Objectives

1. Classification - Transfer Learning

2. Generative Model – Experiment with Autoencoders

Image Data Set

Dog breed classes: 120

Training Image numbers: 327,000

Test Image numbers: 10,157

Cropped Image Dimensions: 299 x 299 x 3

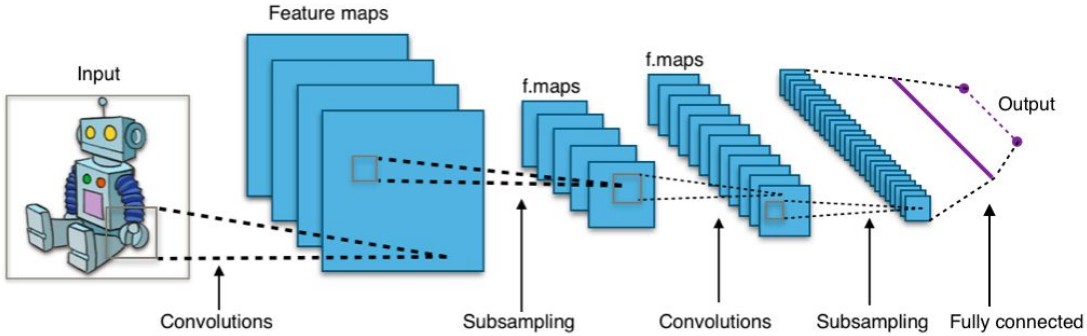
1. Classification

Convolutional Neural Network

Inspired by the visual cortex

Applies a convolution operation to the input, passing the result to the next layer

Share weights in convolutional layers where the same filter is used for each receptive field



1. Classification

Evaluation Metric: Multi Class Log Loss

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j}\log(p_{i,j})$$

Model: Inception v3

- Pre-trained on ImageNet for 1000 different categories
- Smallest out of the pre-trained models at 96 MB
- Batch normalisation – Normalises activations at each layer, which reduces sensitivity to changes of weights in the network
- RELU activation functions – Eliminates vanishing gradient problem, and is computationally efficient

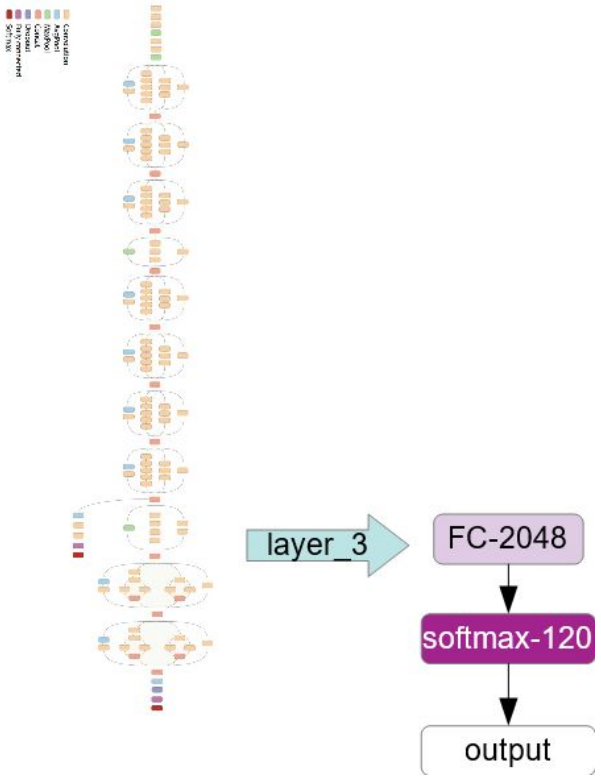
1. Classification

Transfer learning

Storing knowledge gained in one model while solving one problem and applying it to a different but related problem.

Model Architecture:

- Input to final Layer: [None, 2048]
- Weights: [2048 x 120]
- Softmax Layer: [120]
- Cost function: softmax_cross_entropy_with_logits
- Optimizer: Gradient Descent



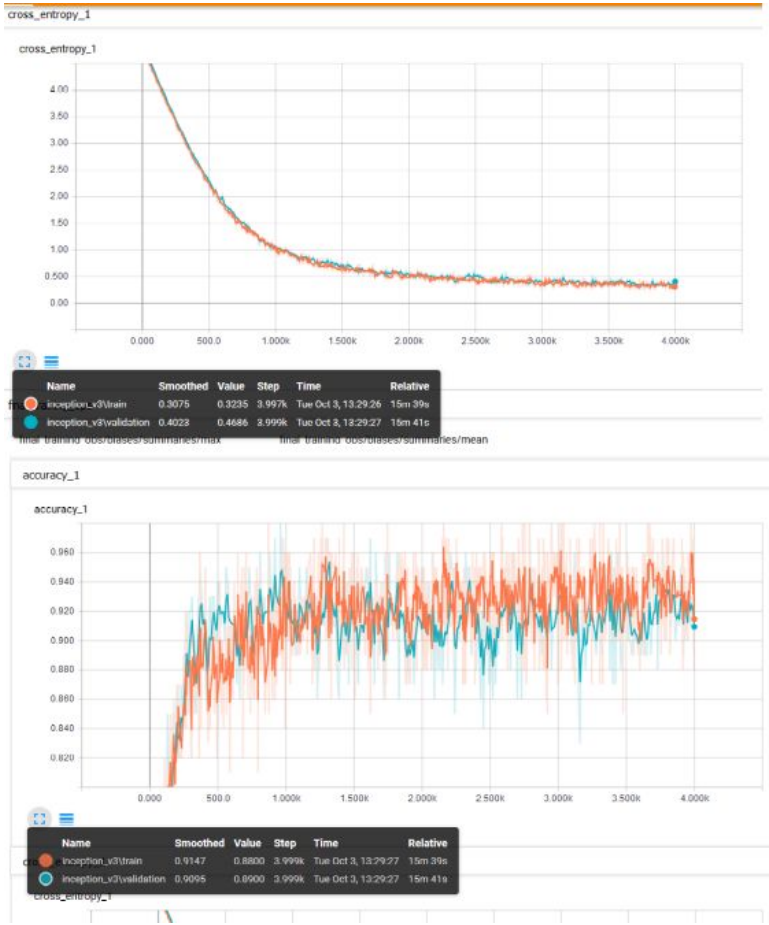
1. Classification

Results

- **Training/Validation Set Loss:** 0.34, 0.47
- **Training/Validation Set Accuracy:** 0.91, 0.9
- **Kaggle Submission Log Loss:** 0.58

Improvements

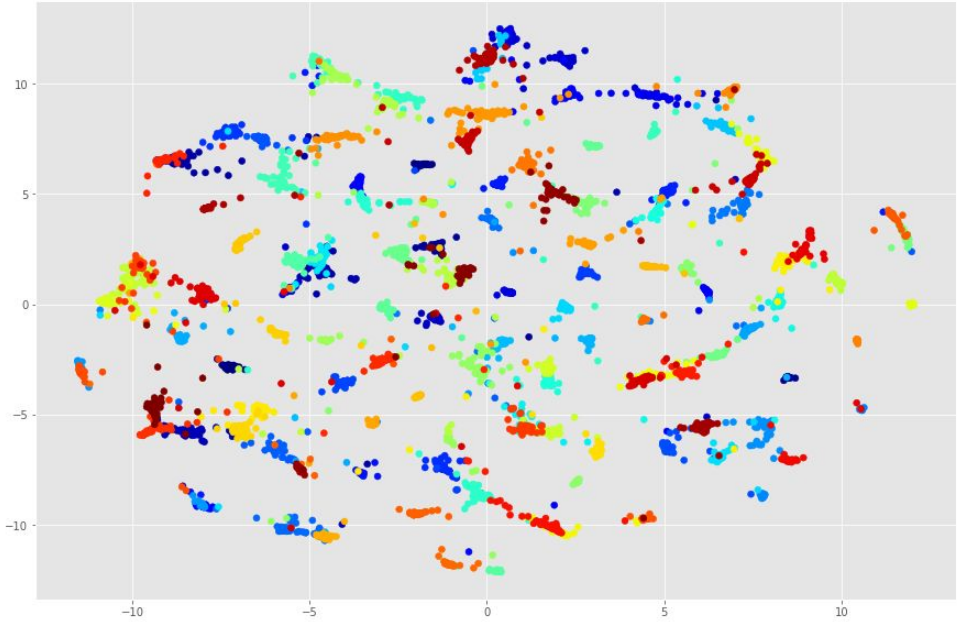
- Image augmentation – Generalises model and reduces overfitting
- Experiment with Inception v5, VGGNet
- Use deeper fully connected layers



1. Classification

Clustering – Latent Manifold

t-SNE: preserve the local distances of the high-dimensional data in some mapping to low-dimensional data



Grid view of clustering

2. Generative Model

Generative Models

Given training data, generate new samples from the same distribution. A form of unsupervised learning.

Why Generative Models?

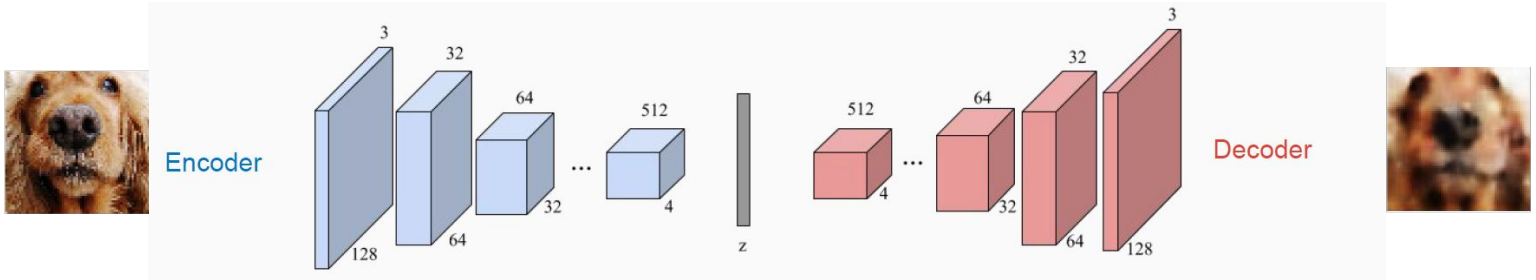
Data created by Generative Models can be used for simulation and planning (Reinforcement Learning)

Enable inference of latent representations that can be useful as general features

2. Generative Model

Variational Autoencoders

- Input data ***x***, reduced to latent variables ***z*** by the Encoder, and reconstructed to ***x-hat*** by the Decoder.
- Generate latent vectors (***z***) that roughly follow a Unit Gaussian Distribution
- Maximise likelihood of original input being reconstructed by the model
- The objective is to optimise:
    - **Generative Loss** - mean squared error that measures how accurately the network reconstructed the images
    - **Latent Loss** - KL divergence that measures how close the latent variables ***z*** match a Unit Gaussian Distribution



2. Generative Model

Variational Encoders (cont.)

But it is intractable to compute ***p(x|z)*** for every ***z***

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

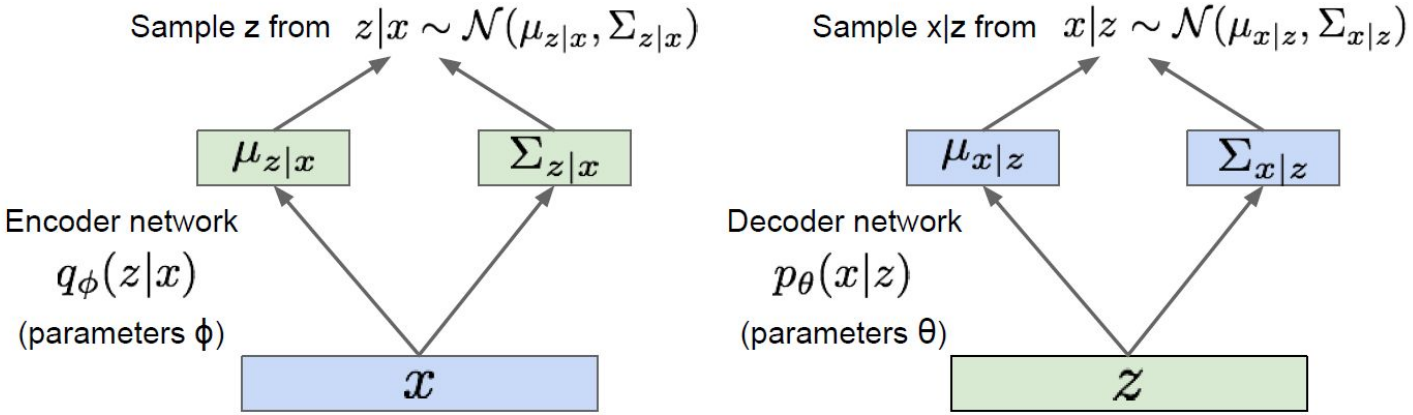
$$p_\theta(z|x) = p_\theta(x|z) p_\theta(z) / p_\theta(x)$$

Posterior density also intractable

Objective: Learn model parameters to maximise likelihood of training data

- In addition to decoder network modeling ***p(x|z)***, define additional encoder network ***q(z|x)*** that approximates ***p(z|x).***
- Allows us to derive a lower bound on the data likelihood that is tractable, which we can optimize

Since we're modeling probabilistic generation of data, encoder and decoder networks are probabilistic



2. Generative Model

$$
\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] && (p_\theta(x^{(i)}) \text{ Does not depend on } z)\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] && \text{(Bayes' Rule)}\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] && \text{(Multiply by constant)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] && \text{(Logarithms)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))
\end{aligned}
$$

Decoder network gives pθ(x|z), can compute estimate of this term through sampling

This KL term (between Gaussians for encoder and z prior) has nice closed-form solution

pθ(z|x) intractable (saw earlier), can't compute this KL term. But we know KL divergence always >= 0.

Reconstruct input data

Make approximate posterior distribution close to prior

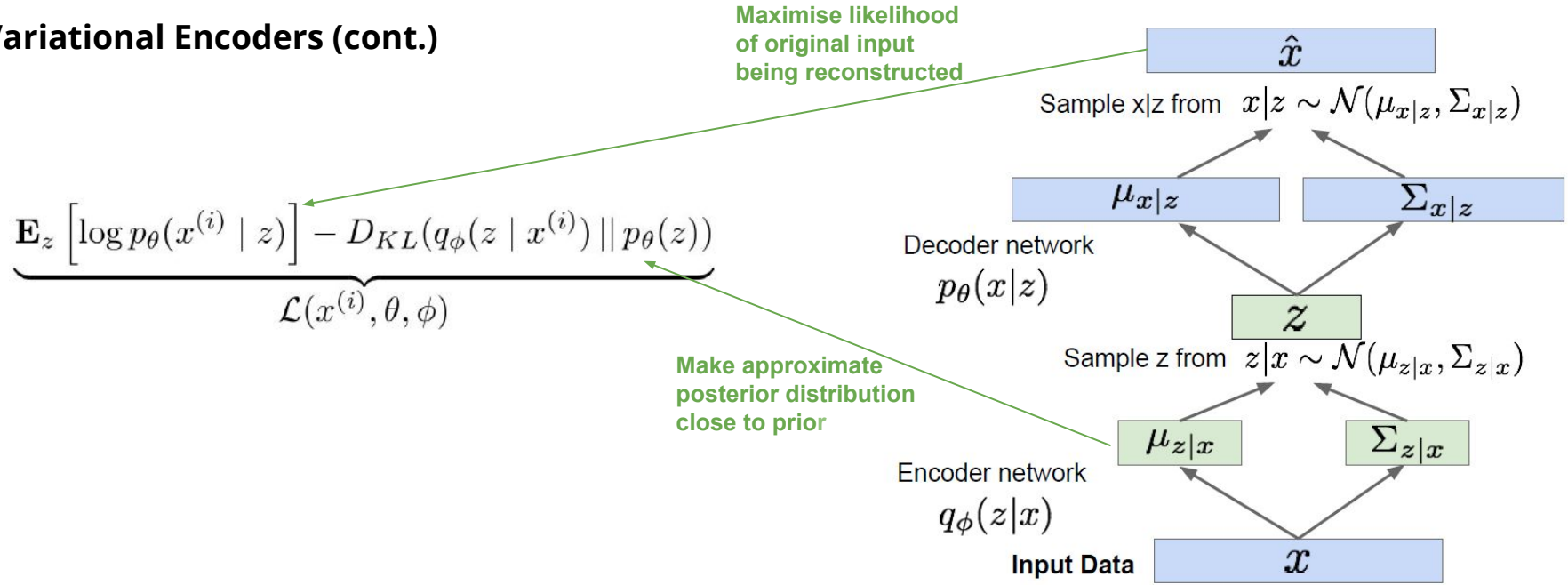
$$\mathcal{L}(x^{(i)}, \theta, \phi)$$

Objective: Maximise this lower bound

$$\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_{i=1}^{N} \mathcal{L}(x^{(i)}, \theta, \phi)$$

2. Generative Model

Variational Encoders (cont.)



More details of the math (can be found at [cs231n_2017_lecture13.pdf]())

2. Generative Model

Variational Autoencoders Configuration

Encoder	Decoder
Input: [None, 64, 64, 3]	Convolutional Layer 1: [None, 16, 16, 128]
Convolutional Layer 1 : [None, 32, 32, 64]	Convolutional Layer 2: [None, 32, 32, 64]
Convolutional Layer 2 : [None, 16, 16, 128]	Convolutional Layer 3: [None, 64, 64, 3]
Convolutional Layer 3 : [None, 8, 8, 256]	Convolutional filter sizes: [5, 3, 3]
Convolutional filter sizes: [5, 3, 3]	
Variational Layer: [None, 8, 8, 256]	
Fully Connected Layer: 256	
Latent Samples: 128	

- Model training
- Image Reconstruction based on Dog Video
- Style Transfer