Team Members: Tholkappian Chidambaram, Robert Maxton, Matthew Zhu, Dushyanth Giridhar, and Abraham Williams

Professor Name: Prof. Shih Yu Chang

Subject Name: CMPE 257 Machine Learning - Summer 2018

Generating RPG Monsters using Variational Autoencoders

Artistic monster figures are common in Role Playing Video Games. These monster figures comes in various shapes and sizes and there is always a need for creating new types of RPG monsters. We are creating new type of RPG Monsters using "Generative Modeling" using variational autoencoders. VAEs are flexible to be used with standard function approximators (neural networks) and can be trained with stochastic gradient descent.

Objective

A representative set of RPG Monster images are downloaded from RPG repositories. These monsters images are used as a training set. Given the dataset of images, the model creates a new data point that "looks like", it came from the training set. Generated RPG monsters have to be artistic and alien. Error in other image domains are considered as features here. In other words, given some space X of possible points (space of all pixel points in mxn images), the model finds P(X), the real-world distribution on X, and draw from that distribution.

Latent Variables

Pixel points in the training set X is high-dimensional, that results in P(X) being complicated function on that high dimensional space. Directly computation of P(X) is a non-starter. We are considering P(X) is probabilistically determined by the values of some latent variable vector z, with smaller dimensionality.

Encoding

Dimension of P(X|z) is still bigger than that of X. Theoretically it is possible to sample enough z values and sum P(X|z) P(z). But z is still likely to be large-dimensional for real problems. Also we need to know P(X|z) very accurately to generate good images. To solve this issue we are checking z values that are interesting - likely to produce non-zero values since most values of P(X|z) = 0. Checking non-zero values of z involves calculating reverse conditional distribution P(z|X).

Derivation

Consider $E_{z/Q}$ is the expectation over {z|Q(z)>0}

As we maximize the equation, second term on left converge to 0 and Q(z|X) $\rightarrow$ P(z|X), the true encoding distribution. This is enforced during z/Q. Q is pulled in one way by backprop from training P but pulled another by direct effect of comparison with P(z). As z, we can also make up P(z) - calling it the second normal N(0,I).

Resummarizing

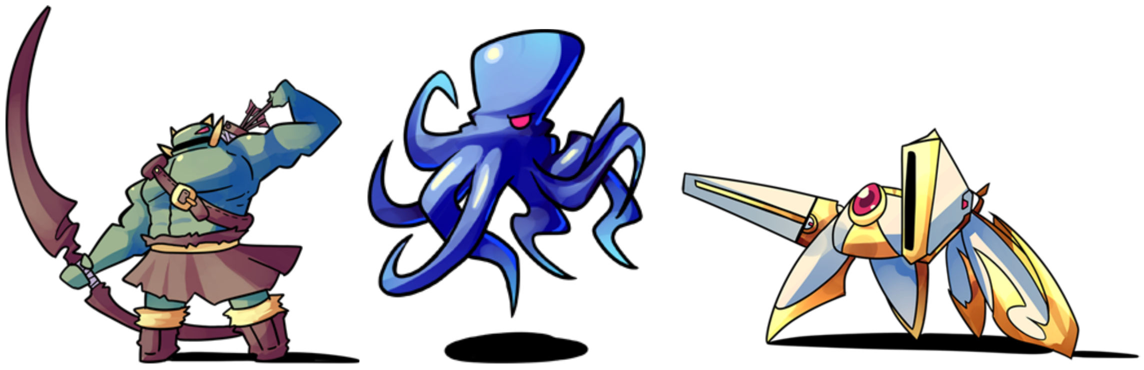
Above equation is implemented in following stages

1. Encoding: Implements Q(z|X), taking points from original domain to latent variable space.
2. Resampling: New points are chosen in latent variable space based on predictions from Q. This is necessary to interpolate between good images.

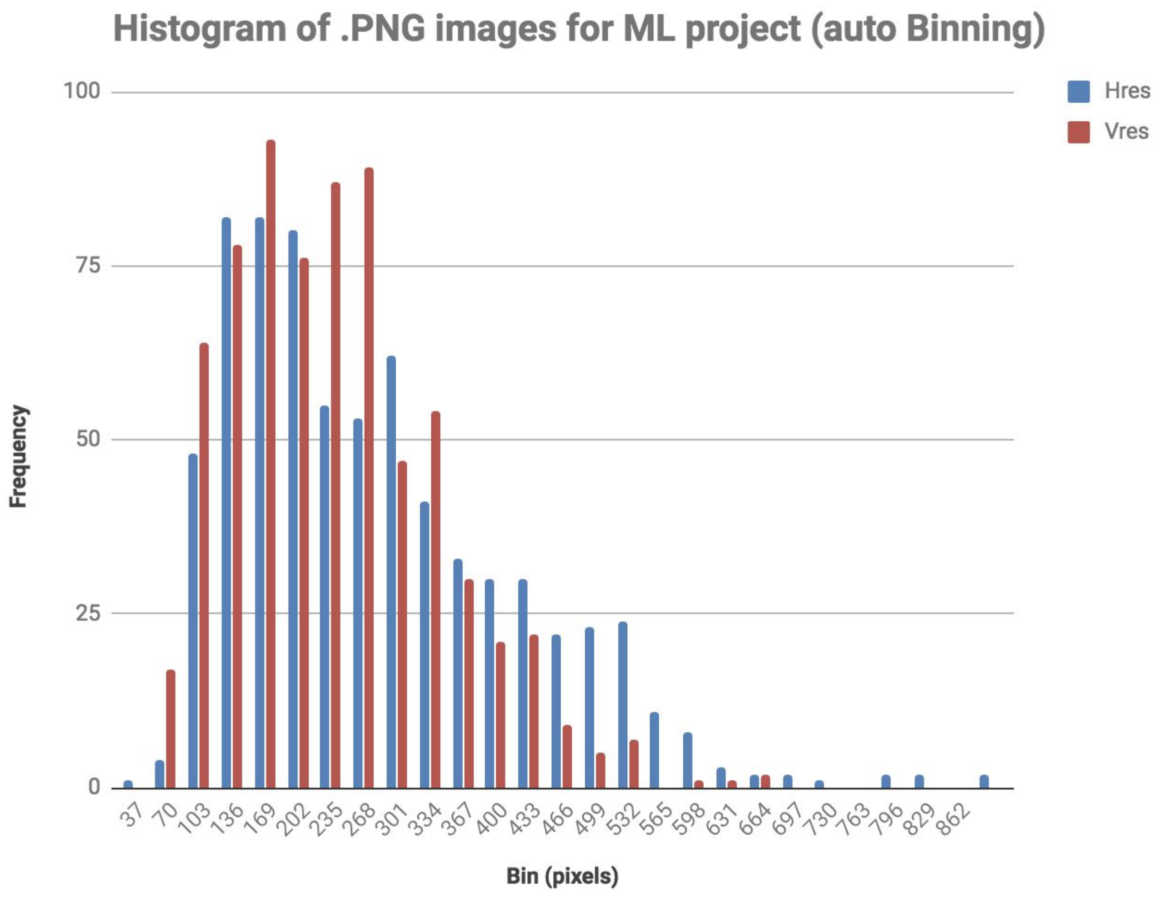
3. Decoding stage: Implements P(X|z) taking points from latent variables back to original domain.
4. Reparametrization: This is to enable backprop across stochastic resampling stage. Instead of choosing at random from distribution as a processing step, take in random values epsilon ε from a standard normal and calculate. This introduces a new input layer.
5. Generation: After generation throw away the encoder and just pick a new sample from standard normal to run through the decoder. P(X|z) tells us that X is most likely given that z, even if that X wasn't in our original dataset, because it has been trained to interpolate.

Training Dataset

The dataset we used for this project was taken, with permission, from the Ækashics Librarium. We were able to gather a total of 703 images in RGBA (red-green-blue-alpha) format, similar to the following:



These images had a variety of resolutions, ranging up to 862 pixels horizontally and 664 pixels vertically. However the bulk of the images were 200 to 300 pixels in each dimension, as shown in the figure below. As a result of this discrepancy in resolution of the dataset, we undertook a few pre-processing steps that are described in more detail in the next section.



Pre-processing Steps

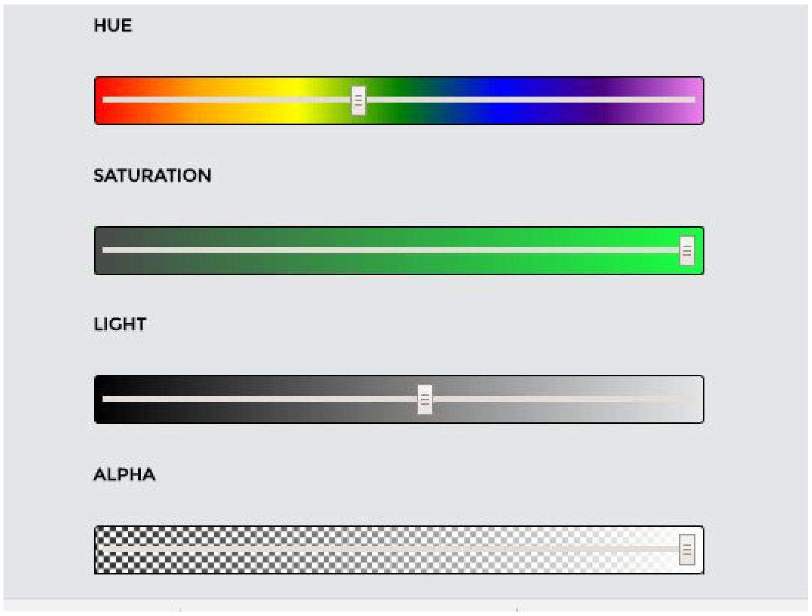
Image Resolution

In order to standardize our dataset, we curated the smaller and larger images that resulted in every image being the same size. We determined that a resolution of 320x480 would be ideal for our dataset, as it allowed us to keep enough detail in the images while also not being too large to work with. The larger images were down sampled, while smaller images were padded with white space. The absolute smallest and largest images were removed from the dataset; otherwise the largest images would lose detail and the smallest images would have a lack of detail at the 320x480.

Color Model

In addition to resolution, we converted the color format of the images. The images were initially in RGBA format, where each pixel had four values (red, green, blue, alpha), where alpha is the opacity. In order to save memory, we wanted to reduce these four values into two by combining values (i.e. $value_1 = red \times 256 + blue$). However if we did this with RGBA, any combination would result in some color being emphasized over another. For example, combining red and blue ($value_1 = red \times 256 + blue$) would give a greater weight to red.

Thus, we chose to convert the images to HSLA format, where each pixel was represented by four values - hue, saturation, lightness, alpha. In order to visually understand why this format is optimal, the scales of each property is shown in the figure below.



After conversion, we encoded these four properties into the following two values:

Hue-Lightness	$\rightarrow$	$value_1 = hue \times 256 + lightness$
Saturation-Alpha	$\rightarrow$	$value_2 = saturation \times 256 + alpha$

The result from the RGBA to HSLA conversion and the encoding is the four properties being represented in two values, where hue and saturation are given a larger weight than lightness and alpha.

<u>Pre-processed training dataset</u>

The final dataset, after standardizing image resolution and color format conversion/encoding, is available through the following link:

https://storage.googleapis.com/mobgen_training_data/train.csv

Model

In this generative model, input layer is the pre-processed image. There are arbitrary n hidden layers that exponentially compresses the image at each step. Image is reduced to latent dimensions for minimum complexity representation. Most of the complexity of the image is contained within its edges. This complexity in the edges is reduced by modeling the edges of the image as "fractals". Fractal dimension was a hyper-parameter of the model. We have set the initial value for this hyper-parameter as 1.25.

Total LATENT_DIM = $2 * (monster\ height)^{fractal\ dim}$

Where monster height = image length since each image is rotated 90 degrees.

Output of the encoder is then decoded for hidden layers in reverse. Measured loss by log-likelihood and KL divergence of the initial and final images are considered as probability distributions. Output of the model is a image in HSLA format which need to converted back to RGBA to maintain consistency with the original training images. Model tuning parameters are provided in the table below.

Parameter	Initial value
IMG_WIDTH	480
IMG_HEIGHT	320
EDGE_FRAC_DIM	1.25
BATCH_SIZE	100
EPOCHS	100
HIDDEN_LAYERS	4
LAYER_DECAY	1.0
ACTIVATION	selu
OPTIMIZER	'adadelta'
EARLY_STOPPING	10
TERMINATE_ON_NAN	False
CHECKDIR	''
PERIOD	10
LOGDIR	''

Google cloud setup

We used Google Clouds' ml-engine to train our model with batch size of 100, running for 50 epochs. Sample invocation command for the training invocation is given below. Input parameters are generated using docopt python library. Models are stored in hdf5 format for efficient I/O processing in the Google cloud. Activation function SELU was used for the model to have model stability.

```
gcloud ml-engine jobs submit training mobgen_mnist_92002

--config "config\config.yaml"
```

```
--package-path .\trainer\
--module-name trainer.task
--region us-east1
--job-dir gs://mobgen_model/out/jobs.hdf5
--staging-bucket gs://mobgen_model
-- train gs://mobgen_training_data/train_mnist.csv 28
     --test_split 0.1
     --check_points
     --img_width 28
     --checkpoints
     gs://mobgen_training_data/out/checkpoints/chk_{epoch:02d}-{
     val_loss:.2f}.hdf5
     --log_dir gs://mobgen_training_data/logs/
     --period 10
     --frac_dim 1.25
     --batch_size 100
     --epochs 50
     --H 6
     --te 15
```

References

Doersch, Carl. *Tutorial on Variational Autoencoders*, 2016.
https://arxiv.org/pdf/1606.05908.pdf.

Various. *RPG Monster images* 2018. http://www.akashics.moe/.