2a) For a convolution operation

$$\text{output size} = \frac{W - K + 2P}{S} + 1$$

where, W = input size
K = Kernel size
P = padding
S = Stride

∴ in our case

$$\text{output} = \frac{7 - 3 + 2\times 2}{2} + 1$$

$$= \frac{8}{2} + 1$$

$$= 5$$

∴ The output will be a 5×5 feature map

For o/p of same size with 5×5 filter

$$7 = \frac{7 - 5 + 2P}{1} + 1$$

$$\Rightarrow 2P = 4 \Rightarrow P = 2$$

∴ padding of size 2 will give same output size as input.

b) For Le Net

input = 1×32×32

Conv1 = 6×5×5

S2 = 2×2 pooling

Conv3 = 16×5×5

S4 = 2×2 pooling

Conv5 = 120×5×5

fc6 = 120 → 84

fc7 = 84 → 10

Calculating number of parameters

For conv1: kernel of 5×5 and bias, 6 channel output
input to conv1 has 1 channel

$\therefore$ parameters $= ((5\times5\times1)+1)\times6$

$= 26\times6$

$= 156$

For conv3: kernel of 5×5 and bias, 16 channel o/p
input to conv3 has 6 channels

$\therefore$ parameters $= ((5\times5\times6)+1)\times16$

$= 2416$

For conv5 : Kernel of 5×5 and bias, 120 channel o/p
input has 16 channels

$\therefore$ parameters = $((5\times5\times16)+1)\times120)$

$= 48,120$

For FC 6 : $120 \rightarrow 84$ & bias

$\therefore$ parameters = $(120+1)\times84$

$= 10,164$

For FC 7 : $84 \rightarrow 10$ & bias

$\therefore$ parameters = $(84+1)\times10$

$= 850$

Pooling layers don't have any learnable parameters.

$\therefore$ Total no. of parameters = $156 + 2416 + 48120 + 10164 + 850$

$= 61,706$

2) given $x_t = 3 \quad f(x_t) = 6 \quad f'(x_t) = 1$

$\eta = 0.1$

Using gradient descent we need to minimize objective function

$\therefore x_{t+1} = x_t - \alpha \cdot \frac{\partial f}{\partial x}$

$= x_t - \alpha f'(x_t)$

$= 3 - 0.1 \times 1$

$= 2.9$

No $f'(x_{t+1})$ does not guarantee that x_{t+1} is the global optimizer as it could be a local optimum

The momentum method is a possible way of counteracting the local minima and converging to a global optimizer. It uses the gradients of the previous steps as momentum to calculate the current gradient.

2d) Feature map: $\begin{bmatrix} 1 & 2 & -1 \\ 0 & 3 & 5 \\ 2 & 1 & 4 \\ 3 & 0 & -2 \end{bmatrix}$

Filter: $\begin{bmatrix} -0.5 & 1 \\ 1.5 & 0 \end{bmatrix}$

Output will have a size: $4-2+1 = 3$

$3-2+1 = 2$

$\therefore 3 \times 2$

let output be denoted by O

$O_{1,1} = 1\times(-0.5) + 2\times1 + 0\times1.5 + 3\times0 = 1.5$

$O_{1,2} = 2\times(-0.5) + (-1)\times1\times1.5\times3 + 5\times0 = 2.5$

$O_{2,1} = 0\times(-0.5) + 3\times1 + 2\times1.5 + 0\times1 = 6$

$O_{2,2} = 3\times(-0.5) + 5\times1 + 1\times1.5 + 4\times0 = 5$

$O_{3,1} = 2\times(-0.5) + 1\times1 + 3\times1.5 + 0\times0 = 4.5$

$O_{3,2} = 1\times(-0.5) + 4\times1 + 0\times1.5 + (-2)\times0 = 3.5$

$\therefore O = \begin{bmatrix} 1.5 & 2.5 \\ 6 & 5 \\ 4.5 & 3.5 \end{bmatrix}$

1a) Threshold functions give us binary values. Hence if the input is greater than the threshold all neurons with that threshold will fire hence such an activation cannot be used with multi class data.
Furthermore a threshold function does not add non-linearity to the network and hence it will limit the ability of the network to understand complex patterns. This is why threshold functions are no longer used in modern deep neural networks. Also threshold functions cannot backpropagate.

1b) Sigmoid & Tanh both face the problem of vanishing gradients. as they squish the output range, Relu does not have this problem as its gradient is exactly 1.
Secondly as Sigmoid & Tanh squish the output between 0 & 1 or -1 & 1 input values for outside this range saturate these functions. Relu does not have this problem and thus allows neurons to express stronger opinions

1c) CNNs use kernels with parameter and hence are much more memory efficient compared to fully connected neural networks especially for large inputs such as images. Moreover really large fully connected networks have huge number of parameters and are difficult to train & may result in overfitting.

The second advantage of CNNs is that they are translation shift invariant and more robust to rotation. As the kernel scans over the image it will fire as long as the desired feature is present somewhere in the image.

1d) large train accuracy & low test accuracy likely implies overfitting of the network.
The 2 things I would like to try are:
Early Stopping: Large networks tend to overfit. Early stopping prevents this as it stops the training after validation accuracy starts decreasing as this would imply the network has stopped learning features & is likely memorizing inputs.
Data Augmentation: Data augmentation randomly manipulates the data to generate new training data. This increases the availability of training data & helps the network in extracting features. As the input data increases this also makes it less likely that the network will memorize features.
A third alternative is to add dropout layers as this will force the network to learn rich features

(e) Convergence after very few epochs with low accuracy. I would like to try:

Data augmentation: It is possible that the model does not have enough data to extract meaningful features. Hence a possible solution is to carry out data augmentation such as random flips or crops or color jitter. This will increase available training data giving the model a better chance to extract features.

Preprocessing & batch normalization: Batch normalizatio & Data preprocessing is likely to help. Batch norm makes the inputs of each layer more predictable improving training speed and accuracy & in turn test accuracy. Data preprocessing such as PCA or whitening will also help as it reduces sensitivity to small weight changes making training more robust.

A third possibility is to increase model complexity as it may be the case that the model is not complex enough to extract features from the data.

f) The critical problem that LSTM solves over vanilla RNN is that of vanishing gradients. For vanilla RNNs as the weights keep getting multiplied over long sequences of input the gradients tend to vanish or explode making them difficult to train.

LSTMs solve this by introducing a gated structure. As the LSTM maintains a separate cell state which is accessed through the input & forget gates. the new input is added through the input gate to the cell state and not recursively multiplied. Hence it solves the vanishing gradients problem

g) LSTMs are recurrent while CNNs are not. given that we have a 1D signal a CNN would treat all convolutions as independent. Hence it would only look at the features convolving with a filter at a given time step independent of all previous data.
CNNs can use multiple filters to learn different features LSTMs on the other hand, being recurrent, will take into account the fact that previous time steps of the signal have some autocorrelation. Thus an lstm will be able to learn time based patterns or repeating patterns in the signal.

h) PCA is a linear transformation while VAE is capable of modelling complex non linear functions. PCA is a direct transformation while VAE can be deep neural networks. Hence they can learn much more complex patterns. Furthermore PCA features are orthogonal by design while VAE features can have corellation
A single layer autoencoder with a linear activation function would perform like a PCA since the PCA is just a linear operation on the data.

i) The KL divergence term is included in VAE to prevent overfitting of the input data in the latent space. This ensures that the latent space is well generalized with good properties to enable meaningful generation. The regularization provides completeness & continuity to the latent space. This provides a structure where points close in the latent space will generate similar outputs & any random point picked from the latent space generates meaningful outputs.

If the KL divergence term is not included the network will greedily overfit the data in the latent space to minimize reconstruction loss. This can result in cases where close by points have completely different meaning and randomly sampled point will likely give garbage output. Such a VAE will not generalize well to testing data

j) No this does not guarantee that the GAN model is well trained. This is because The GAN loss converging to 0 means that $D(x)$ is 0 and $D(G(z))$ is 1. Hence it means that the discriminator is easily able to discriminate between real and fake inputs. This would likely indicate that a convergence failure has occured in the model and so the generator is likely producing garbage output. A well trained GAN should produce realistic results and so this is the opposite of the desired outcome

3)

a) (D)

b) (A)

c) (D)

d) (C)

e) (A) (B)

f) (B) (C)

g) (D)

h) (B)

i) (B)

j) (A) (C)