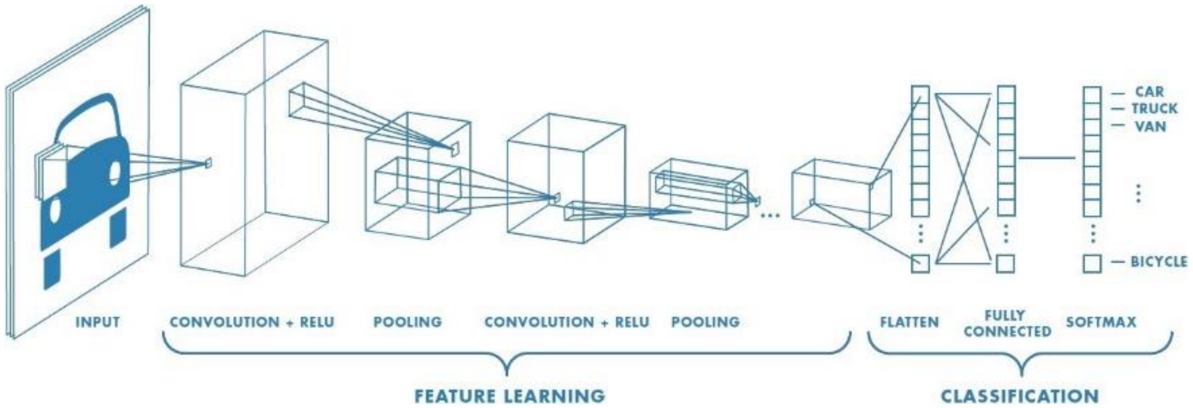
CPSC 340: Machine Learning and Data Mining

Autoencoders and Multi-Label

Fall 2022

Last Time: Convolutional Neural Networks

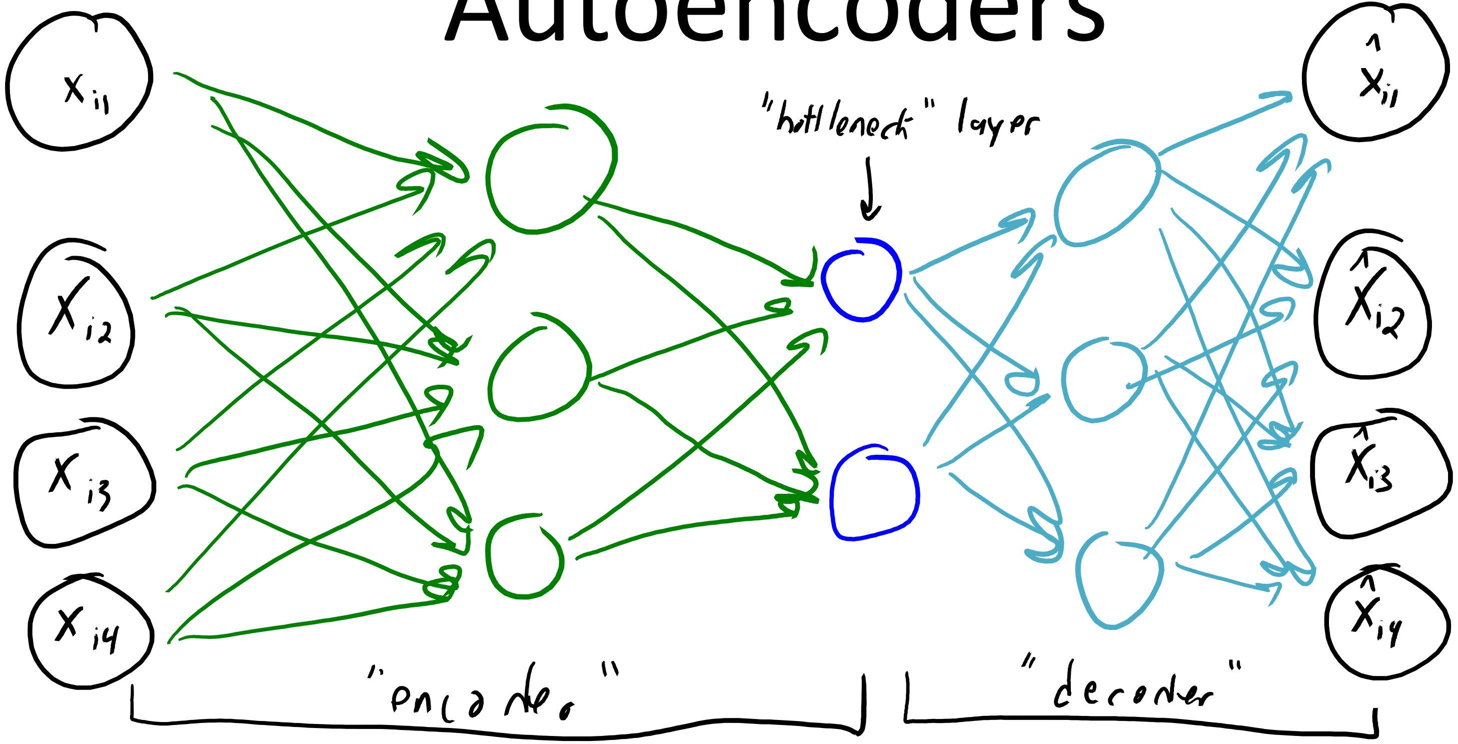
- We discussed convolutional neural network:
  - Neural networks where layers perform several convolutions.
  - Drastically reduces number of parameters and computation time.
  - Gains a degree of translation invariance (“object can appear anywhere”).



- ImageNet: Millions of labeled images, 1000 object classes.
  - Led to popularization of CNNs and deep learning across computer vision.
  - Led to many insights about how to train CNNs and construct architectures.
    - ImageNet + CNNs is arguably most influential computer vision work of all time.

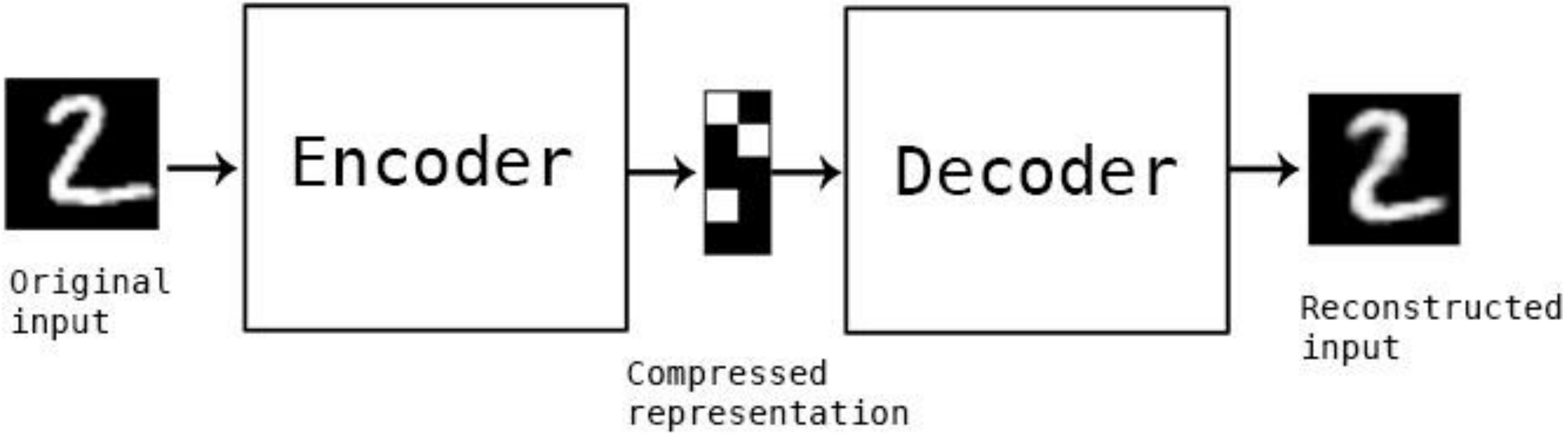
https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Autoencoders



- Autoencoders are neural networks with same input and output.
  - Includes a bottleneck layer: with dimension 'k' smaller than input 'd'.
  - First layers "encode" the input into bottleneck.
  - Last layers "decode" the bottleneck into a (hopefully valid) input.

Autoencoders

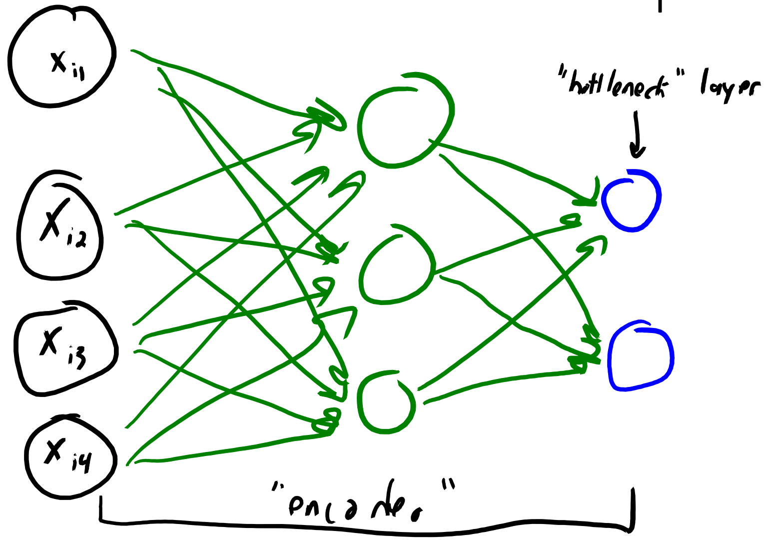


- This is an unsupervised learning method.
  - There are no labels 'y_i'.
- Relationship to principal component analysis (PCA):
  - With squared error and linear network (no non-linear 'h'), equivalent to PCA.
    - Size of bottleneck layer gives number of latent factors 'k' in PCA.
  - With non-linear transforms: a non-linear/deep generalization of PCA.

https://blog.keras.io/building-autoencoders-in-keras.html

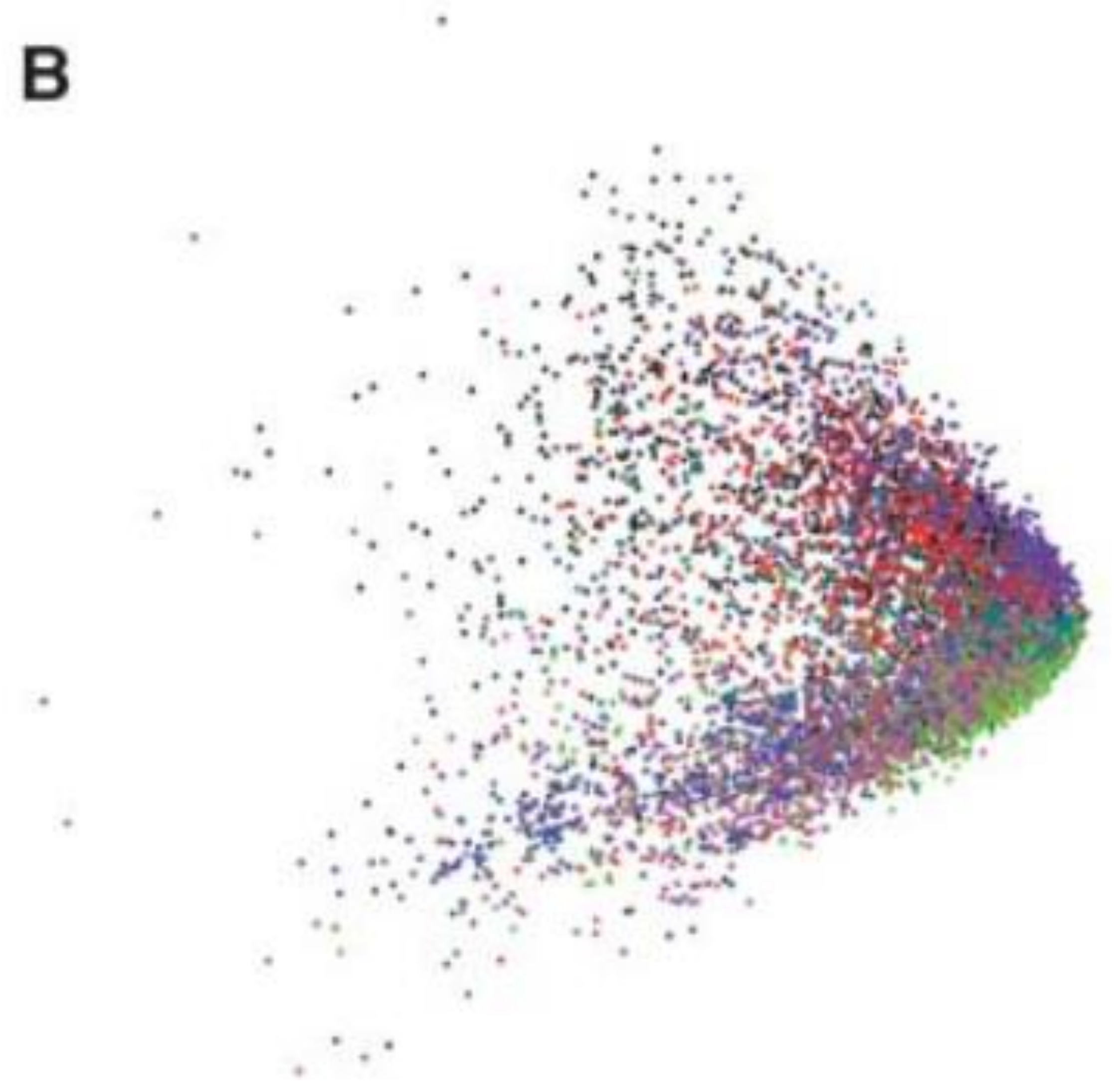
Encoder as Learning a Representation

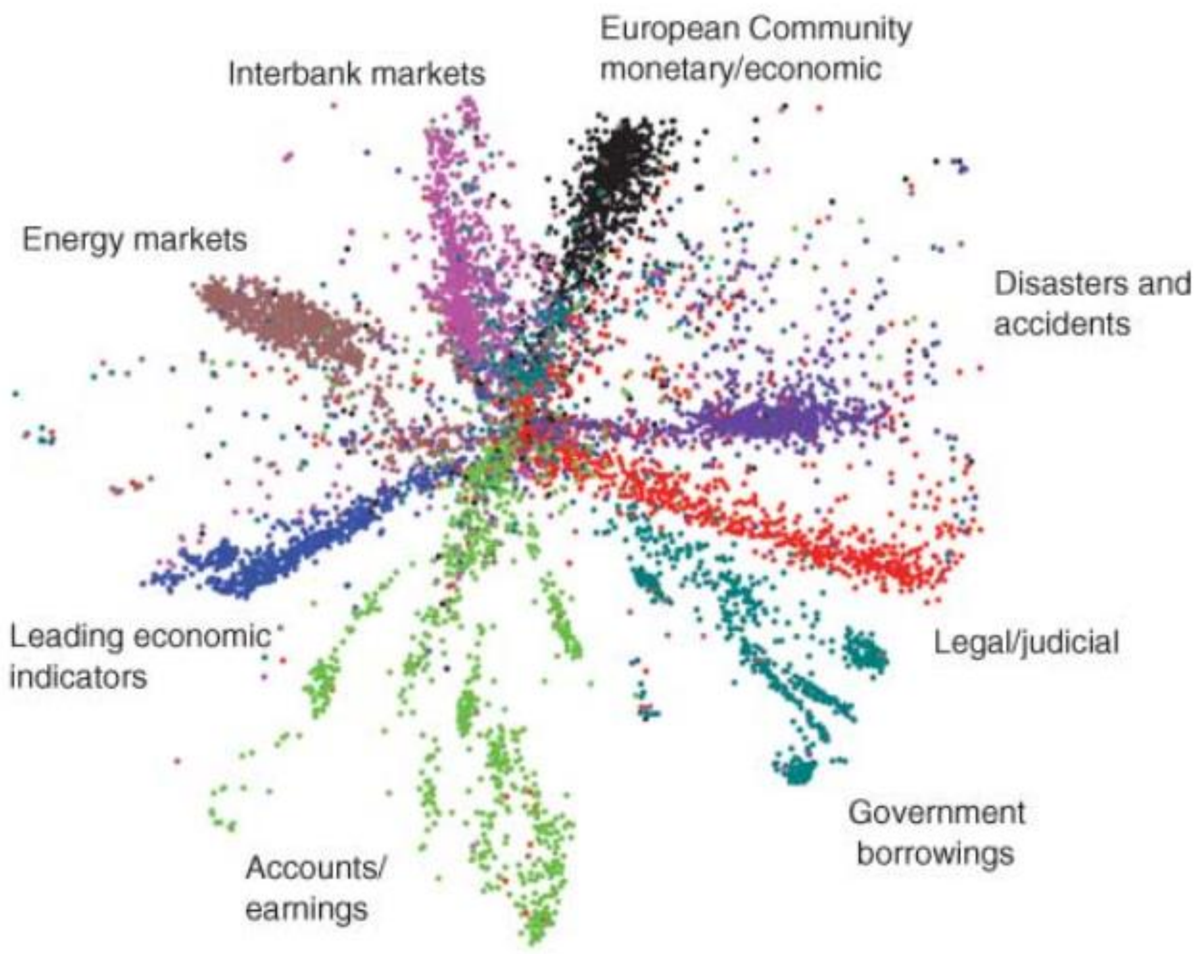
- Consider the encoder part of the network:
  - Takes features ‘x_i’ and makes low-dimensional ‘z_i’.



- Can use encoded z_i for the usual latent-factor tasks:
  - Compression, visualization, interpretation.
  - Can add a supervised ‘y^i’ to final layer of trained autoencoder, fit with SGD.
    - Called “unsupervised pre-training” (often easy to get a lot of unlabeled data).

PCA vs. Deep Autoencoder (Document Data)



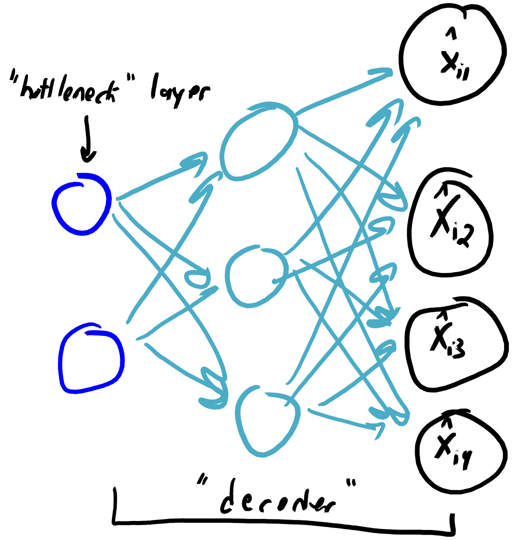


https://www.cs.toronto.edu/~hinton/science.pdf

(this was before t-SNE came out)

Decoder as Generative Model

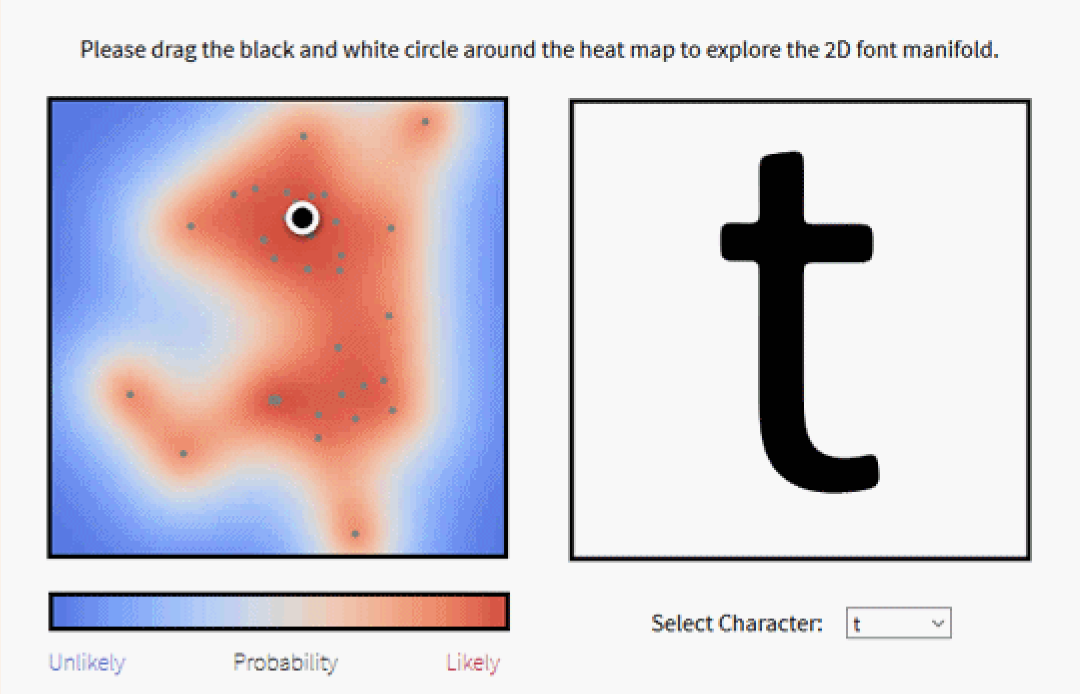
- Consider the decoder part of the network:
  - Takes low-dimensional 'z_i' and makes features '$\hat{x}_i$'.



- Can be used for outlier detection:
  - Check distance to original features to detect outliers.
- Can be used to generate new data:
  - The 'z_i' close to training data might generate new reasonable x_i values.

Font Manifold

- Going from z_i to $\hat{x}_i$ for different fonts:



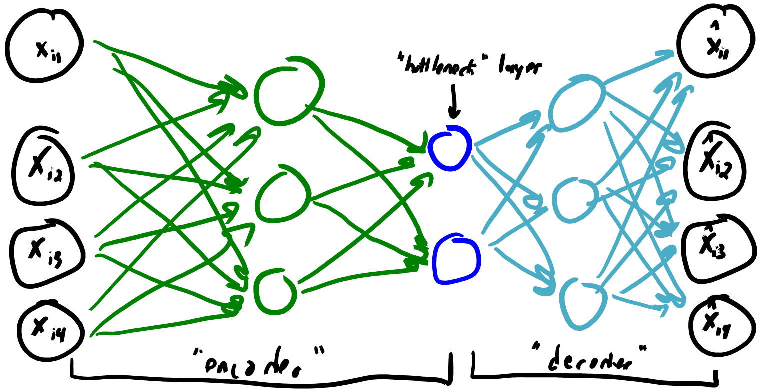
- Demo [here](#).
  - The above was not actually generated by an autoencoder.
  - But the decoder part of autoencoders is trying to do something like this.

http://entangled.systems/fragments/20160729-learning-a-manifold-of-fonts-machine-learning-research-from-2014-by-dr-neill-campbell-provides-an-interactive-exploration-of.html

Neural Networks with Multiple Outputs

- Previous neural networks we have seen only have 1 output ‘y_i’.
- In autoencoders, we have ‘d’ outputs (one for each feature).

$$\left.\begin{aligned}\hat{x}_{i1} &= v_1^T h(W^3 h(W^2 h(W^1 x_i)))\\ \hat{x}_{i2} &= v_2^T h(W^3 h(W^2 h(W^1 x_i)))\\ &\vdots\\ \hat{x}_{id} &= v_d^T h(W^3 h(W^2 h(W^1 x_i)))\end{aligned}\right\} \quad \hat{x}_i = V h(W^3 h(W^2 h(W^1 x_i)))$$



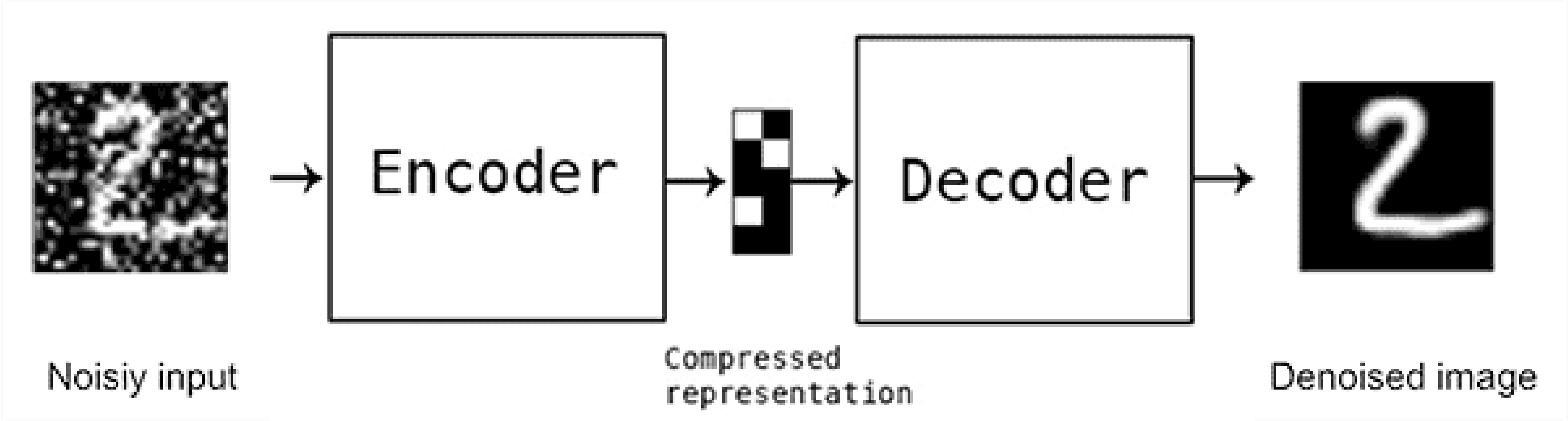
- For training, we add up the loss across all ‘j’:

$$f(W^1, W^2, V) = \sum_{i=1}^{n} \sum_{j=1}^{d} \underbrace{(\overbrace{\hat{x}_{ij}}^{= v_j^T h(W^3 h(W^2 h(W^1 x_i)))} - x_{ij})^2}_{\text{squared error for continuous } x_{ij}} \qquad f(W^1, W^2, V) = \sum_{i=1}^{n} \sum_{j=1}^{d} \underbrace{\log(1 + \exp(-\hat{x}_{ij} x_{ij}))}_{\text{logistic loss for binary } x_{ij} \in \{-1, +1\}}$$

- Fit with SGD (sampling random ‘i’), and usual deep learning tricks can be used.
  - Even though network has multiple outputs, ‘f’ is a scalar so AD works as before.
  - For images, may want to use convolution layers.

Denoising Autoencoders

- A common variation on autoencoders is denoising autoencoders:
  - Use "corrupted" inputs, and learn to reconstruct uncorrupted originals.



  - "Learn a model that removes the noise".
    - Often easy to get lots of training data, just add noise to "clean" data.
    - You can apply the model to denoise new images.
    - Does not necessarily need a "bottleneck" layer.

https://pyimagesearch.com/2020/02/24/denoising-autoencoders-with-keras-tensorflow-and-deep-learning/

Image Colourization

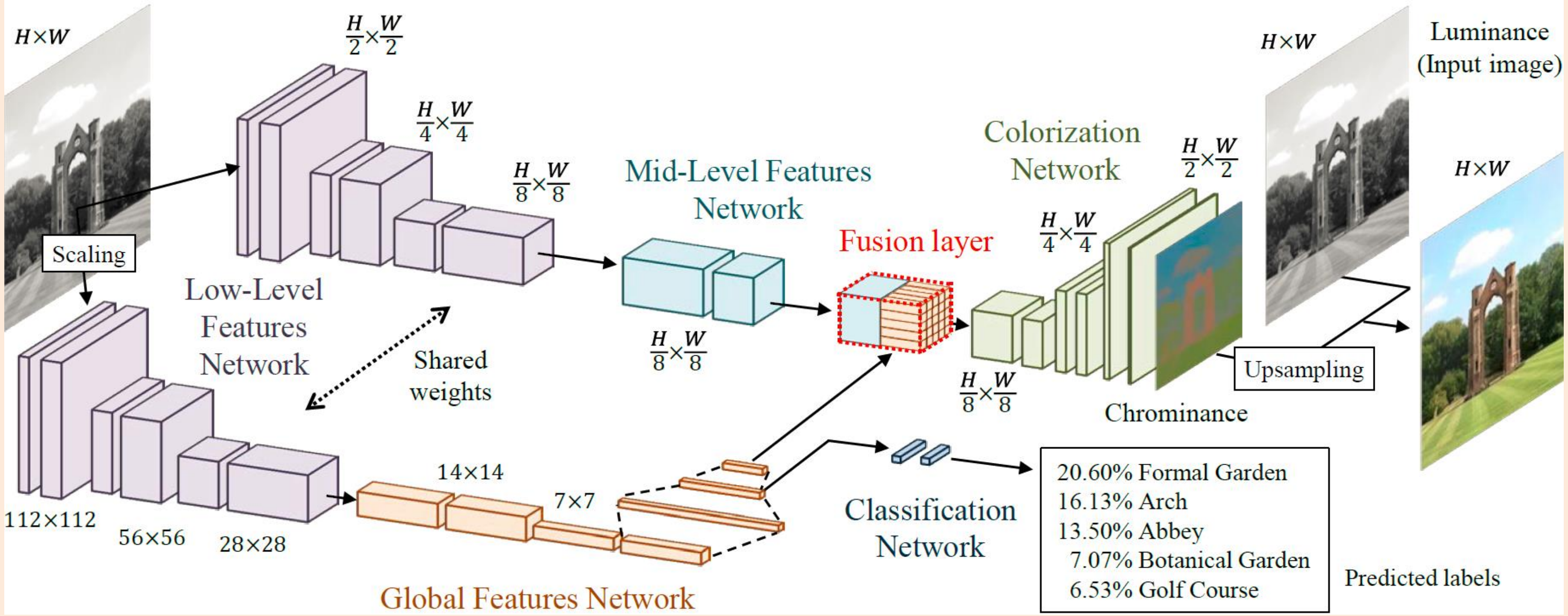
Colorado National Park, 1941 | Textile Mill, June 1937 | Berry Field, June 1909 | Hamilton, 1936

- Gallery: http://iizuka.cs.tsukuba.ac.jp/projects/colorization/extra.html
- Video: https://www.youtube-nocookie.com/embed/ys5nMO4Q0iY

http://iizuka.cs.tsukuba.ac.jp/projects/colorization/en/

Image Colourization

- Instead of noisy inputs, you use de-coloured inputs:



- Another application is super-resolution:
  - Learn to output a high-resolution image based on low-resolution images.

http://iizuka.cs.tsukuba.ac.jp/projects/colorization/en/

Next Topic: Multi-Label Classification

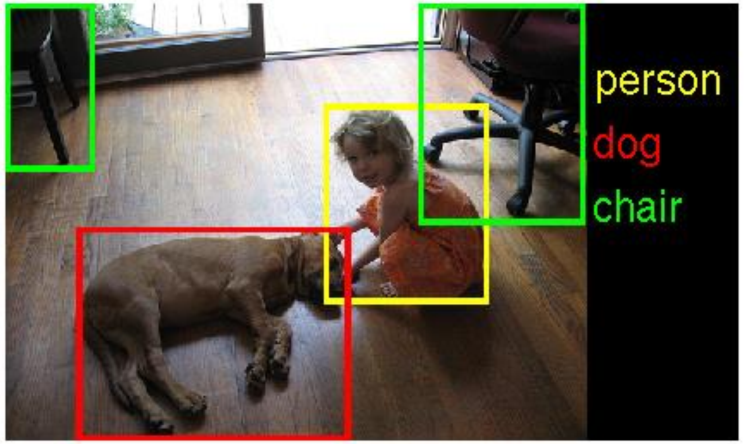
Motivation: Multi-Label Classification

- Consider multi-label classification:

$$X = \underbrace{\begin{bmatrix} \quad & & & \quad \\ & & & \\ & & & \\ & & & \end{bmatrix}}_{d} \Big\} n \qquad Y = \underbrace{\begin{array}{ccccc} \text{cat} & \text{dog} & \text{person} & \text{chair} & \text{mouse} \\ \hline \end{array}}_{} $$

$$Y = \underbrace{\begin{bmatrix} 1 & -1 & 1 & 1 & 1 \\ -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 \\ 1 & 1 & -1 & 1 & -1 \\ -1 & -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 & 1 \end{bmatrix}}_{k} \Big\} n$$

- Which of the ‘k’ objects are in this image?
  - There may be more than one “correct” class label.



http://image-net.org/challenges/LSVRC/2013/

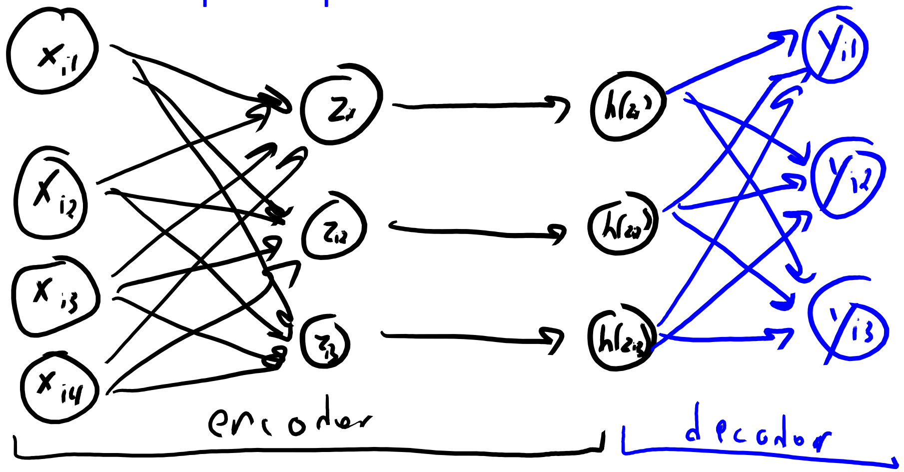
Independent Classifier Approach

- One way to build a multi-label classifier:
  - Train a classifier for each label.
    - Train a neural network that predicts +1 if the image contains a dog, and -1 otherwise.
    - Train a neural network that predicts +1 if the image contains a cat, and -1 otherwise.
    - …
  - To make predictions for the 'k' classes.
    - Apply all each label's binary classifier.
    - Predict all the resulting +1 values as the set of labels.

- Drawbacks:
  - Lots of parameters: k*(number of parameters for base classifier).
  - Each classifier needs to "relearn from scratch".
    - Each classifier needs to learn its own Gabor filters, how corners and light works, and so on.
    - A lot of visual features for "dog" might also help us predict "cat".

Encoding-Decoding for Multi-Label Classification

- Multi-label classification with an encoding-decoding approach:
  - Input is connected to a hidden layer.
  - Hidden layer is connected to multiple output units.



$$\hat{y}_{i1} = v_1^T h(Wx_i)$$

$$\hat{y}_{i2} = v_2^T h(Wx_i)$$

$$\hat{y}_{i3} = v_3^T h(Wx_i)$$

- Prediction: compute hidden layer and activations, compute vector of outputs, take sign element-wise:

$$\text{sign}(Vh(Wx_i))$$

- Number of parameters and cost is O(dk + km) for ‘m’ classes and ‘k’ hidden units.
  - If we trained a separate network for each class, number of parameters and cost would be O(dkm) (‘W’ for each class).
- Might have multiple layers, convolution layers, and so on. And no need to have a “bottleneck” layer.

Encoding-Decoding for Multi-Label Classification

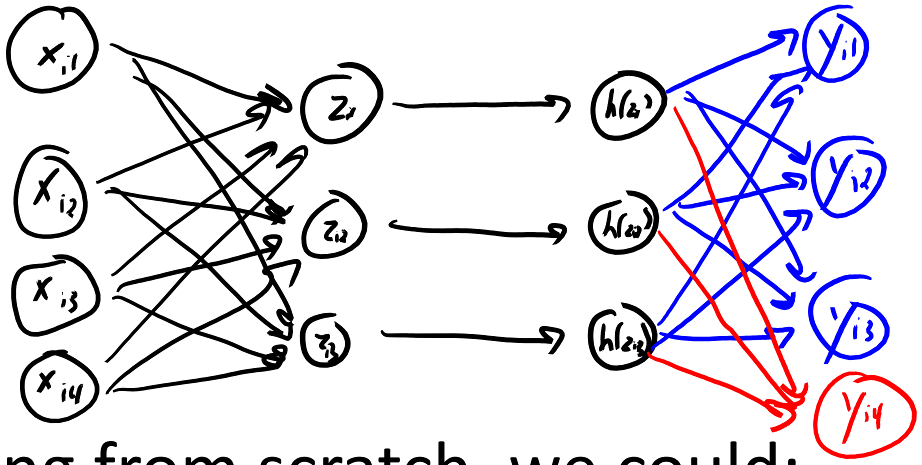
- Using sigmoid likelihood, negative log-likelihood we optimize for MLE:

$$f(W,V) = \sum_{i=1}^{n} \sum_{c=1}^{k} \log(1 + \exp(-y_c^i v_c^T h(Wx^i)))$$

- Use backpropagation or AD to compute gradient, train by SGD.
  - You randomly sample a training example 'i' and compute gradient for all labels.
  - The updates of 'W' lead to features that are useful across classes.
  - The updates of 'V' focus on getting the class labels right given the features.
- Important:
  - Above we are assuming independence of labels given the last layer.
  - But the last layer can reflect dependencies.
    - If "dog" and "human" are frequently together, this should be reflected in the hidden layer.
      - For example, $\hat{y}_{ic}$ for "human" might be higher when we have a high $\hat{y}_{ic}$ value for "dog".

Pre-Training for Multi-Label Classification

- Consider a scenario where we get a new class label.
  – For example, we get new images that contain horses (not seen in training).



- Instead of training from scratch, we could:
  – Add an extra set of weights v_{k+1} to the final layer for the new class.
  – Train these weights with the encoding weights 'W' fixed.
    - This is a simple/convex/easy logistic regression problem.
    - If we already have "features" that are good for many classes, we may be able to learn a new class with very-few training examples!

Pre-Training for Multi-Label Classification

- Using an existing network for new problems is called "pre-training"
  – Typically, we start with a network trained on a large dataset.
  – We use this network to give us features to fit a smaller dataset.
    - "Few-shot learning".

- Depending the setup, you may also update 'W' and the other 'v_c'.
  – Useful if you have a lot of data on the new class.
  – In this case, would typically mix in new examples with old ones.

- Increasing trend in vision and language to using pre-training a lot.
  – No need to learn everything about vision/language for every task!