CAB420: Auto-Encoders

ENCODE YOURSELF

Encoders and Decoders

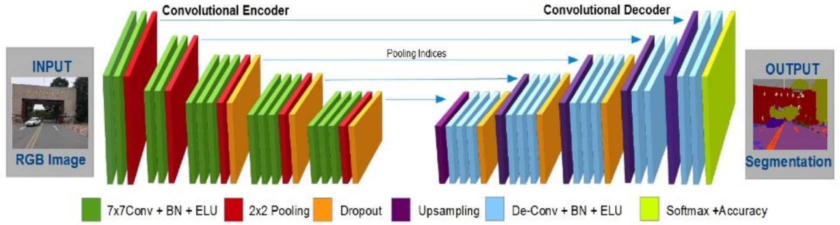
- Common components in deep learning
- Typically operate on signal-based input
  - Images, videos, audio clips
- Often used in a pair
  - Image -> Encoder -> Decoder -> Output (often another image)
- We already seen lots of “encoders”
  - Siamese networks encode the input into an embedding

Encoders and Decoders

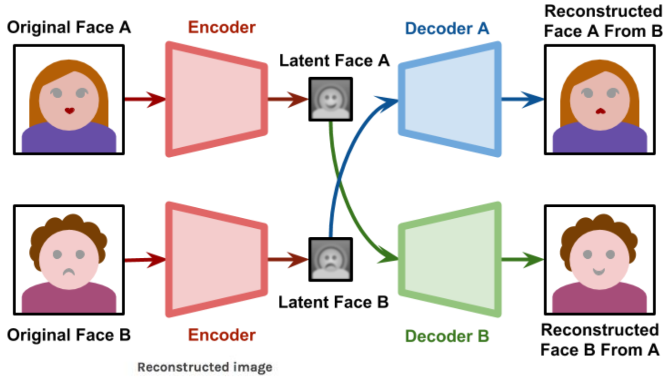
- Encoders
  - Take an input signal, aim to extract a compressed representation
  - Compressed representation may be good for different things depending on how the network is setup
- Decoders
  - Takes the compressed representation
  - Outputs a synthesised signal
    - Can be the original signal (auto-encoder)
    - Can be something else

Encoder-Decoder Applications

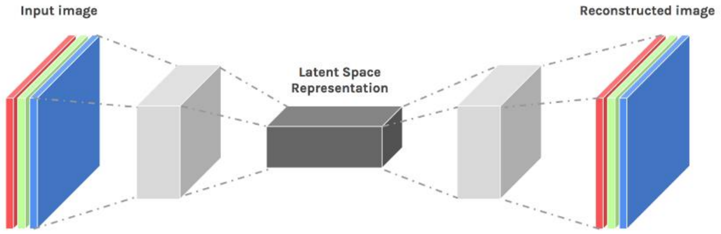
- Image to Image translation



- DeepFakes



- AutoEncoders



Auto-Encoders

- Given an input
  - Encode it into a compact representation
  - Then decode it, getting the original back
- Deep Learning for Dimension Reduction
  - Unlike PCA or LDA, can learn a highly non-linear representation
  - Like PCA, the compressed representation can be used to reconstruct the original signal
- Typically seen as unsupervised learning
  - No explicit ground truth signal or label
  - Target label is the same as the input

Learning Objective

- Auto-encoders try to reconstruct the original signal

$$L_{recon} = \sum_{i}^{N} (x_i - \hat{x}_i)^2$$

  - x_i is the input signal
  - $\hat{x}_i$ is the reconstructed signal
  - N is the size of the signal
- You may also often see an L1 distance used

- Can be seen as a many-to-many regression problem
  - Regress N outputs from N inputs

An Example

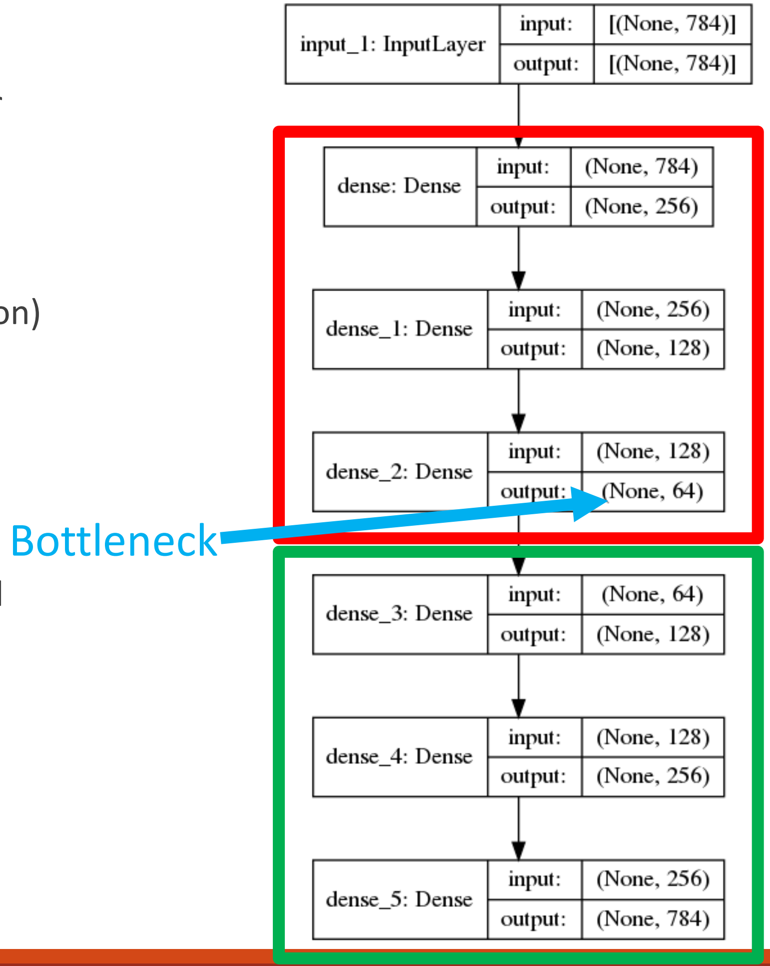
- See ***CAB420_Encoders_and_Decoders_Example_1_AutoEncoders.ipynb***

- Our data
  - Fashion MNIST

- Our Task
  - Encode Fashion MNIST into a compact representation
  - Decode it to reconstruct the original data

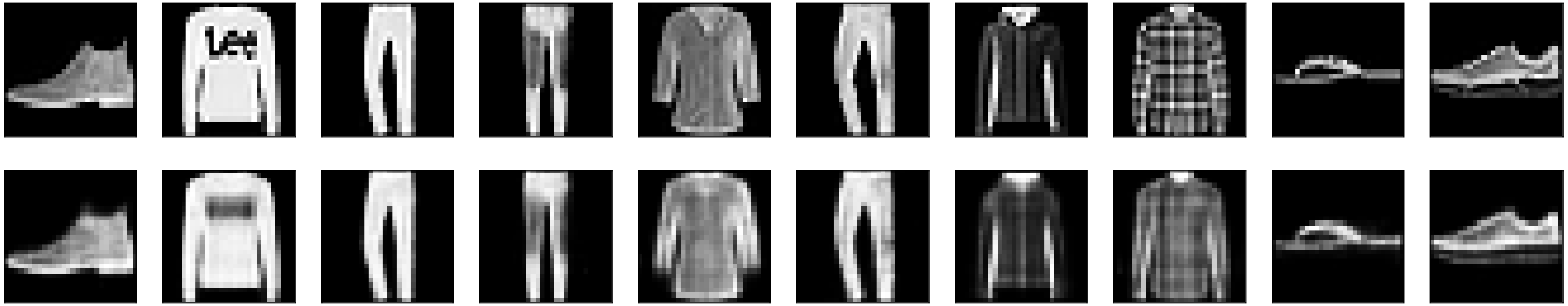
My First Auto Encoder

- Vectorised input
  - 28x28 image to 1x784 vector
  - Encoder
    - Three dense layers
    - Final shape is 1x64 (compressed representation)
  - Decoder
    - Three dense layers
    - Mirrors the encoder
    - Upsample back to 1x784
      - Reconstruct the original signal



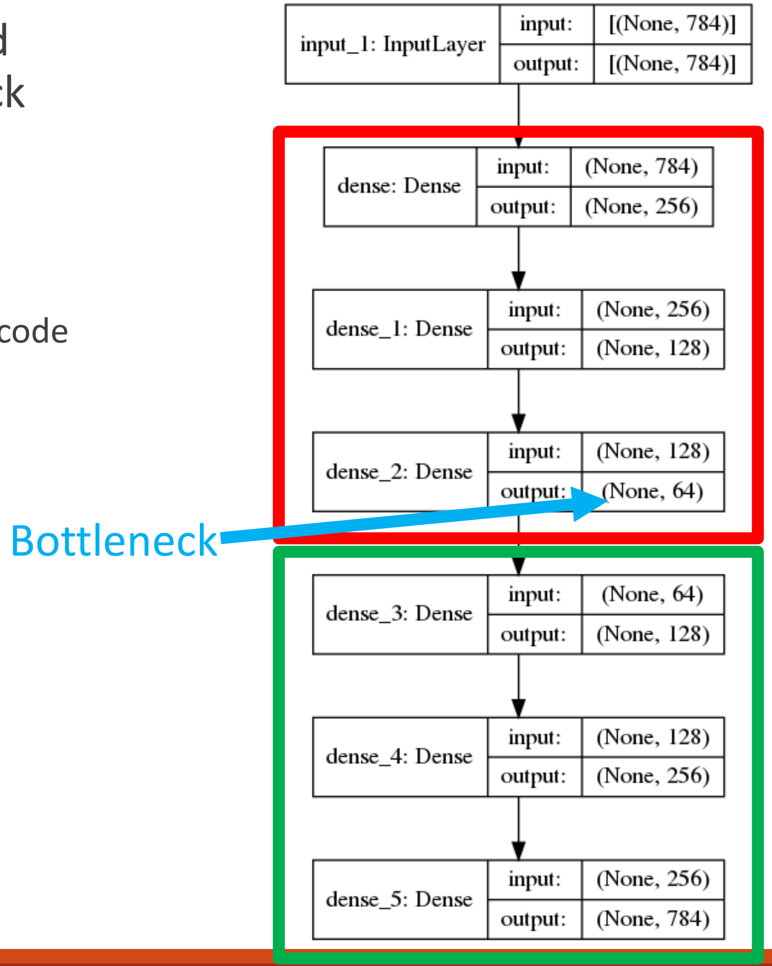
Auto Encoder Output

- Reconstructions contain major details
  - Edges are blurred
  - Textures (text, checked patters) are somewhat lost
  - Broad shape/structure preserved



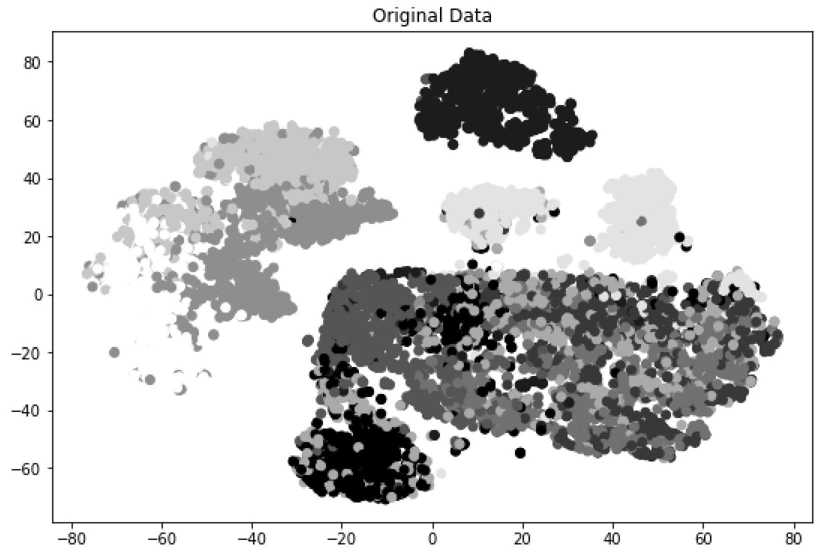
Sparse Autoencoders

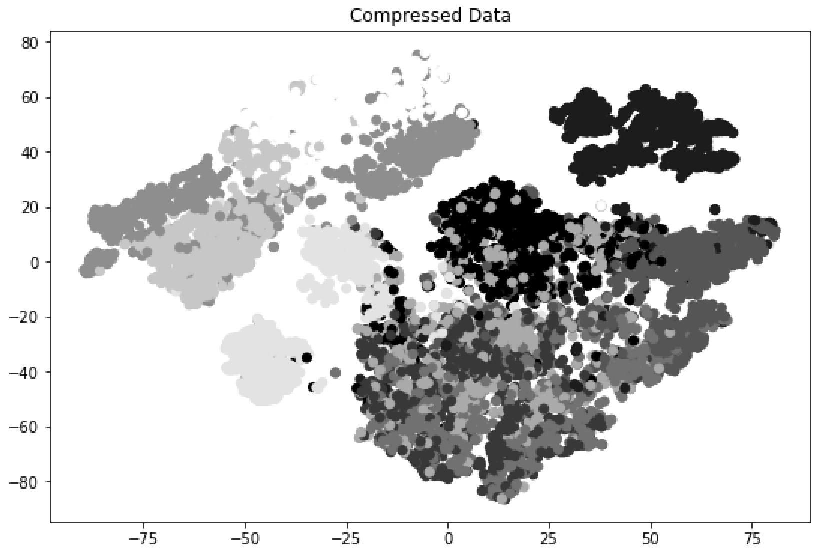
- We may also wish to add sparsity to our bottleneck layer
- L1 penalty
  - Similar to Lasso regression
  - Force network to learn to encode the input with fewer active nodes
  - Hope to promote a more meaningful representation



What is Learnt?

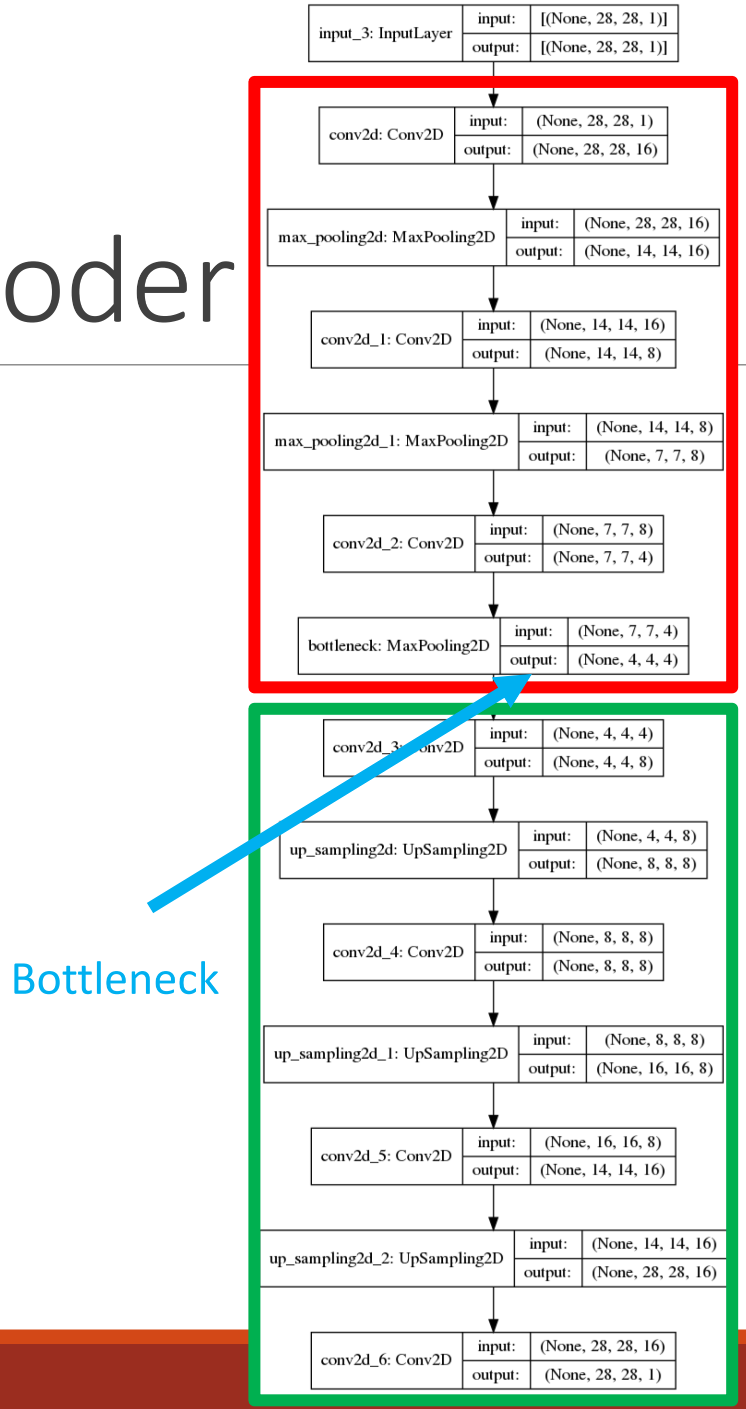
- t-SNE plots of the original data and bottleneck output
- Broad shape similar
- Auto-encoder is unsupervised
  - No class labels, cannot learn a strong boundary between classes





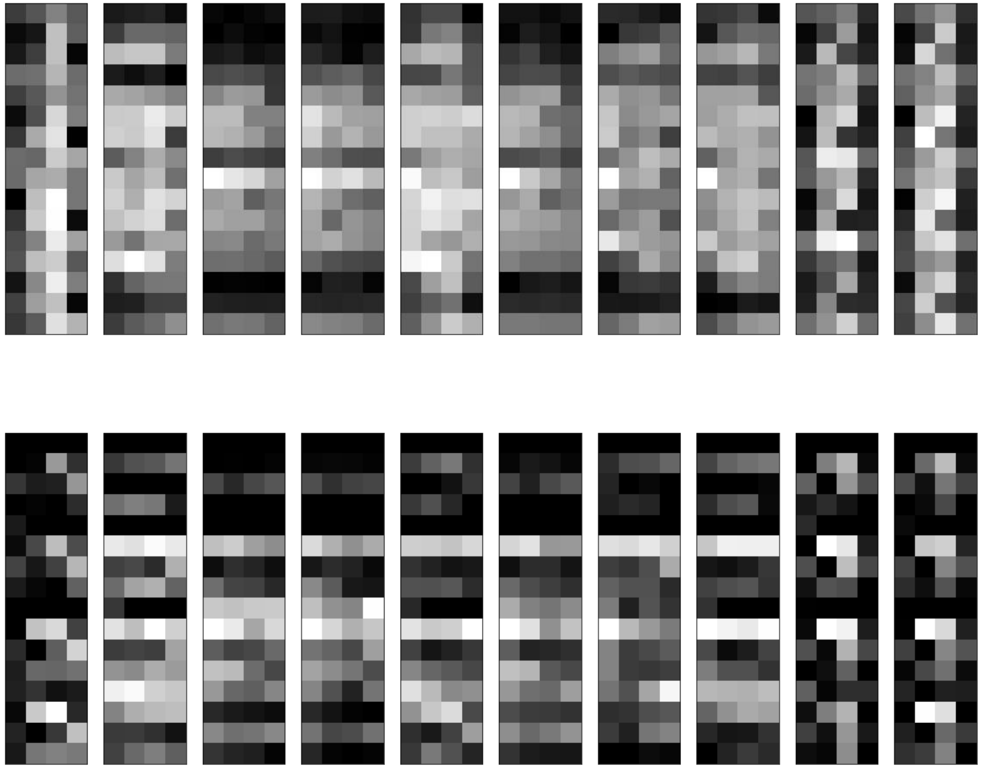
A Better Autoencoder

- Encoder
  - 3 Conv2D layers
    - Number of filters decreasing as we go deeper
- Decoder
  - Reverse of encoder
  - Upsample layers in place of MaxPooling
- Bottleneck
  - 4x4x4 tensor (64 units)



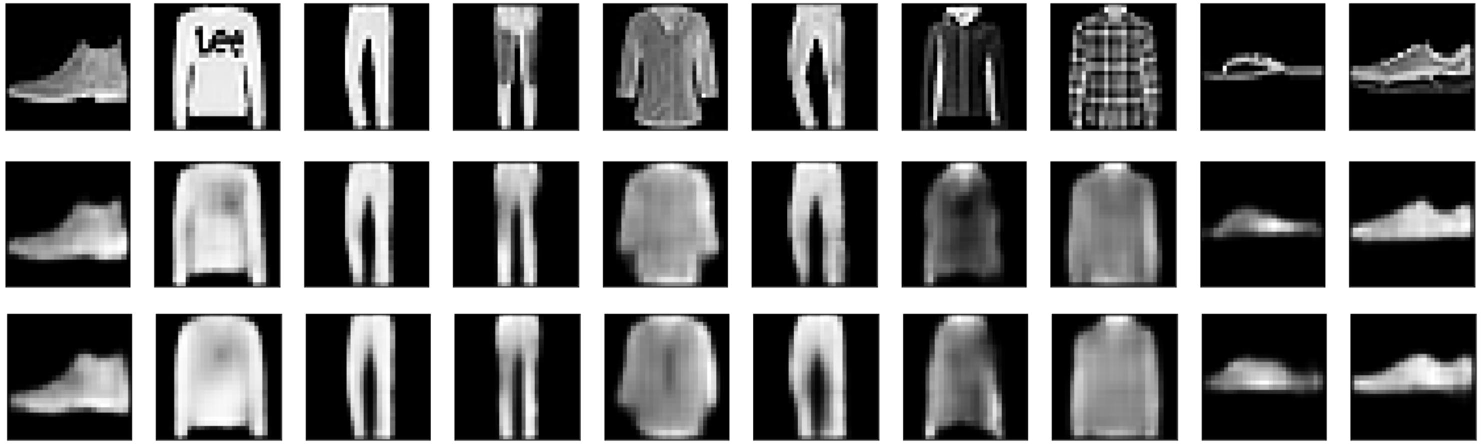
Impact of Sparsity

- Activations for the same input for network
  - Each column are the feature maps for one input sample
  - Without sparsity penalty (top)
  - With sparsity penalty (bottom)
    - Many fewer neurons active



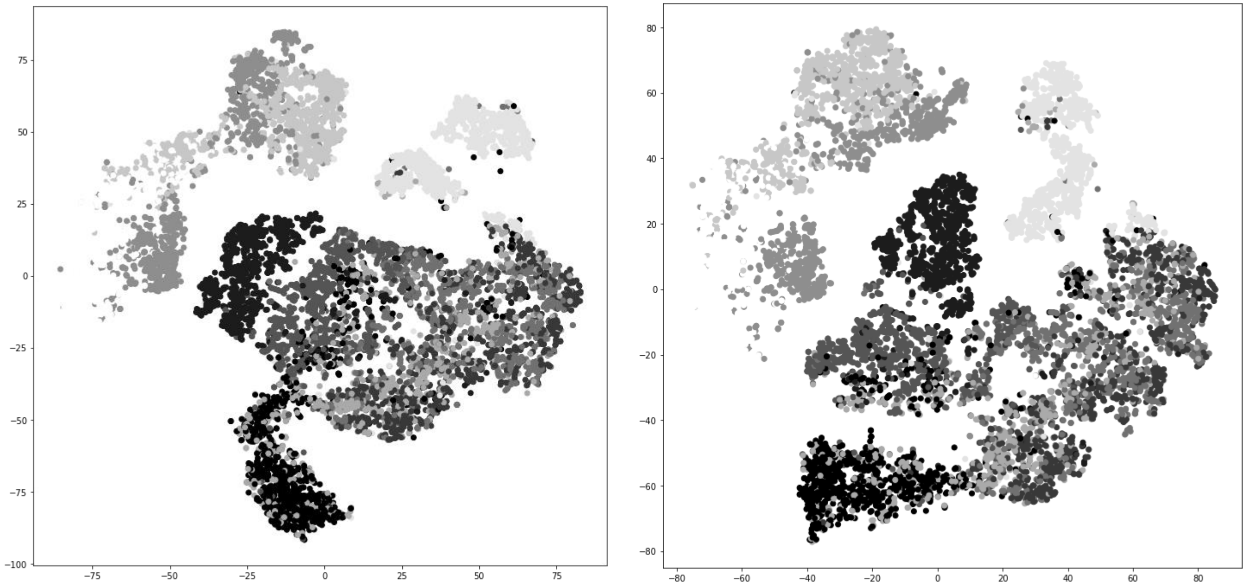
Impact of Sparsity

- Top: Original data
- Middle: Reconstruction without sparse constraint
  - Perhaps contains slightly more fine detail
- Bottom: Reconstruction with sparse constraint



Impact of Sparsity

- t-SNE plots of bottleneck features
- Left: without sparsity constraint
- Right: with sparsity constraint
  - Does a slightly better job disentangling classes
  - Sparsity helps to model to associate specific neurons with specific classes



Why Auto-Encoders?

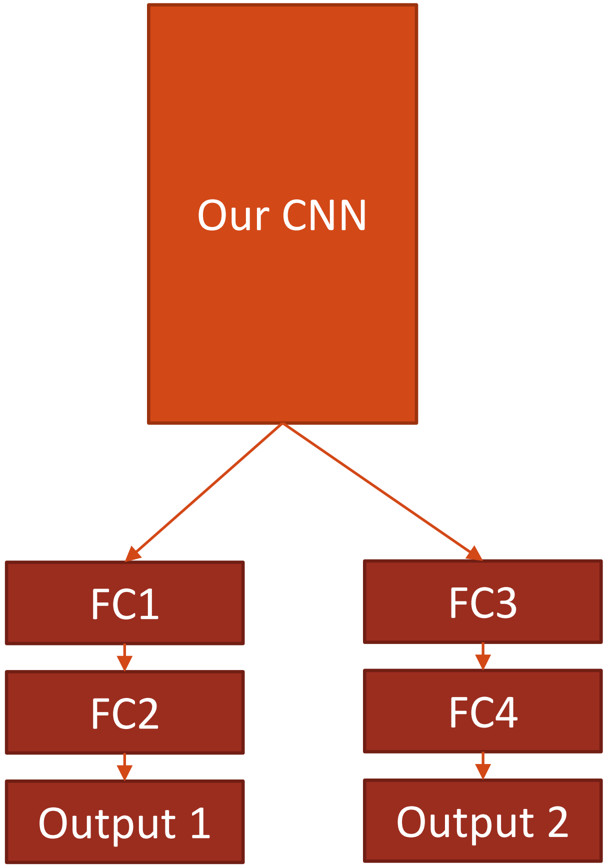
- It does have applications
  - Non-linear data compression/dimension reduction
    - Can stack them to get more compression
  - Anomaly detection
    - Given an input, compress and reconstruct
    - Something normal will be reconstructed well, something abnormal will have a high error
  - As pre-training for a network
    - Reuse the encoder in a classification network

CAB420: Multi-Task Learning

MULTI-TASKING FOR DEEP NETS

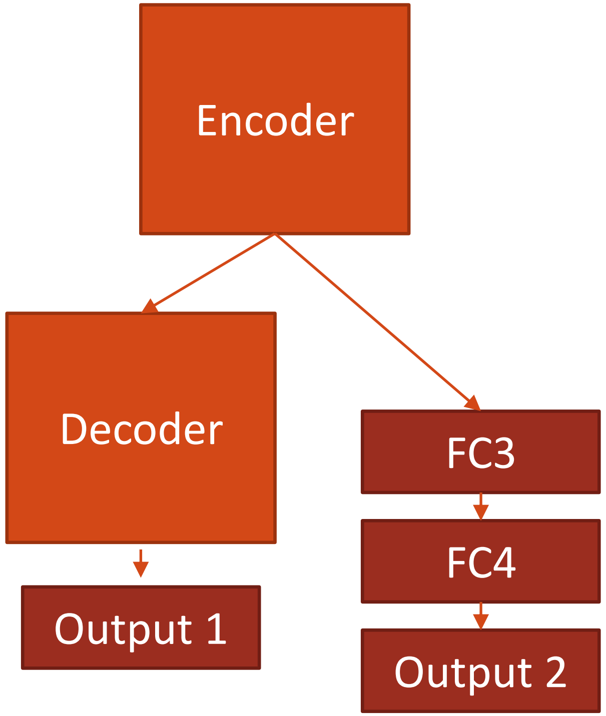
Multi-Task Learning

- We've seen with deep networks that
  - The same network can usually do different things, we just need to change the output shape and/or loss
- Why not then have multiple outputs?
  - Multiple output layers
  - One loss per layer
  - Can have a different loss function for each output
  - Overall loss is just the sum of the losses
    - Can be weighted such that some outputs are more important than others



Multi-Task Learning

- Outputs can come from different parts of the network
  - Outputs can have wildly different shapes
    - Image outputs
    - Numeric Outputs



Multi-Task Learning Objective

- Multiple outputs means multiple losses
- Output 1 (auto-encoder): reconstruction loss

$$L_{recon} = \sum_{i}^{N} (x_i - \hat{x}_i)^2$$

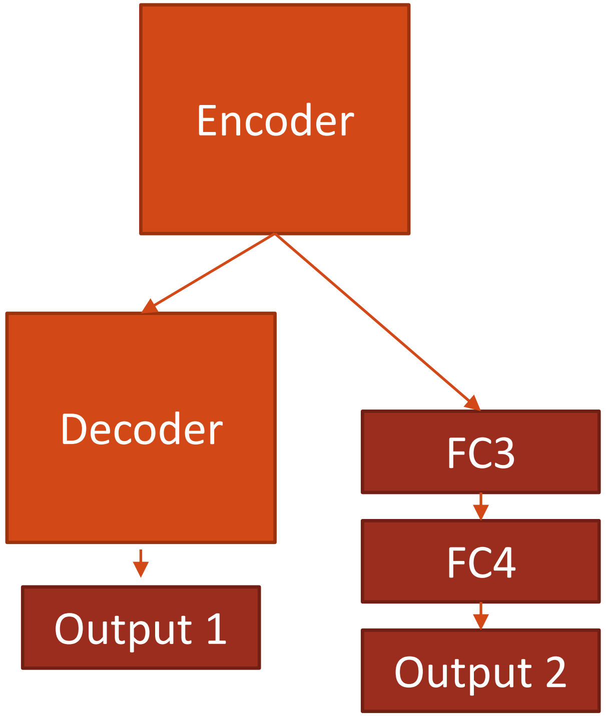
- Output 2 (classifier): categorical cross entropy:

$$L_{CE} = -\sum_{i}^{N} y_i' \log(y_i)$$

- Overall loss combines the two

$$L_{Overall} = \lambda_1 L_{recon} + \lambda_2 L_{CE}$$

- We can set λ_1 and λ_2 as we see fit.
  - If in doubt, $\lambda_1 = \lambda_2 = 1$



Multi-Task Learning

- Pros
  - Usually has a low overhead
    - Can just append a couple of extra layers to get another output
    - Cheaper than having a network for each task
  - Usually helps learning
    - Particularly if tasks are related
    - One task helps regularise the other
- Cons
  - We now need two sets of labels
    - It's hard enough to annotate one set

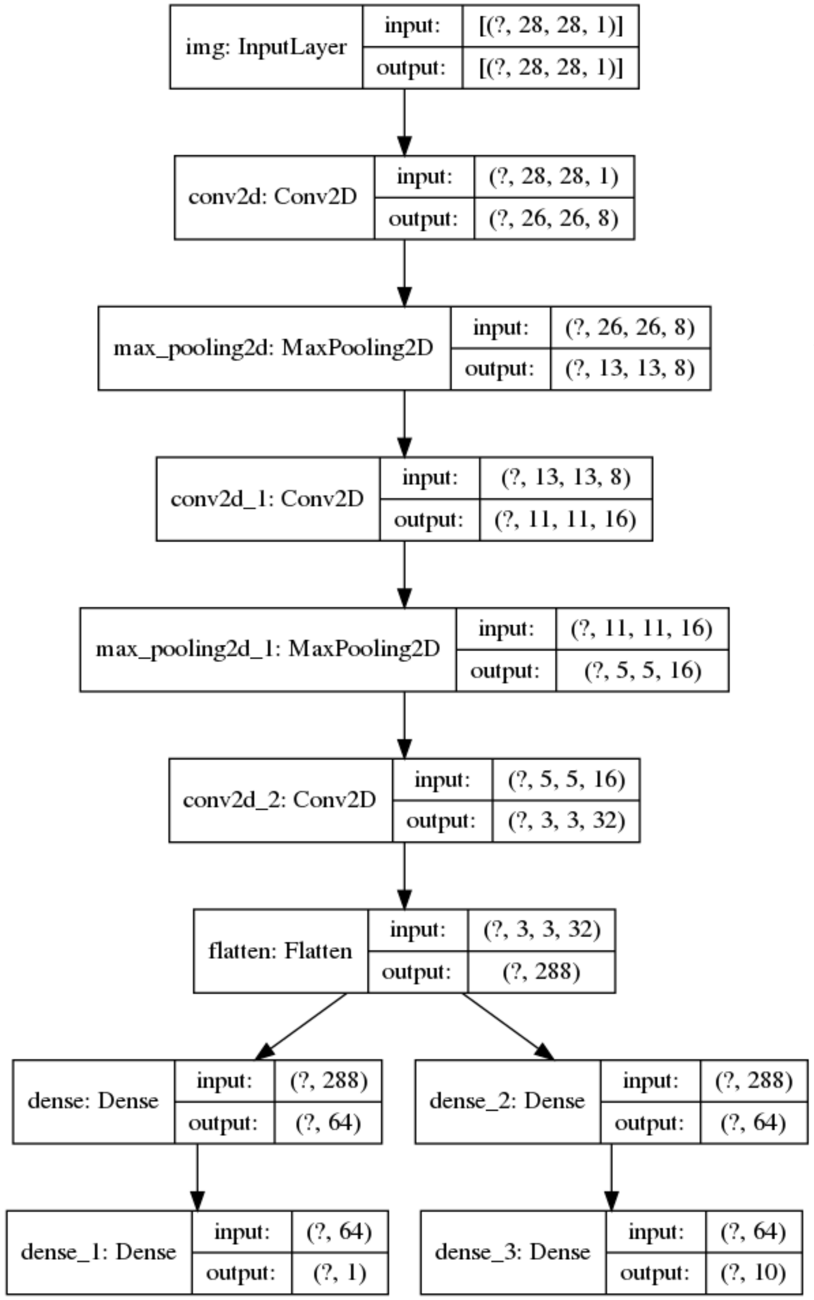
An Example

- See ***CAB420_Encoders_and_Decoders_Example_2_Multiple_Outputs.ipynb***

- Our Task
  - Rotated Digits Datasets
  - Simultaneously estimate
    - The digit (0, 1, 2, …)
    - How much the digit has been rotated by

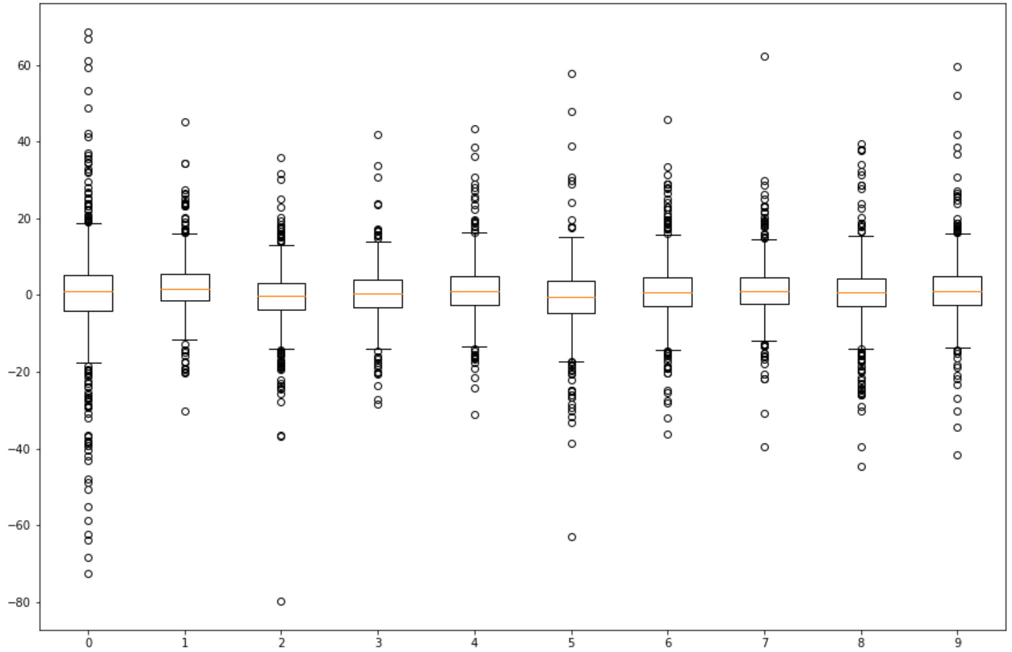
Our Network

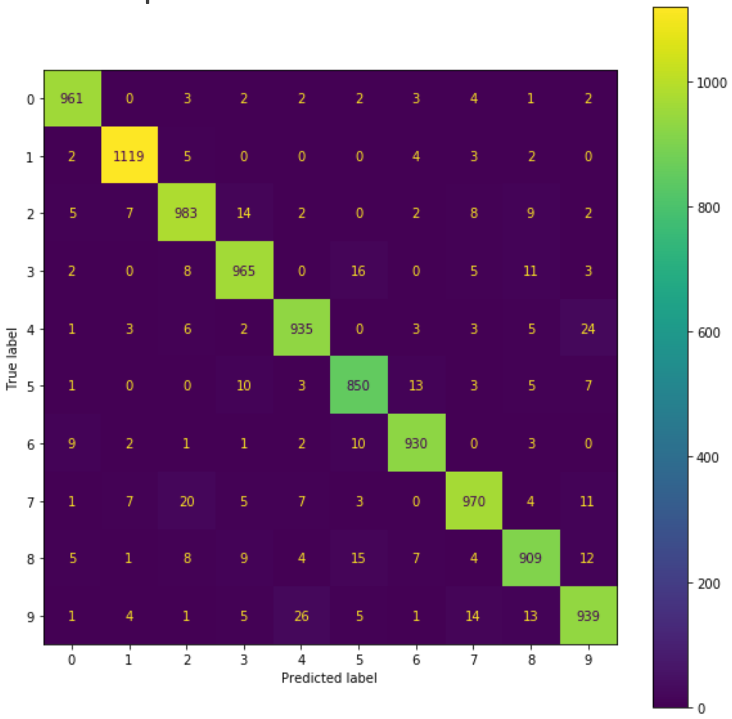
- Simple CNN
  - 3 Convolution layers
- Network branches after last convolution
  - Both branches are two dense layers
  - Different output sizes in each
    - dense_1, size (?, 1) is the angle output
      - Mean Squared Error loss
    - dense_3, size (?, 10) is the digit classification output
      - Categorical Cross Entropy loss



Network Performance

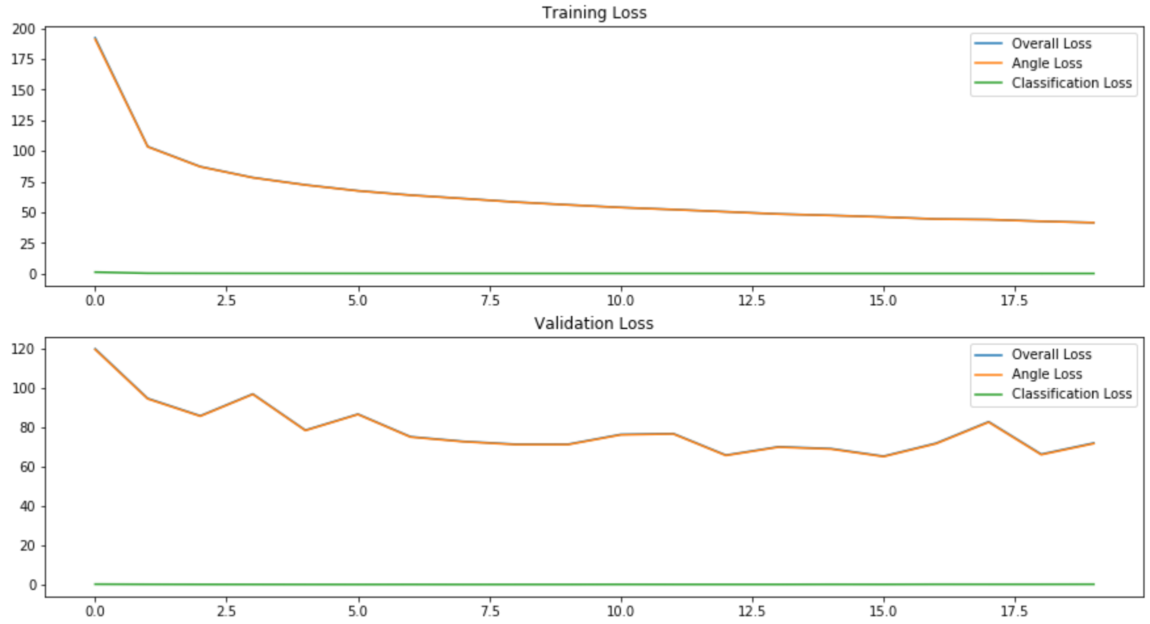
- Overall, both tasks are performed well
- Similar performance to when performing either task individually
  - Unsurprising, tasks are closely related, should help each other





Training Performance

- One loss (MSE) dominates
  - Scale of MSE is much larger than Cross Entropy
    - Means this loss may have more of an impact on learning – bigger values equals bigger gradients
  - Has limited impact here due to very complementary nature of the tasks



Tweaking Loss Weights

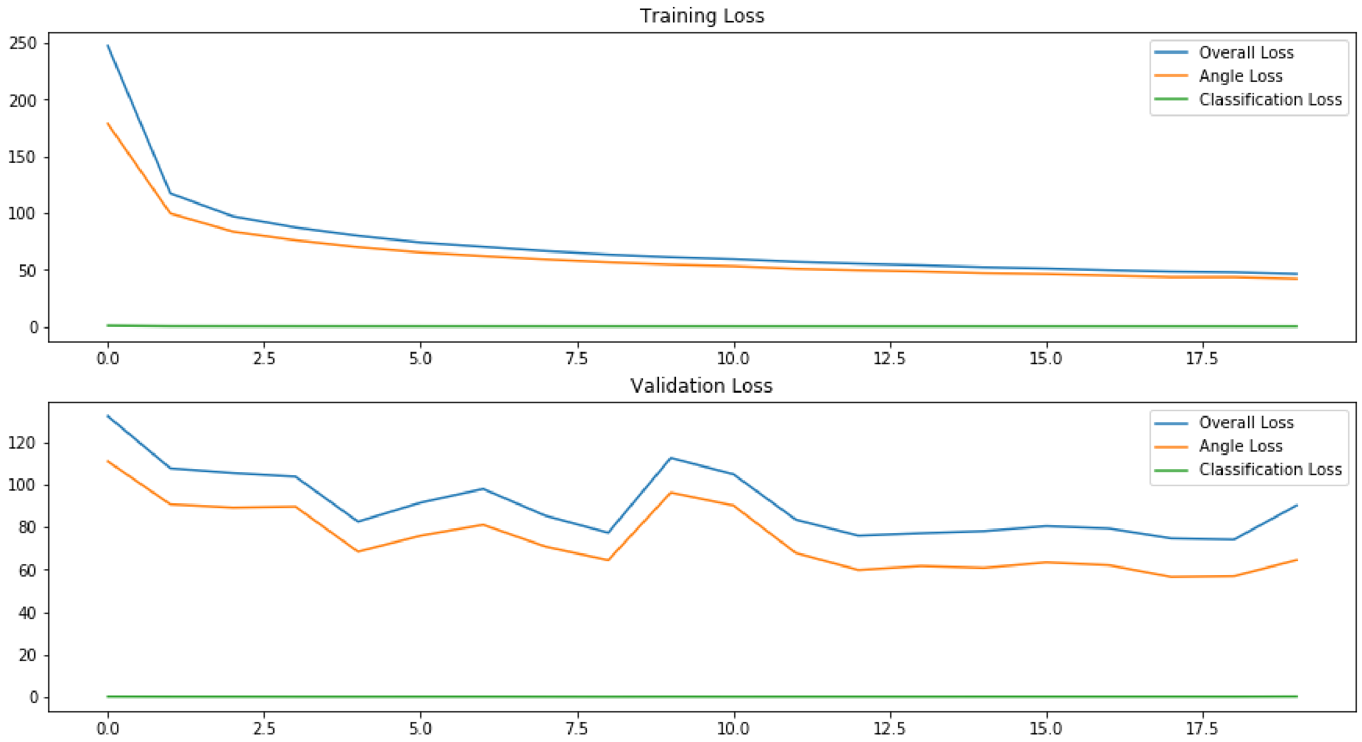
- Our loss can be expressed as

$$L_{Overall} = \lambda_1 L_{MSE} + \lambda_2 L_{CE}$$

- Our first approach set $\lambda_1 = \lambda_2 = 1$
- This time, we'll use $\lambda_1 = 1; \lambda_2 = 100$

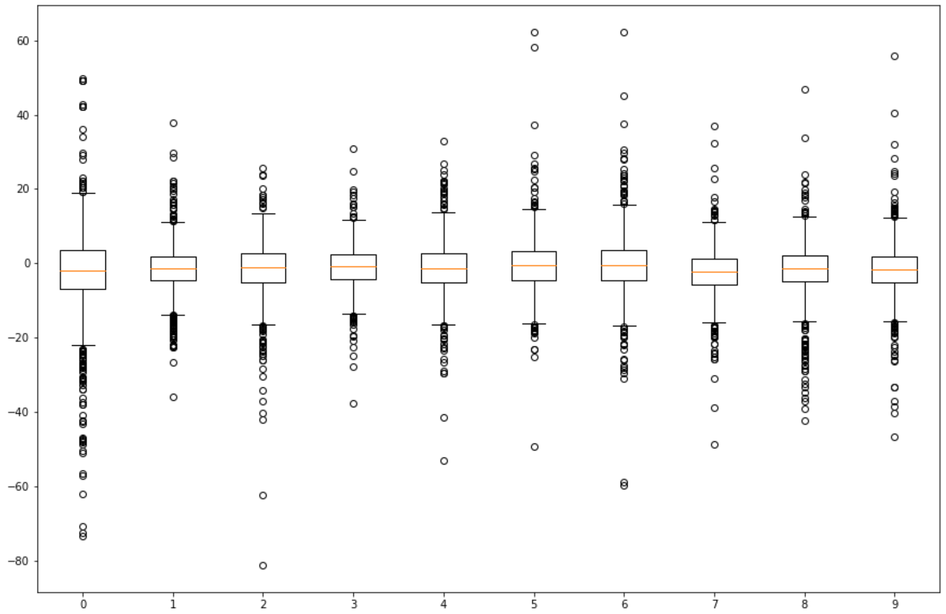
Tweaking Loss Weights

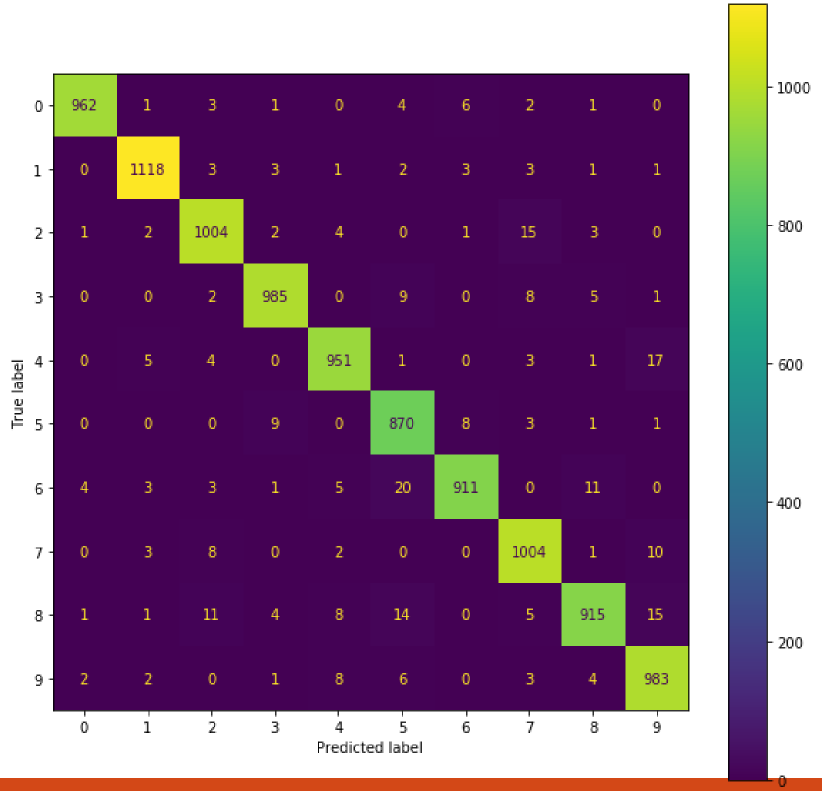
- Change visible in overall loss
  - Classificaiton loss is not scaled when plotting



Network Performance

- Very similar to without loss weights
  - Highly complentary tasks, so little was lost by the imbalanced loss scale





CAB420: Semi-Supervised Learning

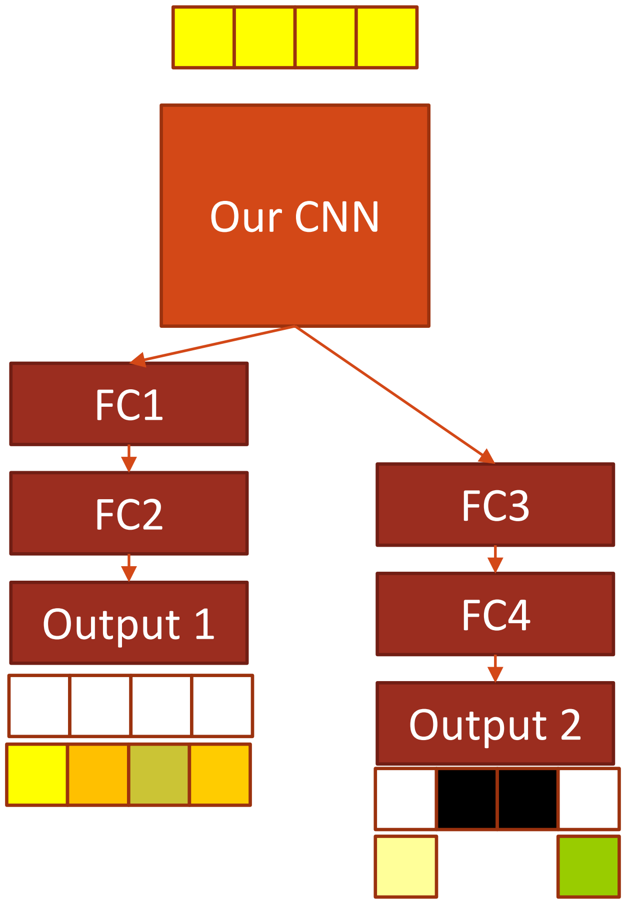
WHEN YOU CAN'T BE BOTHERED LABELLING ALL YOUR DATA

Semi-Supervised Learning

- Pretend we have a dataset which has annotation for two tasks
  - One has full annotation
  - One has data only for some samples
- We wish to learn both tasks
  - We don't wish to do any further annotation

Semi-Supervised Learning

- Modify our loss functions to avoid missing samples
  - Given 4 input samples
  - Output 1 has data for all 4
    - Output for this batch will be the sum of the loss for all four samples
  - Output 2 has data for only 2
    - Output for this batch will be the sum of the loss for the two samples for which data exists



Semi-Supervised Learning

- For each sample, we should have some ground truth signal
  - Need some annotation
  - Can workaround this with
    - Auto-encoders
      - Input becomes an output
    - GANs
- May wish to adjust output weights to reflect what data we have
  - Outputs with limited data may be given a higher weight
  - Encourage learning from whatever data we have

Semi-Supervised Learning Objective

- Output 1, classification objective

$$L_1 = -\sum_{i}^{N} y'_{1,i} \log(y_{1,i})$$

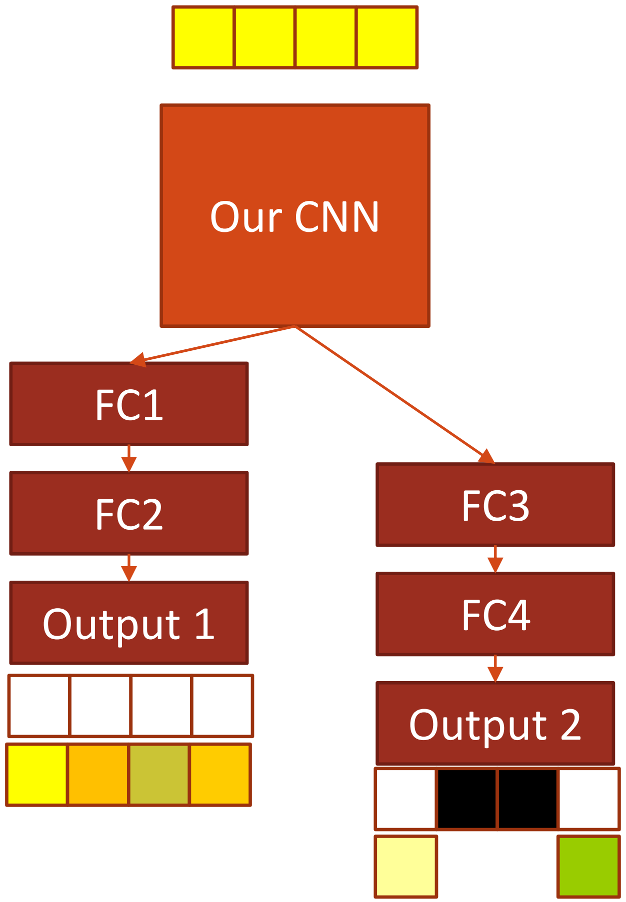
- Output 2, semi-supervised classification objective

$$L_2 = -\sum_{i}^{N} M_i y'_{2,i} \log(y_{2,i})$$

- M_i is a mask variable, equals 1 if we have ground truth, 0 if not
- Overall Loss

$$L_{Overall} = \lambda_1 L_1 + \lambda_2 L_2$$

- If M_i is often 0, L_2 will be small. May need to increase λ_2 to compensate



An Example

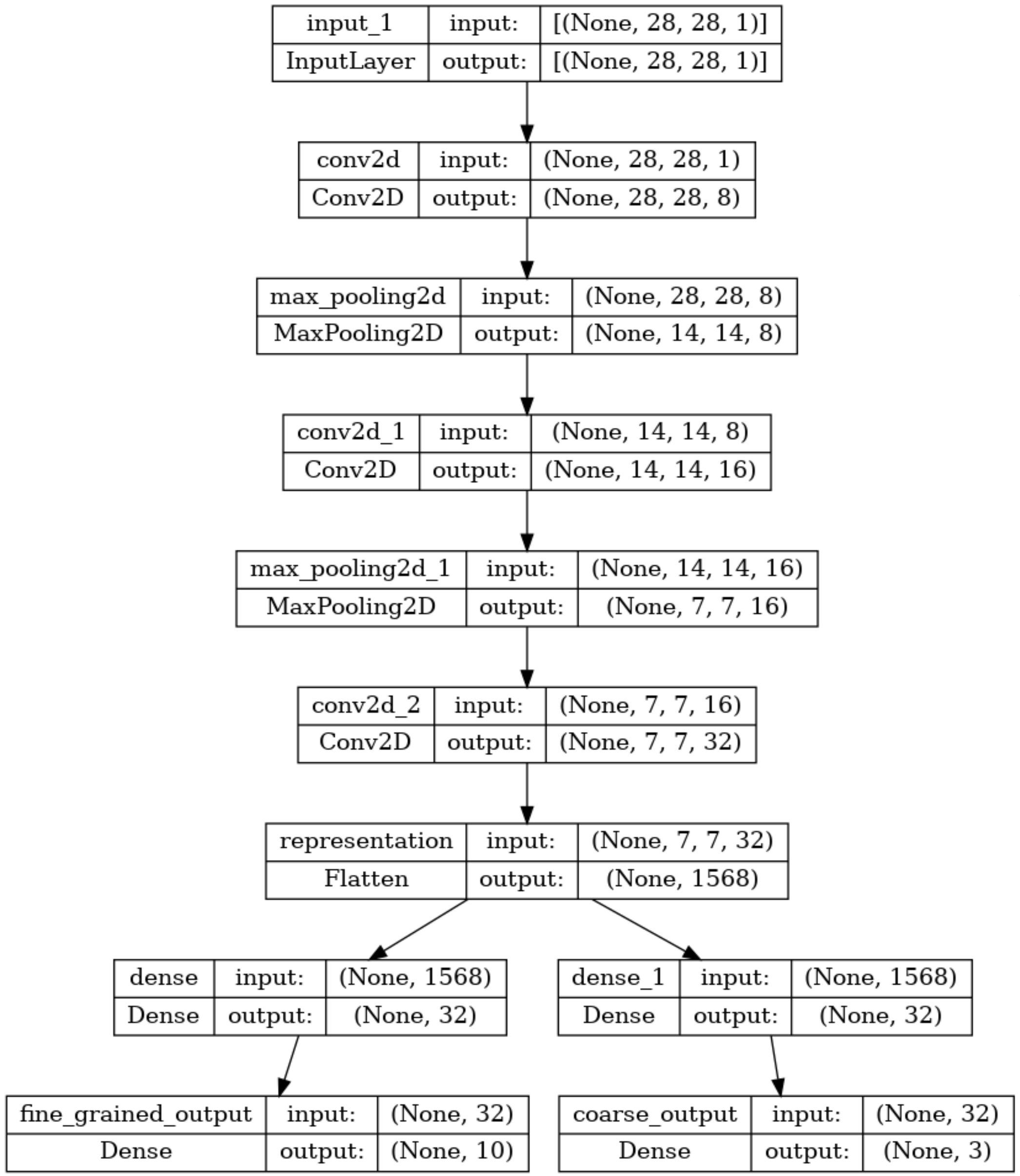
- See ***CAB420_Encoders_and_Decoders_Example_3_Semi_Supervised_Learning.ipynb***

- Our data
  - Fashion MNIST

- Our task
  - Coarse and fine-grained clothing classification
  - Coarse task
    - 3 classes (tops, bottoms, other)
    - Data for all samples
  - Fine-grained task
    - Usual 10-class problems, but with limited data

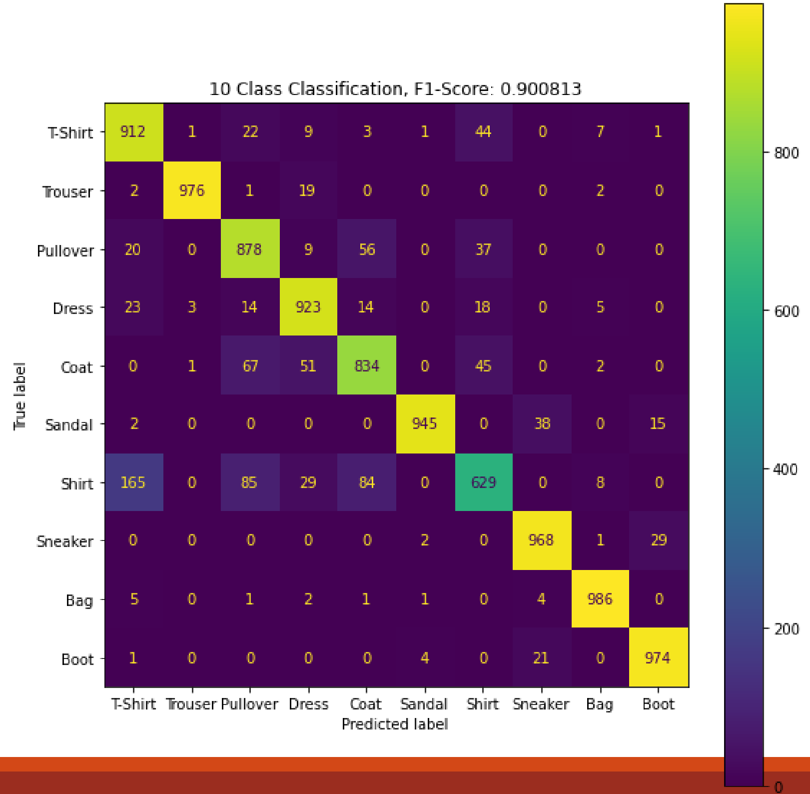
Our Network

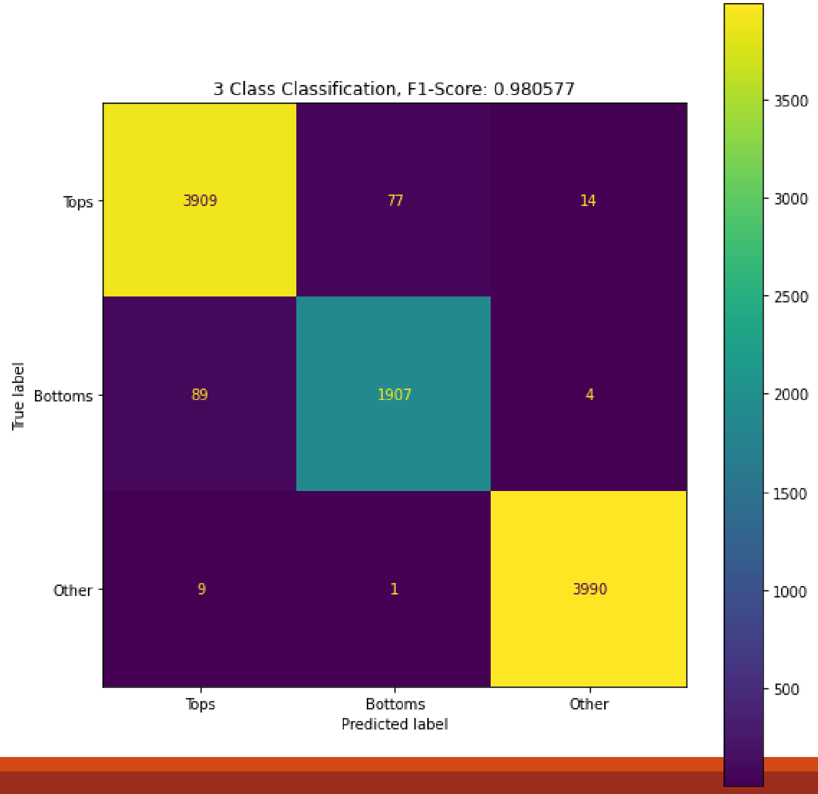
- Modified classification network
- Standard convolution backbone
- Branch at flattened representation
  - One coarse classifier
    - 3 tasks
  - One fine-grained classifier
    - 10 tasks



Training with all the data

- Good performance for both tasks
- Coarse task clearly supporting fine-grained task
  - Note errors in 10-class confusion matrix, the vast majority are confusion between examples within a coarse class





Removing Data

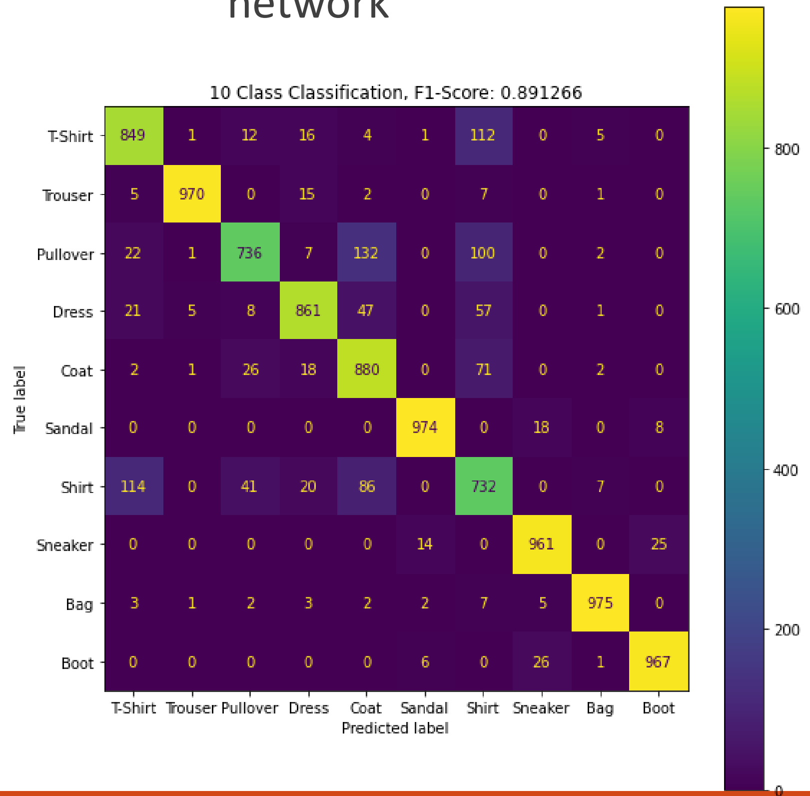
- Remove 75% of the labels
- Labels contain a one-hot representation
- Masks contain all –1 when the sample is removed
- Use a modified loss function that takes the masks as an extra input
  - Exclude masked samples from calculations

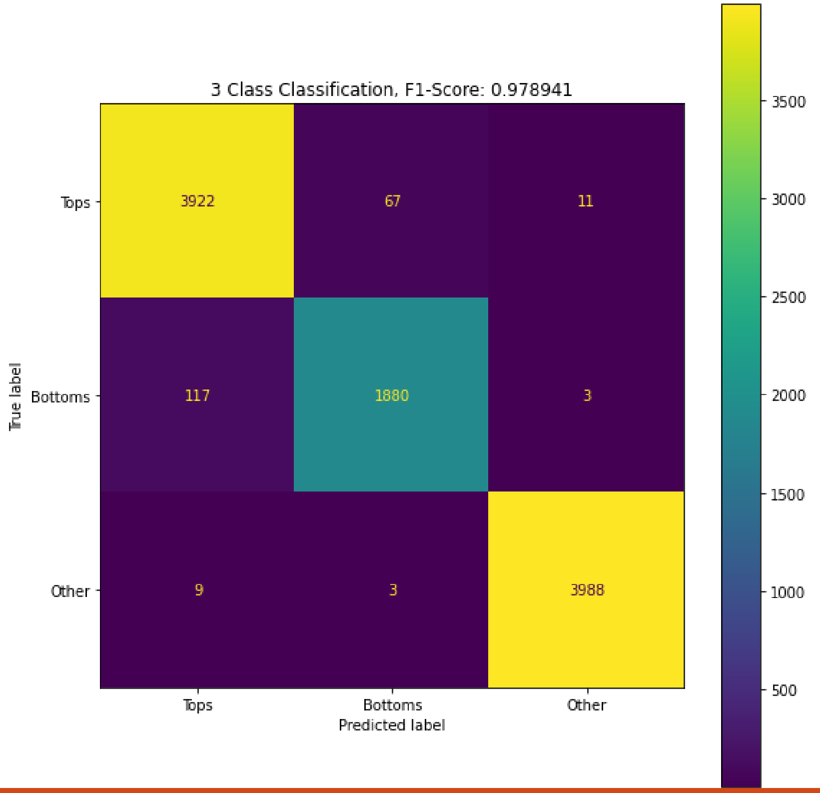
```
Labels
[[0. 0. 0. ... 0. 0. 1.]
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]

Masks
[[ 0.  0.  0. ...  0.  0.  1.]
 [-1. -1. -1. ... -1. -1. -1.]
 [-1. -1. -1. ... -1. -1. -1.]
 ...
 [ 0.  0.  0. ...  0.  0.  0.]
 [-1. -1. -1. ... -1. -1. -1.]
 [ 0.  0.  0. ...  0.  0.  0.]]
```

Training with 25% of the data

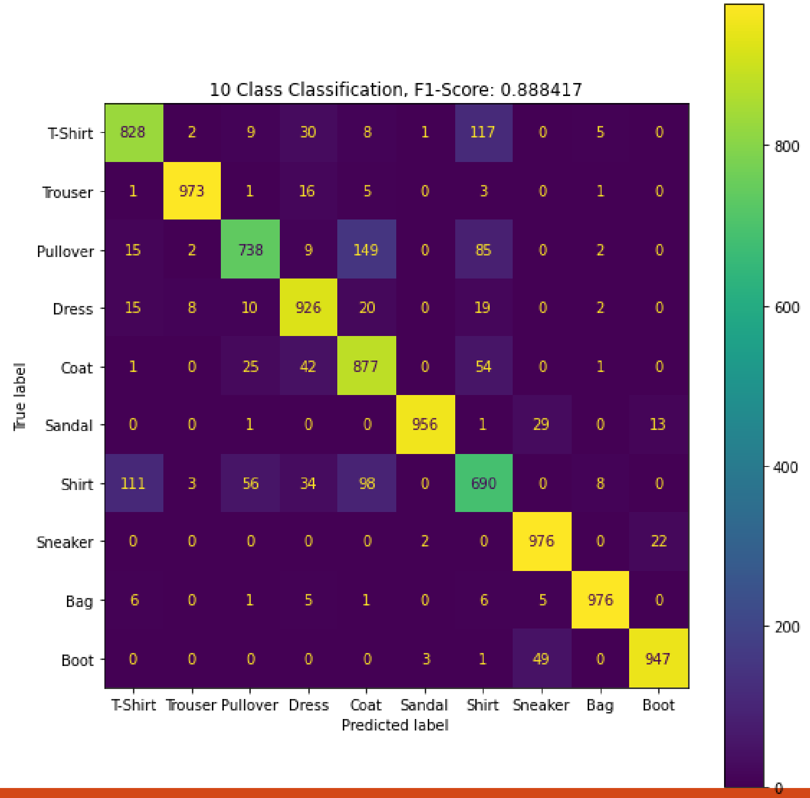
- Network still works well
- Small performance drop is within what we'd expect from simple sample variation when training the network

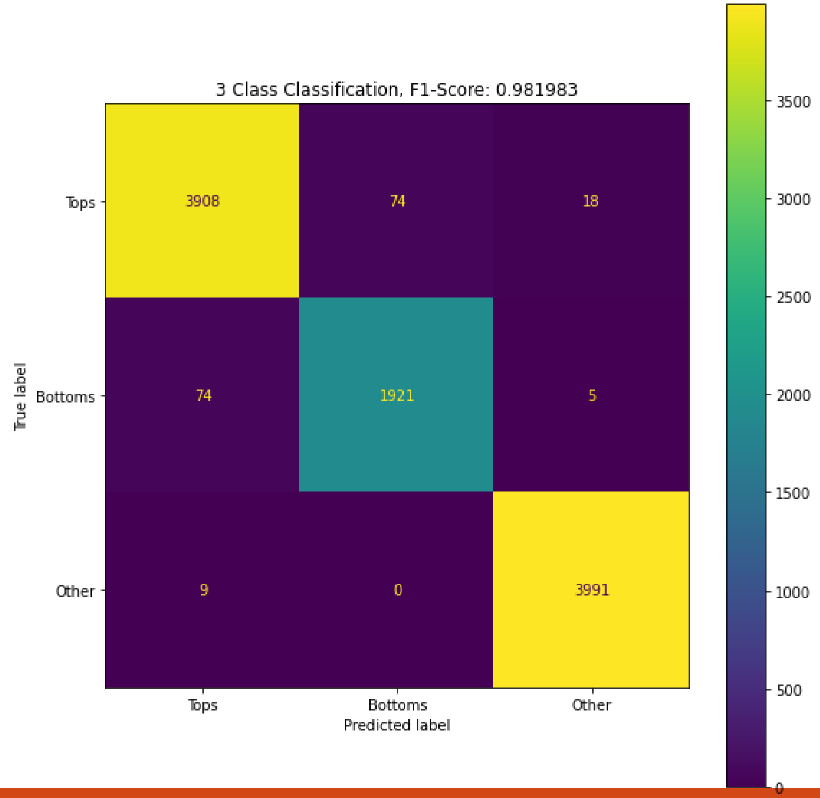




Training with 5% of the data

- Again, performance very similar
  - Perhaps a slight drop in the fine-grained task now





Other Considerations

- We can add class weights
  - May need to increase class weights in relation to the number of labels to improve training
  - May also wish to do this to prioritise one task over the other
    - In our example, the autoencoder really exists as a dummy task to support the classification
    - Thus, classification is far more important and could be weighted more
- We can have multiple tasks with partial data
  - And different tasks may have annotations for different samples

How Realistic is this Setup?

- Note that in this example, we've been greatly helped by the nature of the tasks
  - The "coarse" task is a simplified version of the main task - this helps a lot
- However, this sort of setup is not uncommon
  - Coarse annotation is much easier than fine-grained
  - Such coarse labels can be produced in an automated (or semi-automated) manner

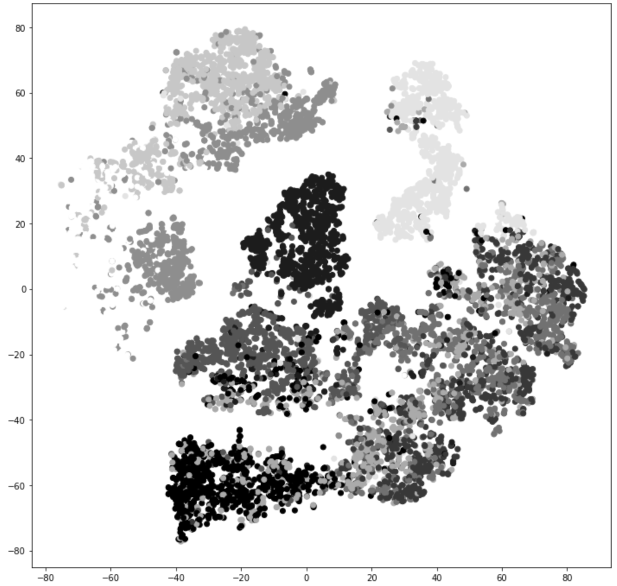
- Unsupervised tasks are also very common in a semi-supervised setup
  - An auto-encoder using all data
  - A classifier using only the available labels
  - Such a network would be geared towards the classification, i.e. no tiny bottleneck in the auto-encoder

CAB420: Variational Auto-Encoders

LEARNING DISTRIBUTIONS

Auto-Encoders

- Learn a compact representation of data by learning how to map from the input to itself via a bottleneck layer
  - Structure of representation is decided by the network (mostly). Though we can influence it using
    - Secondary losses
    - Sparsity constraints
    - Size and shape of the bottleneck

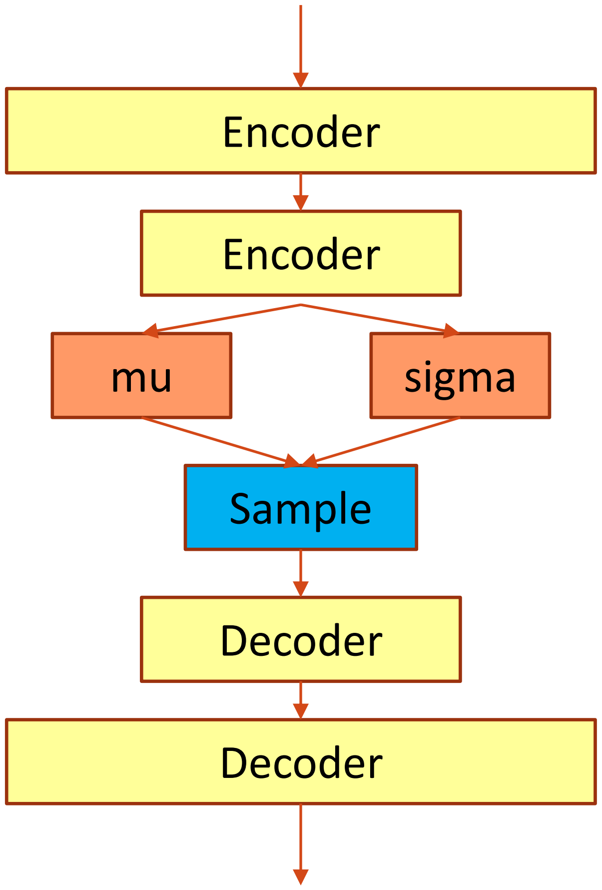


Variational Auto-Encoders

- Generative model
  - i.e. we can sample from it to “create” new data
- Learn a continuous latent space which we can sample from
  - Standard auto-encoders learn a discrete space
    - There can be large gaps in the latent space where no samples can exist

Variational Auto-Encoders

- For an input
  - Compute a mean and std.dev
- The decoder
  - Samples from the distribution described by the mean and std.dev
  - Decodes the sample to try to reconstruct the input
- By sampling we
  - Ensure that for the input, we don't necessarily get the same output
  - Help the decoder to learn the relationship between similar points/inputs

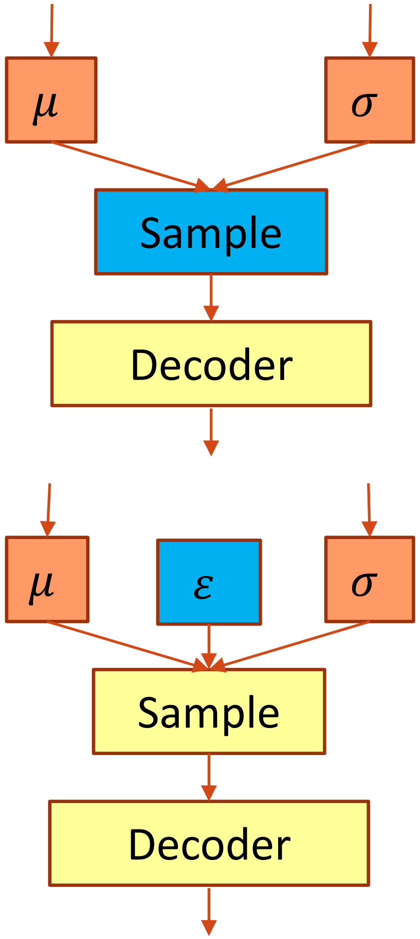


Variational Auto-Encoders

- Sampling and Backpropagation
  - Sampling directly from μ and σ makes backpropagation difficult
- Re-parameterization Trick
  - Leave μ and σ alone
  - Introduce ε
  - Sample becomes

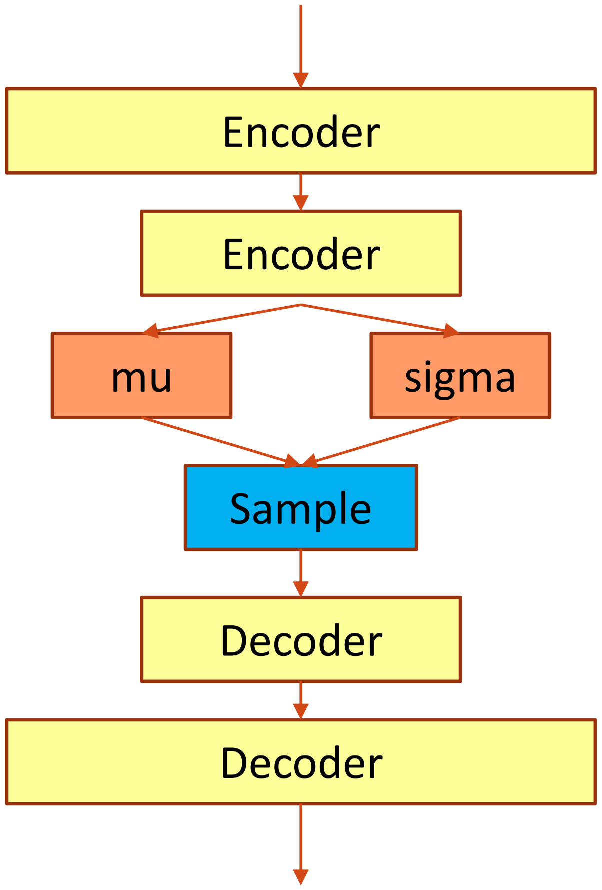
$$\mathrm{z} = \mu + \varepsilon\sigma$$

  - Moved the random node to an input
    - No longer in the main back-prop pathway



Variational Auto-Encoders

- By default, we will learn a discontinuous space
  - Different classes will be in different regions of the latent space
  - No smooth transition from one class to the next
- Place constraints on the learned distributions
  - Use KL Divergence
  - Seek to make our distribution look like a standard normal distribution
    - Ensure that samples are distributed across the latent space
    - Prevent the VAE from “cheating” and packing things into separate corners of the space



Variational Auto-Encoders Objective

- Reconstruction Loss

$$L_{recon} = \sum_{i}^{N} (x_i - \hat{x}_i)^2$$

- KL-Divergence Loss
  - Measures the similarity between two distributions

$$D_{KL}[N(\mu(X), \Sigma(X)) || N(0,1)] =$$
$$\frac{1}{2} \sum_{k} (e^{\Sigma(X)} + \mu^2(X) - 1 - \Sigma(X))$$

  - We are comparing the learned distribution, $N(\mu(X), \Sigma(X))$, with a unit normal distribution, $N(0, 1)$
- Combined Loss

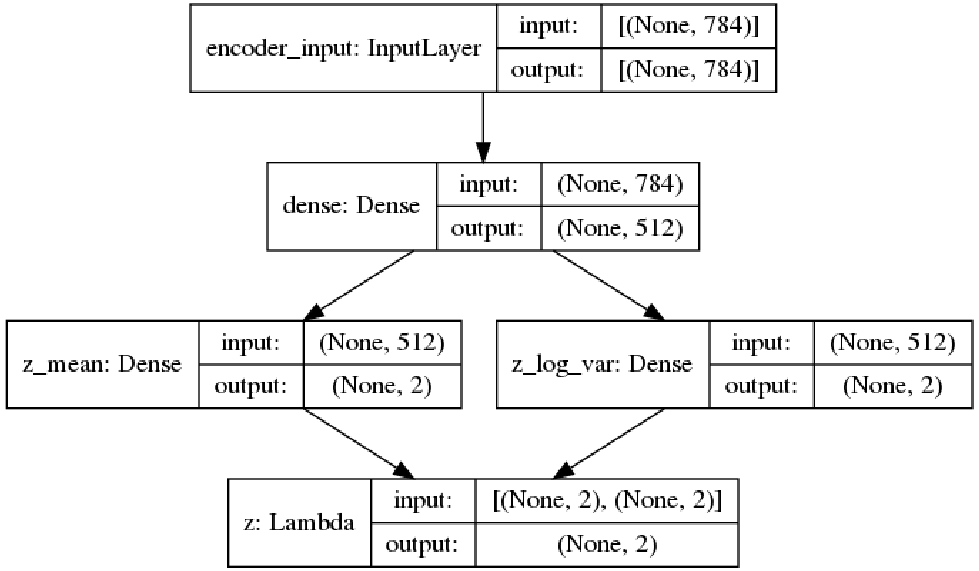
$$L = L_{recon} + D_{KL}$$

An Example

- See ***CAB420_Encoders_and_Decoders_Example_4_VAE.ipynb***
- Our data
  - MNIST
- Our task
  - An autoencoder, reconstruct the original sample

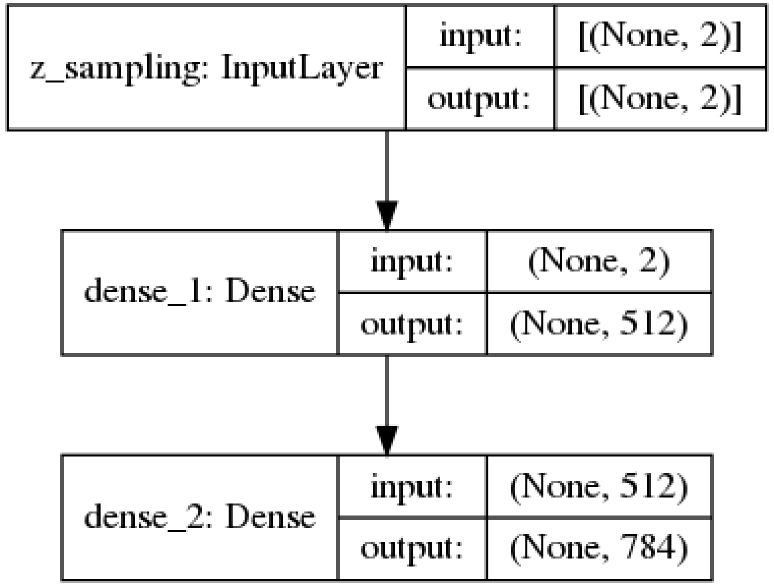
Encoder

- Very simple
  - Vectorised input
  - One common dense layer
  - Braches to learn
    - Mean
    - Variance
  - Sampling layer
    - Take the mean and variance and add a random value to "sample" from the learned distribution



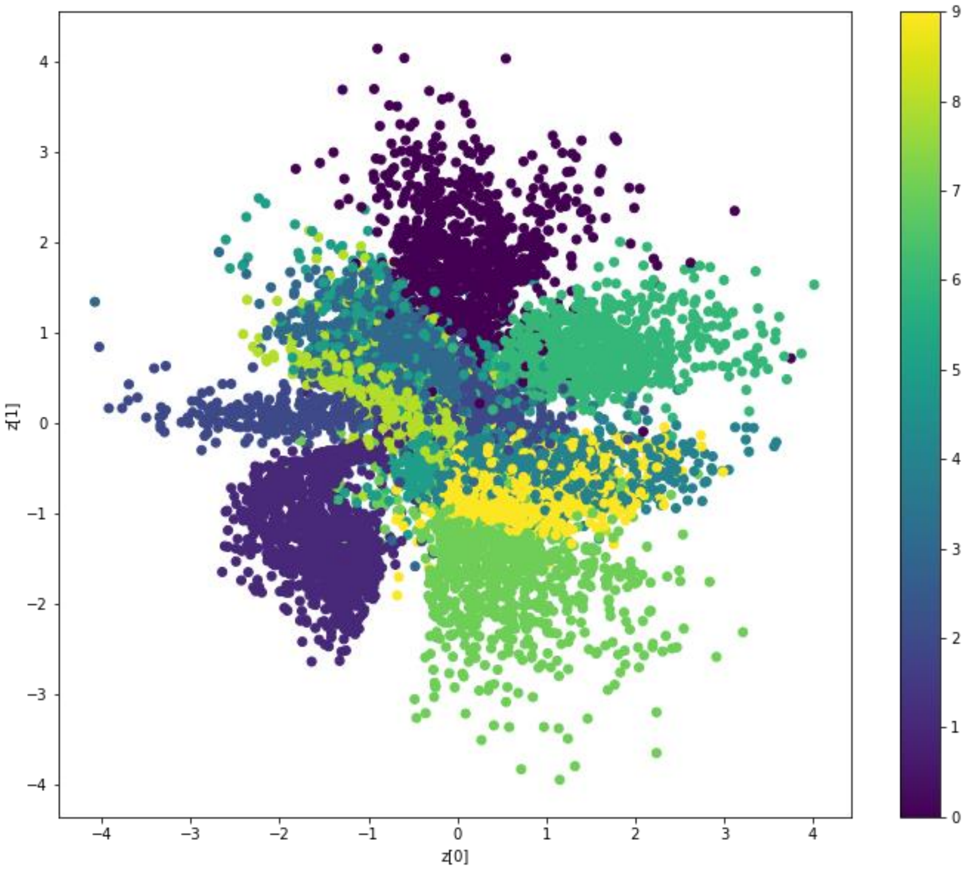
Decoder

- Also, very simple
  - Two dense layers to reconstruct original sample
  - Reconstructing from the "sampled" value
- Encoder and Decoder trained end-to-end
  - Like a regular autoencoder



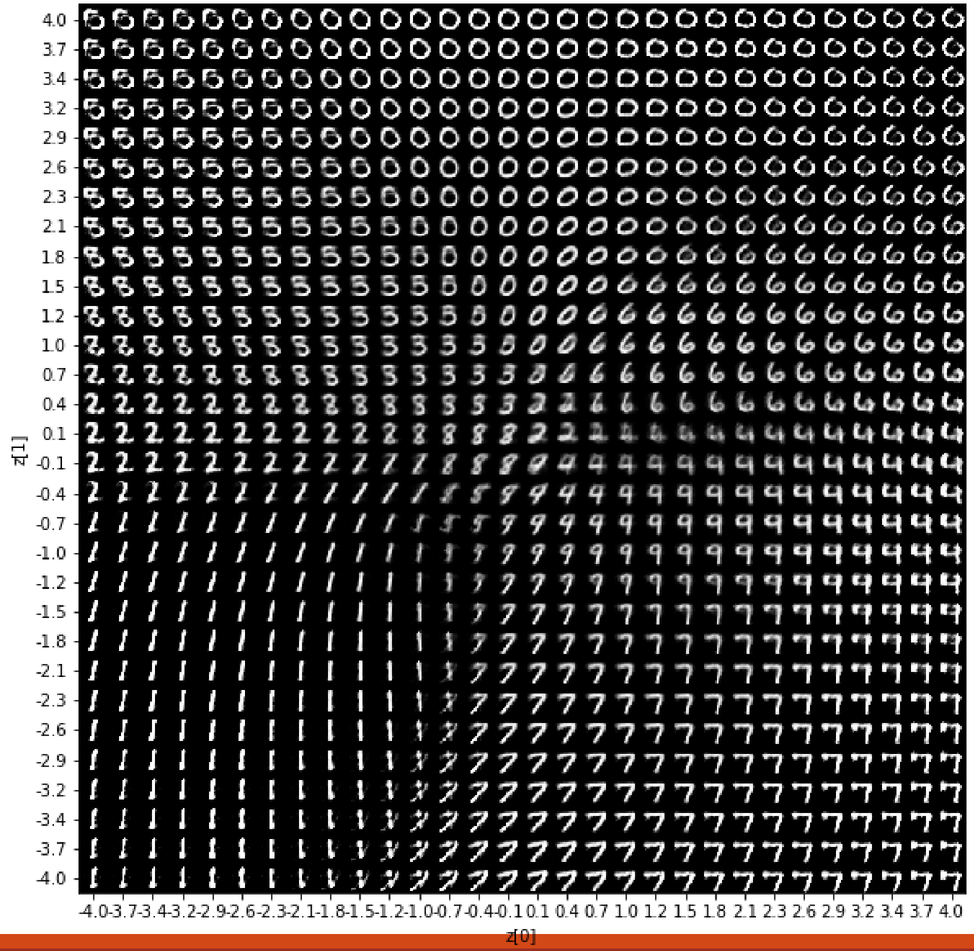
Embedding Space

- Our target distribution is a unit normal distribution
  - Mean of 0
  - Std.dev of 1
- Classes are separated
  - But with no space between them
  - One class blends into the next
  - Model is using the entire feature space
- Other embedding plots we've seen usually contain large spaces
  - Often this is our aim, to separate the classes

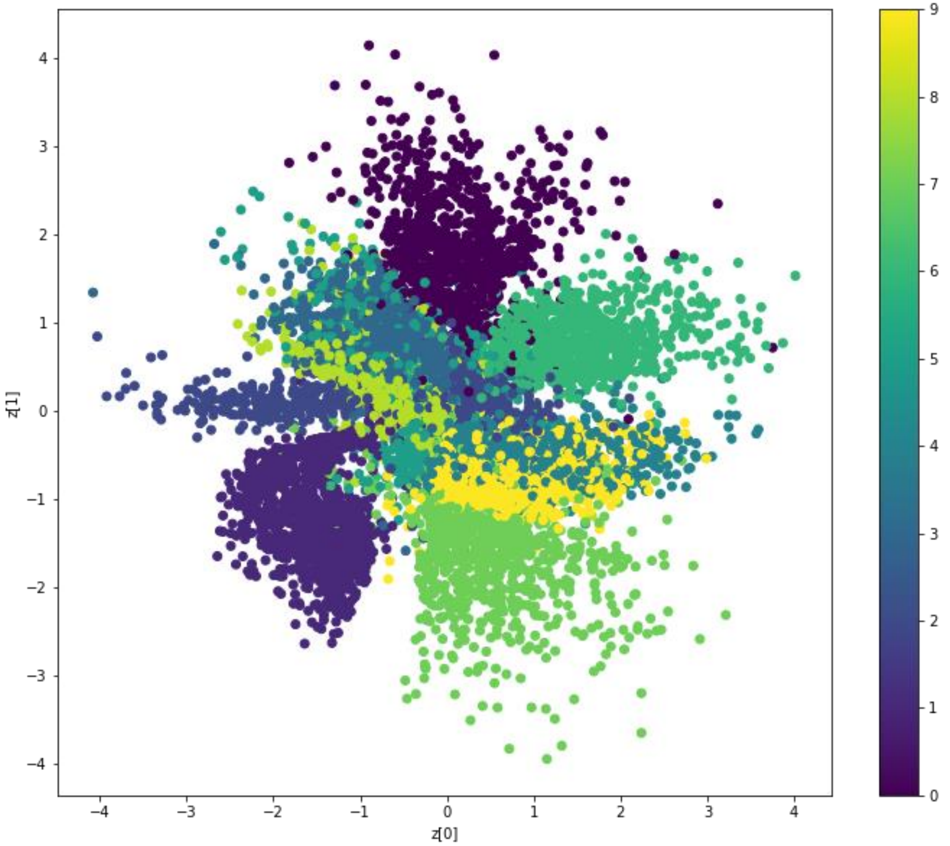


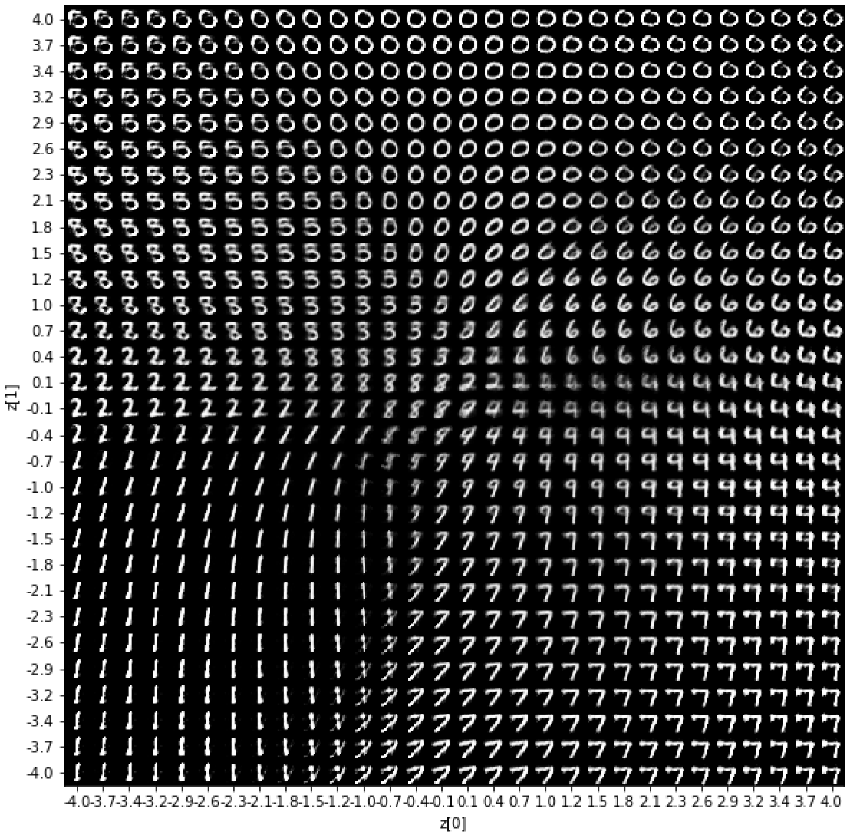
Sampling from the latent space

- We can sample from the learned distribution to create new data samples
  - We can also sample in a uniform grid to see how our classes are distributed about the feature space
  - At class boundaries, one number warps into another



Sampling from the latent space





Considerations

- We've learnt a 2D VAE
  - We've done this for visualisation
  - Richer representations, and better reconstructions are possible with larger networks and bigger representations
- Could use convolution layers to further improve the network
  - Similar to our other autoencoders, but our representation is captured by a mean and a variance

CAB420: Encoder and Decoder Summary

AND OTHER MUSINGS

Encoders and Decoders

- Encoders: given some data
  - Compute a compact representation of that data
- Decoders: given a compact representation
  - Expand out to a data sample
- Auto-encoders aim to reproduce their input at the output via a compressed representation
  - Bottleneck layer
- Encoder-Decoders can be used to do other things
  - Pixel to pixel transforms
- Variational Auto-Encoders (VAE) extend autoencoders by learning a continuous latent space
  - Generative model, i.e. we can sample from the model to create new data

Encoders and Decoders

- Autoencoders get more compact as we go towards the middle
  - Smaller layers, fewer filters, etc
  - This is important for compression, but if that's not our aim, we don't need to do this

Multi-Task and Semi-Supervised Learning

- Neural networks are very adaptable
  - Can do multiple things at the same time
  - Works best if they’re related
- Don’t need data for all tasks all the time
  - Semi-Supervised Learning
  - Need to track which samples we have which data for and mask the loss as needed
- Can adjust loss weights
  - More important tasks can be given higher weights
  - Can use to compensate for missing data

Multiple Inputs

- Just like we can have multiple outputs, we can have multiple inputs
  - Have an encoding stage for each
  - Merge some intermediate features
  - Have another learning stage on the combined representation

Other uses of Encoders and Decoders

- Many other forms of encoder-decoder in machine learning
  - Semantic Segmentation (see additional example)
    - Given an input, produce a segmented output that labels each sample with its class
    - Land use classification, classify an aerial image into building, road, grass, etc.
- Can encode multiple inputs to one output
  - For land use classification, encode input RGB and elevation map
- Can decode to multiple outputs
  - For land use classification, estimate land use and elevation from a single RGB

Generative Models

- Very large topic in Machine Learning
- Generative Adversarial Networks (GANs) have demonstrated great performance for a multitude tasks
  - Generator takes noise and synthesises an input
  - Discriminator receives a real or fake (from the generator) input and tries to determine if it’s real
  - Networks compete with each other
    - Generator trying to fool the discriminator
    - Discriminator trying to correctly determine what’s real and what’s fake
  - The conditional GAN (cGAN) provides the generator with extra stimulus
    - Generate data conditioned on some other thing