read due to missing file, improper permissions, unsupported or invalid format then this method returns an empty matrix as output. The listdir() from *os* module displays the folder names namely banana and apple as shown in Fig 4.

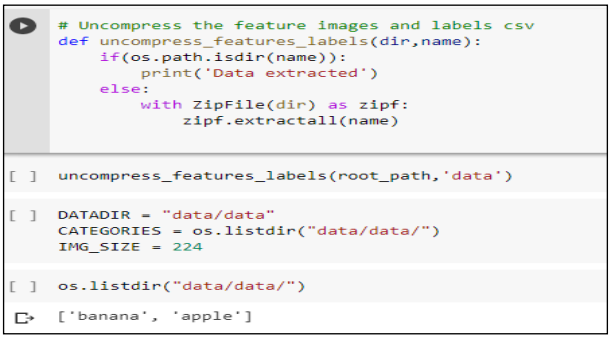

Figure 4. Uncompressed images

tqdm is derived from the Arabic word "*taqadum*" which means "*progress*". A progress bar library supports for nested loops in Jupyter or IPython notebooks. Define training model by setting the path name, category name, resize, and grayscale all the images and before appending them to the list. Now, the training model gets loaded and it is visualised in progress bar as shown in Fig 5.

```
training_data = []

def create_training_data():
    for category in CATEGORIES:

        path = os.path.join(DATADIR,category) # data/data/aeroplane
        class_num = CATEGORIES.index(category) # 0

        for img in tqdm(os.listdir(path)):
            try:
                img_array = cv2.imread(os.path.join(path,img) ,cv2.COLOR_BGR2RGB)
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE), interpola
                training_data.append([new_array, class_num])
            except Exception as e:
                pass

create_training_data()
```

```
100%|██████████| 20/20 [00:00<00:00, 591.58it/s]
100%|██████████| 20/20 [00:00<00:00, 732.83it/s]
```

Figure 5. Progress bar to show dataset loading to a training model

Display the vector dimensional shape of each input image into 224 X 224 X 3 form as shown in Fig 6.

```
print(training_data[0][0].shape)
print(training_data[0][1])
```

```
(224, 224, 3)
0
```

Figure 6. Vector size of dataset

To improve the selection of images for sampling they are randomly shuffled using random packages as shown in Fig 7. Create two list namely - '*images*' and '*labels*' to append them with feature and label while on training data. Input the vector spaced image and import train_test_split module from sklearn model.

```
[11] import random
     random.shuffle(training_data)

     for sample in training_data[:10]:
         print(sample[1])
```

```
0
0
1
0
1
1
1
0
1
1
```

Figure 7. Shuffling

The input image(s) divided into two categories as training and test data for classifying and validation purpose as shown in Fig 8.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

[ ] print(X_train.shape)
    print(X_test.shape)
    print(y_train.shape)
    print(y_test.shape)

    (32, 224, 224, 3)
    (8, 224, 224, 3)
    (32,)
    (8,)

[ ] print(y_train)
    print(y_test)

    [1 0 1 0 0 0 1 1 0 0 1 0 1 1 0 0 0 1 0 1 0 0 1 1 0 1 0 0 0 0 1 0]
    [1 1 0 1 1 1 1 1]
```

Figure 8. Training and Test data.

IV. CNN WITHOUT AUGMENTATION

Import keras, convolutional Neural Network, Dense, Dropout, Flatten, Activation and maxpooling sub modules from tensorflow package for performing classification as shown in Fig 9.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Dense, Dropout, Flatten, Activation, MaxPooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ReduceLROnPlateau
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Figure 9. Importing various modules for Classification

The convolution layer in CNN is same as hidden layer generates a function to extract features from the input images. It is like recognizing certain stroke or pattern from the top to middle and from middle to border of the image in a matrix format. Unlike NN, where the input is a vector, here in CNN the input is a multi-channelled image with three channels. Pooling layer is the next building block of a CNN which reduce the spatial size of the representation. It is done to reduce the amount of parameters and computation in the network. The first fully connected layer takes the inputs from the feature analysis and apply weigh on each spatial size to predict the exact label. The fully connected output layer gives the final probabilities for

each label. Call the sequential function and add CNN network function with filter size set to 32, kernel_size to 3,3, striding to 1,1, set input_shape to a size of 224 X 224 X 3. Set activation function to relu and set pooling to maxpooling as shown in Fig 10.

```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3,3), strides=(1,1), padding='same', input_shape=(IMG_SIZE, IMG_SI
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

model.add(Conv2D(filters=64, kernel_size=(3,3), strides=(1,1), padding='valid'))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

model.add(Flatten())

model.add(Dense(256))
model.add(Activation("relu"))

model.add(Dense(84))
model.add(Activation("relu"))

model.add(Dense(2, activation='softmax'))
```

```
[ ] model.compile(optimizer="adam", loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Figure 10. Defining CNN model

Now fit the model with X_ and y_ train data and undergo validation with X_ and y_ test data with 20 epochs as shown in Fig 11.

```
[ ] model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test))

Train on 32 samples, validate on 8 samples
Epoch 1/20
32/32 [==============================] - 0s 6ms/sample - loss: 0.6860 - acc: 0.5938 - val_loss: 17.9427 - val_acc: 0.1250
Epoch 2/20
32/32 [==============================] - 0s 1ms/sample - loss: 7.5921 - acc: 0.5938 - val_loss: 1.3570 - val_acc: 0.8750
Epoch 3/20
32/32 [==============================] - 0s 1ms/sample - loss: 7.6464 - acc: 0.4375 - val_loss: 1.1946 - val_acc: 0.5000
Epoch 4/20
32/32 [==============================] - 0s 1ms/sample - loss: 0.4986 - acc: 0.7500 - val_loss: 2.7572 - val_acc: 0.1250
Epoch 5/20
32/32 [==============================] - 0s 1ms/sample - loss: 1.0717 - acc: 0.6250 - val_loss: 0.8387 - val_acc: 0.6250
Epoch 6/20
32/32 [==============================] - 0s 1ms/sample - loss: 0.2906 - acc: 0.8750 - val_loss: 0.6709 - val_acc: 0.7500
Epoch 7/20
32/32 [==============================] - 0s 1ms/sample - loss: 0.5372 - acc: 0.6250 - val_loss: 0.7737 - val_acc: 0.7500
Epoch 8/20
32/32 [==============================] - 0s 1ms/sample - loss: 0.2919 - acc: 0.8750 - val_loss: 1.1880 - val_acc: 0.5000
Epoch 9/20
32/32 [==============================] - 0s 1ms/sample - loss: 0.2006 - acc: 0.9062 - val_loss: 1.3446 - val_acc: 0.5000
Epoch 10/20
32/32 [==============================] - 0s 1ms/sample - loss: 0.2368 - acc: 0.9062 - val_loss: 1.0100 - val_acc: 0.5000
Epoch 11/20
```

Figure 11. Fit model for classification

Next, add flatten to CNN layer and set loss function to sparse_categorical_crossentropy. Because categorical _crossentropy uses a one-hot array to calculate the probability, but sparse_categorical _crossentropy (scce) uses a category index. As the proposed paper focus on categorical data, flatten layer set with this parameter. Now generate the classification by calling fit function with epochs to 20 with X_train and y_train data along with validation data as X_test and y_test. To display the classification report and predicted result import classification_report and confusion_matrix modules from sklearn.metrics module as shown in Fig 12.

```
[ ] from sklearn.metrics import classification_report, confusion_matrix

    print('Confusion Matrix')
    print(confusion_matrix(y_test, predict_test))
    print('Classification Report')
    print(classification_report(y_test, predict_test, target_names=CLASS_NAMES))

Confusion Matrix
[[0 1]
 [1 6]]
Classification Report
              precision    recall  f1-score   support

      banana       0.00      0.00      0.00         1
       apple       0.86      0.86      0.86         7

    accuracy                           0.75         8
   macro avg       0.43      0.43      0.43         8
weighted avg       0.75      0.75      0.75         8
```

Figure 12. Classification Report-I

For example, if the confusion matrix results with 0 in any one of the diagonal, then it shows that the input images are fitted/classified to binary labels exactly. Consider the below resultant confusion matrix with:

happy	*sad*
22	0
0	28

22 images classified into label as *happy* and 28 images classified into another label as *sad* indicates all the 50 images are fitted exactly into categories without any misclassification. Thus, the confusion matrix acts as a catalyst or qualifying metric for classification and prediction problem.

```
figure = plt.figure(figsize=(6, 6))
sns.heatmap(con_mat_df, annot=True,cmap=plt.cm.Reds)
plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

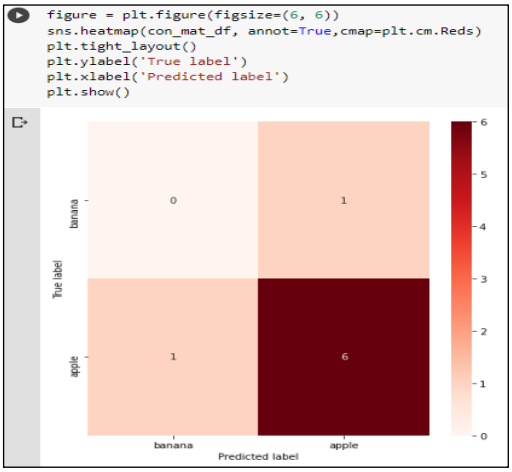

Figure 13. Confusion Matrix-I

Fig 13 shows the visualisation of classification with actual and predicted label in matrix format with 75% of accuracy using CNN model.

Thus the final set of classified images are then exported under the name of fruits.tf in 5 X 5 batch size as shown in Fig 14.

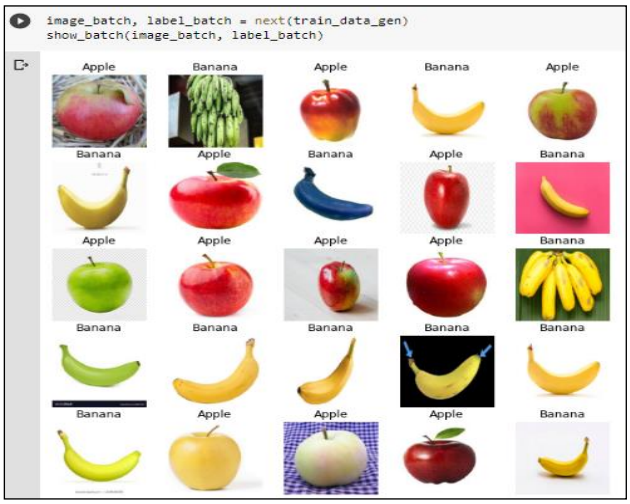

Figure 14. fruits.tf

V. CNN WITH AUGMENTATION

Data Augmentation will generate every single image by rotating at 30 degree each with zoom range set to 0.20. It also flips and rotates the image and generates many copies of image to add it to the training model for improving classification accuracy. Now, set the epoch again to ten and train the classification model to reduce the loss function and gain its accuracy as shown in Fig 15.

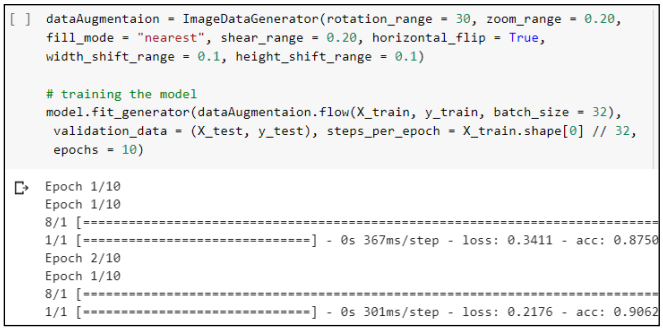

Figure 15. Data Augmentation

The classification report visualises the predicted result and gets displayed in matrix format as shown in Fig 16.

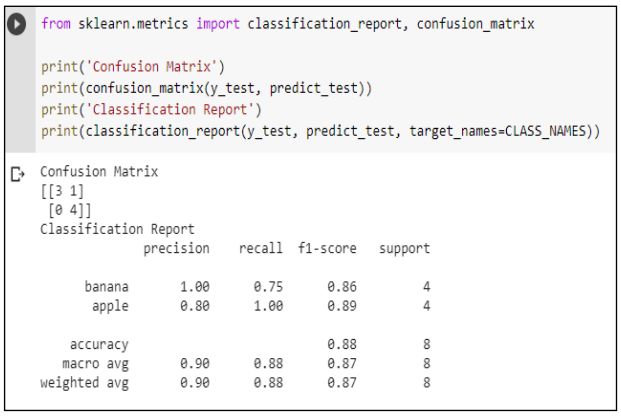

Figure 16. Classification Report-II

Fig 17 shows the resultant matrix after data augmentation and classification. The same is then visualised using Matplotlib module as shown in Fig 18.

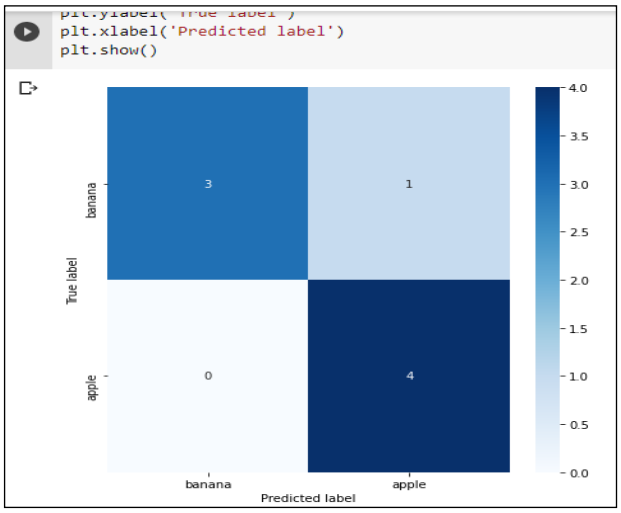

Figure 17. Confusion Matrix-II

Thus the final set of classified images after augmentation are then exported under the name of fruits1.tf in 5X5 batch size as shown in Figure 18.

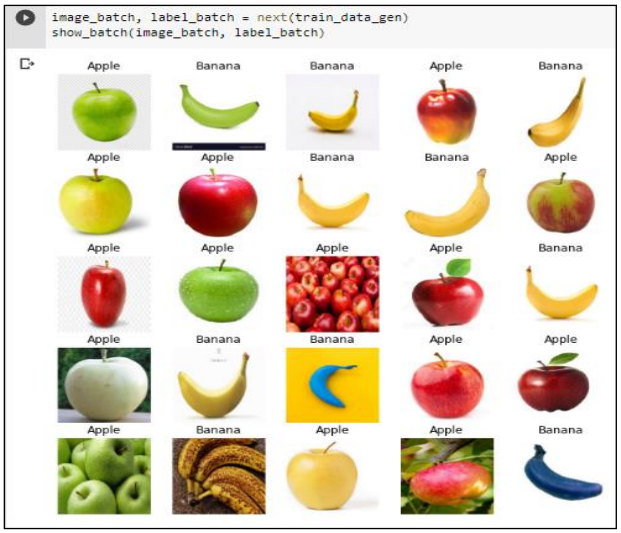

Figure 18. fruits1.tf

VI. CONCLUSION

From Fig 12 and 16 it is clear that the data augmentation reduces the error loss function and improves the performance of classification. In addition, as proposed in this paper, the accuracy has been improved from 75% to 88% due to the inclusion of augmentation technique. In future, this paper can be enhanced by concentrating more on various activation functions like relu, tanh, leaky relu with sigmoid and softmax functions. And it is also necessary to include more number of images to under categorical classification.

VII. REFERENCES

[1]. A. Krizhevsky, I. Sutskever, and G. E. Hinton.(2012). ImageNet classification with deep convolutional neural networks in Proc. Int. Conf. Neural Inf. Process. Syst., pp. 1097_1105.

[2]. Dong, C., Loy, C. C., He, K., & Tang, X. (2016). Image super-resolution using deep Convolutional networks. IEEE transactions on pattern analysis and machine intelligence, 38(2), 295-307.

[3]. G. E. Hinton and R. R. Salakhutdinov.(2006). Reducing the dimensionality of data with neural networks,' Science, vol. 313, no. 5786, pp. 504_507.

[4]. Hasbi Ash Shiddieqy, Farkhad Ihsan Hariadi, Trio Adiono. (2017). Implementation of Deep-Learning based Image Classification on Single Board Computer", International Symposium on Electronics and Smart Devices (ISESD), ISBN 978-1-5386-2779-2, pp. 133-137.

[5]. H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. (2009).Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations,' in Proc. 26th Annu. Int. Conf. Mach. Learn., pp. 609_616.

[6]. J. Jebadurai and J. D. Peter. (2018). Deep CS: Deep Convolutional Neural Network and SVM Based Single Image Super-Resolution, Lecture Notes in Computer Science, vol. 11076, pp. 3-13.

[7]. J. Nagi et al. (2011).Max-pooling convolutional neural networks for vision based hand gesture recognition, Proc. IEEE Int. Conf. Signal Image Process. Appl. (ICSIPA), pp. 342_347.

[8]. K. He, X. Zhang, S. Ren, and J. Sun, (2014).Spatial pyramid pooling in deep convolutional networks for visual recognition, in Proc. Eur. Conf. Comput. Vis., pp. 346–361.

[9]. Laila Ma"rifatul Azizah, Sitti Fadillah Umayah, Slamet Riyadi, Cahya Damarjati, Nafi Ananda Utama . (2017). Deep Learning Implementation using Convolutional Neural Network in Mangosteen Surface Defect Detection, ICCSCE, ISBN 978-1-5386-3898-9, pp. 242-246.

[10]. Min-Chun Yang Yu-Chiang Frank Wang. (2013). A Self-Learning Approach to Single Image Super-Resolution, IEEE Trans. Multimedia, vol. 15, no. 3, pp. 498–508.

[11]. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. (2014). Dropout: A simple way to prevent neural networks from overfitting, J. Mach. Learn. Res., vol. 15, no. 1, pp. 1929_1958.

[12]. Rika Sustika, Asri R. Yuliani, Efendi Zaenudin, Hilman F. Pardede . (2017). On Comparison of

Deep Learning Architectures for Distant Speech Recognition, 2nd International Conferences on Information Technology, Information Systems and Electrical Engineering (ICITISEE), ISBN 978-1-5386-0659-9, pp. 17-21.

[13]. Rui Wang, Wei Li, Runnan Qin and JinZhong Wu. (2017). Blur Image Classification based on Deep Learning, IEEE, ISBN 978-1-5386-1621-5 pp. 1-6.

[14]. Sachchidanand Singh, Nirmala Singh. (2017). Object Classification to Analyze Medical Imaging Data using Deep Learning, International Conference on Innovations in information Embedded and Communication Systems (ICIIECS), ISBN 978-1-5090-3295-2, pp. 1-4.

[15]. Sameer Khan and Suet-Peng Yong, (2017). A Deep Learning Architecture for Classifying Medical Image of Anatomy Object, Annual Summit and Conference, ISBN 978-1-5386-1543-0, pp. 1661-1668.

[16]. Sebastian Stabinger, Antonio Rodr´ıguez-S´anchez. (2017). Evaluation of Deep Learning on an Abstract Image Classification Dataset, IEEE International Conference on Computer Vision Workshops (ICCVW), ISBN 978-1-5386-1035-0, pp. 2767-2772.

[17]. S. Ioffe and C. Szegedy. (2015).Batch normalization: Accelerating deep network training by reducing internal covariate shift, in Proc. Int. Conf. Mach. Learn., 2015, pp. 448_456.

[18]. Teny Handhayani, Janson Hendryli, Lely Hiryantyo, (2017). Comparison of Shallow and Deep Learning Models for Classification of Lasem Batik Patterns, ICICoS, ISBN 978-1-5386-0904-0, pp. 11-16.

[19]. Ye Tao, Ming Zhang, Mark Parsons. (2017). Deep Learning in Photovoltaic Penetration Classification, IEEE Power & Energy Society General Meeting, ISBN 978-1-5386-2213-1, pp. 1-5.

[20]. W. Ouyang et al. (2014). Deep ID-Net: Multi-stage and deformable deep convolutional neural networks for object detection.Online]. Available: https://arxiv.org/abs/1409.3505.

Cite this article as :

Dr. R. Nithya, "Training the Image Classifier with and without Data Augmentation", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 6 Issue 2, pp. 172-178, March-April 2020. Available at doi : https://doi.org/10.32628/CSEIT206245
Journal URL : http://ijsrcseit.com/CSEIT206245