1.

Algorithms don't give exact output but the probability over a class as

a. A model can never be 100% sure like if we train a model to recognise cats and then give a test input as a picture of a dog it will still find some similarities in cats and the given picture however the answer is most likely to be tending to zero.
b. Another reason can be the errors in labelling the train set like in the above example if we have some non cat pictures labeled as cat pictures due to human error. This can result in the model giving a probability and not exact outcome.

2. In SGD we take one input at a time and train the model through it (i.e. the case of minibatch size 1) . However, the catch in this is we get noisy output, to avoid that we use the momentum concept i.e. the moving average.
We have hyperparameters α learning rate and β which controls the exponentially weighted average.

$$V\,dw\,(at\ t\ iteration) = \beta V\,dw\,(at\ t-1\ iteration) + (1-\beta)dw$$
$$V\,db\,(at\ t\ iteration) = \beta V\,db\,(at\ t-1\ iteration) + (1-\beta)db$$

And , $W = W - \alpha V\,dw$ $b = b - \alpha V\,db$

To get the it's analogy with the momentum in physics we take $-V\,dw$ as Force, W as the current momentum. After α time the momentum will update as

New momentum = Old + time*Force

I.e. $Wnew = Wold - V\,dw * \alpha$

3.

a. Mini batches are used to increase the speed of the model when the training set is very large. In this we divide the training set in smaller mini batches and run each mini batch as an epoch and thus train the model in comparatively smaller time. Not using mini batch and training a model will take a lot of time as well have to do several epochs with the same huge training set. While if we take mini batch size as 1 we update the model so frequently that it becomes computationally more expensive and result's in noisy result.
b. On initializing weights symmetrically all the neurons in a hidden layer will start doing the exact same thing thus there would be no use of having multiple neurons in a layer. Therefore, we need to initialize weights randomly.
c. Overfitting happens when the model is too perfect for training data and will thus not generalize the problem and thus will not work fine for other data. Thus we regularize the model by either randomly dropping some parts of the network (dropout regularization) or by decreasing the value of weights (L2 regularization). Other ways are early stopping and data augmentation.
d. Batch normalization is a technique of standardizing the inputs to a layer for each mini-batch. It scales the input in every minibatch such that the mean of the data is zero and standard deviation is one. It thus stabilizes the gradient i.e. less exploding or vanishing values.

4. kernel size or F = 3x3
   P = 1
   S = 2
   InputShape = N1xN2x3
   Output shape = (InputShape - F +2P)/S +1
   And since the output has 8 channels , therefore Output Shape = $\frac{N1+1}{2} * \frac{N2+1}{2} * 8$
   Number of Weights = InputShape * F * OutputShape
   = N1 * N2 * 3 * 3 * 3 * 8 * $\frac{N1+1}{2} * \frac{N2+1}{2}$
   Number of bias = OutputShape = 8 * $\frac{N1+1}{2} * \frac{N2+1}{2}$
   Total number of parameters = (N1 * N2 * 3 * 3 * 3 + 1) * 8 * $\frac{N1+1}{2} * \frac{N2+1}{2}$