Directed Generative Nets Architectures: Review

Andrii Zadaianchuk
Neural Information Processing GTC
Tübingen University

1 Introduction

Image generative models are supposed to capture the distribution of natural images and generate new samples from this distribution.They are important for solving huge variety of learning tasks. For example, the results in such tasks as image compression [61], super resolution [9, 42, 58], inpainting [74, 46], generation of new images from text descriptions [78, 54] have been improved greatly by generative models. In addition, they can simplify and improve the solution of many other machine learning tasks. They are used to improve semi-supervised learning methods [35, 59]. They also are used to predict changes in the environment for reinforcement learning tasks [16] and made slow and expensive computations faster by predicting the result of the computations [10, 68, 5]. However, high-dimensional structured image generation is a challenging task despite almost unlimited unlabeled datasets [50].
Long time the main problem of most generative models (e.g. Bolzman machines) was an inability to scale to big datasets with high dimensional data [20]. In last years, there were several new models such as Generative Adversarial Networks [22], Variational autoencoders [38, 56] and PixelCNN [50] that are scalable enough to apply them to natural images. Despite the principal importance of these models, there were a lot of problems such as unstable training, blurry images or too slow image generation. Inadequate estimation of the quality of the generative models also was a serious problem [67].
A lot of effort was done to make generative models stable, more accurate and faster. Numerous modifications [52, 59, 2, 60, 4] and combinations [45, 44, 24] of these models architectures were developed. An important step in the generation of natural images was the usage of a pre-trained supervised neural network to find the perceptual similarity between images [17, 33, 18]. This approach improved quality of the generated images in many tasks [5, 12, 46].
Most of the scalable generative methods model sampling process (e.g. generator net in GANs) or conditional distribution (e.g. variational decoder in VAE) by a deterministic differentiable neural network that is called generator network [20]. The architecture of generator net effects learning of the whole model. The proper choice of this architecture can accelerate learning and make it more stable. Usage of the sequential or multi-scale generation allows constructing generators that have fewer parameters and learn faster. In this work we review how the architecture of the generator effects different aspects of learning dynamics and quality of generated images. In particular, we look at different modules or structural elements that improve the generative model.

2 Common structural elements

2.1 Generator layers: improved upsampling and no fully connected layers

The convolutional and pooling layers are the important part of deep learning models that process images. Layers that don't change the size of the input were well studied in image classification

[27, 25, 7, 64]. The main gain in the accuracy of image classification models was obtained from the exact composition of the convolutional modules.The usage of similar techniques in generator network such as residual-like connections [71] or gated non-linearities [50] improved quality of the generator. On the other side, the upsampling layers that increase the size of the input were less studied and researchers often use a typical layer composition. However, Wojna [71] showed that the choice of the upsampling layer is important for several different tasks such as super resolution and colorization. Here we will discuss several successfully combined convolutional layer modifications that were used in the generator network.
In DCGAN [52] the main change in the architecture was new layers composition in the generator and the discriminator. The ideas for this architecture are taken from [63] where authors showed that it is possible to create "all convolutional" neural network (e.g. see Figure 3) that contains only convolutional layers and has the same performance as CNN with pooling and fully connected layers. In "all convolutional" network pooling layers are substituted by the strided convolutions with stride $s > 1$. One can interpret the stride convolution layers as the pooling layer where exact shape of the pooling kernel is also a parameter of the model [33] .

2.1.1 Transposed convolution operator

To adapt these convolutions to the generator network we can use the fact that the convolution is a linear operation and can be represented as matrix multiplication. If we have a convolution layer with stride $s > 1$ we can write it as $\boldsymbol{x} * W = \boldsymbol{x} \cdot C(W)$, where $C(W)$ is $n \times m$ matrix and $m < n$. Transposed or fractional-stride convolution [14] is defined as $\boldsymbol{x'} \cdot C(W)^T$. Figure 1 illustrates a transposed convolution operation.

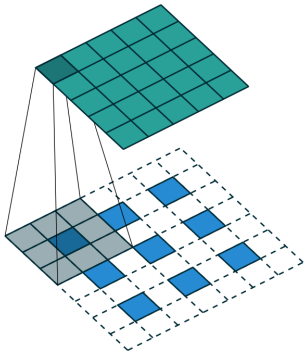


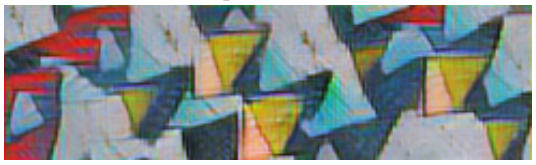
Figure 1: Transposed convolution operation. We add zero padding between image pixels to make the stride $s < 1$ (Figure was taken from [14])

There are several different modifications of transposed convolutions. One of them, Decomposed Transposed Convolution [64] use several low-rank 1D convolutions to simulate 2D transposed convolution. Another one is Separable Transposed Convolution [7]. It is a sequence of transposed convolutions that is done layer by layer and a pointwise convolution with 1×1 kernel that combines the information from channels. These modifications reduce the number of parameters, however, they are a strict subset of Transposed Convolutions. Wojna et al. [71] provided comprehensive compression of the performance of this upsampling layers on five different tasks.

2.1.2 Nonparametric upsampling + convolution

Despite the fact that transposed convolution works well in DCGAN, it tends to create checkerboard patterns in a generated image [48]. Checkerboard artefacts are caused by the fact that transposed convolution kernel parameters fail to adapt completely to the uneven overlap (e.g. when the kernel size is not divisible by the stride). The solution to this problem is to use better upsampling such as nearest-neighbour interpolation in combination with convolution layer. In Figure 2 you see the comparison of these two upsampling procedures. However, such method is more computationally

(a) Transposed convolution



(b) Nearest-neighbor interpolation

Figure 2: Better upsampling helps to avoid checkerboard artifacts (Figure was taken from [48])

expensive, because the convolution that you applied after upsampling has 4 times more parameters for upsampling by the factor of 2. One of the ways to deal with it is just to sum every four feature maps before doing convolution (e.g. bilinear additive upsampling [71]). Another way is to do the convolution with the bigger number of feature maps and than rearrange them in bigger spacial representation (e.g. subpixel convolution [62])
In addition, Dosovitsiy et al. [13] proposed to insert additional convolutional layers between upsampling+convolutional layers. He argued that the quality of the generated images was significantly improved by incorporating additional convolution layers between upconvolutional layers. This idea was used in many successful generative models such as state-of-the-art "Plug and Play" generative model. [12, 41, 46, 47, 58]. Look at Figure 4 for the illustration of this architecture.

2.1.3 Properties of convolutional layers

The exact properties of convolutional layers such as the number of feature maps, size of the kernel, stride and the number of layers depend on the size of the generated image and amount available of data to learn the parameters of the model. However, there are some typical choices such as stride $s = \frac{1}{2}$ or upsampling with the factor 2 and simultaneous shrinkage of the number of feature maps with the same factor. You start from small 2×2 or 4×4 representation and increase the resolution with each layer until you get your image size. Thus the number of layers is approximately $\log_2 n$, where the generated image has size $n \times n$. Exact parameters of two frequently used architectures are shown in Figure 3 and 4.

2.2 Non-linear activation functions

Non-linear transformations are the important part of a neural network. In Figure 5 you see different non-linear activation functions that were used by deep learning community. Here we discussed several of them that were used in generator networks.

2.2.1 Rectified Linear Unit (ReLU)

The ReLU activation function is given by

$$\mathrm{ReLU}(x) \;=\; \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}. \tag{1}$$

This activation function was successfully used as non-linear activation in CNN for supervised tasks [40]. Comparable to previously used activation functions this activation speeds up training by fast gradient calculation and is more robust to noise. For generative models, the ReLU was recommended

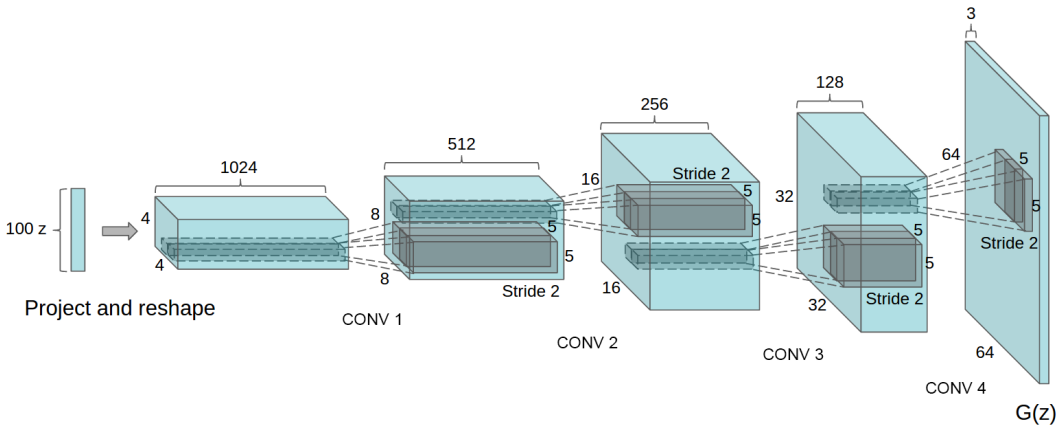


Figure 3: DCGAN generator used for modelling of hotel room images. Four fractionally strided or transposed convolutions were used to produce a 64×64 pixel image. Notably, no fully connected or pooling layers are used. (Figure was taken from [52])

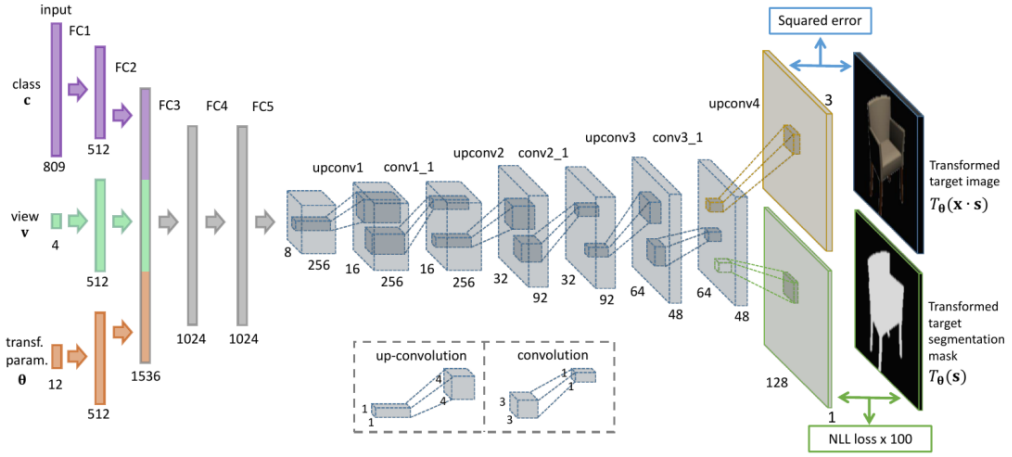


Figure 4: The architecture of a deep network that generates 128×128 pixel images. Layer names are shown above: FC - fully connected, upconv - upsampling+convolution, conv - convolution. (Figure was taken from [13])

by Radford [52], who used it in generator part of DCGAN. Later many different researchers [33, 79] followed this recommendation.

2.2.2 Leaky Rectified Linear Units (lReLU)

There are several different types of the ReLU such as a lReLU [43], Parametric lReLU [26] and Randomised lReLU [72] that use small linear negative part.

$$\text{leakyReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}, \tag{2}$$

where α is a near-zero constant for leaky ReLU and a model parameter for Parametrized leaky ReLU. Such activation functions are sensitive to high negative activations and thus they are noisier.

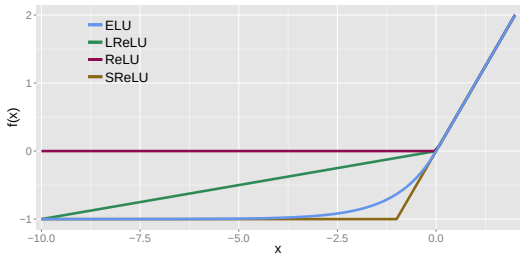


Figure 5: The rectified Linear Unit (ReLU), the leaky ReLU (LReLU, $\alpha = 0.1$), the shifted ReLUs (SReLUs), and the exponential linear unit (ELU, $\alpha = 1.0$). (Figure was taken from [8])

2.2.3 Exponential Linear Unit (ELU)

The definition of the ELU is given by

$$\mathrm{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leq 0 \end{cases}, \tag{3}$$

where α is hyperparameter.
This type of non-linearity is developed by Clevert [8]. The main difference from the ReLU is that for negative arguments the ELU has small negative activation. This activation exponentially saturates, thus it is robust to noise and in addition has close to zero mean activation.
Yang [74] compared effectiveness of ELU in regression network for image inpainting task. He stated that the ELU makes training more stable than the leaky ReLU layers as it can handle large negative responses. In recent work on combining GAN and VAE Mescheder et al. [45] also used the ELU as a good alternative to the ReLU and leaky ReLU.

2.3 Normalization

2.3.1 Batch normalization

Training of a deep learning model can be a challenging task. It is complicated by the fact that the distribution of each layer's inputs changes during training as we change the parameters of previous layers. To deal with this problem one can perform normalization of all the layers for each training mini-batch [30]. Suppose that we have a mini-batch of layer activations $\boldsymbol{H}$ (where each row contains activations from one example). To normalize $\boldsymbol{H}$ we replace it with

$$\boldsymbol{H}' = \boldsymbol{\gamma}\frac{\boldsymbol{H} - \boldsymbol{\mu}}{\boldsymbol{\sigma}} + \boldsymbol{\beta},$$

where $\boldsymbol{\mu}$ is a vector containing the mean of each unit, $\boldsymbol{\sigma}$ a vector containing the standard deviation of each unit and $\boldsymbol{\gamma}, \boldsymbol{\beta}$ are trainable parameters of the model. Such method of adaptive reparametrization is called batch normalization. It allows to choose higher learning rate and to be less careful with initialization of the network parameters. In addition, batch normalization acts as a regularizer improving the generalization of the model.
It is important to implement batch normalization operations as part of the computation graph and to account them during calculation of the gradient. For such modified architecture of the neural network, the gradient will never propose an operation that acts simply to increase the standard deviation or mean of mini-batch activation because cost function will not decrease in this case.
Radford [52] used batch normalization in generator and discriminator networks to avoid collapsing of the generator network outputs to a single point. He applied batch normalization for all layers except the last layer of the generator network and the first layer of the discriminator network. This helped to avoid oscillations and instability in GAN training.
Despite the fact that the batch normalization significantly improves learning dynamics, it also causes some problems. When the batch size is small normalizing constants will fluctuate and the output of the generator will depend on these fluctuations [19]. Salimans et al. [59] proposed to use reference batch that is ones defined and then used to compute normalizing constants during the training for all

mini-batches. It is computationally expensive because it doubles the amount of computations needed for feedforward propagation through the network, so it is better to use it only in generator network. Ioffe [29] also tried to solve this problem. He developed batch renormalization algorithm which ensures that the layer activations depend only on a single example. It consists of several steps. First, you calculate global mean and variance that are a moving average of mean and variance across several mini-batches. Then you correct values in the current mini-batch in order to make them closer to the global mean and variance. During inference, they use these global mean and variance as mean and variance of mini-batch or instance. In contrast to [59], this method doesn't need additional computations and significantly improves the training on small mini-batches.

2.3.2 Instance normalization

Instance normalization is an alternative to batch normalization where normalization is done to every sample rather than to the whole batch. Ulyanov [69] showed that instance normalization can significantly improve fast style transfer [68, 33]. Recently Huang et al. [28] provide a new interpretation to the effectiveness of instance normalization in style transfer. He said that instance normalization performs style normalization by normalizing feature statistics, which have been found to carry the style information of an image. Using this interpretation he developed an adoptive instance normalization layer that adjusts mean and variance of the content input to match those of the style input. Usage of such layer allowed him to create fast and flexible style transfer algorithm that is as fast as the feedforward style transfer algorithm and is able to transfer arbitrary style.

2.4 Sequential generation

One common approach to generate complex images is to construct them part by part. For models with latent variables, it means that we generate sequentially T groups of latent variables with size k instead of generating simultaneously one group of size $K = kT$. Such models generate the next part of the image conditioned to the parts that were generated previously. Most of them use the spatial attention mechanism (e.g. spacial transformer module [32]) that works similar to the fovea of the human eye. The DRAW model [23] was one of the first models that try to generate natural images (though they managed to generate only Street View House Numbers) part by part. In every time step t, we use encoder RNN to produce hidden state h_t^{enc} that integrates information about the previously produced sequence of latent variable samples $z_{1:t-1}$ and a new piece of the image x. Using this hidden state the model can produce a new sample z_t that is used by sequential decoder to produce new hidden state h_t^{dec} and later to produce new canvas c_t with a generated piece of the image (see Figure 6). Later Rezende and Mohamed [55] used the similar architecture to achieve one-shot generalization ability.

Special kind of sequential generation is when generated parts are independent. When parts of the image are not connected (e.g. background and foreground) we can generate them in parallel. Yan [73] used this idea in disCAE model (see Figure 7). He used disentangled latent representations of background and foreground image to account independence of background and foreground.

2.4.1 Spatial Transformer Module

Spatial Transformer module was used in several sequential generative models [15, 54, 55]. It is differentiable and can be trained with back-propagation. At the same time it allows applying large family of spacial transformations (e.g. affine transformation) that means that we can localize important part of the image (e.g. eyes in a face image) and transform it to the canonical view. This module (see Figure 8) consist of localization network that learns the parameters of transformation and the differentiable sampling mechanism that transforms regular grid of output to the transformed grid according to the transformation parameters.

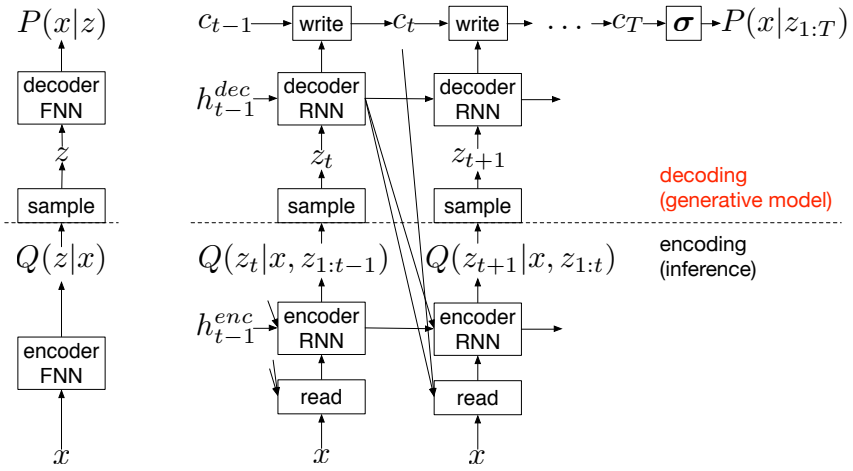


Figure 6: **Left: Conventional Variational Auto-Encoder**. During generation, a sample z is drawn from a prior $P(z)$ and passed through the feedforward decoder network to compute the probability of the input $P(x|z)$ given the sample. During inference the input x is passed to the encoder network, producing an approximate posterior $Q(z|x)$ over latent variables. During training, z is sampled from $Q(z|x)$ and then used to compute the total description length $KL\big(Q(Z|x)||P(Z)\big) - \log(P(x|z))$, which is minimised with stochastic gradient descent. **Right: DRAW Network**. At each time-step a sample z_t from the prior $P(z_t)$ is passed to the recurrent decoder network, which then modifies part of the canvas matrix. The final canvas matrix $_T$ is used to compute $P(x|z_{1:T})$. During inference the input is read at every time-step and the result is passed to the encoder RNN. The RNNs at the previous time-step specify where to read. The output of the encoder RNN is used to compute the approximate posterior over the latent variables at that time-step. (Figure was taken from [23])

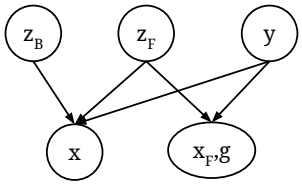


Figure 7: Graphical model representations of attribute-conditioned image generation models with disentangled latent space(disCAE). (Figure was taken from [73])

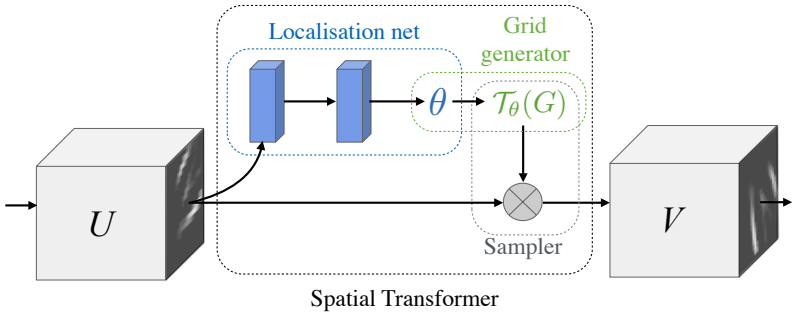


Figure 8: The architecture of a spatial transformer module. The input feature map U is passed to a localization network which regresses the transformation parameters θ. The regular spatial grid G over V is transformed to the sampling grid $\mathcal{T}_\theta(G)$, which is applied to U, producing the warped output feature map V. The combination of the localization network and sampling mechanism defines a spatial transformer. (Figure was taken from [32])

2.5 Multi-scale generation

Finding the mapping from a random noise sample to high-dimensional data such as a 256×256 image is difficult. One way to simplify the process of generation is to split it into the sequence of more simple generations. In such process, the input of the next generator is a random noise sample combined with the output of the previous generator. There are quite a few generator architectures that use such process of generation [11, 70, 78, 5, 68]. Below we describe several examples in detail.

We can use Laplacian pyramid representation to construct a sequence of smaller images. The Laplacian pyramid is a linear invertible image representation. It consists of a set of images with a band-pass filtered version of an original image and a low-frequency residual. The intervals for the band is uniform in logarithmic scale. Denton [11] used this representation to step by step generate an image. In Fig. 9 you see the architecture of LAPGAN generator. In LAPGAN generators transform sample from the latent random variable to the Laplacian pyramid coefficients using previously generated image representation. This modification allowed to use convolutional layers for the first time in GAN models.

The similar architecture was successfully used in Photographic Image Synthesis from segmentation maps [5]. Chen et al. used Cascaded Refinement Networks that consists of several modules. Each module operates at a given resolution. It learns to modify image representation of smaller scale using a segmentation map as additional input. Using such architecture and perceptual loss [33] authors achieved significantly more realistic image synthesis than the prior state of the art. Look at Fig. 19 in Supplementary materials for qualitative comparison of images.

Most of the multi-scale models use low-resolution image and refinement of this image on different scales. However, it is possible to use completely different representations. For example, another way to exploit the multi-scale structure of natural images is to split them into style and structure of the image [70]. Structure of the image can be represented by surface normal maps. First, we learn the generative model to generate the surface normal map and then we use the generated normal map and additional noise sample to generate image (see Figure 10). The disadvantage of this work is that we need to have ground truth surface maps in addition to the images in our dataset. However, it can be used in realistic computer graphics simulation.

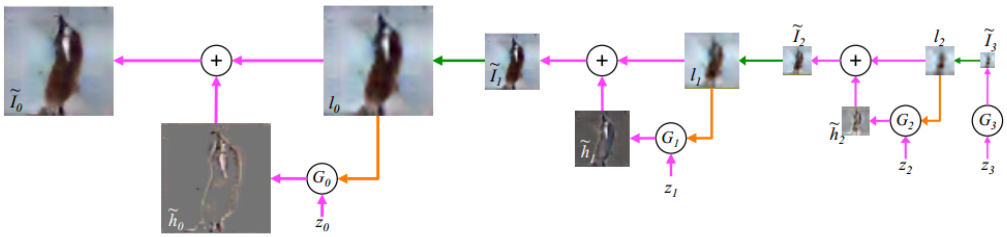


Figure 9: Architecture of LAPGAN. (Figure was taken from [11])

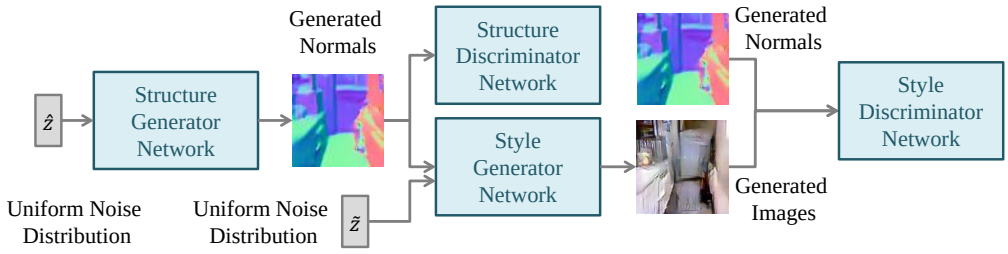


Figure 10: Full model of the S^2-GAN. It can directly generate RGB images given $\hat{z}, \tilde{z}$ as inputs. During joint learning, the loss from Style-GAN is also passed down to the Structure-GAN. (Figure was taken from [70])

3 Model Specific structural elements

3.1 Generative Adversarial Networks

Generative Adversarial Networks or GANs [22] are differentiable generator networks that use additional discriminator network to distinguish between outputs from the generative network and real images from train data. We generate sample $\boldsymbol{x}$ from the evaluated distribution in two steps: first, we generate vector $\boldsymbol{z}$ from simple prior distribution $p(\boldsymbol{z})$ and than we use differentiable mapping $G(\boldsymbol{z}, \boldsymbol{\theta^g})$ to generate sample $\boldsymbol{x}$. Typical example of $p(\boldsymbol{z})$ is uniform distribution $U[-1, 1]$. During training, we update the parameters $\boldsymbol{\theta^d}$ of discriminator $D(\boldsymbol{x}, \boldsymbol{\theta^d})$ to estimate the probability that sample came from the training data whereas parameters $\boldsymbol{\theta^g}$ of the generator $G(\boldsymbol{z}, \boldsymbol{\theta^g})$ are updated to maximize the probability of $D(\boldsymbol{x}, \boldsymbol{\theta^d})$ making a mistake. This corresponds to a minimax two player game.

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]. \tag{4}$$

The main advantage of differentiable generator networks is that it is possible to use complex neural networks as generator function $G(\boldsymbol{z})$ and train the model using backpropagation. In case of GANs, the equilibria for a minimax game is not a minimum of $V(D, G)$. Equilibrium of minimax game is the local minimum for generator parameters $\boldsymbol{\theta^g}$ and local maximum for discriminator parameters $\boldsymbol{\theta^d}$. Therefore, they are saddle points of V. Unfortunately, there is no theoretical guaranty that such game will converge to the saddle point of V when we use gradient updates to train it [21]. In practice, there were many problems with convergence and stability of training first GANs models.
The choice of the generative network architecture in GANs effects the quality of generated samples as well as stability of the training. Thus many efforts were done to find out what structures of $G(\boldsymbol{z})$ ensure stable training and provide good quality samples.

3.1.1 Addition of decreasing noise

One empirically found method to improve stabilization of GANs is to make the discriminator's job harder. There are many ways to make learning of discriminator more difficult. For example one can use early stopping of discriminator optimization or flipping of some labels in data. For example, Salimans [59] used one-sided label smoothing. This method involves changing the labels from 0 and 1 to 0 and 0.9, decreasing confidence of discriminator. Thus it adds some noise to the answers.
More general approach is called instance noise injection(first described in appendix of [65]). The main idea of this method is injecting some random noise to real and fake data. This allows to train discriminator until convergence or to take more gradient steps between subsequent updates to the generator.
An important step in understanding theoretical reasons for improvements due to instance noise injection was done in current work [1] on training dynamics of GANs. The author argued that KL, inversed KL and JSD are useless when supports of real distribution $p_r(x)$ and modelled $p_g(x)$ lie in manifolds that don't have full dimension and don't perfectly align. Using them as a measure of similarity between distributions can be inadequate in high dimensional space where probability is concentrated mainly in a smaller dimensional manifold. Arjovsky proposed to use a Wasserstein metric as a measure of similarity in GANs [2].

Definition 3.1. *The definition of the Wasserstein metric $W(p(x), q(x))$ for $p(x)$ and $q(x)$ two distributions over $\mathcal{X}$ is*

$$W(p(x), q(x)) = \inf_{\gamma \in \Gamma} \int_{\mathcal{X} \times \mathcal{X}} \|x - y\|_2 d\gamma(x, y),$$

where Γ is the set of all possible joints on $\mathcal{X} \times \mathcal{X}$ that have marginals $p(x)$ and $q(x)$.

This metric, in addition to the similarities of the shapes $p(x)$ and $q(x)$, incorporates a notion of distance between manifolds: when the supports of $p(x)$ and $q(x)$ get closer it goes to 0.
In addition to the usage of Wasserstein metric as better criteria of distribution similarity, he proved the following theorem, that links the usage of additional noise in JSD to the minimization of Wasserstein metric.

Theorem 3.1. *Let $x \sim p_r(x)$ and $x' \sim p_g(x')$ and ϵ be a random vector with mean 0 and variance V. Let us denote $x + \epsilon \sim p_{r+\epsilon}$ and $x' + \epsilon \sim p_{g+\epsilon}$. If $p_{r+\epsilon}$ and $p_{g+\epsilon}$ have support contained on a ball of diameter C, then*

$$W(p_r, p_g) \leq 2V^{\frac{1}{2}} + 2C\sqrt{JSD(p_{r+\epsilon} \| p_{g+\epsilon})}. \tag{5}$$

To sum up, with decreasing noise variance minimization of JSD will also minimize the Wasserstein metric. In practice [79, 59] is it better to insert the same independent additional noise to every layer of the generative network.

3.2 VAE

Variational autoencoders [37, 56] are another type of differentiable generator networks that use the generator in combination with inference network. First, latent variable $\boldsymbol{z}$ is sampled from some prior $p_{\boldsymbol{\theta}}(\boldsymbol{z})$, then this sample is transformed by the generator network to the parameters of the conditional distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) = p(\boldsymbol{x}; G(\boldsymbol{z}))$. During training, we use additional network $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ for approximate inference.
The training process is based on optimisation of the log-likelihood lower bound :

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}^{(i)}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})\right] - D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_{\boldsymbol{\theta}}(\boldsymbol{z})) \leq \log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}). \tag{6}$$

The first term in Equation 6 is the reconstruction log-likelihood typical for autoencoders models. The second term is the regularizer that pushes approximate inference distribution $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})$ to the simple prior distribution over latent variable $p_{\boldsymbol{\theta}}(\boldsymbol{z})$.
There were several problems with the quality of the samples and properties of the latent variables for the fist VAE models. The samples were blurry and most of the dimensions of the latent variable were useless. Here we looked at architectural changes that helped to deal with such problems.

3.2.1 Over-pruning

Bruda et al. [4] showed that VAEs has the problem of factor over-pruning. Many dimensions of the latent variable $\boldsymbol{z}$ are not used to learn anything about data and become inactive. This leads to decreased model capacity and a suboptimal generative model. There were several attempts to deal with this problem [36, 3] based on restricting the regularizer term in variational lower bound. However, these approaches change lower bound and take away the principled regularization scheme of VAE. Yeung el al. [76] proposed epitomic variational autoencoder(eVAE). This model uses groups of mutually exclusive latent factors (parts of the latent vector $\boldsymbol{z}$) that compete to explain the data. The authors called these groups epitoms. The process of selection epitomes consists of two parts. First, you sample from the discrete distribution an epitome selector variable y. Then you use the mask $\boldsymbol{m_y}$ that is zero everywhere except K dimensions that correspond to chosen epitom. Look at Figure 11 to see the process of generation. In this way, Epitomic VAE models are forced to learn latent space as multiply shared subspace. Authors showed that every subspace corresponds to a different specialization (see Figure 11).

3.3 PixelCNN

Pixel CNN [50] is an autoregressive generative model. It models joint distribution over pixels of an image by modelling the product of conditional distributions. Every pixel is generated conditioned on the previously generated pixels. Formally, the PixelCNN models the following distribution:

$$p(\boldsymbol{x}) = \prod_{i=1}^{n^2} p(x_i|x_1, ..., x_{i-1}), \tag{7}$$

where x_i is a pixel of the image $\boldsymbol{x}$.
Every conditional distribution is modelled by the same convolutional neuron network with masked convolutions. Look at Figure 12 to see the visualisation of PixelCNN and masked convolution.

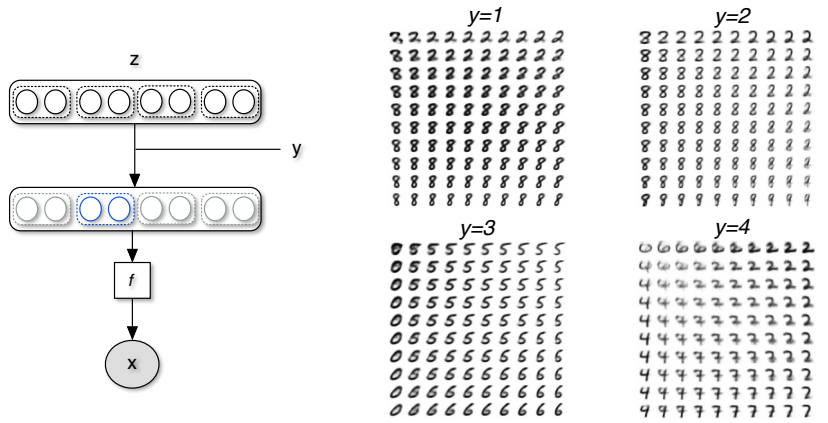


Figure 11: **Left:** Illustration of an epitomic VAE with dimension D=8, epitome size K=2 and stride S=2. In this depiction, the second epitome is active. **Right:** Learned manifolds on MNIST for 4 different epitomes in a 20-d eVAE with size $K = 2$ and stride $s = 1$. We observe that each epitome specializes on a coherent subset of examples; this enables increasing the diversity of the samples generated while maintaining quality of the samples when the latent dimension is large. (Figure was taken from [76])

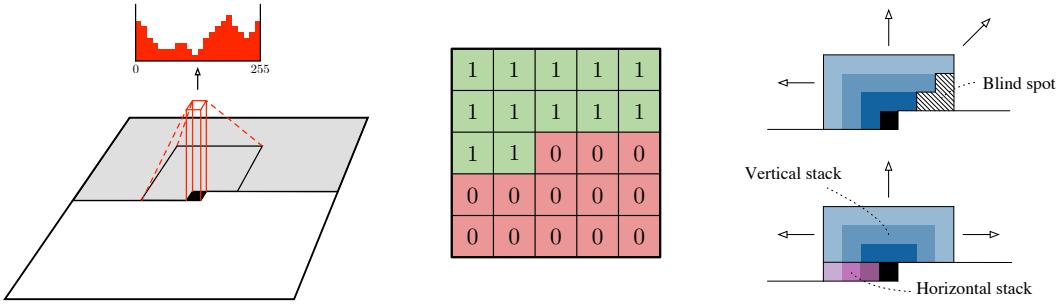


Figure 12: **Left**: A visualization of the PixelCNN that maps a neighborhood of pixels to prediction for the next pixel. To generate pixel x_i the model can only condition on the previously generated pixels $x_1, \ldots x_{i-1}$. **Middle**: an example matrix that is used to mask the 5x5 filters to make sure the model cannot read pixels below (or strictly to the right) of the current pixel to make its predictions. **Right**: Top: PixelCNNs have a *blind spot* in the receptive field that can not be used to make predictions. Bottom: Two convolutional stacks (blue and purple) allow to capture the whole receptive field. (Figure was taken from [51]).

There were several problems with PixelCNN. Sampling from this model is slow because you need to generate pixels one by one. Secondly, the quality of the PixelCNN model was worse that similar PixelRNN model. The reason for this was blind spots caused by the usage of masked convolutions.

3.3.1 Speed of the model

Several different modifications to the model architecture were proposed to speed up the PixelCNN model. Salimans et al. [60] proposed to use a discretized logistic mixture model [66] to speed up training. This model allows using information that one value of pixel intensity is close to other values. The authors also proposed to use downsampling with skip connections similar to U-net architecture arguing that it is faster than the usage of dilated convolution [34, 49] whereas their performances are the same. Later Ramachandran et al. [53] used caching to speed up PixelCNN++ [60] in 182 times.

3.3.2 Dealing with blind spots

The usage of stacked masked convolutions causes the significant portion of pixels to be ignored. Oord et al. [51] solved this problem by substituting 2D convolution by combination if two vertical and horizontal convolutions (see Figure 12). Vertical stack conditions on all the rows above, whereas horizontal stack conditions on the current row. Later the information from both stacks is combined in the special way to avoid conditioning on the future pixels. This modification in combination with the gated convolutional layers improved performance of PixelCNN model made it comparable with PixelRNN model.

4 Task Specific structural elements

4.1 Image-to-image translations

We can formulate many problems in computer vision, computer graphics and image processing as a translation of input image to the output image. For example, super resolution that is the translation of low-quality image to the realistic high-quality picture and colorization that is the transformation of grayscale image to the RGB image. Often such translations are one-to-many: for one input there is the distribution of outputs (e.g. in image synthesis for one segmentation map there are a lot of different possible images). In this situation, we want to generate one of the possible output images from a generative model conditioned to the input image. Here we consider the structural elements that use the information about an input image in the most efficient way.

4.1.1 Skip connections in image-to-image conditional generative models

There are many image-to-image translations where the structure in the output is roughly aligned with the structure in the output. In this case, we often want to encode some efficient and compact representation of the input and in the same time allow the generator to use the structure of the input. U-net architecture [57] is the implementation of these two ideas. It contains skip connection between each i layer to the $n - i$ layer, where n is the total number of layers. In Figure 13 we show the main difference between encoder-decoder network and U-net. Such skip connection were used to solve many different tasks [31, 58, 80, 33, 60]. Isola el al. [31] showed that skip connection is an important element of their model architecture (see Figure 14)

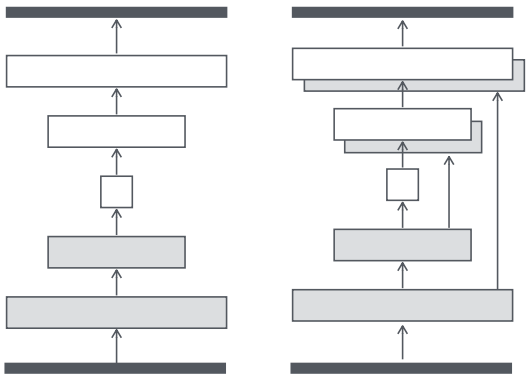


Figure 13: Two choices for the architecture of the generator. The “U-Net" [57] is an encoder-decoder with skip connections between mirrored layers in the encoder and decoder stacks. (Figure was taken from [31])

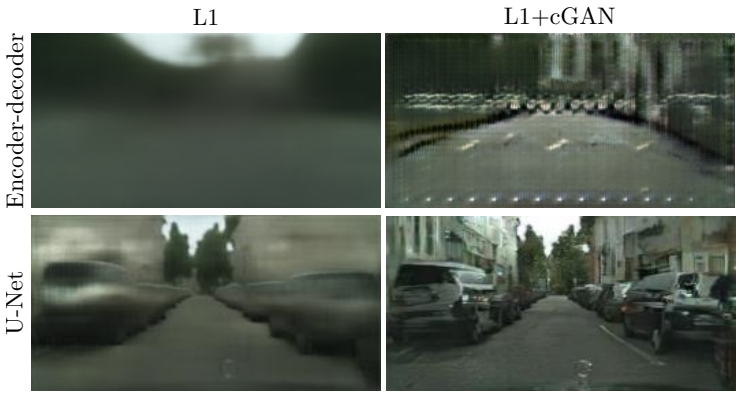


Figure 14: Adding skip connections to an encoder-decoder to create a “U-Net" results in much higher quality results. (Figure was taken from [31])

4.1.2 Residual Connections

Residual Connections [25] were first used to construct much deeper neural networks. Such networks were more accurate for image recognition tasks. The main idea of the residual connections is the addition of the input image to the output of the convolutional module (see figure 15). In this way, convolutional layer learns transformation to the residuals with reference to the layer inputs.
Residual connections were often used in generative networks for super resolution task [42, 33]. However, in super resolution models they were used when the convolution transformation preserves input size and the number of feature maps, which is not the case for most generator networks where we gradually increase the input size to the output image size. Wojna et al. [71] adopt residual connections to the upsampling layer. They proposed to apply an additional upsampling operation without convolutional layer to the input to create the "identity" input image that contains the same information as the input. Then they add the upsampled input to the output of upsampling+convolution layer to allow the convolution to learn only residual function with reference to the upsampled input (see Figure 16). They compared the model results with such residual like connections showing that it performs better for most of the tasks and upsampling methods.

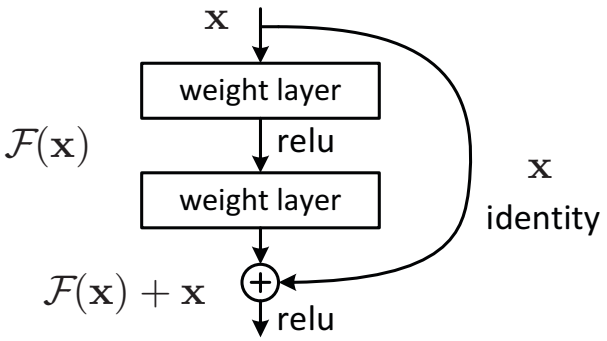


Figure 15: Residual learning: a building block. (Figure was taken from [25])

4.1.3 Dilated Convolutions

To find the best mapping from one image to another it is important to process image on different scales. The commonly used approach was to use many convolutional layers such that receptive fields of deeper layers were broader and feature maps more high level. However, such approach is computationally expansive. An alternative way of expanding receptive fields is the usage of dilated convolutions. Yu and Koltun [77] showed that incorporating of dilated convolutions to the network architecture simplifies multi-scale context aggregation. Their context aggregation network with

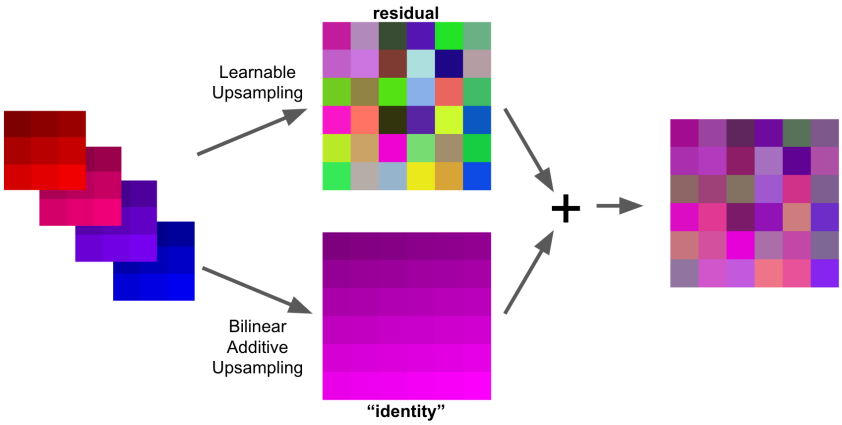


Figure 16: Residual connection for upsampling layer with Bilinear Additive Upsampling layer. (Figure was taken from [71])

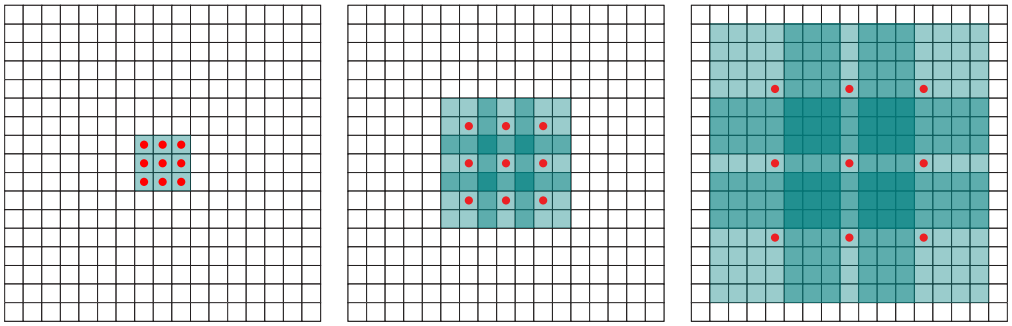
dilated convolutions achieved state-of-the-art accuracy in semantic segmentation.
Dilated convolution is the generalization of convolution operator that supports the exponential expansion of the receptive field without loss of resolution. To see the main difference with convolution operator let us compare their definitions. The discrete convolution operator $*$ can be defined as

$$(F * k)(\mathbf{p}) = \sum_{\mathbf{s}+\mathbf{t}=\mathbf{p}} F(\mathbf{s})\, k(\mathbf{t}). \tag{8}$$

Let l be a dilation factor and let $*_l$ be defined as

$$(F *_l k)(\mathbf{p}) = \sum_{\mathbf{s}+l\mathbf{t}=\mathbf{p}} F(\mathbf{s})\, k(\mathbf{t}). \tag{9}$$

Thus the discrete convolution $*$ is simply the 1-dilated convolution. Look at Figure 17 to see how we use dilated convolutions to achieve an exponential expansion of receptive field.



(a) (b) (c)

Figure 17: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) 1-dilated convolution; each element in F_1 has a receptive field of 3×3. (b) 2-dilated convolution; each element in F_2 has a receptive field of 7×7. (c) 4-dilated convolution; each element in F_3 has a receptive field of 15×15. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly. (Figure was taken from [77])

Later Chen et al. [5] used dilated convolutions in context aggregation networks (CANs) to develop the fast and general approximation to many image processing operators. CAN architecture consists of several dilated convolution layers with exponentially bigger dilations and receptive fields (see Figure 20 in Supplementary materials). The authors showed that dilated convolutions are crucial to achieve fast and accurate image to image translation. In addition, Yang et al. [75] use dilated convolutions in the decoder of VAE architecture to control the effective context from previously generated words. They first time showed that VAE can outperform LSTM language models.

5 Suggestion for the future architecture modifications

5.1 Self-normalizing neural nets

Klambauer et al. [39] show that it is possible to change non-linearity used in a neural network such that the resulting network will be self-normalized without batch normalization. The SELU activation function that is needed for self-normalisation is given by

$$\text{SELU}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leq 0 \end{cases} . \tag{10}$$

Usage of SELU as activation function for generator networks could be useful to speed up computations, however, more experiments are needed to find out how effective such networks are in stabilization of generative models.

5.2 Gated activation functions

The idea to scale the activations in one layer by the coefficient that also depends on the parameters of the network came from LSTM architecture. There are several multiplicative units that allow LSTM layer to model more complex interactions. This idea was used in PixelRNN [50] and later adopted to PixtelCNN architecture [9]. The gated activation function for the convolutional layer can be written as

$$\mathbf{y} = \tanh(W_{k,f} * \mathbf{x}) \odot \sigma(W_{k,g} * \mathbf{x}), \tag{11}$$

The authors claimed that this change in combination with usage of convolutional stacks highly improved the log-likelihood of the model. In addition, a similar gating structure is useful for improving the supervised image classification task as was shown by the winners of ImageNet 2017 competition[27]. Their Squeeze-and-Excitation module adds additional learnable gating factors for every feature map (see Figure 18) making some feature maps more important than others. However,

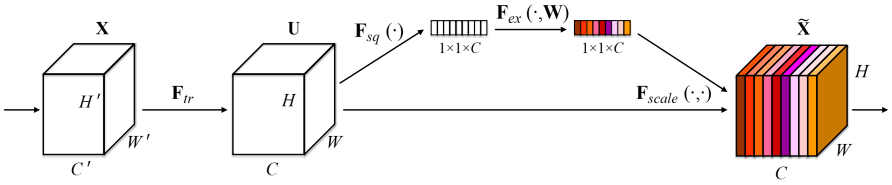


Figure 18: A Squeeze-and-Excitation module. Colours represent different weights. (Figure was taken from [27])

we have not seen the usage of gated activation functions in other generative models (e.g. VAEs or GANs) so we think it is good to check how gated activation functions will effect performance of other generative networks.

Conclusion

Image generative models are important for solving many different tasks such as unsupervised learning or image-to-image translations. The architecture of the generator effects learning of the whole model and the quality of the generated images. There are many ways to make the architecture of the model better. Some of them modify the layer composition of the network, whereas other combine several networks to make generation of high-resolution images easier. Most of the modifications are useful for many different generative models such as GAN, VAE and PixelCNN. However, there are several structural changes that are specific to the model type. In addition, there are some architecture modifications that are shown to be useful in other deep learning tasks. In image generation, the efficiency of these modifications should be checked in the future.

References

[1] M. Arjovsky and L. Bottou. Towards Principled Methods for Training Generative Adversarial Networks. *arXiv:1701.04862 [cs, stat]*, Jan. 2017. arXiv: 1701.04862. 3.1.1

[2] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein GAN. *arXiv:1701.07875 [cs, stat]*, Jan. 2017. arXiv: 1701.07875. 1, 3.1.1

[3] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio. Generating Sentences from a Continuous Space. *arXiv:1511.06349 [cs]*, Nov. 2015. arXiv: 1511.06349. 3.2.1

[4] Y. Burda, R. Grosse, and R. Salakhutdinov. Importance Weighted Autoencoders. *arXiv:1509.00519 [cs, stat]*, Sept. 2015. arXiv: 1509.00519. 1, 3.2.1

[5] Q. Chen and V. Koltun. Photographic Image Synthesis with Cascaded Refinement Networks. *arXiv:1707.09405 [cs]*, July 2017. arXiv: 1707.09405. 1, 2.5, 4.1.3, **??**, 19

[6] Q. Chen, J. Xu, and V. Koltun. Fast Image Processing with Fully-Convolutional Networks. *arXiv:1709.00643 [cs]*, Sept. 2017. arXiv: 1709.00643. 20

[7] F. Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. *arXiv:1610.02357 [cs]*, Oct. 2016. arXiv: 1610.02357. 2.1, 2.1.1

[8] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *arXiv:1511.07289 [cs]*, Nov. 2015. arXiv: 1511.07289. 5, 2.2.3

[9] R. Dahl, M. Norouzi, and J. Shlens. Pixel Recursive Super Resolution. *arXiv:1702.00783 [cs]*, Feb. 2017. arXiv: 1702.00783. 1, 5.2

[10] L. de Oliveira, M. Paganini, and B. Nachman. Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis. *arXiv:1701.05927 [hep-ex, physics:physics, stat]*, Jan. 2017. arXiv: 1701.05927. 1

[11] E. Denton, S. Chintala, A. Szlam, and R. Fergus. Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks. *arXiv:1506.05751 [cs]*, June 2015. arXiv: 1506.05751. 2.5, 9

[12] A. Dosovitskiy and T. Brox. Generating Images with Perceptual Similarity Metrics based on Deep Networks. *arXiv:1602.02644 [cs]*, Feb. 2016. arXiv: 1602.02644. 1, 2.1.2

[13] A. Dosovitskiy, J. T. Springenberg, M. Tatarchenko, and T. Brox. Learning to Generate Chairs, Tables and Cars with Convolutional Networks. *arXiv:1411.5928 [cs]*, Nov. 2014. arXiv: 1411.5928. 2.1.2, 4

[14] V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. *arXiv:1603.07285 [cs, stat]*, Mar. 2016. arXiv: 1603.07285. 2.1.1, 1

[15] S. M. A. Eslami, N. Heess, T. Weber, Y. Tassa, D. Szepesvari, K. Kavukcuoglu, and G. E. Hinton. Attend, Infer, Repeat: Fast Scene Understanding with Generative Models. *arXiv:1603.08575 [cs]*, Mar. 2016. arXiv: 1603.08575. 2.4.1

[16] C. Finn, I. Goodfellow, and S. Levine. Unsupervised Learning for Physical Interaction through Video Prediction. *arXiv:1605.07157 [cs]*, May 2016. arXiv: 1605.07157. 1

[17] L. A. Gatys, A. S. Ecker, and M. Bethge. A Neural Algorithm of Artistic Style. *arXiv:1508.06576 [cs, q-bio]*, Aug. 2015. arXiv: 1508.06576. 1

[18] L. A. Gatys, A. S. Ecker, and M. Bethge. Texture and art with deep neural networks. *Current Opinion in Neurobiology*, 46:178–186, Oct. 2017. 1

[19] I. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016. 2.3.1

[20] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. 1

[21] I. J. Goodfellow. On distinguishability criteria for estimating generative models. *arXiv:1412.6515 [stat]*, Dec. 2014. arXiv: 1412.6515. 3.1

[22] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. *arXiv:1406.2661 [cs, stat]*, June 2014. arXiv: 1406.2661. 1, 3.1

[23] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra. DRAW: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015. 2.4, 6

[24] I. Gulrajani, K. Kumar, F. Ahmed, A. A. Taiga, F. Visin, D. Vazquez, and A. Courville. PixelVAE: A Latent Variable Model for Natural Images. *arXiv:1611.05013 [cs]*, Nov. 2016. arXiv: 1611.05013. 1

[25] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, Dec. 2015. arXiv: 1512.03385. 2.1, 4.1.2, 15

[26] K. He, X. Zhang, S. Ren, and J. Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *arXiv:1502.01852 [cs]*, Feb. 2015. arXiv: 1502.01852. 2.2.2

[27] J. Hu, L. Shen, and G. Sun. Squeeze-and-Excitation Networks. *arXiv:1709.01507 [cs]*, Sept. 2017. arXiv: 1709.01507. 2.1, 5.2, 18

[28] X. Huang and S. Belongie. Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization. *arXiv:1703.06868 [cs]*, Mar. 2017. arXiv: 1703.06868. 2.3.2

[29] S. Ioffe. Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models. *arXiv:1702.03275 [cs]*, Feb. 2017. arXiv: 1702.03275. 2.3.1

[30] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]*, Feb. 2015. arXiv: 1502.03167. 2.3.1

[31] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-Image Translation with Conditional Adversarial Networks. *arXiv:1611.07004 [cs]*, Nov. 2016. arXiv: 1611.07004. 4.1.1, 13, 14, **??**

[32] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial Transformer Networks. *arXiv:1506.02025 [cs]*, June 2015. arXiv: 1506.02025. 2.4, 8

[33] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. *arXiv:1603.08155 [cs]*, Mar. 2016. arXiv: 1603.08155. 1, 2.1, 2.2.1, 2.3.2, 2.5, 4.1.1, 4.1.2

[34] N. Kalchbrenner, A. v. d. Oord, K. Simonyan, I. Danihelka, O. Vinyals, A. Graves, and K. Kavukcuoglu. Video Pixel Networks. *arXiv:1610.00527 [cs]*, Oct. 2016. arXiv: 1610.00527. 3.3.1

[35] D. P. Kingma, D. J. Rezende, S. Mohamed, and M. Welling. Semi-Supervised Learning with Deep Generative Models. *arXiv:1406.5298 [cs, stat]*, June 2014. arXiv: 1406.5298. 1

[36] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling. Improving Variational Inference with Inverse Autoregressive Flow. *arXiv:1606.04934 [cs, stat]*, June 2016. arXiv: 1606.04934. 3.2.1

[37] D. P. Kingma, T. Salimans, and M. Welling. Variational Dropout and the Local Reparameterization Trick. *arXiv:1506.02557 [cs, stat]*, June 2015. arXiv: 1506.02557. 3.2

[38] D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*, Dec. 2013. arXiv: 1312.6114. 1

[39] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-Normalizing Neural Networks. *arXiv:1706.02515 [cs, stat]*, June 2017. arXiv: 1706.02515. 5.1

[40] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 2.2.1

[41] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum. Deep convolutional inverse graphics network. In *Advances in Neural Information Processing Systems*, pages 2539–2547, 2015. 2.1.2

[42] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. *arXiv:1609.04802 [cs, stat]*, Sept. 2016. arXiv: 1609.04802. 1, 4.1.2

[43] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013. 2.2.2

[44] A. Makhzani and B. Frey. PixelGAN Autoencoders. *arXiv:1706.00531 [cs]*, June 2017. arXiv: 1706.00531. 1

[45] L. Mescheder, S. Nowozin, and A. Geiger. Adversarial Variational Bayes: Unifying Variational Autoencoders and Generative Adversarial Networks. *arXiv:1701.04722 [cs]*, Jan. 2017. arXiv: 1701.04722. 1, 2.2.3

[46] A. Nguyen, J. Clune, Y. Bengio, A. Dosovitskiy, and J. Yosinski. Plug & Play Generative Networks: Conditional Iterative Generation of Images in Latent Space. *arXiv:1612.00005 [cs]*, Nov. 2016. arXiv: 1612.00005. 1, 2.1.2

[47] A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. *arXiv:1605.09304 [cs]*, May 2016. arXiv: 1605.09304. 2.1.2

[48] A. Odena, V. Dumoulin, and C. Olah. Deconvolution and Checkerboard Artifacts. *Distill*, 1(10), Oct. 2016. 2.1.2, 2

[49] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. *arXiv:1609.03499 [cs]*, Sept. 2016. arXiv: 1609.03499. 3.3.1

[50] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel Recurrent Neural Networks. *arXiv:1601.06759 [cs]*, Jan. 2016. arXiv: 1601.06759. 1, 2.1, 3.3, 5.2

[51] A. v. d. Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu. Conditional Image Generation with PixelCNN Decoders. *arXiv:1606.05328 [cs]*, June 2016. arXiv: 1606.05328. 12, 3.3.2

[52] A. Radford, L. Metz, and S. Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *arXiv:1511.06434 [cs]*, Nov. 2015. arXiv: 1511.06434. 1, 2.1, 3, 2.2.1, 2.3.1

[53] P. Ramachandran, T. L. Paine, P. Khorrami, M. Babaeizadeh, S. Chang, Y. Zhang, M. A. Hasegawa-Johnson, R. H. Campbell, and T. S. Huang. Fast Generation for Convolutional Autoregressive Models.

arXiv:1704.06001 [cs, stat], Apr. 2017. arXiv: 1704.06001. 3.3.1

[54] S. E. Reed, Z. Akata, S. Mohan, S. Tenka, B. Schiele, and H. Lee. Learning what and where to draw. In *Advances in Neural Information Processing Systems*, pages 217–225, 2016. 1, 2.4.1

[55] D. J. Rezende, S. Mohamed, I. Danihelka, K. Gregor, and D. Wierstra. One-Shot Generalization in Deep Generative Models. *arXiv:1603.05106 [cs, stat]*, Mar. 2016. arXiv: 1603.05106. 2.4, 2.4.1

[56] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. *arXiv:1401.4082 [cs, stat]*, Jan. 2014. arXiv: 1401.4082. 1, 3.2

[57] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. *arXiv:1505.04597 [cs]*, May 2015. arXiv: 1505.04597. 4.1.1, 13

[58] M. S. M. Sajjadi, B. Schölkopf, and M. Hirsch. EnhanceNet: Single Image Super-Resolution Through Automated Texture Synthesis. *arXiv:1612.07919 [cs]*, Dec. 2016. arXiv: 1612.07919. 1, 2.1.2, 4.1.1

[59] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved Techniques for Training GANs. *arXiv:1606.03498 [cs]*, June 2016. arXiv: 1606.03498. 1, 2.3.1, 3.1.1, 3.1.1

[60] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications. *arXiv:1701.05517 [cs, stat]*, Jan. 2017. arXiv: 1701.05517. 1, 3.3.1, 4.1.1

[61] S. Santurkar, D. Budden, and N. Shavit. Generative Compression. *arXiv:1703.01467 [cs]*, Mar. 2017. arXiv: 1703.01467. 1

[62] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang. Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network. *arXiv:1609.05158 [cs, stat]*, Sept. 2016. arXiv: 1609.05158. 2.1.2

[63] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for Simplicity: The All Convolutional Net. *arXiv:1412.6806 [cs]*, Dec. 2014. arXiv: 1412.6806. 2.1

[64] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception Architecture for Computer Vision. *arXiv:1512.00567 [cs]*, Dec. 2015. arXiv: 1512.00567. 2.1, 2.1.1

[65] C. K. Sønderby, J. Caballero, L. Theis, W. Shi, and F. Huszár. Amortised MAP Inference for Image Super-resolution. *arXiv:1610.04490 [cs, stat]*, Oct. 2016. arXiv: 1610.04490. 3.1.1

[66] L. Theis, R. Hosseini, and M. Bethge. Mixtures of conditional Gaussian scale mixtures applied to multiscale image representations. *PLoS ONE*, 7(7):e39857, July 2012. arXiv: 1109.4389. 3.3.1

[67] L. Theis, A. v. d. Oord, and M. Bethge. A note on the evaluation of generative models. *arXiv:1511.01844 [cs, stat]*, Nov. 2015. arXiv: 1511.01844. 1

[68] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. Lempitsky. Texture Networks: Feed-forward Synthesis of Textures and Stylized Images. *arXiv:1603.03417 [cs]*, Mar. 2016. arXiv: 1603.03417. 1, 2.3.2, 2.5

[69] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance Normalization: The Missing Ingredient for Fast Stylization. *arXiv:1607.08022 [cs]*, July 2016. arXiv: 1607.08022. 2.3.2

[70] X. Wang and A. Gupta. Generative Image Modeling using Style and Structure Adversarial Networks. *arXiv:1603.05631 [cs]*, Mar. 2016. arXiv: 1603.05631. 2.5, 10

[71] Z. Wojna, V. Ferrari, S. Guadarrama, N. Silberman, L.-C. Chen, A. Fathi, and J. Uijlings. The Devil is in the Decoder. *arXiv:1707.05847 [cs]*, July 2017. arXiv: 1707.05847. 2.1, 2.1.1, 2.1.2, 4.1.2, 16

[72] B. Xu, N. Wang, T. Chen, and M. Li. Empirical Evaluation of Rectified Activations in Convolutional Network. *arXiv:1505.00853 [cs, stat]*, May 2015. arXiv: 1505.00853. 2.2.2

[73] X. Yan, J. Yang, K. Sohn, and H. Lee. Attribute2image: Conditional Image Generation from Visual Attributes. *arXiv:1512.00570 [cs]*, Dec. 2015. arXiv: 1512.00570. 2.4, 7

[74] C. Yang, X. Lu, Z. Lin, E. Shechtman, O. Wang, and H. Li. High-Resolution Image Inpainting using Multi-Scale Neural Patch Synthesis. *arXiv:1611.09969 [cs]*, Nov. 2016. arXiv: 1611.09969. 1, 2.2.3

[75] Z. Yang, Z. Hu, R. Salakhutdinov, and T. Berg-Kirkpatrick. Improved Variational Autoencoders for Text Modeling using Dilated Convolutions. *arXiv:1702.08139 [cs]*, Feb. 2017. arXiv: 1702.08139. 4.1.3

[76] S. Yeung, A. Kannan, Y. Dauphin, and L. Fei-Fei. Tackling Over-pruning in Variational Autoencoders. *arXiv:1706.03643 [cs]*, June 2017. arXiv: 1706.03643. 3.2.1, 11

[77] F. Yu and V. Koltun. Multi-Scale Context Aggregation by Dilated Convolutions. *arXiv:1511.07122 [cs]*, Nov. 2015. arXiv: 1511.07122. 4.1.3, 17

[78] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. Metaxas. StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks. *arXiv:1612.03242 [cs, stat]*, Dec. 2016. arXiv: 1612.03242. 1, 2.5

[79] J. Zhao, M. Mathieu, and Y. LeCun. Energy-based Generative Adversarial Network. *arXiv:1609.03126 [cs, stat]*, Sept. 2016. arXiv: 1609.03126. 2.2.1, 3.1.1

[80] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. *arXiv:1703.10593 [cs]*, Mar. 2017. arXiv: 1703.10593. 4.1.1

Supplementary materials

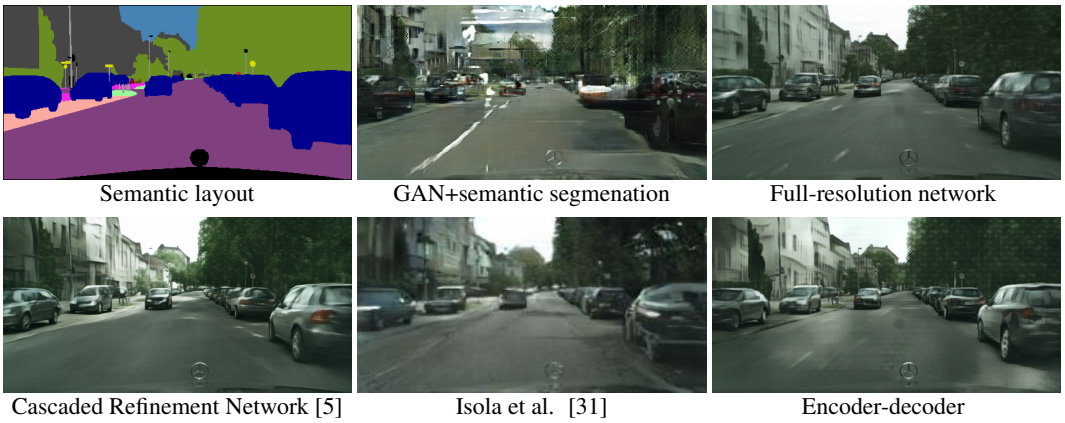


Figure 19: Qualitative comparison of different generative models on the Cityscapes dataset. (Figure was taken from [5])

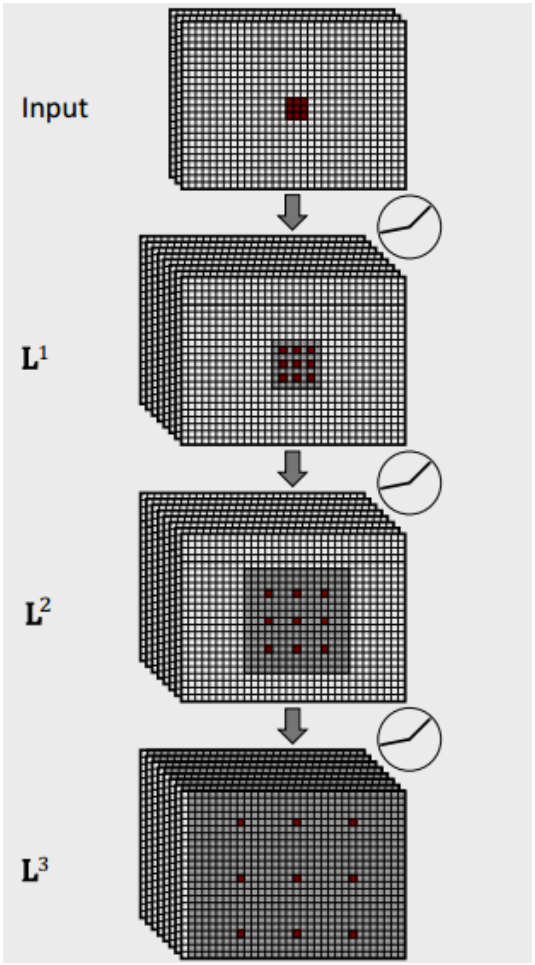


Figure 20: Schematic illustration of several first layers of CAN. The red pixels show the application of dilated convolutions. The shaded gray pixels show the receptive field of a single element. Circles show the nonlinear transformation between layers. (Figure was taken from [6])