1a)

**Shift Invariance**

Works regardless of image position on the frame

f(x) = 1

f(s(x)) = 1

**t**

**Shift equivariance**

Output shifts in the way as the object

fi(x)= 1 if pixel i ∈ object, 0 otherwise

fi(s(x))= 1 if pixel i ∈ object, 0 otherwise

Alternatively

s(f(x) = f(s(x))

1b)

CNNs are approximately shift equivariant due to convolution.

CNNs are approximately shift invariant because of pooling/striding.

1c)

The goal of the auxiliary losses is to predict the same label as the final network would predict but by using the module’s output. Due to the ‘vanishing gradient issue’ that happens in large networks (pre batch normalization) it is difficult/slow to train in first layers. This helps augment that and facilitates training throughout the entire network.

1d)

1x1 convolution performs convolution on a single feature of the input but across all the channels. It acts like a multi-layer perceptron and can reduce the number of channels in a network without losing any data.

s

1e)

* 5x5
* 3x3
* 7x7
* 4x4

1f)

Output neuron = 1 x1

Layer 3 = 3x3

Layer 2 = 5x5

Layer 1 = 7x7

Input = 9x9

Hence, support = 9x9 = 81

4-layer complexity

(Assume padding of 1)

4 = Layers

9 = 3x3 kernel

W = Image width

H = Image height

F = number of feature maps

C = Number of channels of the input image

(Multiplicative operations: 4\*9 \* W \* H \* C = 36 \* W \* H \* C \* F + 36 WHF \* (3F))

(Assuming H = W = N : 36 \* N^2 \* (CF + 3F^2)

I think we assume input C channels, and F output channels after the first filter

d

And C -> F for the 1 layer replacement

1-layer complexity

(Assume padding of 4 to keep image size)

81 = 9x9 kernel

W = Image width

H = Image height

F = number of feature maps

(Multiplicative operations: 81 \* W \* H \* C \* F)

(Assuming padding 1 and H=W=N, then roughly 9\*9\*(N-6)^2 = 81\*(N-6)^2 operations?)

# Side note

4-layer has 36 parameters whereas 1 layer has 81 parameters (excluding biases)

2a)

Above answer is not “deep learning” based, use Alex Net

Alternative answer:

Use a VAE to perform dimensionality reduction on the data. Assign closest matching labels for each unlabelled data, then perform classification (on the original? Or the encoded version?)

Alternative answer:

One deep learning approach to semi-supervised learning is to have a teacher and student network. The teacher network is trained to output labels for the unlabelled data and the student network uses those labels to learn classification on the entire dataset, and possibly provide feedback on the accuracy of those labels.

See this google paper from 2020 which does this: [[2003.10580] Meta Pseudo Labels (arxiv.org)](https://arxiv.org/abs/2003.10580)

Or – alternative:

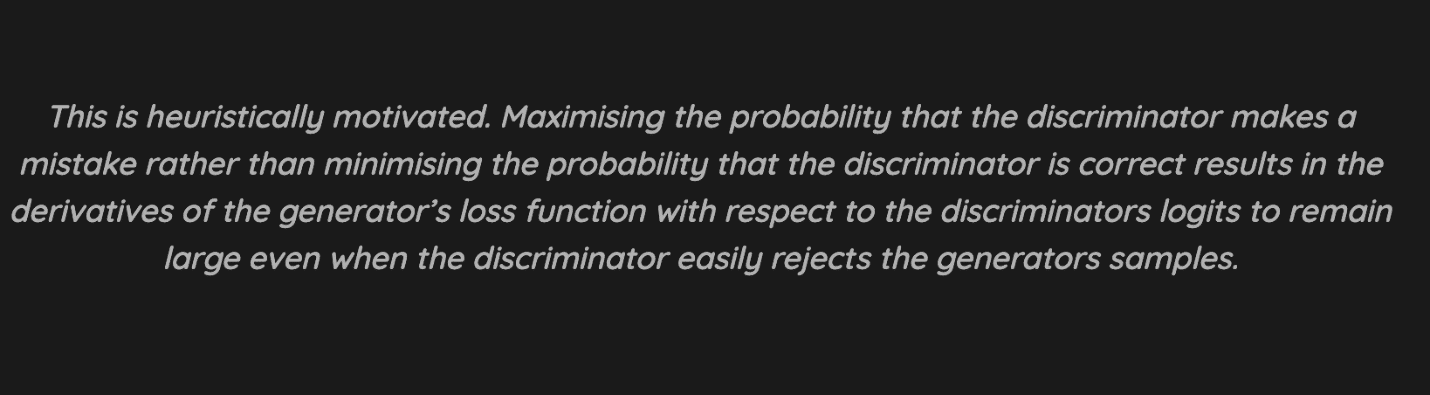
Use self-supervised learning pretraining. So initialise the weights of the network with e.g. a contrastive loss on the unlabelled data. Then finetune the network on the labelled data.

2b)

* Early on D(G(z)) is closer to 0
* -log(D(G(z)) provides higher loss and therefore stronger gradients early in training (when D(G(z)) close to 0), where it is most important (answer provided from comments)

(not because of higher loss, because of higher gradient)

[ This was also a question in the quiz and was explained heuristically]



* Optimally the value of D(G(z)) would be 0.5, since given an optimal generator, the discriminator would not be able to tell the difference between real and generated data, so it would output an average of 0.5

(Not 1 because otherwise the discriminator has collapsed and provides no more useful data)

(Answer from comments)

* Yingzhen provides mathematical proof for the equilibrium, see GAN lecture note equation (7)~(9). With Pθ(x)=Pdata(x), D(G(z)) would be 0.5

3a)

GNNs are not examinable.

Le Net graph convolutional network (😀)

<https://piazza.com/class/kf7uh4qe592wv?cid=139>

3bi)

One to Many – Musical generation, produce a piece of music from a single note, Image captioning

Many to One – Sentiment analysis, Alex Net

Many to Many – Language Translation, part-of-speech tagging for each word in a document

One to One – Traditional neural network

3bii)

Time step pooling

Average/Max

We could use the average of all the previous hidden states for the current time step.

Bidirectional

Attention?

Le Net?

3biii)

Although RNNs and LSTMs both have a hidden state that get passed between timestamps, LSTMs also have a cell state. The derivative of the hidden state will decay/explode exponentially if the network weights do not equal 1. In the cell state however, there is no exponentially fast decaying factor and it therefore does not explode/vanish.

This means it can use longer sequences (up to 50).

(Shouldn’t this at least mention the reset gate?)  
The addition of the reset gate inside the LSTM controls how much of the previous cell state is carried forwards. In backpropagation, this means the gradient decays with respect to the sigmoid output instead of the weights (which can be much smaller)

3biv)

I think this is correct but I am pretty unsure

I don’t think this is examinable, since it wasn’t taught

(I think given that we studied RNN models we should be expected to know how a LSTM works)

The input and forget gates. They jointly decide what data we want to keep and pass on to the next cell from the previous cells data and the new data that has been inputted.

OR: a shot in the dark here but maybe peephole connections (not examinable) [Peephole connection - Machine Learning Glossary](https://machinelearning.wtf/terms/peephole-connection-lstm/) ? Allow to not only pass the previous hidden state, but also the cell state.

3bv)

* Reset Gate
* Update Gate

The GRU network only passes 1 hidden state between cells. An LSTM passes 2 (the hidden state and the cell state). I.e., no cell state for a GRU.

GRUs have fewer parameters, are easier to modify and quicker to compute than LSTMs.