CS230

A method to improve classifier performance using Generative Adversarial Networks(GANs) based data augmentation on the Kannada MNIST dataset. url:https://github.com/prateik/cs230project

Prateik Harwalker
Stanford University
prateik@stanford.edu

Abstract

In this paper, a data augmentation method using Generative Adversarial Networks(GANs) to improve the performance of a Convolution Neural Network(CNN) for the task of classification of Kannada MNIST digits is presented. This technique replaced 100 images from the original dataset with synthetic generated data. The test accuracy is shown to improve from 52.28% to 70.07 % and the CNN F1 score(macro avg.) has improved from 0.51 to 0.70.

1 Introduction

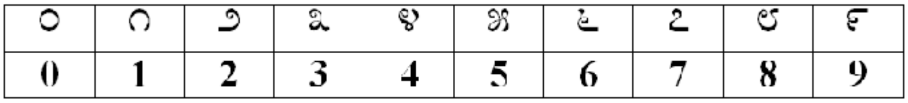
Traditionally, Machine Learning/Deep Learning techniques require a large amount of training data to achieve good results in terms of the test accuracy of the model. It would be interesting to investigate if we could achieve comparable test accuracy utilizing only a subset of the training dataset. This would help in reducing the amount of raw training data that would be required for building models. Therefore, exploring the idea of using data augmentation and GANs for developing synthetic training data is interesting. The input to the algorithm is a gray scale image which is 28 by 28 pixels in dimensions. A Convolutional Neural Network (CNN) is used to output a predicted digit out of 10 classes.

2 Related work

Data augmentation using GANs is a relatively new concept and has been explored to a certain extent in [8] where they have explored generating medical images using GANs and training a ResNet based architecture using this synthetic data. In [9] they have results to prove that adding synthetic GAN based images to liver lesion classification has improved the classification performance. The CNN model for the Kannada MNIST has been studied in detail in this article [12].

3 Dataset and Features

Kannada[1] is a regional language spoken in South India by over 44 million people. The Kannada MNIST dataset has been recently (Aug 2019) published as part of the paper Kannada-MNIST. [2].



೦	೧	೨	೩	೪	೫	೬	೭	೮	೯
0	1	2	3	4	5	6	7	8	9

Figure 1: Kannada digits and the corresponding digits in English.[13]

CS230: Deep Learning, Winter 2019, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

The images are 28X28 pixels (gray-scale) with a split of 60000 train and 10000 test image data. The GAN based architecture was used to generate Kannada MNIST digits and then the dataset was prepared by labelling the correct class for every image that was human readable as shown in Figure 2.

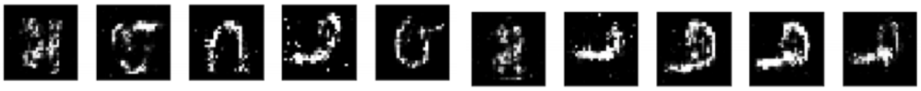


Figure 2: Sample Kannada digits generated using a GAN.[13]

The training data was normalized by multiplying with a factor (1/255) before training the model. Keras based ImageDataGenerator was used for Data Augmentation with the following specifications:

Parameter	Value
rescale	1/255
rotation range	10
width shift range	0.25
height shift range	0.25
shear range	0.1
zoom range	0.25
horizontal flip	False

Table 1: The parameters used for the Keras based Data Augmentation.

4 Methods

Based on the dimensions of the input dataset and the output classes , a Convolutional Neural Network [7] with the architecture defined in Figure 4 was used for the classification task. Since the output has multiple classes in the output layer, the Categorical Cross Entropy Loss function is used as defined below:

$$-\Sigma_{c=1}^{M} y_{i,c} \log(p_{i,c})$$

Figure 3: The categorical cross entropy loss function used as there are 10 classes in the output layer.

where M is the number of classes(10), y i,c is the ith observation of the cth catergory and p i,c is the probability predicted by the model for the ith observation to belong to the cth category.

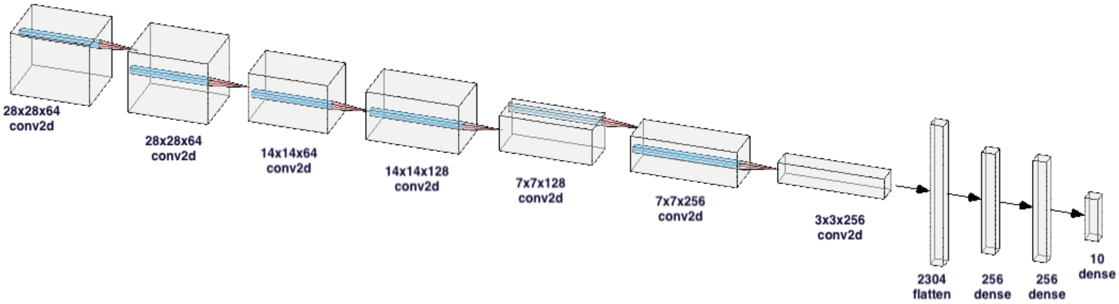


Figure 4: The Architecture used for the CNN and the model specification is defined below.[7]

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 64)        640
_________________________________________________________________
batch_normalization (BatchNo (None, 28, 28, 64)        256
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 28, 28, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 28, 28, 64)        36928
_________________________________________________________________
batch_normalization_1 (Batch (None, 28, 28, 64)        256
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 28, 28, 64)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 64)        0
_________________________________________________________________
```

```
dropout (Dropout)            (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 128)       73856
_________________________________________________________________
batch_normalization_2 (Batch (None, 14, 14, 128)       512
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 128)       147584
_________________________________________________________________
batch_normalization_3 (Batch (None, 14, 14, 128)       512
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 14, 14, 128)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 7, 7, 256)         295168
_________________________________________________________________
batch_normalization_4 (Batch (None, 7, 7, 256)         1024
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 7, 7, 256)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 7, 7, 256)         590080
_________________________________________________________________
batch_normalization_5 (Batch (None, 7, 7, 256)         1024
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 7, 7, 256)         0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 256)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 3, 3, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 2304)              0
_________________________________________________________________
dense (Dense)                (None, 256)               590080
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
batch_normalization_6 (Batch (None, 256)               1024
_________________________________________________________________
dense_1 (Dense)              (None, 10)                2570
=================================================================
```

For the Generative Adversarial Network [4], the Generator and Discriminator are defined as Fully Connected Neural Networks as defined in Figure 5.

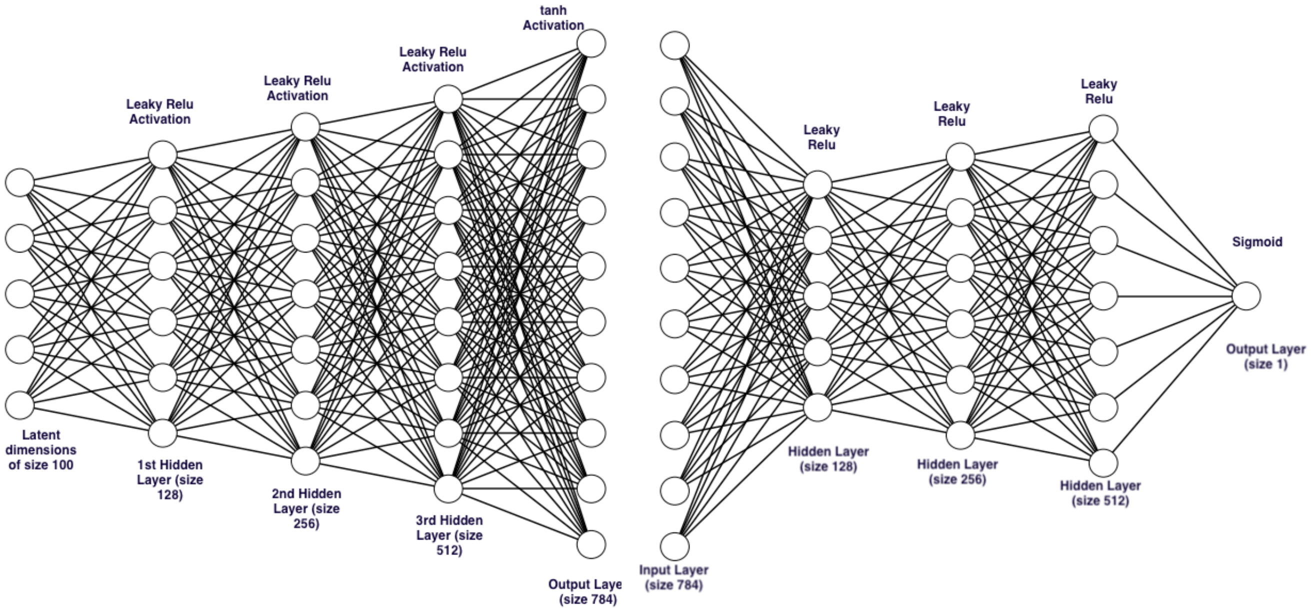


Figure 5: The Neural Network on the left in the Generator architecture which used Leaky Relu activations for the hidden layers and the Tanh activation for the output layer. The Discriminator (NN on the right) uses the Leaky Relu activations for the hidden layers and the Sigmoid for the output layer.

5 Hyperparameter Tuning

1. Two different learning rates 0.002 and 0.02 were tested and the former displayed superior performance metrics on the test dataset.

2. Several batch sizes were experimented with as shown in the table below.

Batch Size	Test Accuracy(%)	Test Accuracy(%with GAN)
64	52.28	70.07
512	66.15	72.8
1024	75.37	68.89
2048	78.89	68.07

Table 2: The test accuracy of the CNN model before and after including synthetic data in the training for various batch sizes

6 Experiments/Results/Discussion

The CNN based classifier trained with the parameters listed in Table 3.

Hyperparameter	Value
learning rate	0.002
mini-batch size	64
epochs	50
Optimizer	RMSProp

Table 3: The hyperparameters used for the training.

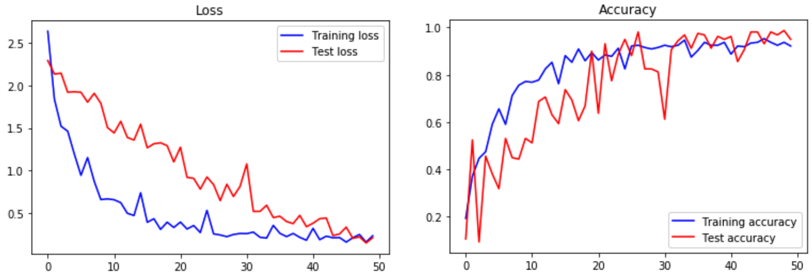


Figure 6: X axis is the number of epochs and Y axis is the value. The left image is the comparison of the CNN training and testing loss and the right image represents the accuracy of the CNN. Both of them are for the real training data only.

One thing to note is that the generated images have significantly higher noise as compared to the original dataset. A batch of 100 samples in the training data was replaced with the GAN based synthetic data. One observation of interest was the fact that the introduction of noisy images resulted in many small value pixels which would have been zeros in the original training data and thus the sparsity of the input images was reduced and more number of non-zero values were present.

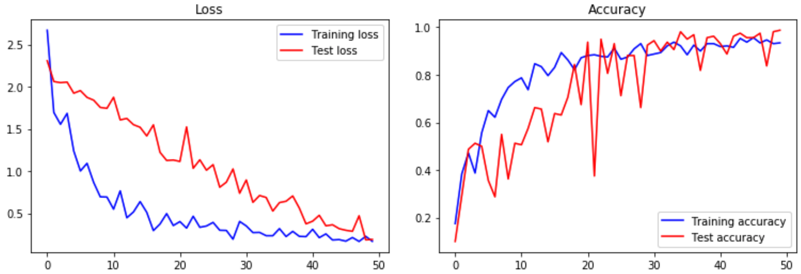


Figure 7: X axis is the number of epochs and Y axis is the value . The left image is the comparison of the CNN training and testing loss and the right image represents the accuracy of the CNN. Both of them are for the real and synthetic data combined.

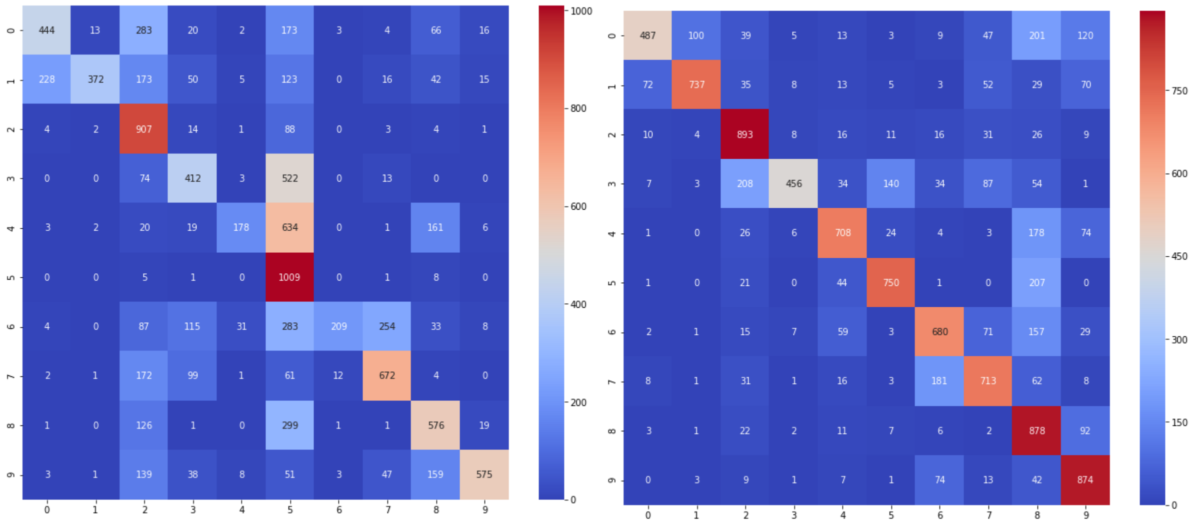
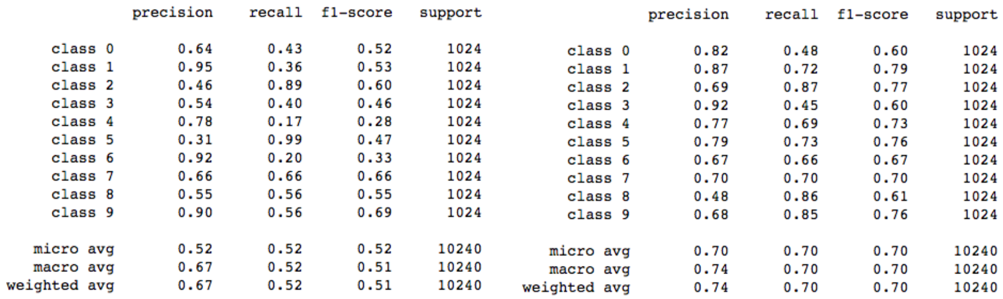


Figure 8: The left image is the confusion matrix of the CNN trained with real training data only. The image on the right is the confusion matrix of the CNN trained with a mixture of real and synthetic data.



```
               precision    recall  f1-score   support

     class 0       0.64      0.43      0.52      1024
     class 1       0.95      0.36      0.53      1024
     class 2       0.46      0.89      0.60      1024
     class 3       0.54      0.40      0.46      1024
     class 4       0.78      0.17      0.28      1024
     class 5       0.31      0.99      0.47      1024
     class 6       0.92      0.20      0.33      1024
     class 7       0.66      0.66      0.66      1024
     class 8       0.55      0.56      0.55      1024
     class 9       0.90      0.56      0.69      1024

   micro avg       0.52      0.52      0.52     10240
   macro avg       0.67      0.52      0.51     10240
weighted avg       0.67      0.52      0.51     10240
```

```
               precision    recall  f1-score   support

     class 0       0.82      0.48      0.60      1024
     class 1       0.87      0.72      0.79      1024
     class 2       0.69      0.87      0.77      1024
     class 3       0.92      0.45      0.60      1024
     class 4       0.77      0.69      0.73      1024
     class 5       0.79      0.73      0.76      1024
     class 6       0.67      0.66      0.67      1024
     class 7       0.70      0.70      0.70      1024
     class 8       0.48      0.86      0.61      1024
     class 9       0.68      0.85      0.76      1024

   micro avg       0.70      0.70      0.70     10240
   macro avg       0.74      0.70      0.70     10240
weighted avg       0.74      0.70      0.70     10240
```

Figure 9: The left image is the performance metrics of the CNN trained with real training data only. The image on the right is the performance metrics of the CNN trained with a mixture of real and synthetic data.

7 Conclusion/Future Work

To summarize, it is possible to use GANs for data augmentation, but the generated images would need to be post processed to reduce the noise in the images to make them usable. The CNN classfier test accuracy improved by 17.79 %upon introducing GAN based synthetic data into the training dataset. It would be interesting to explore the idea of generating a larger GAN based synthetic dataset and performing comparitive analysis.

8 Contributions

The project was taken up as an individual.

References

[1] Web Reference:https://en.wikipedia.org/wiki/Kannada

[2] Kannada-MNIST: A new handwritten digits dataset for the Kannada language, Prabhu, Vinay Uday, arXiv preprint arXiv:1908.01242, 2019

[3] Website:https://medium.com/@ayman.shams07/data-augmentation-tasks-using-keras-for-image-data-and-how-to-use-it-in-deep-learning-d4dd24e8ca19

[4] Web Reference:https://machinelearningmastery.com/how-to-develop-a-generative-adversarial-network-for-an-mnist-handwritten-digits-from-scratch-in-keras/

[5] Web Reference:https://www.kaggle.com/raoulma/mnist-image-class-tensorflow-cnn-99-51-test-acc

[6] Web Reference:https://github.com/prateik/cs230project

[7] Web Reference: https://www.kaggle.com/bustam/cnn-in-keras-for-kannada-digits

[8] *GAN Augmentation: Augmenting Training Data using Generative Adversarial Networks*, Christopher Bowles, Liang Chen, Ricardo Guerrero, Paul Bentley, Roger Gunn, Alexander Hammers, David Alexander Dickie, Maria Valdés Hernández, Joanna Wardlaw, Daniel Rueckert, arXiv:1810.10863 [cs.CV], 2018

[9] *SYNTHETIC DATA AUGMENTATION USING GAN FOR IMPROVED LIVER LESION CLASSIFICATION*, Maayan Frid-Adar1, Eyal Klang,Michal Amitai,Jacob Goldberger, Hayit Greenspan1, arXiv:1801.02385v1 [cs.CV], 2018

[10] Web Reference: https://www.omniglot.com/writing/kannada.htm

[11] Web Reference: https://github.com/keras-team/keras/issues/6444

[12] Web Reference: https://towardsdatascience.com/a-new-handwritten-digits-dataset-in-ml-town-kannada-mnist-69df0f2d1456

[13] Web Reference: https://www.researchgate.net/figure/Kannada-digits-their-equivalent-english-digits$_{f}ig1_{2}$74248237