CInC Flow: Characterizable Invertible 3×3 Convolution

Sandeep Nagar[1] **Marius Dufraisse**[2] **Girish Varma**[1]

[1]Machine Learning Lab, International Institute of Information Technology, Hyderabad, India
[2]Computer Science Dept., École Normale Supérieure (ENS), Paris-Saclay, France

Abstract

Normalizing flows are an essential alternative to GANs for generative modelling, which can be optimized directly on the maximum likelihood of the dataset. They also allow computation of the exact latent vector corresponding to an image since they are composed of invertible transformations. However, the requirement of invertibility of the transformation prevents standard and expressive neural network models such as CNNs from being directly used. Emergent convolutions were proposed to construct an invertible 3×3 CNN layer using a pair of masked CNN layers, making them inefficient. We study conditions such that 3×3 CNNs are invertible, allowing them to construct expressive normalizing flows. We derive necessary and sufficient conditions on a padded CNN for it to be invertible. Our conditions for invertibility are simple, can easily be maintained during the training process. Since we require only a single CNN layer for every effective invertible CNN layer, our approach is more efficient than emerging convolutions. We also proposed a new coupling layer for more flexibility and expressiveness, Quad-coupling. We benchmark our approach and show similar performance results to emergent convolutions while improving the model's efficiency. Code available on GitHub[a].

[a]`https://github.com/Naagar/Normalizing_Flow_3x3_inv`

1 INTRODUCTION

The availability of large datasets has resulted in improved machine learning solutions for more complex problems. However, supervised datasets are expensive to create. Hence unsupervised methods like generative models are increasingly worked on. Generative models that have been proposed can be broadly categorized under Likelihood-based methods and Generative Adversarial Methods. For example, an optimization algorithm could directly minimize the former's negative log-likelihood of the unsupervised examples. At the same time, in the latter, the loss function itself is modelled as a discriminator network that is trained alternatively. Hence likelihood-based methods directly optimize the probability of examples. In contrast, in GANs, the function being optimized is implicit and hard to reason about.

An essential type of Likelihood-based Generative models is normalizing flow-based models. Normalizing flow-based models transform a latent vector usually sampled from a continuous distribution like the Gaussian by a sequence of invertible functions to produce the sample. Hence even though the latent vector distribution is simple, the sample distribution could be highly complex, provided we are using an expressive set of invertible transformations. Also, the invertibility of the model implies that one can find the exact latent vector corresponding to an example from a dataset. All other approaches to generative modelling can compute the latent vector, for example, only approximately.

The ability of a normalizing flow based model to express complex real-world data distributions depends on the expressive power of the invertible transformations used. In supervised models in vision tasks, complex, multilayered CNNs with different window sizes are used. CNN's with larger window size helps in spatial mixing of information about the images, resulting in expressive features. Glow used invertible 1×1 convolutions to build normalizing flow models Kingma and Dhariwal [2018]. For a 1×1 convolution (if it is invertible), the inverse is also a 1×1 convolution. We show that this approach does not generalize to larger window sizes. In particular, the inverse of an invertible 3×3 convolution necessarily depends on all the feature vector dimensions, unlike CNNs, which only require local features.

Emerging convolutions proposed a way of inverting convolutions with large window sizes Hoogeboom et al. [2019]. The inverse is not a convolution and is computed by a linear equation system that can efficiently be solved using back substitution. However, for obtaining an invertible convolution, they required 2 CNN filters to be applied. Hence for every effective invertible convolution, they are required to do two convolutions back to back. We propose a simple approach using padding of obtaining invertible convolutions, which only uses a single convolutional filter. Furthermore, we are able to give a characterization (necessary and sufficient conditions) for the convolutions to be invertible. This allows us to optimize over the space of invertible convolutions during training directly. We have compared our method with Emerging convolution (Hoogeboom et al. [2019]) and Autoregressive convolution (Germain et al. [2015]) in Appendix-C

Main Contributions.

- We give necessary and sufficient conditions for a 3×3 convolution to be invertible by making some modifications to the padding (see Section 3.1).
- We also propose a more expressive coupling mechanism called Quad-coupling (see Section 3.2).
- We use our characterization and Quad-coupling to train flow-based models that give samples of similar quality as previous works while improving upon the run-time compared to the other invertible 3×3 convolutions proposed (see Section 4).

2 RELATED WORKS

Normalizing flows. A normalizing flow aims to model an unknown data distribution (Kingma and Dhariwal [2018], Durkan et al. [2019a,b]), that is, to be able to sample from this distribution and estimate the likelihood of an element for this distribution.

To model the probability density of a random variable x, a normalizing flow apply an invertible change of variable $x = g_\theta(z)$ where z is a random variable following a known distribution for instance $z \sim \mathcal{N}(0, I_d)$. Then we can get the probability of x by applying the change of variable formula

$$p_\theta(x) = p(f_\theta(x)) \left(\left| \frac{\partial f_\theta(x)}{\partial x^T} \right| \right)$$

where f_θ denotes the inverse of g_θ and $\left| \frac{\partial f_\theta(x)}{\partial x^T} \right|$ its Jacobian.

The parameters θ are learned by maximizing the actual likelihood of the dataset. At the same time, the model is designed so that the function g_θ can be inverted and have its Jacobian computed in a reasonable amount of time.

Glow. RealNVP defines a normalizing flow composed of a succession of invertible steps (Dinh et al. [2017]). Each of these steps can be decomposed into layers steps. Improvements for some of these layers where proposed in later articles (Kingma and Dhariwal [2018], Hoogeboom et al. [2019]).

Actnorm: The actnorm layer performs an affine transformation similar to batch normalization. First, its weights are initialized so that the output of the actnorm layer has zero mean and unit variance. Then its parameters are learned without any constraint.

Permutation: RealNVP proposed to use a fixed permutation to shuffle the channels as the coupling layer only acts on half of the channels. Later, Kingma and Dhariwal [2018] replaced this permutation with a 1×1 convolution in Glow. These can easily be inverted by inverting the kernel. Finally, Hoogeboom et al. [2019] replaced this 1x1 convolution with the so-called emerging convolution. These have the same receptive convolution with a kernel of arbitrary size. However, they are computed by convolving with two successive convolutions whose kernel is masked to help the inversion operation.

Coupling layer: The coupling layer is used to provide flexibility to the architecture. The Feistel scheme (Hoang and Rogaway [2010]) inspires its design. They are used to build an invertible layer out of any given function f. Here f is learn as a convolutional neural network.

$$y = [y_1, y_2], \quad y_1 = x1, \quad y_2 = (x_2 + f(x_1)) * \exp(g(x_1))$$

Where we get x_1 and x_2 by splitting the input x along the channel axis.

Invertible Convolutional Networks. Complementary to normalizing flows, there has been some work done designing more flexible invertible networks. For example, Gomez et al. [2017] proposed reversible residual networks (RevNet) to limit the memory overhead of backpropagation, while (Jacobsen et al. [2019]) built modifications to allow an explicit form of the inverse, also studying the ill-conditioning of the local inverse. Ho et al. [2019] proposed a flow-based model that is the non-autoregressive model for unconditional density estimation on standard image benchmarks

Invertible 1×1 Convolution: Kingma and Dhariwal [2018] proposed the invertible 1×1 convolution replacing fixed permutation (Dinh et al. [2017]) that reverses the ordering of the channels. Hoogeboom et al. [2019] proposed normalizing flow method to do the inversion of 1×1 convolution with doing some padding on the kernel and two distinct autoregressive convolutions, which also provide a stable and flexible parameterization for invertible 1×1 convolutions.

Invertible n×n Convolution: Reformulating n×n convolution using invertible shift function proposed by Truong et al. [2019] to decrease the number of parameters and remove the additional computational cost while keeping the range of the receptive fields. In our proposed method, there is no need for the reformulation of standard convolutions. Hoogeboom et al. [2019] proposed two different methods to produce the invertible convolutions : (1) Emerging Convolution and (2) Invertible Periodic Convolutions. Emerging requires two autoregressive convolutions to do a standard convolution, but our method requires only one convolution as compare to the method proposed by Hoogeboom et al. [2019] and increase the flexibility of the invertible n×n convolution.

3 OUR APPROACH

We propose a novel approach for constructing invertible 3×3 convolutions and coupling layers for normalizing flows. We propose two modifications to the existing layers used in previous normalizing flow models:

- convolution layer: instead of using 1×1 convolutions or emerging convolutions, we propose to use standard convolutions with a kernel of any size with a specific padding.
- coupling layer: we propose to use a modified version of the coupling layer designed to have a bigger receptive field.

We also show how invertibility can be used to manipulate images semantically.

3.1 INVERTIBLE 3×3 CONVOLUTION

We give necessary and sufficient conditions for an arbitrary convolution with some simple modifications on the padding to be invertible. Moreover, the inverse can also be computed by an efficient back substitution algorithm.

Definition 1 (Convolution)**.** *The convolution of an input X with shape $H \times W \times C$ with a kernel K with shape $k \times k \times C \times C$ is $Y = X * K$ of shape $(H - k + 1) \times (W - k + 1) \times C$ which is equal to*

$$Y_{i,j,c_o} = \sum_{l,h<k} \sum_{c_i=1}^{C} I_{i+l,j+k,c_i} K_{l,k,c_i,c_o} \tag{1}$$

.

In this setting, the output Y has a smaller size than the input to prevent this input is padded before applying the convolution.

Definition 2 (Padding)**.** *Given an image I with shape $H \times W \times C$, the (t, b, l, r) padding of I is the image $\hat{I}$ of shape $(H + t + b) \times (W + l + r) \times C$ defined as*

$$\hat{I}_{i,j,c} = \begin{cases} I_{i-t,j-l,c} & \textit{if } i - t < H \textit{ and } j - l < W \\ 0 & \textit{otherwise} \end{cases} \tag{2}$$

As zero padding does not add any bias to the input, the convolution between a padded input $\hat{I}$ and a kernel K is still a linear map between the input and the output. As such, it can be described as matrix multiplication.

An image I of shape (H, W, C) can be seen as an vector $\vec{I}$ of $\mathbb{R}^{H \times W \times C}$. In the rest of this paper we will always use the following basis $I_{i,j,c} = \vec{I}_{c+Cj+CHi}$. For any index $i \leq HWC$, let (i_y, i_x, i_c) denote the indexes that satisfy $\vec{I}_i = I_{i_y,i_x,i_c}$. Note that $i < j$ iff $(i_y, i_x, i_c) \prec (j_y, j_x, j_c)$ where $\prec$ denotes the lexicographical order over $\mathbb{R}^3$. If $C = 1$ this means that the pixel (j_y, j_x) is on the right or below the pixel (i_y, i_x).

Definition 3 (Matrix of a convolution.)**.** *Let K be a kernel of shape $k \times k \times C \times C$. The matrix of a convolution of kernel K with input X of size $H \times W \times C$ with padding (t, b, l, r) is a matrix describing the linear map $X \mapsto \hat{X} * K$.*

Characterization of invertible convolutions: We consider convolution with top and left padding only. For such convolutions, we give necessary and sufficient conditions for it to be invertible. Let K be the kernel of the convolution with shape $3 \times 3 \times N \times N$ where 3×3 is the window size, and N is the number of channels. Note that number of input channels should be equal to the number of output channels for it to be invertible.

Lemma 1. *Let $y = M\hat{x}$,*

M is a lower triangular matrix with all diagonal entries $= K_{3,3}$

Where the matrix M is which produces the equivalent result when multiplied with a vectorized input ($\hat{x}$).

Proof. Consider any entry in the upper right half of M. That is (i, j) such that $i < j$ according to the ordering given in the definition of M. $M_{i,j}$ is nothing but the scalar weight that needs to be multiplied to the jth pixel of input when computing ith pixel of the output. The linear equation relating these two variable is as follows:

$$y_i = \sum_{l=0}^{3} \sum_{k=0}^{3} K_{3-l,3-k} x_{i_x-l,i_y-k}$$

From this equations follows that if $j_x > i_x$ or $j_y > i_y$ then the ith pixel of the output does not depend on the jth pixel of the input and thus $M_{i,j} = 0$. This also justifies that all diagonal coefficients of M are equal to $K_{3,3}$ □

We first describe our conditions for the case when $N = 1$. We prove the following theorem.

Theorem 1 (Characterization for $N = 1$)**.**

M is invertible iff $K_{3,3} \neq 0$.

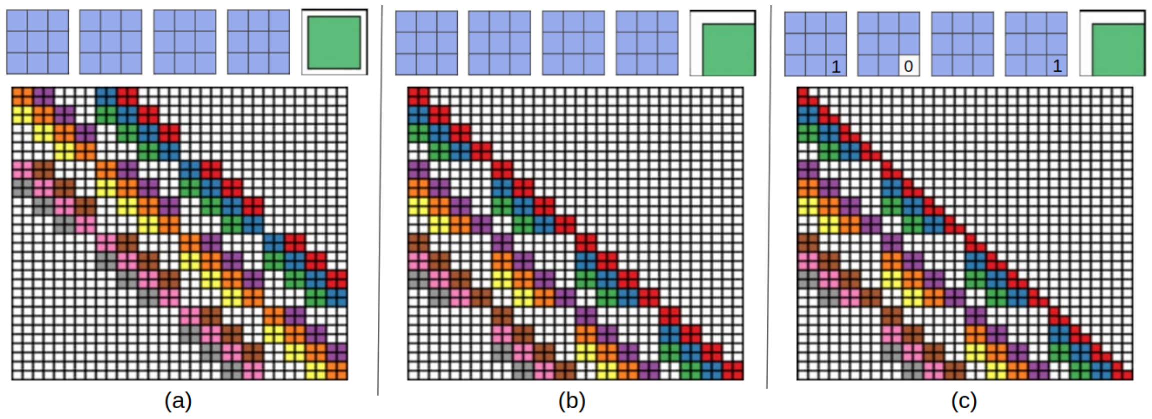


Figure 1: (a).Top: first four is kernel matrix and fifth is input matrix with the *standard* padding that give the bottom convolution matrix, Bottom: the convolution matrix corresponding to a convolution with kernel of size 3 applied to an input of size 4×4 padded on both sides and with 2 channels. Zero coefficients are drawn in white, other coefficient are drawn using the same color if they are applied to the same spatial location albeit on different channels. (b) Top: an *alternative* padding scheme that results in a block triangular matrix M, Bottom: The matrix corresponding to a convolution with kernel of size 3 applied to an input of size 4×4 padded only on one side and with 2 channels. (c) Top: an *masked alternative* padding scheme that results in a triangular matrix M, Bottom: the matrix corresponding to a convolution with kernel of size 3 applied to an input of size 4×4 padded only on one side and with 2 channel. One of the weight of the kernel is masked. Note that the equivalent matrix M is *triangular*.

Proof. The proof of the theorem uses Lemma 1. Since M is lower triangular, the determinant is nothing but the product of diagonal entries, which is $= K_{3,3}^{h*w}$ where h, w is the dimensions of the input/output image. □

At its core, the convolution layer is a linear operation. However, we have no guarantees regarding it as invertibility. The result z of the convolution of input x with kernel k can be expressed as the product as x with a matrix M. When zero-padding is used around the input so that x and z have the same shape, the matrix M is not easily invertible because the determinant of M can be zero (see matrix M in Figure 1(a)).

However, when padding only on two sides (left and top), the corresponding M is blocked triangular (see Figure 1(b)). To further ensure invertibility and speed up the inversion process, we also mask part of the kernel so that the matrix corresponding to the convolution is triangular, see in Figure 1(c) and for more details which $K_(n, n)$ we need to mask see Appendix-B. In this configuration, the Jacobian of the convolution can also be easily computed. For further details of the padding the input, see Appendix-A.

3.2 QUAD-COUPLING

The coupling layer is used to have some flexibility as its functions can be of any form. However, it only combines the effects of half channels. To overcome this issue we designed a new coupling layer inspired from generalized Feistel (Hoang and Rogaway [2010]) schemes. Instead of dividing the input x into two blocks we divide it into four $x = [x_1, x_2, x_3, x_4]$ along the feature axis. Then we keep x_1 unchanged and use it to modify the other blocks in an autoregressive manner (see Figure 2):

$$y_1 = x_1 \tag{3}$$
$$y_2 = (x_2 + f_1(x_1)) * \exp(g_1(x_1)) \tag{4}$$
$$y_3 = (x_3 + f_2(x_1, x_2)) * \exp(g_2(x_1, x_2)) \tag{5}$$
$$y_4 = (x_4 + f_3(x_1, x_2, x_3)) * \exp(g_3(x_1, x_2, x_3)) \tag{6}$$

where $(f_i)_{i \leq 3}$ and $(g_i)_{i \leq 3}$ are learned. The output of the layer is obtained by concatenating the $(y_i)_{i \leq 4}$.

4 EXPERIMENTAL RESULTS

The architecture is based on Hoogeboom et al. [2019]. We modified the emerging convolution layer to use our standard convolution. We also introduced the Quad-coupling layer in place of the affine coupling layer. Finally, we evaluate the model on a variety of models and provide images sampled from the model. For detailed overview of the architecture see Figure 3.

Training setting: To train the model on Cifar10, we used the 3 level (L) and depth (D) of 32 and lr 0.001 for the 500 epochs. To train on ImageNet32, $L = 3$, $D = 48$, lr 0.001 for the 600 epochs and for ImageNet64, $L = 4$, $D = 48$, lr 0.001 for the 1000 epochs. See Figure 3 for the model architecture.

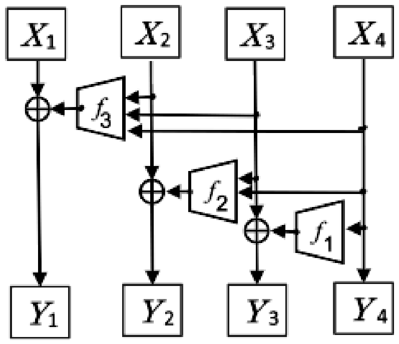


Figure 2: The Quad-coupling layer, each input block X_i has the same spatial dimension as the input X but only one quarter of the channels. Each of the function f_1, f_2 and f_3 is a 3 layer convolutional network. $\bigoplus$ symbolizes a component-wise addition. The multiplicative actions are not represented here.

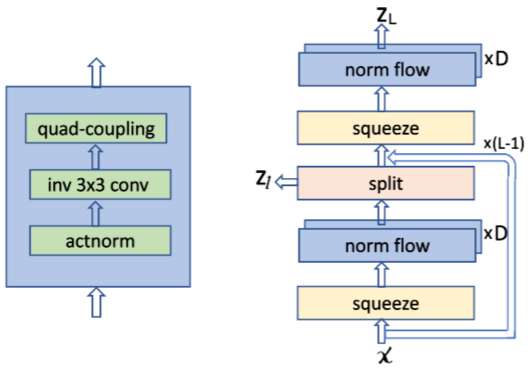


Figure 3: Overview of the model architecture. Left, the flow modules we propose: containing inv 3×3 convolution. The diagram on the right shows the entire model architecture, where the flow module is now grouped. The squeeze module reorders pixels by reducing the spatial dimensions by a half, and increasing the channel depth by four. A hierarchical prior is placed on part of the intermediate representation using the split module as in (Kingma and Dhariwal [2018]). x and z denote input and output. The model has L levels, and D flow modules per level.

Quantitative results: The Comparison of the performance of 3×3 invertible convolution with the emerging convolution (Hoogeboom et al. [2019]) for the cifar10 dataset in Table 1. The performance of our layers was tested on CIFAR10 (Krizhevsky et al. [2009]), ImageNet (Russakovsky et al. [2015]) as well as on the galaxy dataset (Ackermann et al. [2018]) see Table 2. We also tested our architecture on networks with a smaller depth ($D = 4$ or $D = 8$) see Table 3 which could be used when computational resources are limited as their sampling time is much lower. In this case, using standard convolution and Quad-coupling offers a more considerable performance improvement than with bigger models (see Table 3).

	Emerging 3×3 Inv. conv	Our 3×3 Inv. conv
Affine	3.3851	3.4209
Quad	**3.3612**	**3.3879**

Table 1: Comparison of the performance (in bits per dimension) achieved on the Cifar10 dataset with different coupling architectures.

	Glow	Emerging	3×3	Quad
Cifar10	3.35	3.34	**3.3498**	**3.3471**
ImageNet32	4.09	4.09	**4.0140**	4.0377
ImageNet64	3.81	3.81	3.8946	3.8514
Galaxy	—	2.2722	2.2739	**2.2591**

Table 2: Performance achieved on the Cifar10 and Imagenet datasets after a limited number of epochs (500 for Cifar10, 600 for ImageNet32,ImageNet64 and 1000 for Galaxy). Emerging results were obtained by using the code provided in Hoogeboom et al. [2019], 3×3 is replacing the emerging convolutions by our 3×3 invertible convolutions and Quad uses Quad-coupling on top of this.

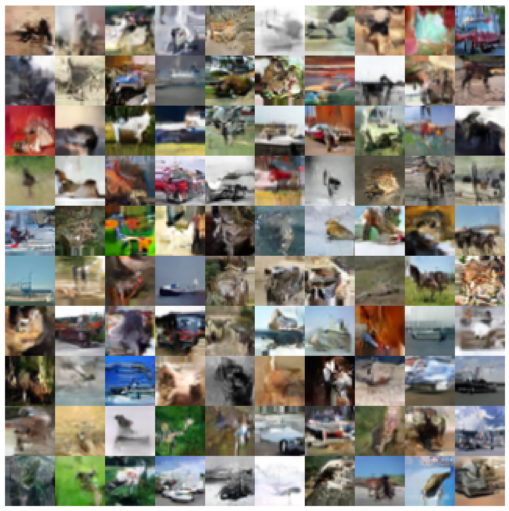


Figure 4: Sample images generated after training on the cifar dataset.

Sampling Times: We compared our method's sampling time (Table 4) against Glow (Kingma and Dhariwal [2018]) and Emerging Convolutions (Hoogeboom et al. [2019]). Our convolution still requires solving a sequential problem to be inverted and, as such, need to be inverted on CPU, unlike Glow that can be inverted while performing all the computation on GPU. This explains the gap between the sampling time of our model compared to Glow. However, it is still roughly two times faster than emerging convolutions; this comes from the need to solve two inversion problems to invert one emerging convolution layer. The Quad-coupling layer does not affect sampling time too much.

Dataset	Emerging		Ours		Depth
	Performance	Sampling time	Performance	Sampling time	
Cifar10	3.52	2.45	**3.49**	**1.31**	4
Imagenet32	4.30		**4.25**		
Cifar10	3.47	4.94	**3.46**	**2.76**	8
Imagenet32	4.20		**4.18**		

Table 3: Performance with smaller networks, when computational resources are limited. The performance is expressed in bits per dimension and the sampling time is the time in seconds needed to sample 100 images. All networks were trained for 600 epochs.

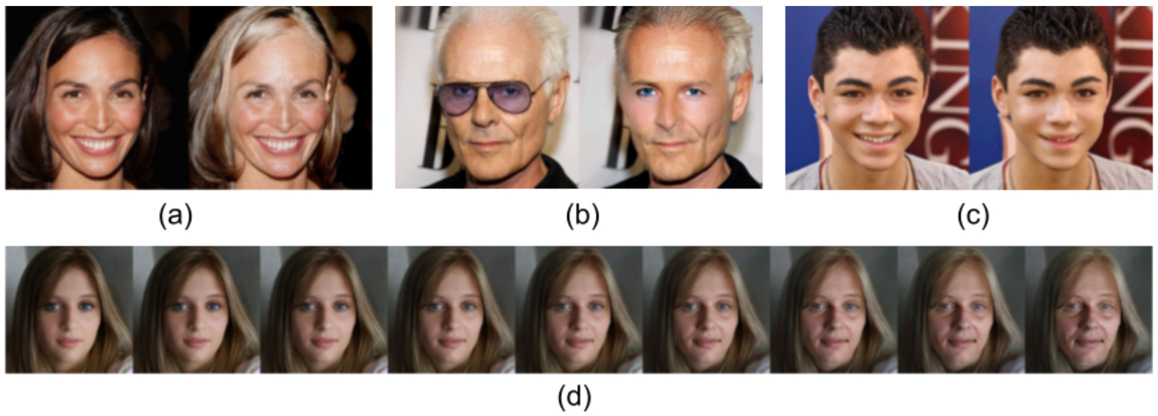


Figure 5: From left to right : the result obtained when using the network to change hair colour (a), remove glasses (b), and visage shape (c). For every example, the original image is shown on the left. Fig.(d) here, we can see the result of gradually modifying the age parameter. The original image is the fourth from the left (middle one).

	Glow	Emerging	3×3	Quad
Cifar10	0.58	18.4	9.3	10.8
Imagenet32	0.86	27.6	14.015	16.1
Imagenet64	0.50	160.72	82.04	84.06

Table 4: Time in seconds to sample 100 images. Results were obtained with Glow running on GPU and the other methods running on one CPU core.

Interpretability results: To show the interpretability of our invertible network, we used the Celeba dataset (Liu et al. [2015]) which provides images of faces and attributes corresponding to these faces. In Figure 4 are the randomly generated fake sample images for the cifar10 dataset. The covariance matrix between the attributes of images in the dataset and their latent representation indicates how to modify the latent representation of an image to add or remove features. Examples of such modifications can be seen in Figures 5(a, b, c) and 5(d).

5 CONCLUSION

In this paper, we propose a new method for Invertible n×n Convolution. Coupling layers solve two problems for normalizing flows: they have a tractable Jacobian determinant and can be inverted in a single pass. We propose a new type of coupling method, Quad-coupling. Our method shows consistent improvement over the Emerging convolutions method, and we only need a single CNN layer for every effective invertible convolution. This paper shows that we can invert a convolution with only one effective convolution, and additionally, the inference time and sampling time improved notably. We show the inversion of 3×3 convolution and the generalization of the inversion for the n×n kernel. Furthermore, we demonstrate improved quantitative performance in terms of log-likelihood on standard image modelling benchmarks.

References

Sandro Ackermann, Kevin Schawinski, Ce Zhang, Anna K Weigel, and M Dennis Turp. Using transfer learning to detect galaxy mergers. *Monthly Notices of the Royal Astronomical Society*, 479(1):415–425, 2018.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Cubic-spline flows. *arXiv preprint arXiv:1906.02145*, 2019a.

Conor Durkan, Artur Bekasov, Iain Murray, and George

Papamakarios. Neural spline flows. *arXiv preprint arXiv:1906.04032*, 2019b.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pages 881–889, 2015.

Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 2722–2730. PMLR, 09–15 Jun 2019.

Viet Tung Hoang and Phillip Rogaway. On generalized feistel networks. In *Annual Cryptology Conference*, pages 613–630. Springer, 2010.

Emiel Hoogeboom, Rianne van den Berg, and Max Welling. Emerging convolutions for generative normalizing flows. *arXiv preprint arXiv:1901.11137*, 2019.

Jörn-Henrik Jacobsen, Jens Behrmann, Richard S. Zemel, and Matthias Bethge. Excessive invariance causes adversarial vulnerability. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Diederik P. Kingma, Tim Salimans, and Max Welling. Improving variational inference with inverse autoregressive flow. *CoRR*, abs/1606.04934, 2016. URL http://arxiv.org/abs/1606.04934.

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10215–10224, 2018.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. volume 1, page 7. In Technical report, 2009.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

Thanh-Dat Truong, Khoa Luu, Chi Nhan Duong, Ngan Le, and Minh-Triet Tran. Generative flow via invertible nxn convolution. *CoRR*, abs/1905.10170, 2019. URL http://arxiv.org/abs/1905.10170.

A INPUT PADDING

To make sure the matrix (M) block triangular, padding of $(k+1)/2$ on top and left of input (x) is applied where k is the size of kernels.

B MASKING OF KERNELS

Considering the assumption that the numbers of input channels is equals to the output channels (N) then masking of the kernels depends only on the numbers of channels (N), let the kernel size is $k \times k$ and the $K_{k,k}^{a,b}$ are the entries corresponding to block $(N \times N)$ diagonals entire in the convolution matrix (M), where a and b are $a, b \in 1, 2, ..., N$. $K_{k,k}^{a,b}$ is a square matrix (C) of size $N \times N)$ and the diagonal entries of the block of matrix M are the diagonal of C and to ensure the invariability of the M, we have to set all the entries $C_{a,b}$ to zero when $a > b$ and one when $a = b$.

C COMPARISON OF OUR METHOD WITH THE EXISTING INVERTIBLE NORMALIZING FLOW METHODS

See figure 6.

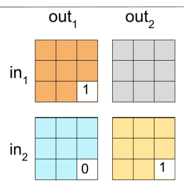
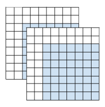
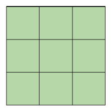
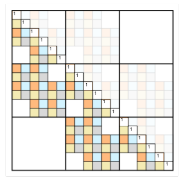
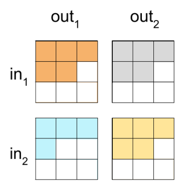
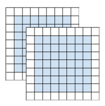
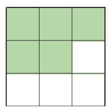
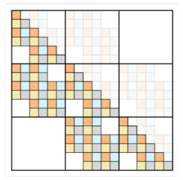
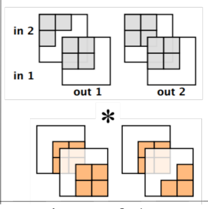
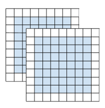
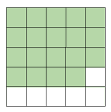
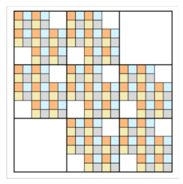
	Filters	Padding	Receptive Field	Convolution Matrix	Observations
CInC (Ours) Convolutions					n = #in,out channels. #learnable parameters **$9n^2$ - n(n-1)/2** #convs = **1** Invertibility is guaranteed in training since diagonal entries of matrix are 1s.
Autoregressive Convolutions					#learnable parameters **$5n^2$** #convs = **1** Number of learnable parameters are reduced by almost 50% resulting in lesser expressive power.
Emerging Convolutions					#learnable parameters **$10n^2$** #convs = **2** Having more convolutions will increase runtime during generation as well as latent vector computation passes.

Figure 6: Comparison of the speed and utilization of parameters with Autoregressive convolutions (Germain et al. [2015], Kingma et al. [2016]) and Emerging (Hoogeboom et al. [2019])

Automated Seed Quality Testing System using GAN & Active Learning

Sandeep Nagar[1], Prateek Pani[1], Raj Nair[2], and Girish Varma[1]

[1] Machine Learning Lab, International Institute of Information Technology, Hyderabad, India
[2] AdTech Corp, Hyderabad, India
sandeep.nagar, prateek.pani@research.iiit.ac.in,
raj@adtechcorp.in, girish.varma@iiit.ac.in

Abstract. Quality assessment of agricultural produce is a crucial step in minimizing food stock wastage. However, this is currently done manually and often requires expert supervision, especially in smaller seeds like corn. We propose a novel computer vision-based system for automating this process. We build a novel seed image acquisition setup, which captures both the top and bottom views. Dataset collection for this problem has challenges of data annotation costs/time and class imbalance. We address these challenges by i.) using a Conditional Generative Adversarial Network (CGAN) to generate real-looking images for the classes with lesser images and ii.) annotate a large dataset with minimal expert human intervention by using a Batch Active Learning (BAL) based annotation tool. We benchmark different image classification models on the dataset obtained. We are able to get accuracies of up to 91.6% for testing the physical purity of seed samples.

Keywords: Agriculture · Quality Testing · Generative Methods · Active Learning · Automation · Computer Vision · Image Classification.

1 Introduction

Quality checking is an essential step to ensuring food grain supplies. It ensures that stocks with different compositions of defective grains are not mixed and can be processed, packaged, and sold for appropriate uses, from high quality, economy packaging to animal feeds. However, quality checking for small grains and seeds like corn, rice is a tedious task done mainly through experts by visual inspection. Manual inspections are not scalable because of the human resources required, inconsistency between inspectors, and slower processing pipeline. An automated approach to seed/grain quality testing can solve many of these problems, leading to better usage and distribution of food stocks.

With the advent of Deep Neural Networks (DNNs), data-driven models have become increasingly adept at image classification/detection tasks. DNN models have been used in literature for seed quality testing problems [10–12]. However, there are some significant impediments to their widespread adoption. DNNs require large-scale datasets for giving high accuracies, which matches human inspectors. However, creating large-scale, good-quality datasets is challenging. Annotating seed data sets with defects requires experts with specialized knowledge. Also, a considerable imbalance of seeds can

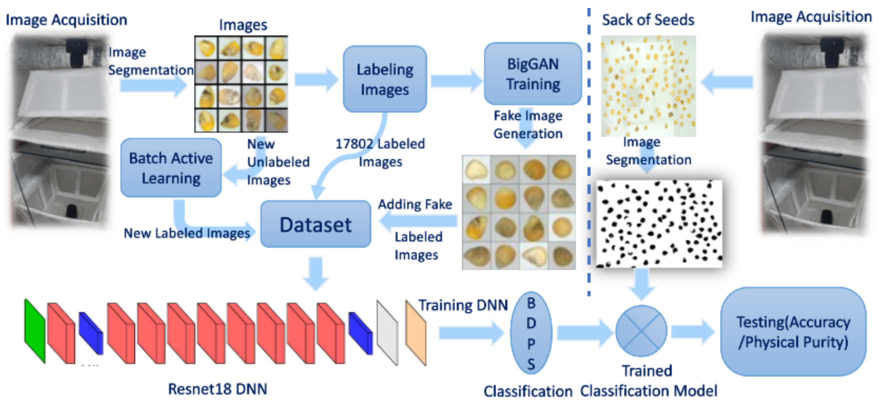


Fig. 1. Overview of our proposed system.

occur in a sample with a specific defect, which results in small datasets highly skewed to non-defective classes, with minimal images in some defective categories. Another main problem is that the defective part of the seed might not be visible from the top view.

We propose addressing class imbalance and annotation costs using machine learning techniques such as generative methods and BAL. We use active learning to select the most informative images from unlabeled data to be labelled by experts, leading to lesser annotation. We use generative methods, expressly conditional generative adversarial networks, to generative images of each class to address the class imbalance problem. We also develop a novel hardware design, which examines the seeds from the top and bottom, leading to better classification accuracy.

Active learning uses to select an initial batch of images to be annotated. After a round of annotation, the next batch is determined based on the uncertainty of classification remaining conditioned on the initial batch of annotation. Batch Active Learning also ensures that the images within the same batch are diverse compared to basic Active Learning methods. We build an annotation tool, which shows the next batch obtained by Active Learning. It also incorporates UI elements like suggestions for labels from the partially trained model on images labelled so far, which leads to short annotation. Using this tool, we build a dataset of 26K corn seed images, classified into pure seeds and three other defective classes (broken, silkcut, and discoloured).

However, the dataset created is highly imbalanced, with images of pure seeds being 40% while some of the defective classes being as low as 9% with 4 classes in total. We use a Conditional Generative Adversarial Network (CGAN) to address the class imbalance problem. A CGAN (BigGAN) trained on the labelled base dataset conditioned on the labels to generate good quality seed images. Then the CGAN is used to generate images that are indistinguishable from real images for each class. Hence this results in a more balanced dataset. Finally, an image classifier DNN is trained on this dataset (with the fake images added), leading to better accuracy of 80%.

Main Contributions. We build a computer vision-based system for large-scale automated quality checking of corn seeds.

1. We give a better hardware design, which takes images from the top and bottom to inspect defective/pure seeds (see Section 3.1).
2. We build an annotation tool using Batch Active Learning and specific UI elements to accelerate the annotation process (see Section 3.2).
3. We use Conditional Generative Adversarial Networks to generate fake images of each class, leading to a larger and more balanced training dataset (see Section 3.3).
4. We build a dataset of 26K corn seed images labeled as pure, broken, silkcut, and discolored (see Section 4.1).
5. We train an image classifier on the dataset with generated and real images, leading to improved accuracy (see Section 4).

2 Related Works

ML for Seed Quality Testing and Agriculture. Machine vision for precision agriculture has attracted research interest in recent years [3, 9, 8, 12]. Plant health monitoring approaches are addressed, including weed, insect, and disease detection [3]. With the success of DNNs, different approaches have been proposed to tackle problems of corn seed classification [12, 10, 11]. Fine-grained objects (seeds) are visually similar by a rough glimpse, and details can correctly recognize them in discriminative local regions.

Generative Methods for Class Imbalance. Generative models can not only be used to generate images [17], but adversarial learning showed good potential in restoring balance in imbalanced datasets [18]. Generative models can generate samples that are difficult for the object detector to classify. Creating a balanced dataset is a problem because the availability of one type of sample (seed) with defects or impurity compared to others is not always the same. While creating a new dataset, the imbalance of the instance, and we applied the fake image generation to overcome this. We use a generative model for the Image-to-image translation with conditional adversarial networks[15].

Batch Active Learning for Fast Annotation. Manual data annotation can be very slow and costly needs expertise for the same. There is no single standard format when it comes to image label/annotation. In our dataset, images contain only seeds, and each image is labelled one class out of four classes. Labelling the fine-grained image is challenging due to the significant intraclass variance and slight inter-class variance to recognize hundreds of sub-categories belonging to the same basic-level category. The aim of Active Learning (AL) is to discover the dependence of some variable (y) on an input variable (x) [2]. We use the Batch Active Learning (BatchBALD) [1] to label the images with the help of a small no. of manually labelled images.

3 Approach

We approach the problem of seed quality testing by first building a camera setup and preprocessing pipeline to obtain individual seed images from a sample which are then labelled (see Section 3.1). Then, since the data is highly imbalanced and to minimize the expert labelling effort, we propose two methods: i.) use Active learning-based UI tool to aid the creation of a larger dataset with the least effort from the expert human intervention (see Section 3.2) ii.) using Conditional Generative Adversarial Networks (CGAN) to generate images to solve the class imbalance problem (see Section 3.3).

3.1 Hardware Setup and Primary Dataset Creation.

Our seed quality testing approach makes use of a camera setup shown in Figure.2 below. In this setup, we use two cameras, one on the top and one bottom. We place a bunch of seeds in the middle on transparent glass, take a picture one from each camera and save it to the computer connected to the camera. To block the other side of the transparent glass, we use a thin white film. While capturing the top view of the seeds, we put the thin film below the glass, which works as a white background, similar to the bottom view. Thus, the top camera gives the picture of a top view of each seed, and similarly, the bottom camera capture the bottom picture of seeds to train the classification model. We use the top and bottom images of seeds as individual input images, which can get two independent predictions, increasing the accuracy.

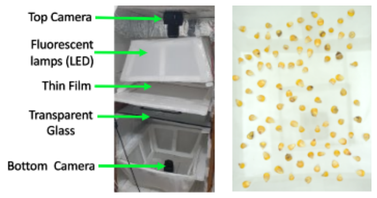


Fig. 2. Image capturing setup and sample image.

For seed classification, individual corns in the captured image should be accurately segmented first. During placing the seeds on the transparent glass, we mind the gap between the neighbour seeds to avoid the overlapping of seeds segmentation [13] of the image done by the Watershed method because this transform decomposes an image completely and assigns each pixel to a region or a watershed, and image segmentation can be accomplished simultaneously. After the segmentation, the expert's labelling of each seed image is done for the 17802, considering the top and bottom two different seed images. To classify new seeds after the training model, we take a picture of a sack of seeds and use the segmentation to detect the location [13] of each seed and give it as the input to the model classification.

3.2 Batch Active Learning (BAL) for Fast Annotation.

Data efficiency is a crucial problem in Deep Learning. Active learning, a sub-field in Machine learning, is centred around attaining data efficiency. To avoid the tedious data labelling of a large dataset, in Active Learning, we iteratively query the most informative points from a set of unlabelled points. However, in practical Active learning, rather than acquiring single points, we query a batch of points from an unlabelled set of most informative and diverse points. The question is which subset of points of the unlabelled set should be added to the training set so that the model would learn the fastest when trained on this updated training set than picking any other subset of the unlabelled points.

Moreover, since we can label multiple images in a single screen shown as a grid to the annotator (see Fig 4), there is a possibility of sampling points belonging to the same class of the underlying distribution. In such a scenario, we will have a model being accurate in one class but not in all other classes. To avoid this situation, we query the least confidence, and the queried set should be diverse. Batch Active Learning (BAL) was proposed to solve such problems, which we use to build an annotation tool.

3.3 Conditional Image Generation to Balance Dataset

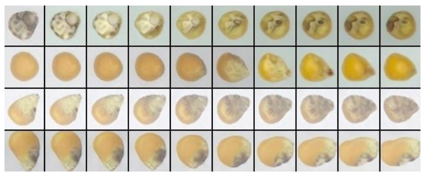


Fig. 3. Example of interpolation two GAN generated images. A gradual linear transformation from one seeds to another using latent vector(z) interpolation.

Generative Adversarial Networks (GANs) are one of the state-of-the-art approaches for image generation. Existing works primarily aim at synthesizing data of high visual quality [15, 16]. We use BigGAN [15], as it gives good quality images in similar datasets. Training generative adversarial networks (GANs) using too little data typically leads to discriminator (D) over-fitting [14]. GANs training is dynamic and sensitive to nearly every aspect of its setup (from optimization parameters to model architecture). **Image Interpolation:** Exploring the structure of the *latent space* (z) for a GANs model is interesting for the problem domain. It helps develop an intuition for what has been learned by the generating model (see Figure 3). We use image interpolation to evaluate whether the inverted codes are semantically meaningful. We interpolate one type of seed to other classes in a large diversity. As we can see in Figure 3 left to right, how smoothly the interpolation works, which validates that the GAN has learned a good latent representation for images in the dataset.

4 Results

4.1 Dataset & Experimental Details

Our primary dataset contains four classes: *broken*, *discolored*, *pure*, and *silkcut* (B, D, P, S), having different instances for each class (Imbalance). Images are taken for both sides of the corn, *top-view*(8901) and *bottom-view*(8901), as explained above in the hardware setup section-3.1. The primary dataset is highly imbalanced. To analyze the class confusion for seeds, we explore the confusion matrix in the result section4. Instances of each class and their % in the primary dataset are as follows: *broken* 5670 (32%), *discolored* 3114 (17.4%), *pure* $7267(40.8\%)$, *silkcut* $1751(09.8\%)$. We use two methods to add more images into the primary dataset: i.) adding GAN generated images to balance the dataset and ii.) adding more captured images labeled using the Batch Active Learning method.

In case of adding fake images to balance the dataset, we split the primary dataset into two parts train and validation in a 70:30 ratio, ensuring that each class has the same % of instances on each set. The train set is used to train the BigGAN model, and after adding fake images generated by BigGAN into the train set, this new train set is used to train the classification model. Finally, the Validation set is used for testing the classification model only.

We generated fake images as follows: *broken* 2937, *discolored* 5823, *pure* 2937, *silkcut* 5823 instances and added them into the train set to balance the data set. In the case of adding newly captured images labelled using the *Batch Active Learning* method after image segmentation, new 9000 labelled images are added into the *primary* dataset. This new dataset contains 26,802 images split into Train and Validation set $80 : 20$, respectively. The train set is used to train the classification model, and the Validation part is used to test the classification model.

We train different Convolutional Neural Networks (CNN) models on this dataset in Section 4.4. First, we use transfer learning; the models are initialized with weights learned from ImageNet Classification [6]. Then, we trained the DNNs on the train set of the primary dataset to fine-tune the model and compare the validation accuracy for a different model (see table1). Since Resnet18 has the highest accuracy, we trained it on three different datasets: i.) the primary dataset, ii.) with fake images generated using BigGAN, and iii.) after adding the newly annotated images labelled by the Batch Activation Learning method.

4.2 Batch Active Learning (BAL)

We start with the corn seed *primary* dataset with 17,802 images and do a 90%-10% train-validation split to train the BAL model. We also have an unlabelled dataset of corn seeds of 26,777 to label using BAL. Specifically, we use the BatchBALD [1] method. An active learning cycle involves retraining a model on the data annotated so far. During training, we use early stopping to avoid over-fitting. The used acquisition function is based on model uncertainty and ensures that the queried images are diverse in the predicted distribution models uncertainty entropy. Next, pick 5000 images that have the highest entropy (most informative points). To ensure diversity, perform k-means clustering on these 5000 images until stabilization to find the 1000 cluster centres. After that, find the points in the dataset closest to each of these 1000 centres. This spits out 1000 images queried by the above acquisition function along with their predicted labels. We then render these images and their predicted labels via an annotation tool that we have built for human annotation (see Figure.4). Next step, the training set is updated by accumulating these images and the labels that the annotator makes. A reinitialized model is then trained on the updated training set, and the cycle continues.

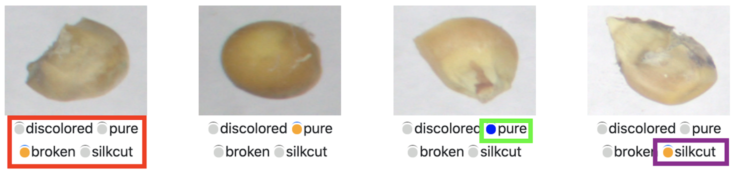


Fig. 4. The user interface of the annotation tool. A single-screen shows a batch of images. If the suggestion is wrong, the user can relabel it. After a few active learning cycles, the model suggestions become more accurate, resulting in less effort from the annotator. Purple box: label suggested by the model.

We record the validation accuracy for every cycle and find that with labelling only 9000 images of the 26,777 original unlabelled set, there is a significant increase in the validation accuracy from 46.83% to 73.97% which is shown in Figure 5. Moreover,

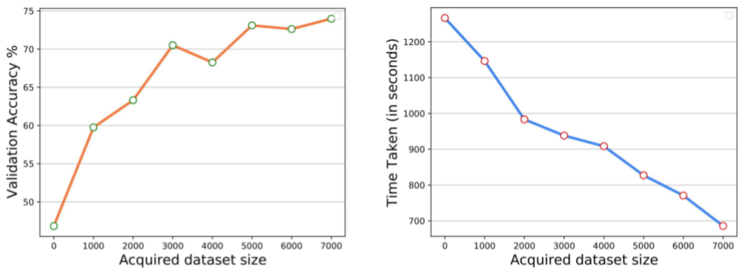


Fig. 5. (Left) Over 8 AL cycles shows an increase from 46.83 to 73.97% accuracy. (Right) Time taken to annotate shows a drastic decrease in annotation time over 8 AL cycles.

the annotation time is taken to annotate the first queried batch of 1000 images fell drastically from 1266.42 seconds to 682.02 seconds (which is the time taken to annotate the 8th queried batch) as reflected in Figure 5, thereby decreasing the annotation cost and efficiency significantly. After the 9th cycle, the total labelled instances in the dataset are 26802 after adding new labelled images.

4.3 Conditional Image Generation (CGAN)

We trained the conditional GANs to generate the fake images to add to the dataset and reduce the class instances' imbalance. To train BigGAN, the *primary* dataset is split into two sets, train and validation $70 : 30$, respectively. The train set is used to train the BigGAN and, after adding the fake images, to train the classification model. The validation set is used only to test the classification model. We trained the Generator (G) and Discriminator (D) of the BigGAN for 250 epochs, and after training, we passed a random noise and label to the G to generate the fake images. Hyper-parameters used are as follows: input image resolution$=$ $256 \times 256 \times 3$, learning rate $= 2e^{-4}$, batch size $= 16$, dimension of the latent vector space $\dim(z) = 128$. BigGAN is trained in alternate phases for D, G and we ensure that each input batch contains an equal no. of images from each class. Figure 6 shows and compares that the image generated is indistinguishable from real images.

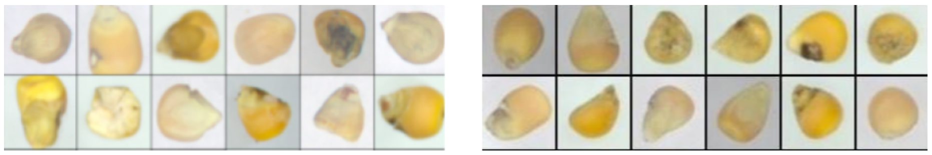


Fig. 6. Sample images from primary dataset (left) & ones generated using BigGAN (right).

4.4 Classification

We have validated the effectiveness of Active Learning in Section 4.2 and analyzed the quality of images generated by the Conditional GAN in Section 4.3. Here we discuss the accuracy of the classification model trained on the dataset obtained.

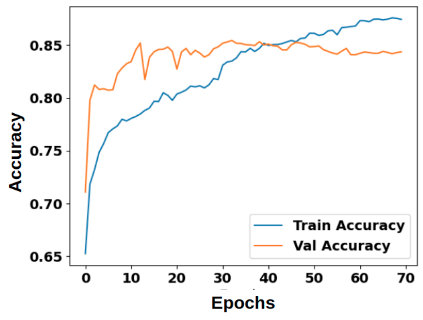
The highest validation accuracy of the primary dataset is 71.02% (see Table 1) among different deep learning models. In the case of adding fake images into the train set of the primary dataset to balance the dataset, the validation accuracy after training the classification model on the new train set increases the form 71.02% to 79.24% (see Table 2), thereby validating the approach of using CGAN to solve the class imbalance

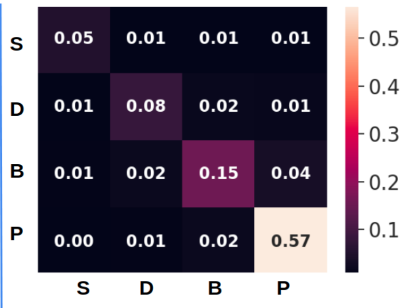
Model	Acc. %
resnet18	**71**
squeezenet	71
resnet50	70
mobilenet	68
wideResnet50	69

Table 1. Validation accuracy for different CNN models on *primary* dataset.

# fake images	Acc.%	Phys. Purity %
Zero	71.00	80.58
20K	**79.24**	**88.25**
40K	78.23	87.68
100K	78.88	87.24

Table 2. Validation accuracy comparison before and after adding the fake images to dataset(for resnet18 DNN) solving imbalance. *Accuracy:* four class classification accuracy. *Physical Purity*: Pure vs Impure(Two class classification)





	S	D	B	P
S	0.05	0.01	0.01	0.01
D	0.01	0.08	0.02	0.01
B	0.01	0.02	0.15	0.04
P	0.00	0.01	0.02	0.57

Fig. 7. (Left) Train and Validation accuracy of the ResNet18 using pre-trained model. (Right) Confusion Matrix: heat map of validation images (4946) after training the Resnet18 on train set of new dataset (26802). Ground Truth: rows, Predicted: columns. B: broken, D: discolored, S: silkcut, P: pure.

problem. The graph in Figure 7 plots the training and validation accuracy for the Resnet 18 on the dataset after adding new images labeled by the BAL. This further improves the accuracy to 85.24% and we also get the *physical purity* (*pure* vs *impure* classification) accuracy to be 91.62%. The classwise accuracies are as follows: *broken* 71.20%, *discolored* 69.08%, *pure* 94.94%, *silkcut* 75.82%.

Some images in the dataset have high-class ambiguity. To analyze, we used the confusion matrix given in Figure 7 for the validation set. Each matrix entry gives the % of images of specific ground truth (rows) and a specific predicted class (columns). As can be seen from Figure 7, the classes *pure* and *broken* are most confusing followed by *discolored* and *broken*.

5 Conclusions & Discussion

We propose a novel computer vision-based automated system that can be used for corn seed quality testing. A novel image acquisition setup is used so that two different viewpoints are obtained for every seed. Furthermore, we decrease the human intervention required for labelling by building a BAL based UI tool. We also address the class imbalance problem by using Conditional GANs (BigGAN) to generate more images of classes with a small dataset. We believe similar approaches can be used for quality testing of various seeds and vegetables and can decrease wastage and human intervention.

References

1. Kirsch, A., Amersfoort, J. Van., Gal, Y.: BatchBALD: Efficient and Diverse Batch Acquisition for Deep Bayesian Active Learning. In: 33rd Conference on Neural Information Processing Systems (NeurIPS), Vancouver, Canada (2019)
2. Houlsby, N., Huszár, F., Ghahramani, Z. and Lengyel, M.: Bayesian active learning for classification and preference learning.
3. Mavridou E., Vrochidou E., Papakostas G. A., Pachidis T., Kaburlasos, V. G.: Machine Vision Systems in Precision Agriculture for Crop Farming. Journal of Imaging **5**(12), 89 (2019)
4. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Li, F.-F.: ImageNet: a Large-Scale Hierarchical Image Database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248-255. IEEE, Miami, FL, USA (2009)
5. Cireşan, D. C., Meier, U., Masci, J., Gambardella, L. M., Schmidhuber, J.: High-performance neural networks for visual object classification. (2011) https://arxiv.org/abs/1102.0183v1
6. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Adv. in neural information processing systems, 25, pp.1097-1105 (2012)
7. He, K., Zhang, X., Ren, S. and Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR) pp. 770-778. IEEE, Las Vegas, NV, USA (2016)
8. Asif, M.J., Shahbaz, T., Rizvi, S.T.H., Iqbal, S.: Rice Grain Identification and quality analysis using image processing based on principal component analysis. In: 2018 International Symposium on Recent Advances in Electrical Engineering (RAEE), pp. 1-6, IEEE, Islamabad, Pakistan (2018)
9. Tian, H., Wang, T., Liu, Y., Qiao, X. and Li, Y.: Computer vision technology in agricultural automation-A review. Information Processing in Agriculture **7**(1), pp.1-19 (2020)
10. Velesaca, H.O., Mira, R., Suárez, P.L., Larrea, C.X., Sappa, A.D.: Deep Learning based Corn Kernel Classification. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops pp. 66-67. IEEE, Seattle, WA, USA (2020)
11. Khaki, S., Pham, H., Han, Y., Kuhl, A., Kent, W., Wang, L.: Deepcorn: A semi-supervised deep learning method for high-throughput image-based corn kernel counting and yield estimation. Knowledge-Based Systems **218**, 106874 (2021)
12. Khaki, S., Pham, H., Han, Y., Kuhl, A., Kent, W., Wang, L.: Convolutional neural networks for image-based corn kernel detection and counting. Sensors, **20**(9), 2721 (2020)
13. Li, X., Dai, B., Sun, H., Li, W.: Corn classification system based on computer vision. Symmetry, **11**(4), 591 (2019)
14. Karras, T., Aittala, M., Hellsten, J., Laine, S., Lehtinen, J., Aila, T.: Training generative adversarial networks with limited data. (2020) https://arxiv.org/abs/2006.06676
15. Brock, A., Donahue, J., Simonyan, K.: Large Scale GAN Training for High Fidelity Natural Image Synthesis. In: 7th International Conference on Learning Representations (ICLR) New Orleans, LA, USA (2019)
16. Wang, T.C., Liu, M.Y., Zhu, J.Y., Tao, A., Kautz, J., Catanzaro, B.: High-resolution image synthesis and semantic manipulation with conditional GANs. In: Proceedings of the IEEE conference on CVPR pp. 8798-8807, IEEE, Salt Lake City, UT (2018)
17. Sampath, V., Maurtua, I., Aguilar Martín, J.J. et al. A survey on generative adversarial networks for imbalance problems. In: computer vision tasks. J Big Data **8**, 27 (2021)
18. Nazki, H., Lee, J., Yoon, S., Park, D. S.: Image-to-Image Translation with GAN for Synthetic Data Augmentation in Plant Disease Datasets. In: Korean Institute of Smart Media, **8**(2), pp. 46–57 (2019)