Generating and Scoring Cosmology Images

Prashanth Balasubramanian[1], Sara Javadzadeh[2], Simon Buus Jensen[3], and Parijit Kedia [4]

[1,2,3,4]Group: Intergalatic Generators, Department of Computer Science, ETH Zurich

***Abstract*—In this paper we propose a method of generating and scoring cosmological images, using a Variational Auto-Encoder (VAE) and a Convolutional Neural Network (CNN) respectively. We use a dataset that has a varying range of similarity scores to a "prototypical cosmology image" and one with positive and negative examples of cosmology images. With these two datasets we were able to generate thousands of images resembling real galaxies and achieve a score of 0.63485 on the Kaggle leaderboard. The VAE proved to be highly sensitive to parameter tuning, however within a certain range of hyperparameters and training method, the outcome was superior. The results were compared to the output of a Deep Convolutional Generative Adversarial Network [5] (DCGAN) and the CNN far outperformed the DCGAN in learning similarity scores, using the architecture detailed in Section IV.**

I. Introduction

The aim of this project is to perform two sub-tasks using two separate data sets (labeled and scored):
Generate images which resemble a prototypical cosmology image and to estimate the similarity to a prototypical cosmology image for a given set of images.
Deep Convolutional Generative Adversarial Network (DCGAN) [5] was used as a first attempt and the final solution comprised a Variational Auto-Encoder (VAE) and a Convolutional Neural Network (CNN). In the following sections we will elaborate our models and results for each of the proposed methods.

Challenges

Training images are on the order of (1000×1000). This led to a large number of neurons in the first layer of our attempts. Training such networks is time consuming and requires specialized hardware with the added difficulty of adjusting sensitive parameters. On the other hand, each image is extremely sparse. Meaning that the model should capture sparse elements effectively and represent them correctly. This was solved by using a standard definition VAE.

II. Related Work

Recent studies have shown promising results for image completion using multi-layer neural networks[1], continued with conditional image generation capable of training more efficiently by introducing Gated Pixel CNNs [2] ,[3].
Generative adversarial networks (GANs) were shown to be effective for the problem of feature representation in unsupervised learning.[5] The advantage with a GAN is that no heuristic cost function is needed. On the other hand, a disadvantage is that it can be unstable to train and sensitive to parameter tuning.

Here, we developed a separate approach for each sub-task, both of which could be potentially fused.

An impressive image feature extraction approach is introduced as Variational Auto Encoders(VAEs). VAEs claim to distinguish semantic features in contrast to element-wise features. Semantic features tend to identify features that are closer to human visual perception[7]. Extracted semantic features at first phase are embedded in a compressed representation which is trained for all training images. Then, using this latent representation, the decoder network will try to reconstruct the image. Since the latent representation is very low dimensional (compared to original images), it can be interpreted as a lossy compression. Therefore, reconstructed images will never be the same as any of the images in training set, but generated from an 'understanding' of what those images look like.

III. Generating Cosmology Images

Our model for generating cosmology images was inspired by a Composite Pattern Producing Neural Network (CPPN) [4]. However, a VAE proved to produce superior results, with images greatly resembling real galaxies, as shown in 2. A VAE [6] is a type of generative model that uses multiple layers of neurons to encode a distribution of observed and latent (unobserved) variables. In this type of generative modeling, we aim to model a high dimensional space, eg: A cosmology image, as a human readable, low dimensional vector. We assume that our data set N i.i.d. samples of some variable x. In this case, these N samples correspond the number of pixels in a cosmology image. The images provided being 1000x1000 pixels led to the input to the network being 1 million neurons. Since this was infeasible due to a lack of memory, the images were downsized to 128x128 pixels. This provides a probabilistic interpretation of the decoder network, where given a latent variable or 'code' z we generate a sample x in the data space. Similarly, the role of the encoder would be to take a sample x from data space and give us a latent z sampled from the posterior density distribution $p_\theta(z|x)$. The architecture for this is shown in 1.

The outputs of the last layer in 1 are taken to be the mean of a Bernoulli distribution, the variational lower bound for which is given by:

$$\mathcal{L} = \mathbb{E}_{z\sim Q(z|X)} \log P(X|z) - D(Q(z|X)||P(z)) \quad (1)$$

When P is a Bernoulli distribution, the log likelihood is given by

$$\log P(X|z) = \sum_{i}^{N} X^{(i)} \log y^{(i)} + (1X^{(i)}) \cdot \log(1y^{(i)}) \quad (2)$$

where N is the batch size and $y^{(i)}$ is the reconstruction from the latent code $z^{(i)}$. The KL divergence between a gaussian Q with mean μ and standard deviation σ and a standard normal distribution P is given by:

$$D(Q||P) = -\frac{1}{2} \sum_{j}^{J} \left(1 + \log((\sigma_j)^2) - (\mu_j)^2 - (\sigma_j)^2\right) \quad (3)$$

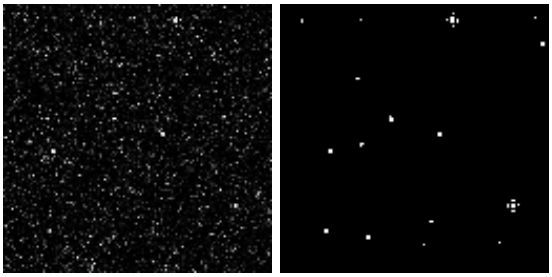

Fig. 2. Generated Cosmology Images - Dense and Sparse

After multiple runs, the architecture used was as follows:

- 128x128 images, i.e., 16384 neurons in the input layer
- 1000 neurons in each hidden layer
- z dimensionality between 60 and 64
- Batch size 64
- 10000 training iterations

The last detail of the VAE to be discussed here is the re-parametrization trick. Since the reconstruction error term is estimated by sampling $z \sim q_\phi(z|x)$, using gradient training methods through the sampling process is problematic.[7] suggests and alternate method, i.e., allowing $z = g_\phi(x, \epsilon)$ to be a deterministic function of ϕ and ϵ be some independent noise. Therefore we let $q_\phi(z|x) = \mathcal{N}(z; \mu, \sigma^2 I)$. So the outputs of our encoder network are the μ and σ of our approximate posterior. Now in order to train with gradient methods, we re-parameterize as described in [7], letting $z = \mu + \sigma\epsilon$ where $\epsilon \sim \mathcal{N}(0, 1)$.

Implementation: The VAE is self contained in a jupyter notebook and the architecture of the network is exactly as described above. After several rounds of trial and error with convolutional VAE's and different activation functions, the encoder/decoder architecture from [**?**] proved to be the most effective. Experimenting with the VAE parameters proved to be challenging as they were extremely sensitive to change.

IV. Scoring Cosmology Images

For scoring the given set of query images we create and train a CNN. The task of predicting scores between 0 and 8 for cosmology images can be categorized as a linear regression task, as the set of possible score values is continuous. In the following section, we describe the procedures taken and the architecture of the CNN. The scored data set is split into a train set (90 % of the images - 8640 images) and a test set (10 % of the images - 960 images) during training. By devoting part of the data to a test set, we make sure the network isn't overfitting the training set.

1) Architecture: The architecture of the CNN is inspired by Aymeric Damien's neural network tensorflow examples [8]. The network consists of the following layers:

- An input layer of size equal to the image height $\times$ image width
- A convolutional layer (A 5×5 kernel, 1 input channel, 32 output channels, stride 1)
- A rectified linear unit activation function (This layer will remove all negative values)
- A max pool layer (A 2×2 kernel. This layer will halve the size of the input image)
- A convolutional layer (A 5×5 kernel, 32 input channel, 64 output channels, stride 1)
- A rectified linear unit activation function (This layer will remove all negative values)
- A max pool layer (A 2×2 kernel. This layer will halve the size of the image further)
- A fully connected layer (100 units)
- A fully connected output layer (1 output unit consisting of the predicted score value)

The size of the input image height and width is during training down scaled to 140×140 pixels. The original size, 1000×1000 pixel, increased the size of the network and the amount of memory required, making it infeasible to perform batch training efficiently. However, the leaderboard scores indicate that by using 140×140 images, the amount of information lost is minimal, as relevant features of cosmology images in this particular dataset are captured.

The loss function is defined as the sum of the absolute value of the differences between the predicted score logits denoted by $\hat{y}$, and the true score labels denoted by y, i.e. :

$$loss(\hat{y}, y) = \sum_{i=0}^{n} ||\hat{y}_i - y_i||$$

The initial values of the weights and biases is set using Xavier initialization [10]. Simply speaking the Xavier initializer uses the number of units in the network's layers to set the weights and biases to relevant initial values. Thereby decreasing the convergences time and increasing the chance of finding a good local minimum.

Implementation: The CNN is written in python 2.7 and uses Tensorflow version 1.1. In order to run the CNN model one simply has to change the path to the cosmology image data directory in the config.py file and run the command:

```
$ python main.py
```

This will begin training the model. Other python library dependencies are:

- Pandas

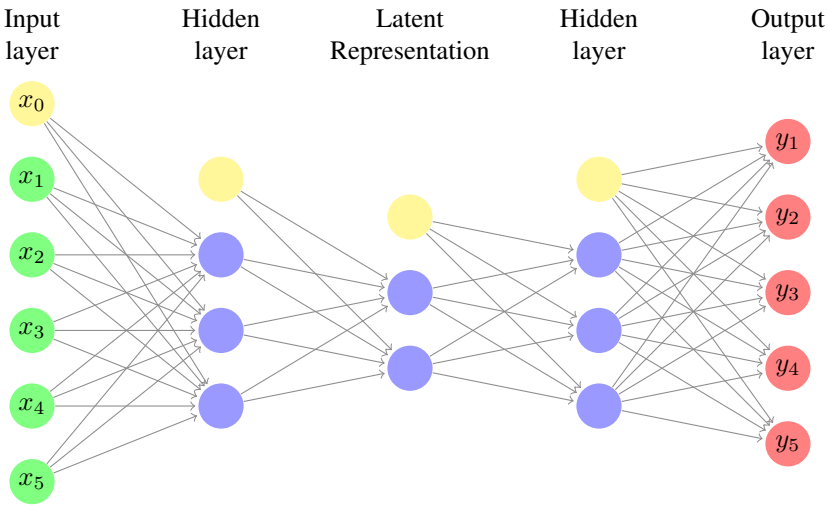

Fig. 1. Variational Auto-Encoder Architecture

- Pickle
- Numpy
- PIL.Image

In the config.py file it is also possible to set the desired down scaled size of the input images, batch size, number of epochs, learning rate or the Adam optimizer etc. While training. The predictions will be outputted to a output folder, which will automatically be created during model start up. A prediction will be every x epoch, where x can be specified in the config.py predict_step variable.

V. Comparison

Deep Convolutional Generative Adversarial networks consists of a generator which learns to produce images of a category given a training set of images of the same category (or generate images conditioned on a class with images of several classes as training set), and a discriminator which learns to distinguish real images from the ones produced by generator. Each network improves at each training step and at the same time challenges the other network. The generator learns to produce more realistic images and the discriminator learns features from images to more accurately distinguish between the target category (here, cosmology) by an output y which denotes the probability that a generated image is a real image. The discriminator network is fed with both real and generated images without prior knowledge. This network is updated at each step, to both, minimize y for generated images and maximize y for real images by maximizing the binary cross entropy[7]:

$$\mathcal{L} = log(Dis(x)) + log(1 - Dis(Gen(z)) \tag{4}$$

Where Dis(x) is the probability that an image x belongs to the real dataset (not generated). Gen(z) is a generated image given a latent z representation. The generator network, on the other hand wants to trick the discriminator into predicting high y values for generated images. Therefore, at each step, the generator network is updated to maximize this y value (discriminator output) for generated images by maximizing the same function. We exploited the GAN to do both sub-tasks in this project. The Generator learns to generate images similar to a real cosmology image and the discriminator additionally generates similarity scores.

However, the scores generated by the DCGAN proved to server as a baseline for improvement. These scores ranked at about 1.2 on the kaggle leaerboard, indicating a poor performance. A few samples of the images generated by the generator are shown in V. It can be seen that the Generator is unable to learn features from this dataset, even after 10000 iterations. Hence, we proceeded to use a VAE and a CNN to fit the needs of this project.

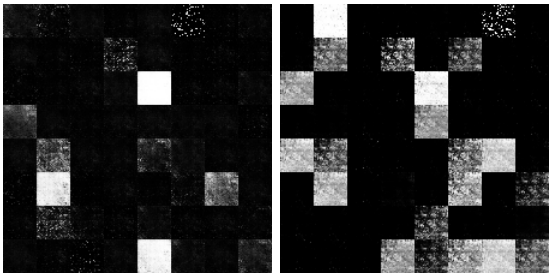

Fig. 3. Cosmology Images generated by DCGAN

VI. Results

During the generation of cosmology images, the VAE proved to be superior to convolutional VAE's or the DCGAN. The model is highly sensitive to parameter tuning, primarily the dimensionality of the latent vector. 4 shows some poorly generated galaxy images when the dimensionality of z was set to 35. The first images were generated using the labeled data alone, i.e., considering 1000 positive examples of galaxy images, followed by more than 2600 images of galaxy images from the scored dataset as well. Using more samples does increase the quality of images generated, however including

negative training samples such as random images, generates a substantial amount of noise in the generated images.

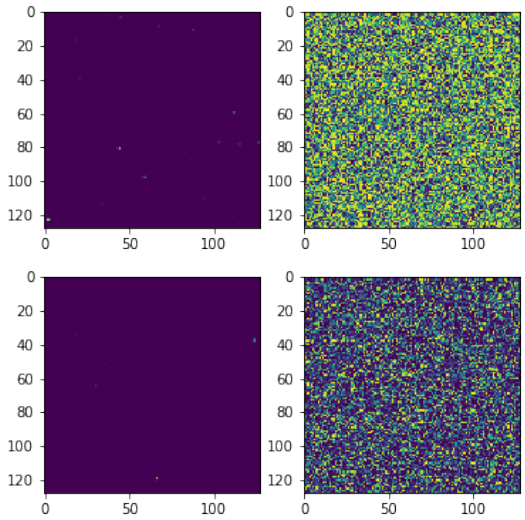

Fig. 4. Examples of parameter sensitivity in the Galaxy VAE. These samples were generated when the hidden dimensionality was 100 and the dimensionality of the z vector was 35

A plot of the log-likelihood, KL Divergence described in Equations 2 and 3 are shown in 5. It can be seen that the KL divergence and the negative variational lower bound that we are trying to minimize (Tensorflow does not support a maximizing optimizer yet), converge around 10000 iterations. The actual loss values reported per 1000 iterations with different latent dimensionalities and number of hidden neurons, was highly sensitive.

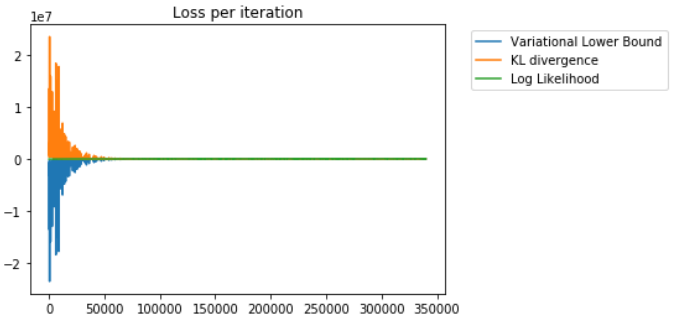

Fig. 5. Loss, KL Divergence and Log Likelihood per 1000 iterations

For the scoring of images, we use Tensorflow's Adam optimizer. The Adam optimizer is a stochastic gradient descent optimizer with an optional decaying momentum parameter. We set the learning rate to 0.0002 and the initial momentum value to 0.5. We also experimented with initializing the values with random values using a Gaussian distribution with 0 mean and 0.01 standard deviation. But we found that even this made the starting weights of the network too large and as the signal grows and the input passes through each layer, the predicted score at the output layer becomes far too large.

VII. Discussion

Utilizing the given labeled data during the training could have yielded benefits while scoring, e.g. giving the labeled data very low scores for non-galaxy images and scores between 2.5 - 6 for galaxy images or by using a semi-supervised learning approach. Another improvement could have been to increase the network size and test whether this would make the network be able to estimate the scoring function even more accurately. In terms of the depth of the network, we believe two convolutional layers are sufficient for scoring images such as the given galaxy images, as they do not contain a lot of distinct features and are very sparse due to a large amount of black pixels. In addition, we envisaged a simple neural network that learns a mapping for 2D (x,y) euclidean pair coordinates to RGB color values, in order to upscale the images from 128x128 to 1000x1000. The code for this is also contained in a jupyter notebook titled $Scale_up.ipynb$. However, due to the lack of variation in color in a galaxy image, the network was unable to learn this function effectively. However, somewhat substantial results were produced for other color images, as can be seen in the notebook. This could potentially be improved, to learn a binary mapping instead of RGB, for galaxy images in particular.

References

[1] Aaron van den Oord, Nal Kalchbrenner and Koray Kavukcuoglu - "Pixel recurrent neural networks"
[2] Tim Salimans, Andrej Karpathy, Xi Chen, Diederik P. Kingma and Yaroslav Bulatov - "PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and other modifications"
[3] Aaron van den Oord et al. - "Conditional Image Generation with PixelCNN Decoders"
[4] Kenneth O. Stanley -"Compositional Pattern Producing Networks: A Novel Abstraction of Development"
[5] Alec Radford et.al. - "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks"
[6] Carl Doersch -"Tutorial on Variational Auto-Encoders"
[7] Kingma et. al. - "Auto-Encoding Variational Bayes"
[8] github: Tensorflow-Examples
[9] Xavier Glorot, Yoshua Bengio - "Understanding the difficulty of training deep feedforward neural networks"
[10] github: Variational Autoencoder
[11] github: Pixel-CNN Model