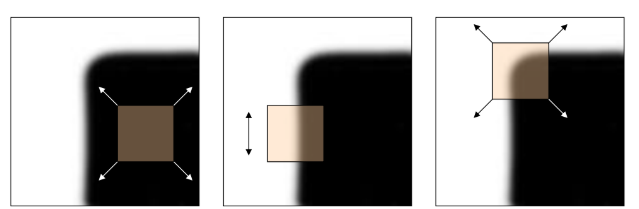
1. **Harris Corner Detection**

**i) Basic Idea in Corner Detection:**

* Recognize corners by looking at small window.
* Shift in any direction to give a large change in intensity. (brightness)

****

1. **(2) (3)**

**(1)** If the window is on a flat region (like a wall), the intensity doesn’t change much in any direction.

**(2)** If the window is on an edge (like the side of a rectangle), the intensity changes a lot in one direction but not much in the perpendicular direction.

**(3)** If the window is on a corner, the intensity changes a lot in all directions.

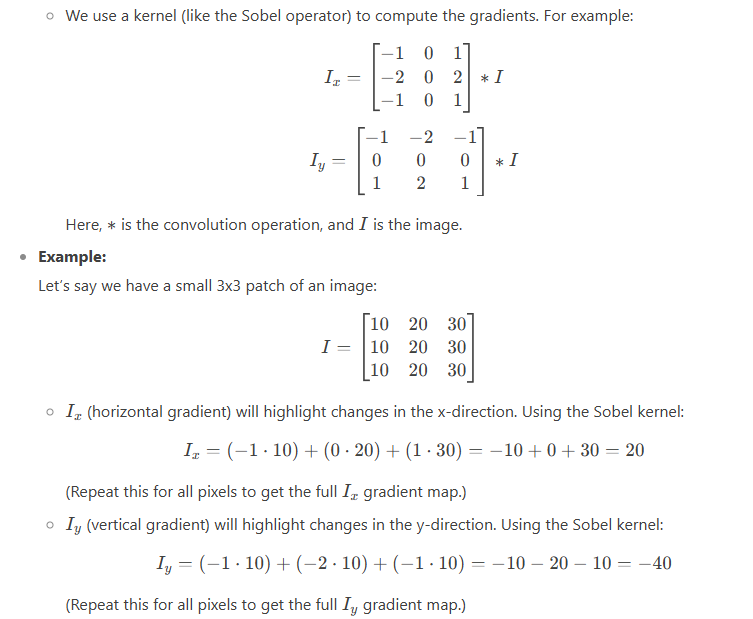
**ii) Harris Detection Algorithm**

**Step 1: Convert the Image to Grayscale**

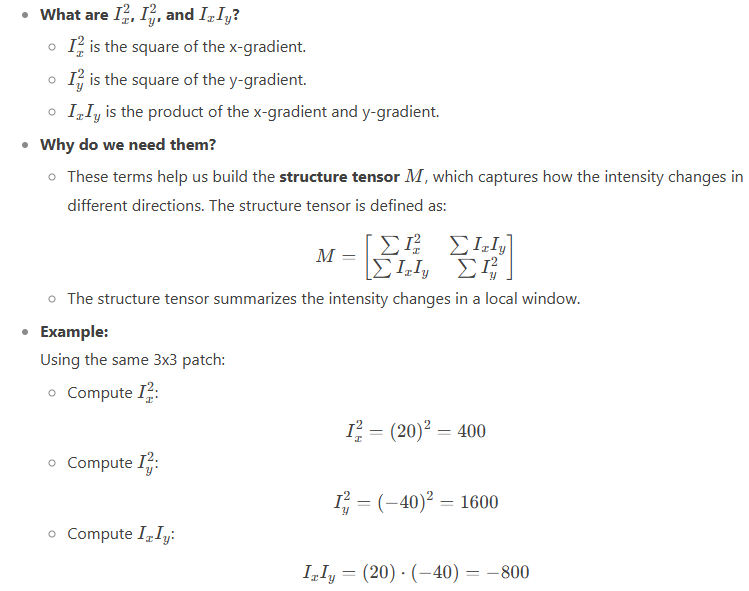
Corners are based on intensity changes, so we first convert the image to grayscale (black and white) to simplify the process.

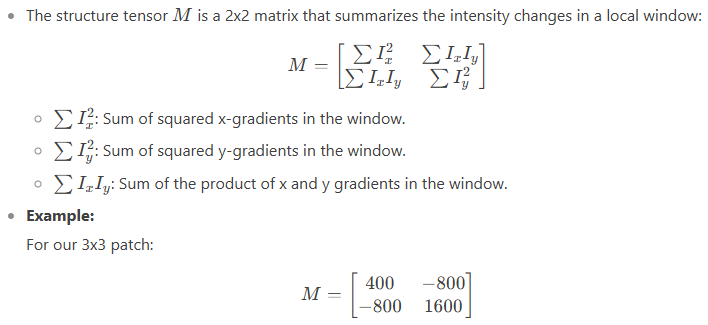
**Step 2: Compute Intensity Gradients**

* **What are**
  + is the gradient (derivative) of the image intensity in the x-direction (horizontal).
  + is the gradient (derivative) of the image intensity in the y-direction (vertical).
  + These gradients tell us how quickly the intensity changes as we move in the x or y direction.
* **Why do we need them?**
  + Corners are points where the intensity changes significantly in both directions. By computing  and , we can measure these changes.
* **How to compute them?**

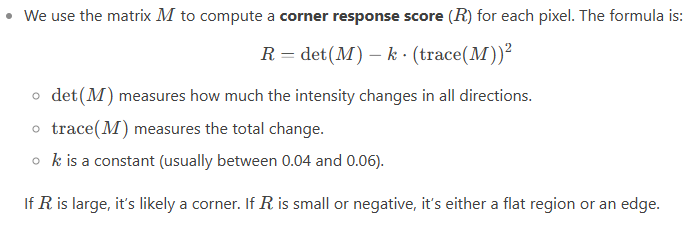
****

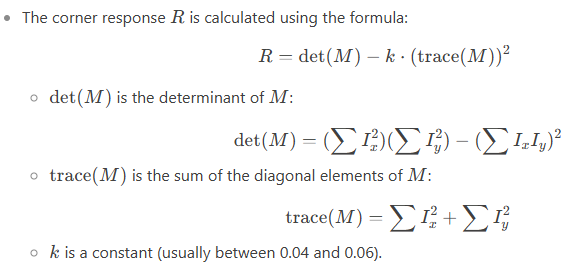
**Step 3: Create a Matrix to Capture Intensity Changes**

****

****

**Step 4: Measure Corner Response**

****

****

**Step 5: Thresholding**

We set a threshold value. If **R** is above this threshold, we mark it as a corner.

1. **Flat Region:**

R is close to 0, or R=0

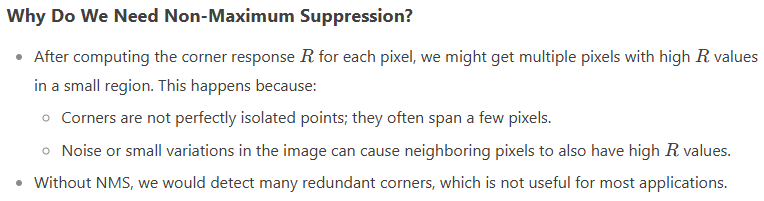
1. **Edge:**

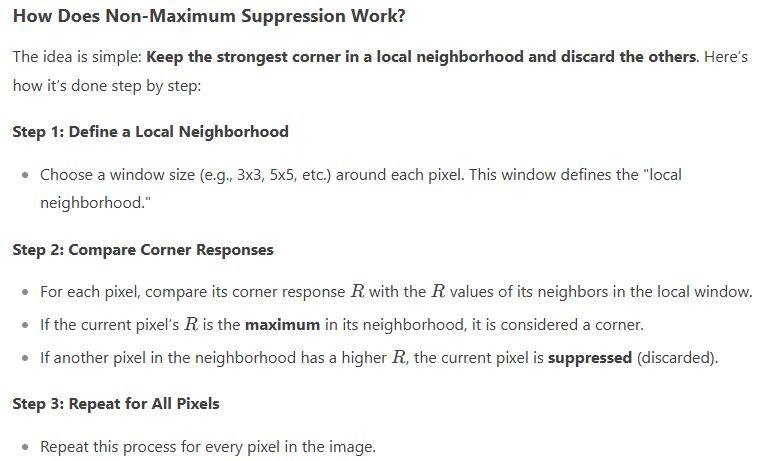
R is Negative, this happens when one of Ix​ or Iy​ is large, but the other is small (intensity changes significantly in one direction but not the other).

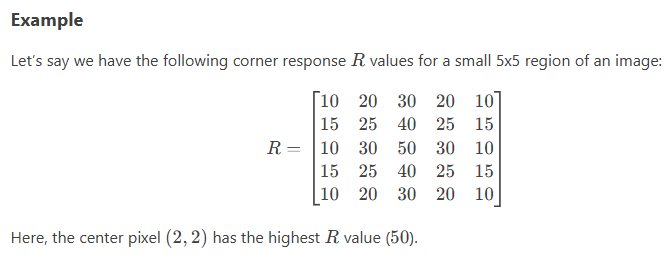
1. **Corner:**

R is large and positive (greater than threshold). This happens when both Ix and Iy are large (intensity changes significantly in both directions).

**Step 6: Non-Maximum Suppression**

****

****

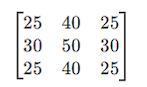
****

**Step 1: Define a Local Neighborhood**

* Let’s use a 3x3 window for simplicity

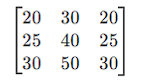
**Step 2: Compare Corner Responses**

* For the center point (2, 2), compare it’s R=50 with it’s 8 neighbors.

****

Since 50 is the maximum in this neighborhood, the center pixel is kept as a corner.

* For the pixel (1, 2) with R=40, compare it’s R=40 with it’s neighbors:



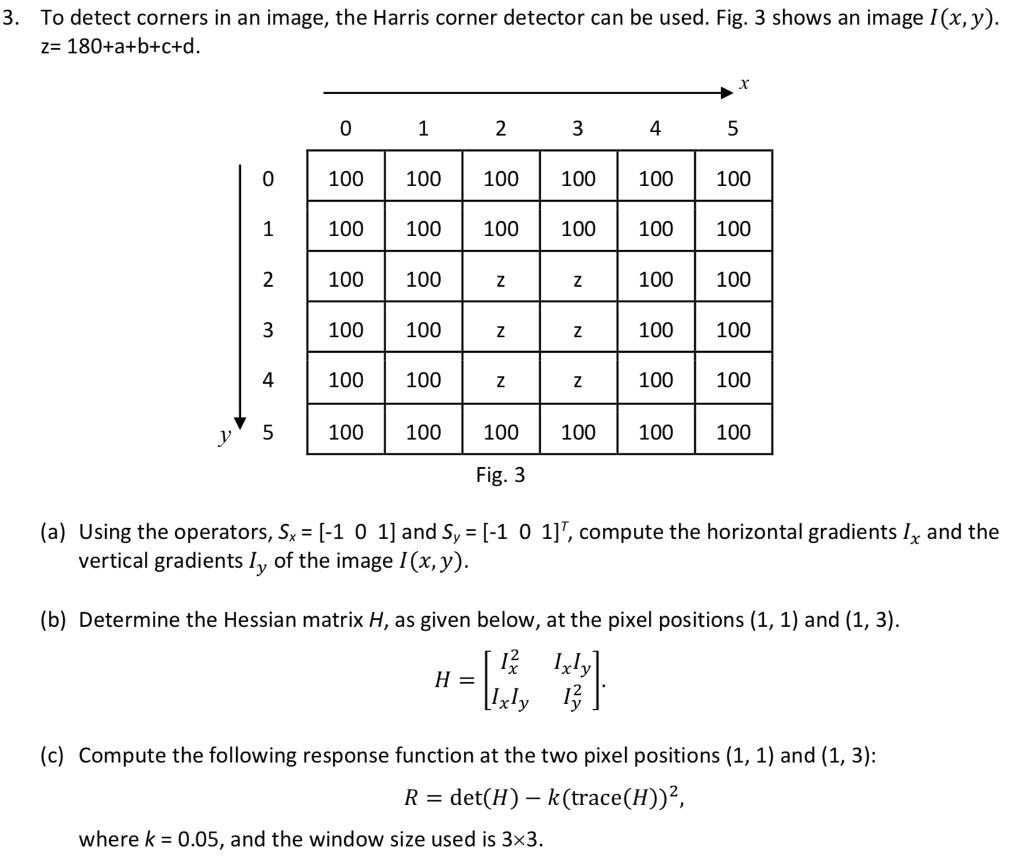
The center pixel (2, 2) has R=50, which is higher than 40. So the pixel (1, 2) is suppressed.

* Repeat this process for all the pixels.

**Step 3: Result after NMS**

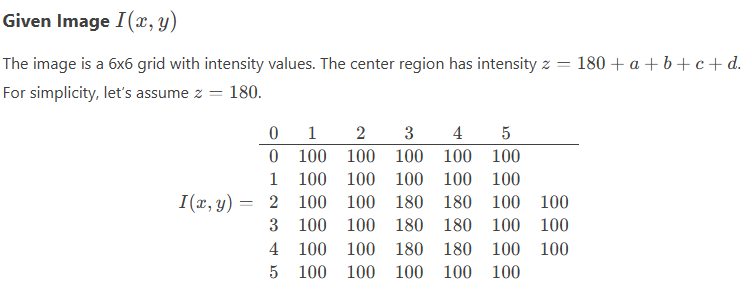
After applying NMS, only the strongest corners remain. In this example, only the center pixel (2, 2) with R=50 is kept as a corner. All other pixels in its neighborhood are suppressed.

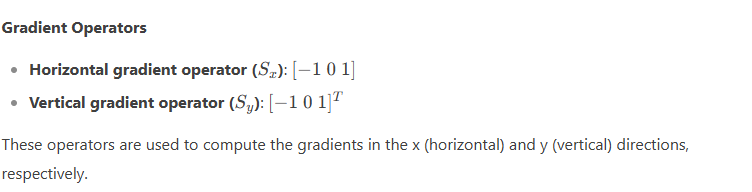
**Example Questions of Harris Corner Detection:**

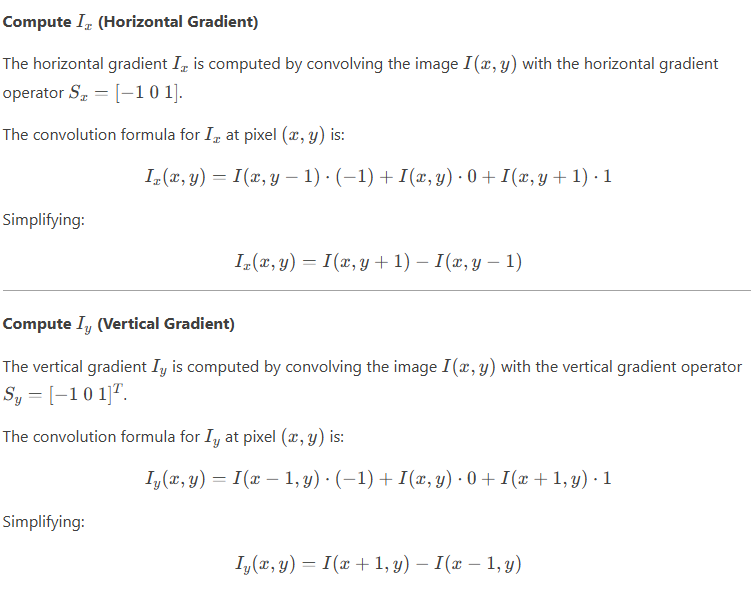
**Question 1:**

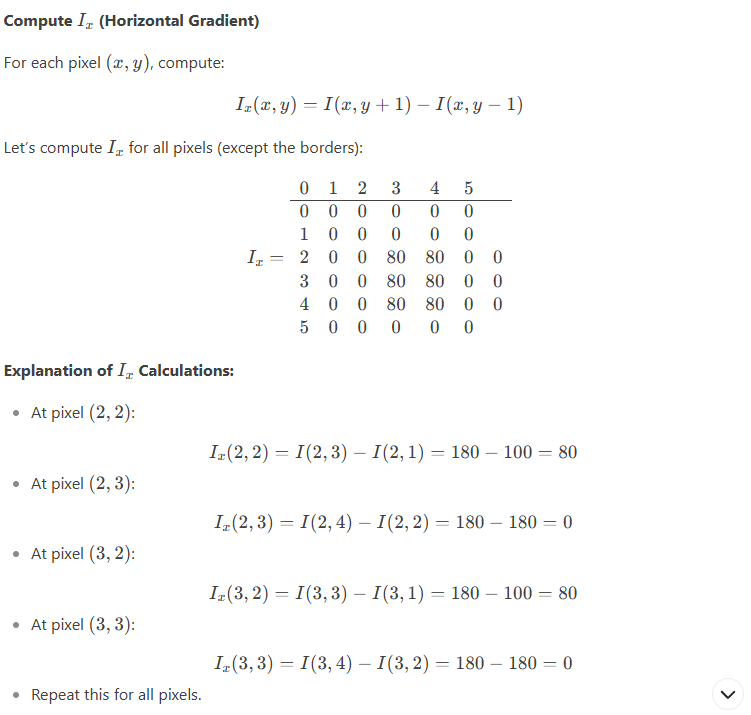
**Solution:**

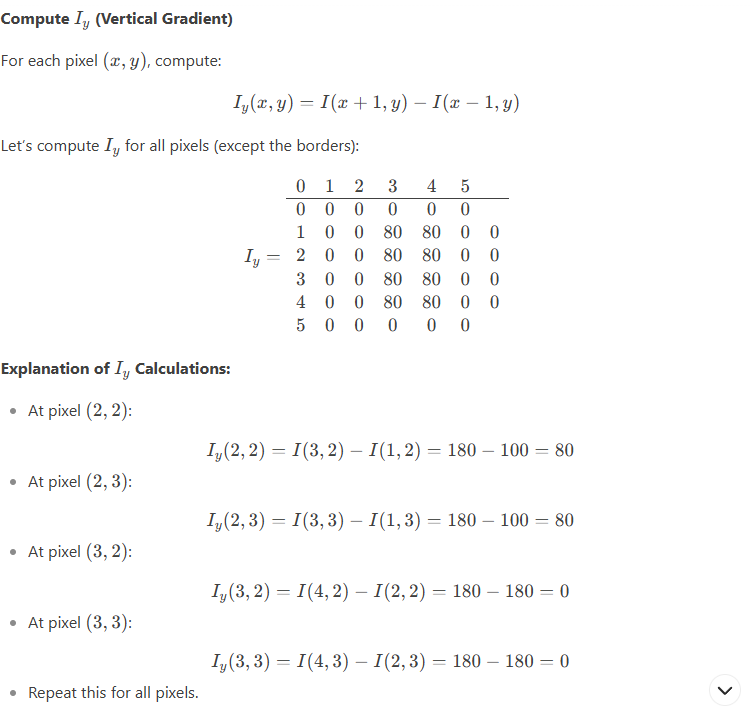
****

****

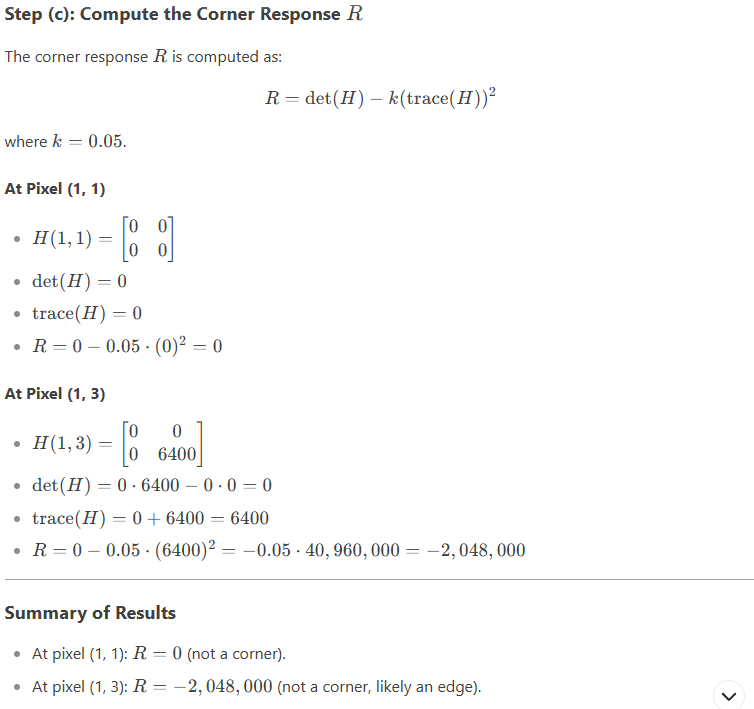
****

****

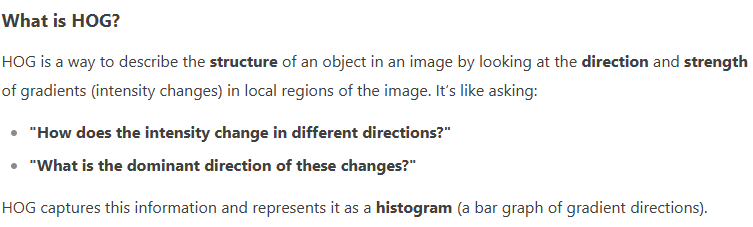
****

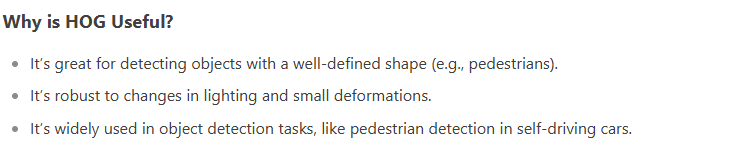
****

****

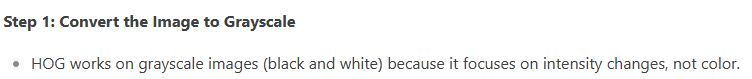
****

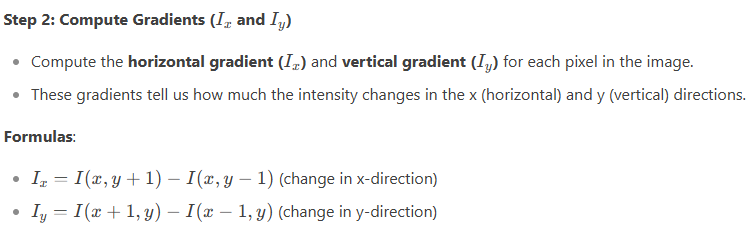
1. **Histogram of Oriented Gradients (HOG)**

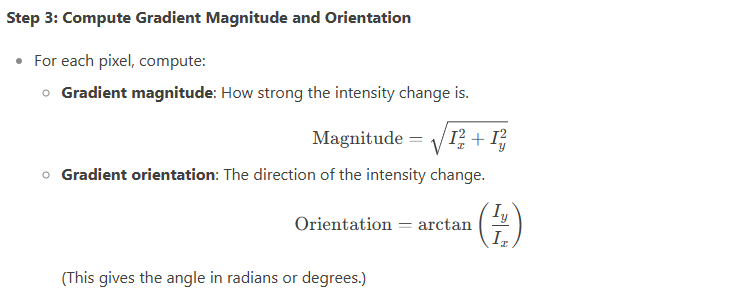
****

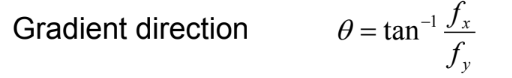
****

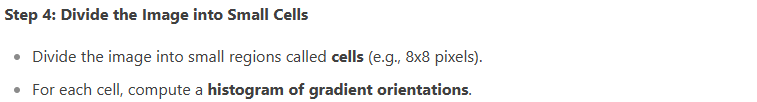
****

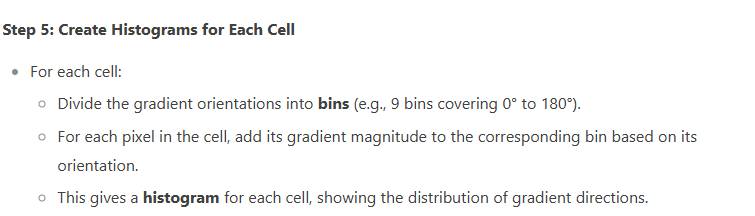
****

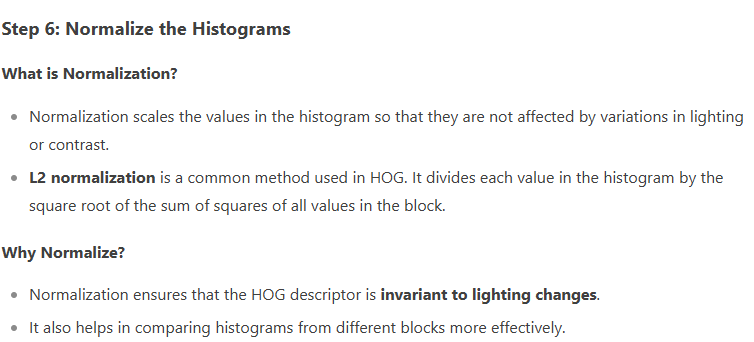
****

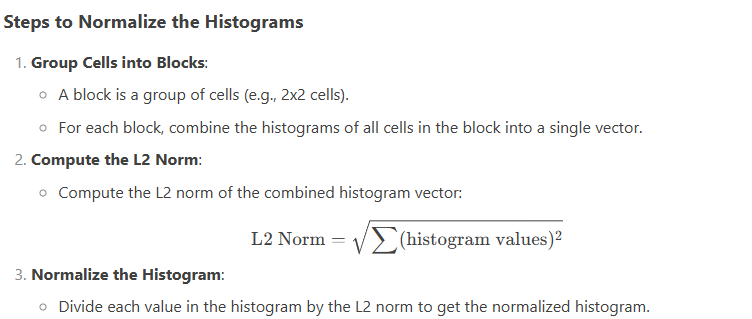
****

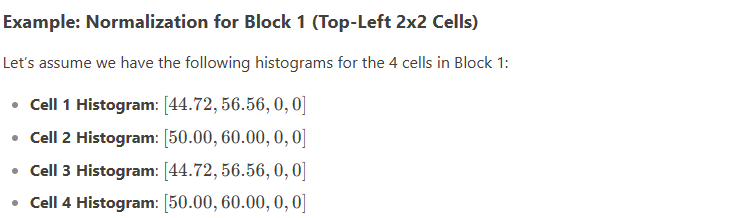
****

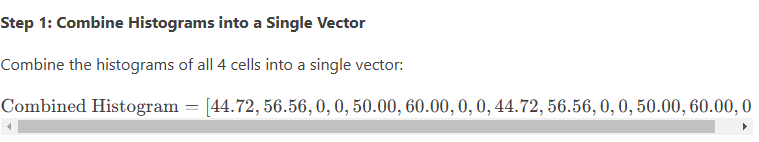
****

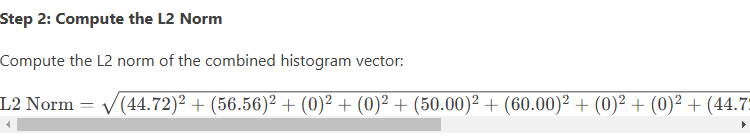
****

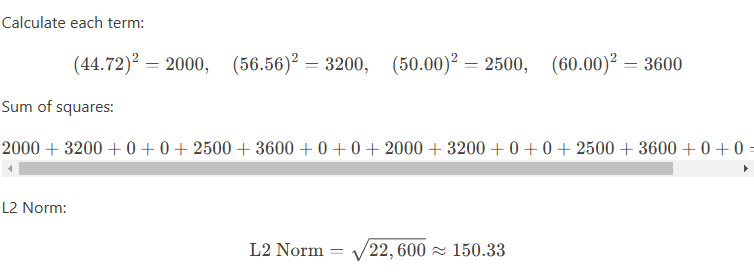
****

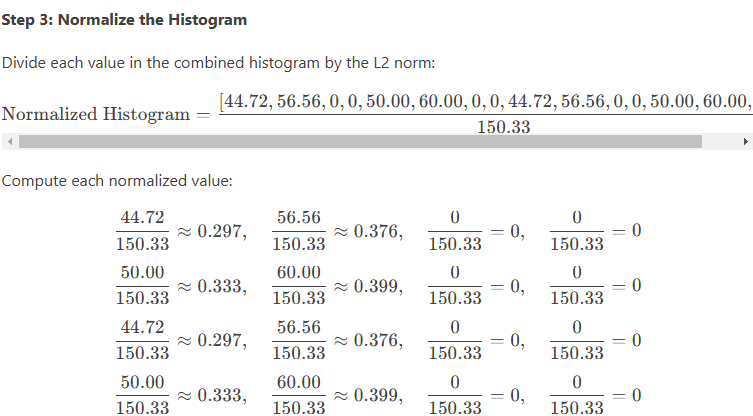
****

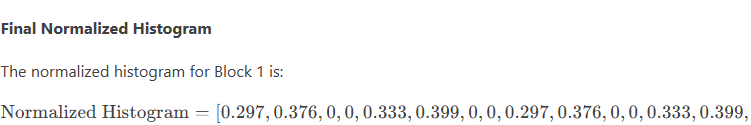
****

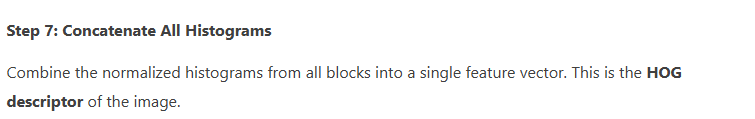
****

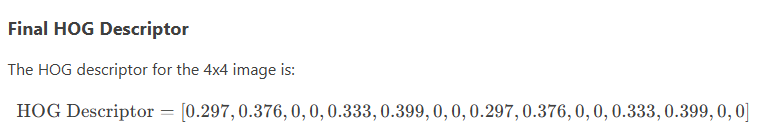
****

****

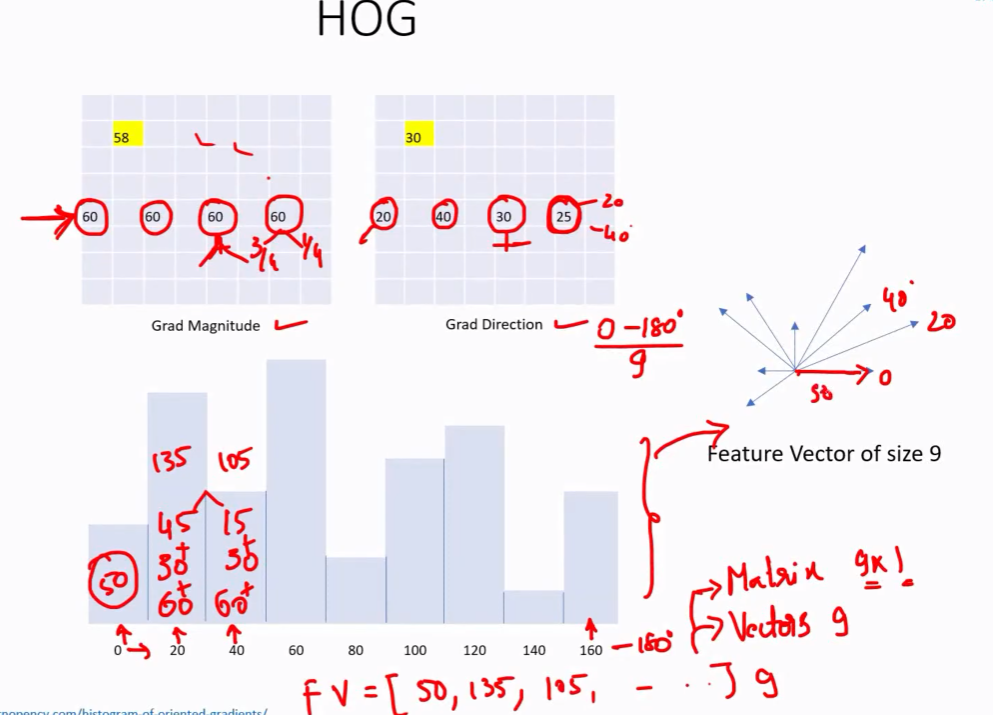
****

****

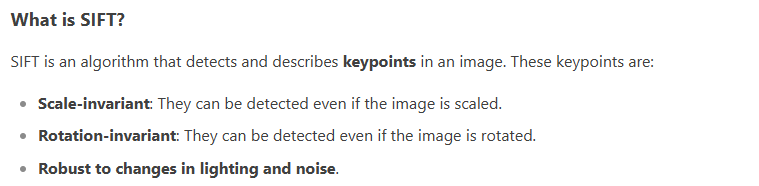
****

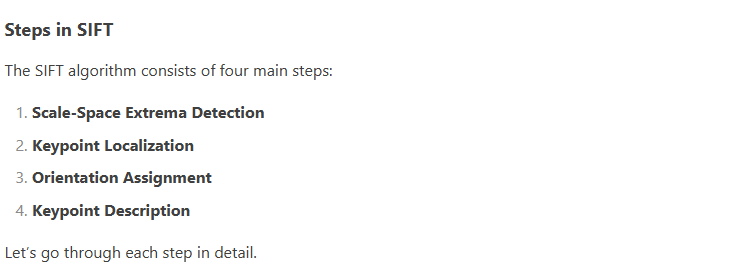
****

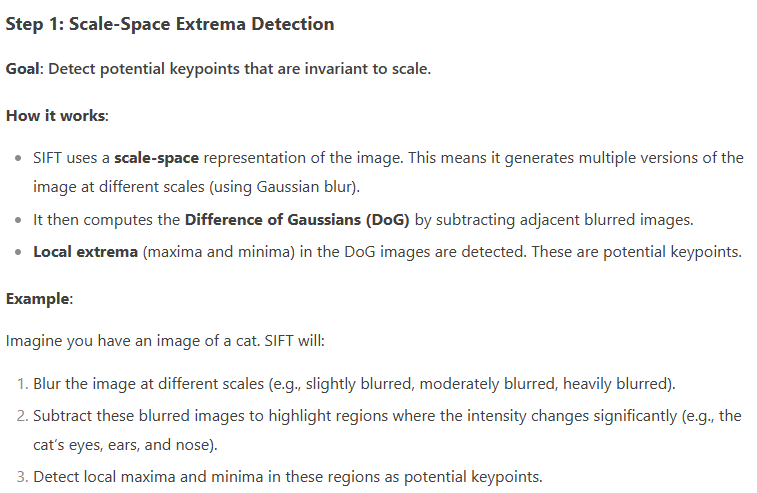
**Note: this HOG Descriptor contains values of single block of image only, we have to combine all the normalized values of histograms of every block.**

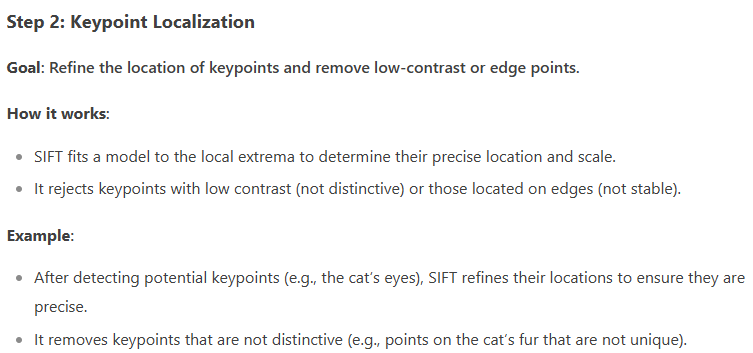
****

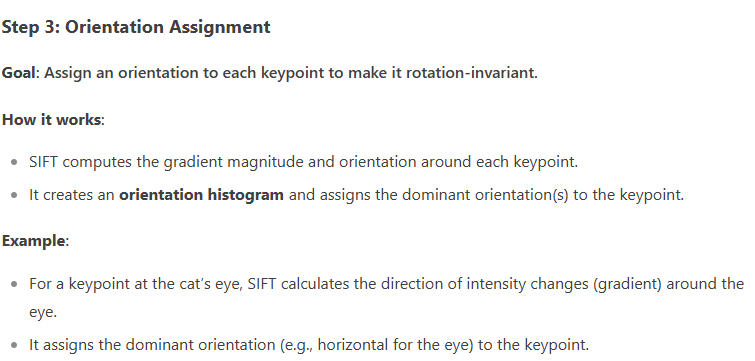
1. **SIFT**

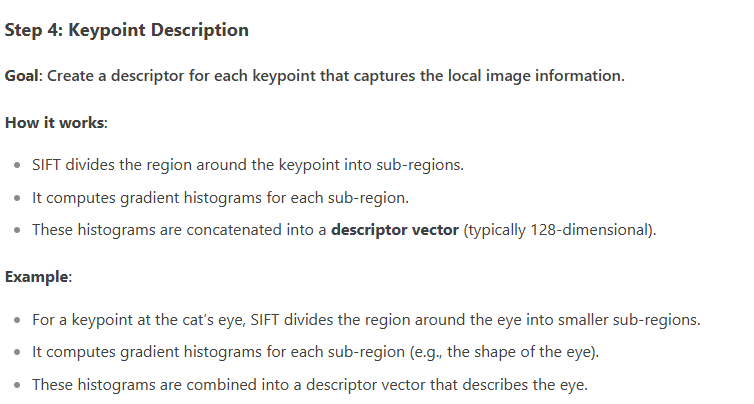
****

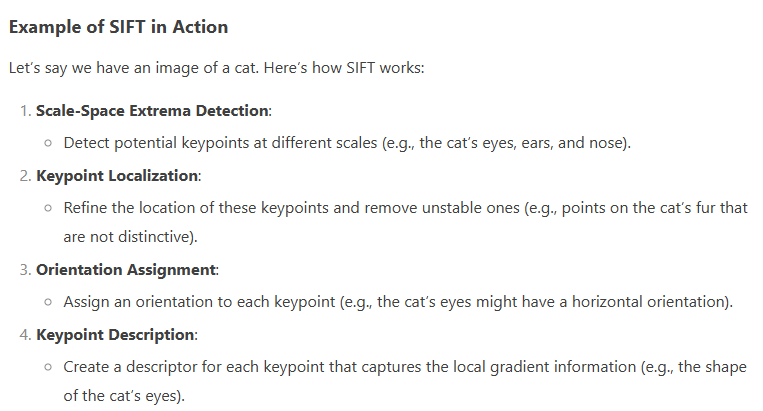
****

****

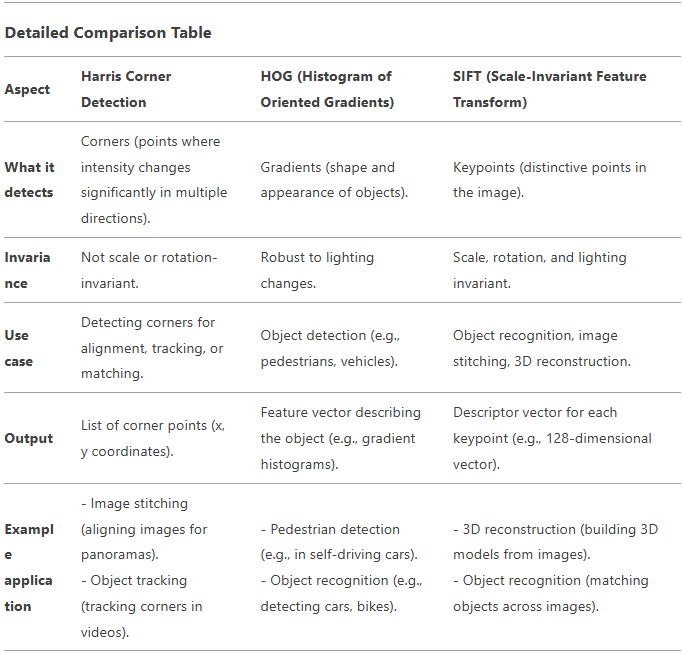
****

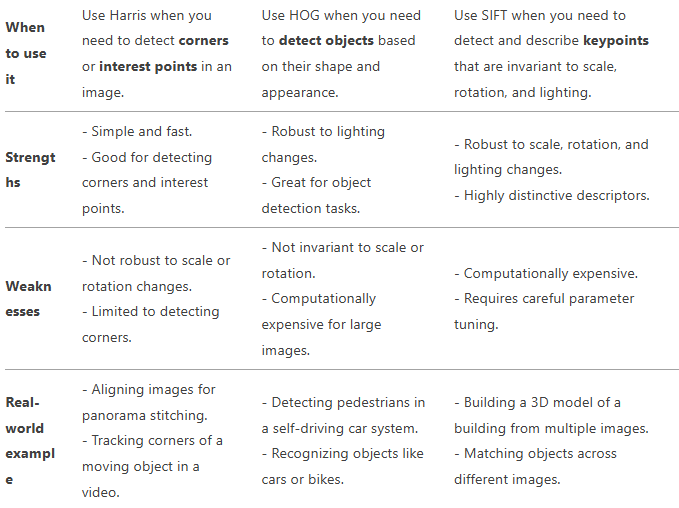
****

****

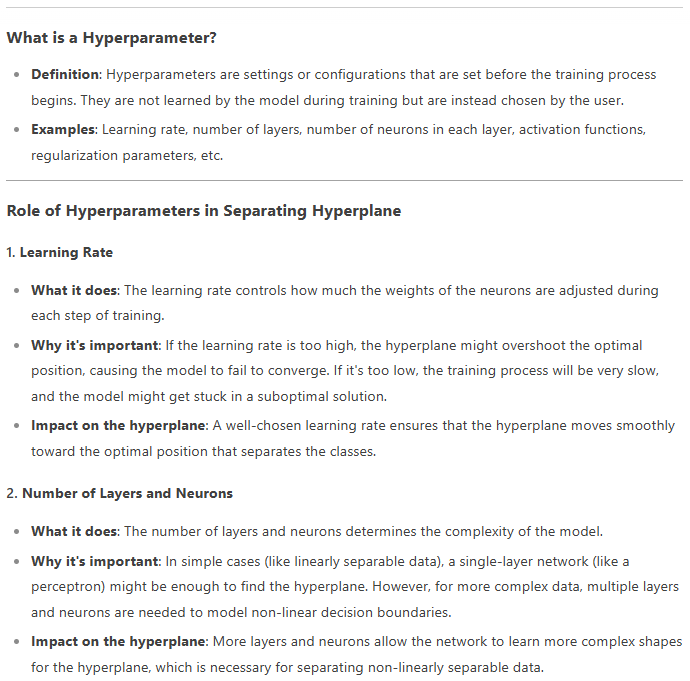
****

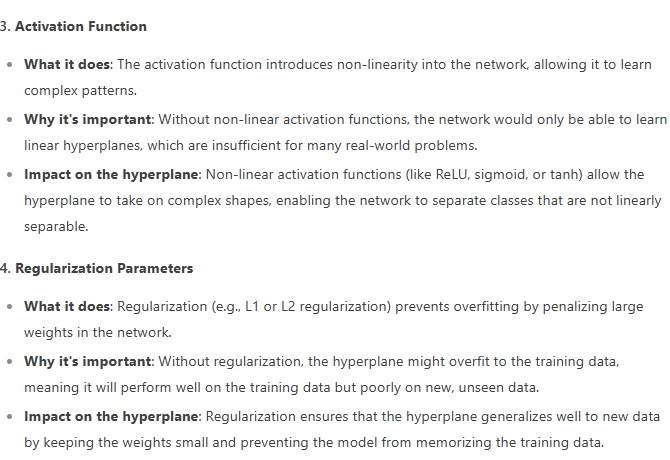
**Sample Question of SIFT with Solution:   
**

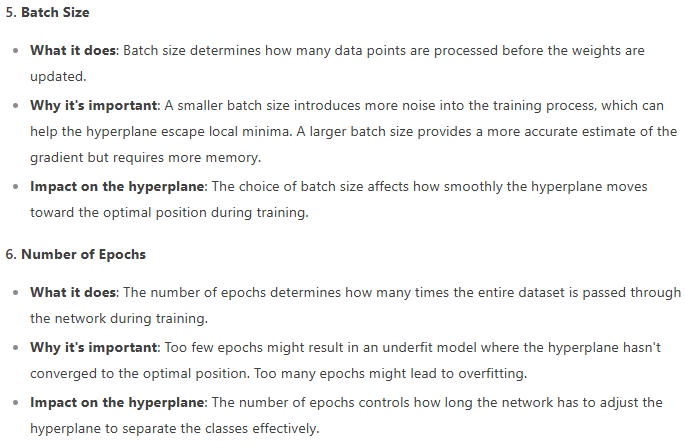
****

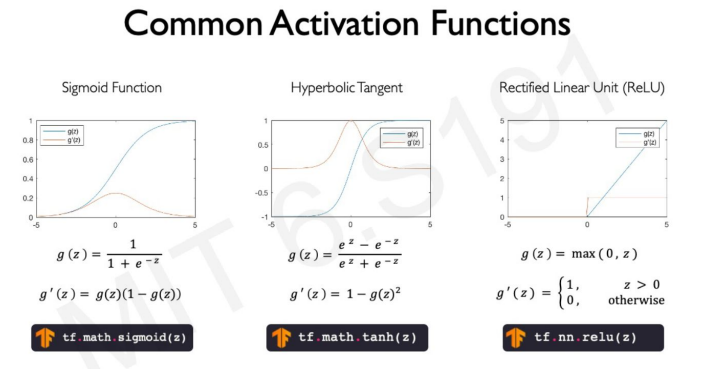
****

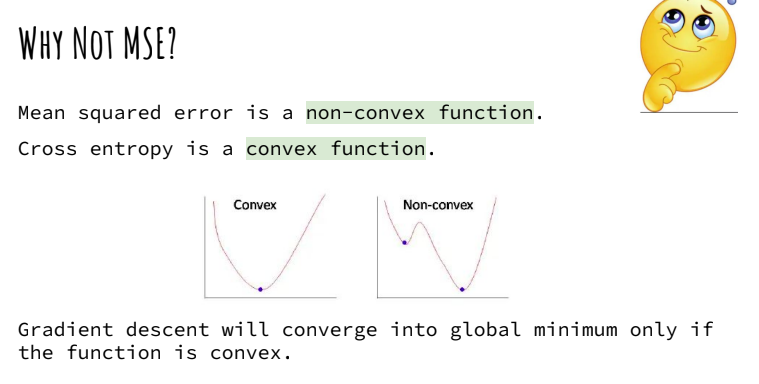
**Neural Network :**

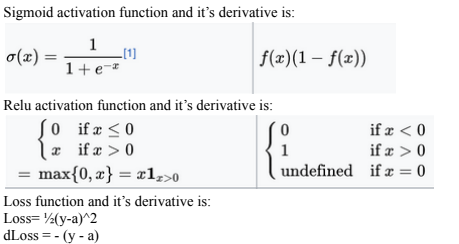
****

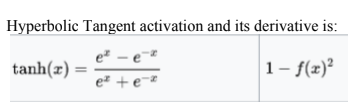
****

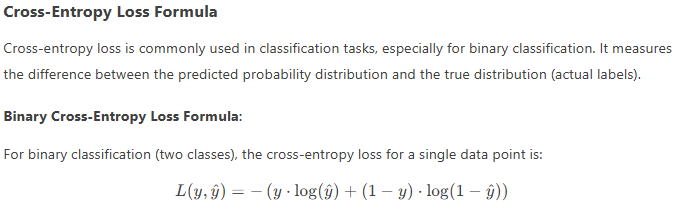
****

****

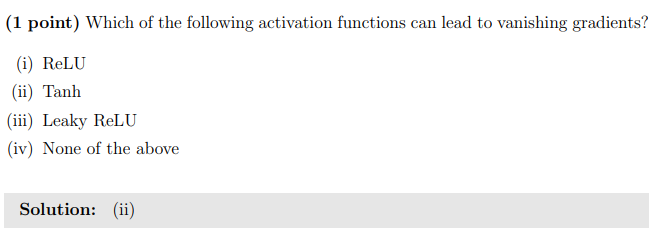
****

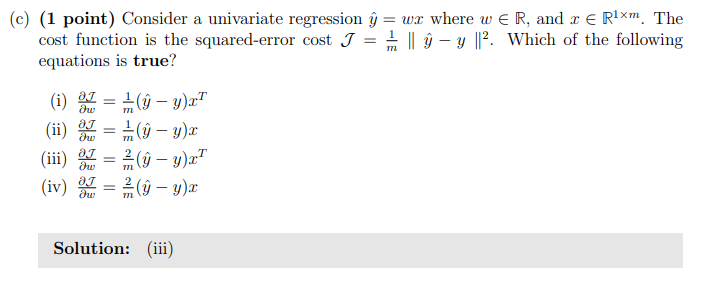
****

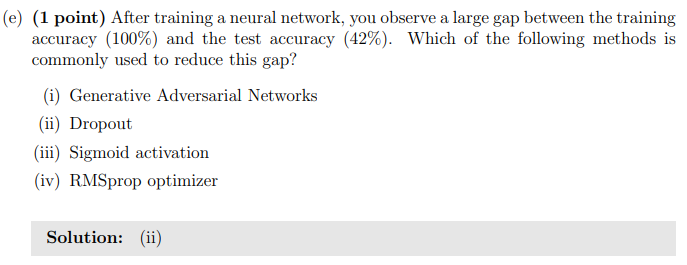
****

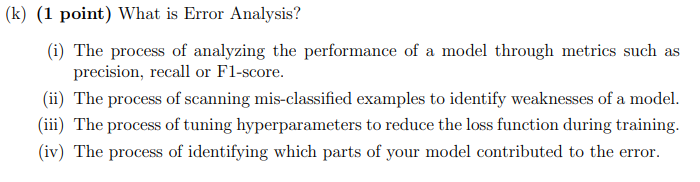
****

**Example Questions (Neural Networks)**

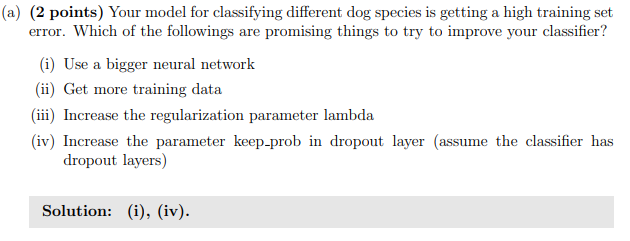


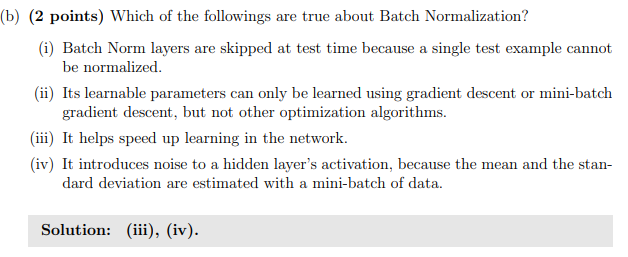


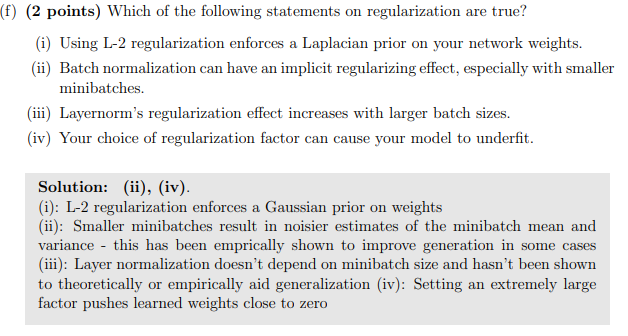


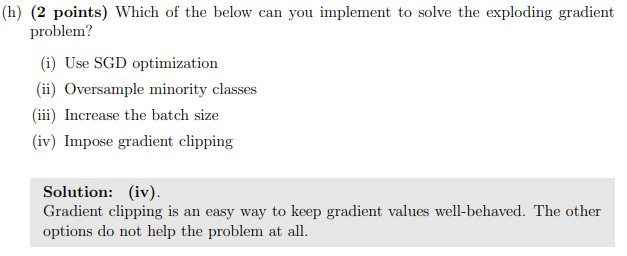


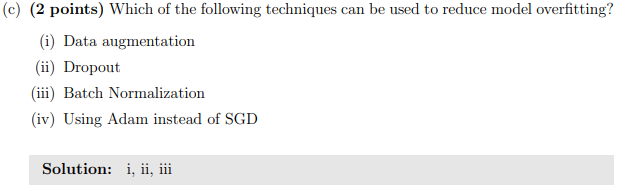
**Solution : (ii)**

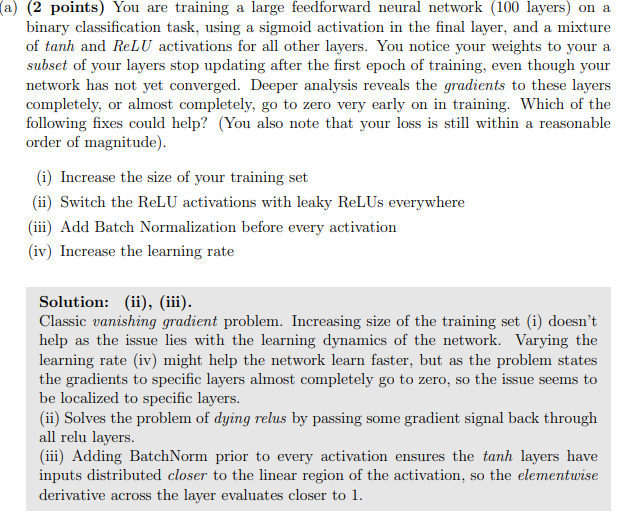


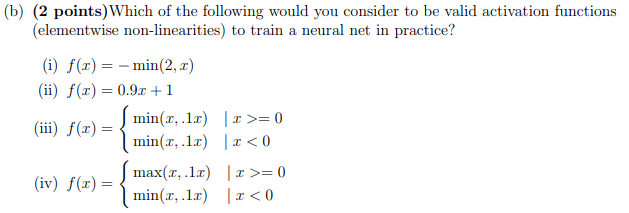


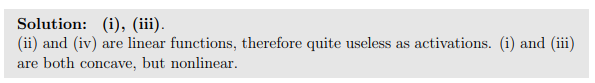


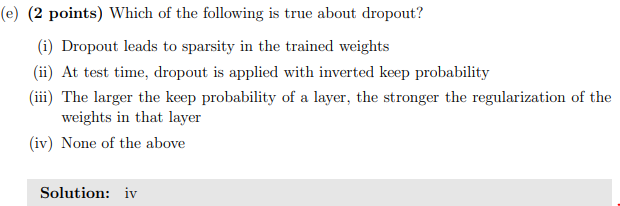




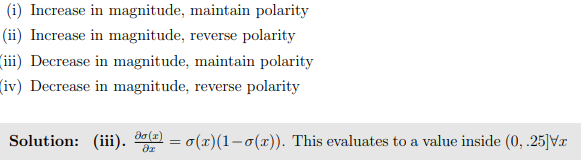


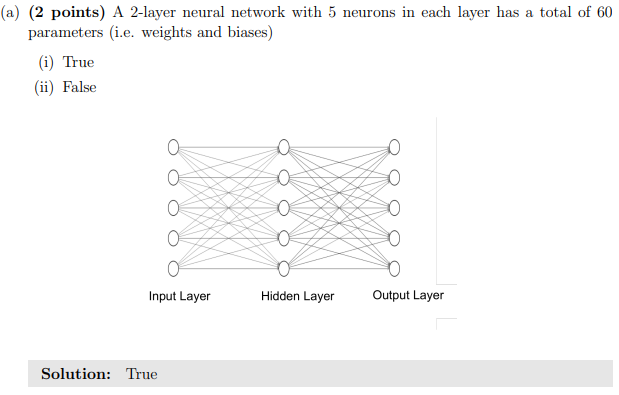


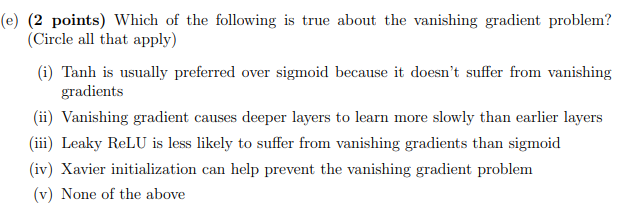




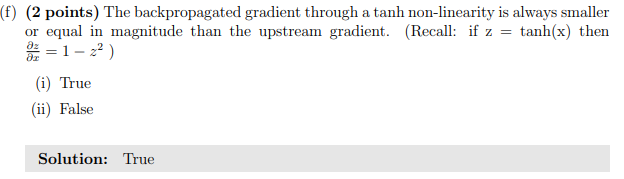


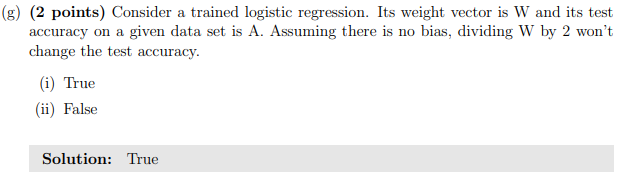


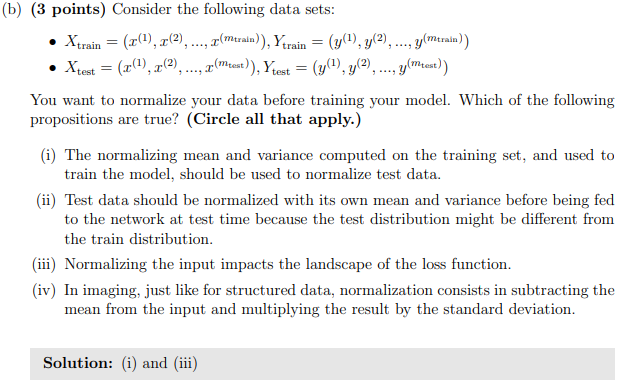


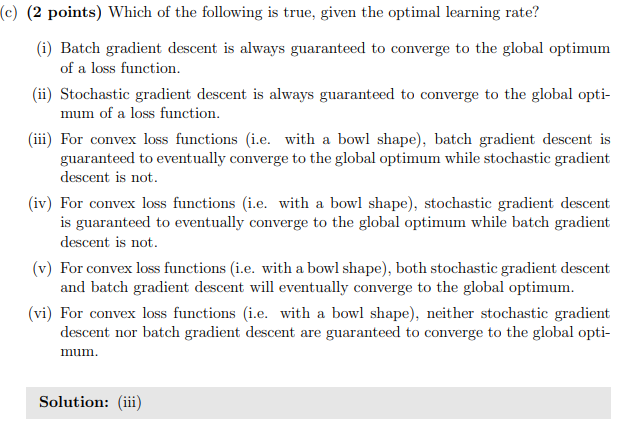


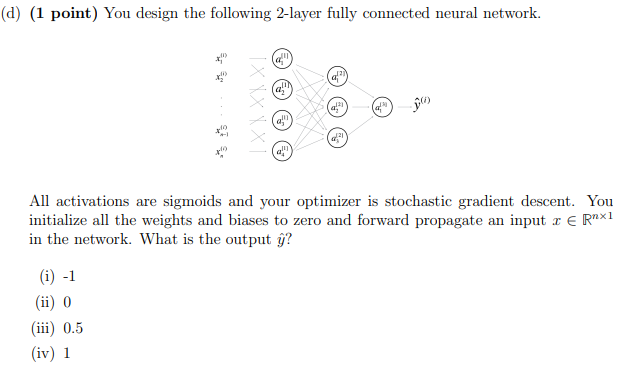






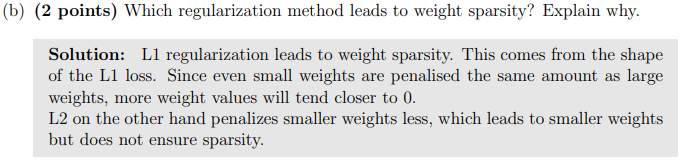


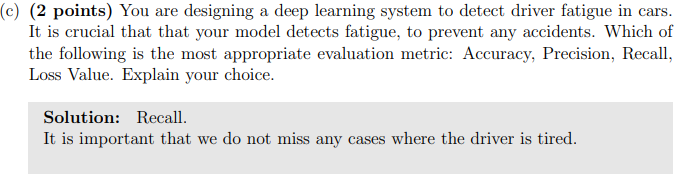


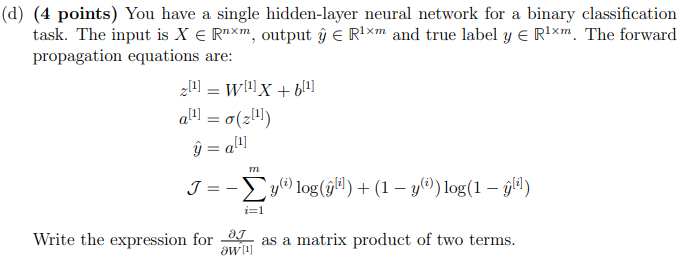


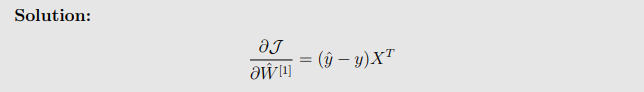
**Short Questions(Neural Networks):**

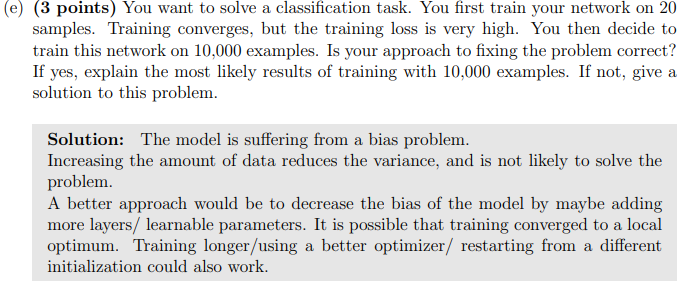




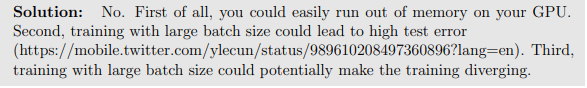




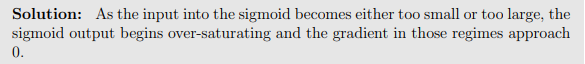


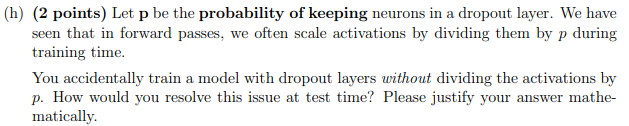


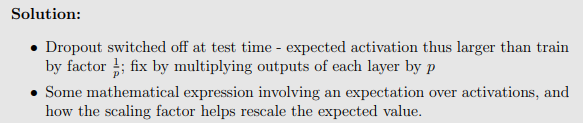


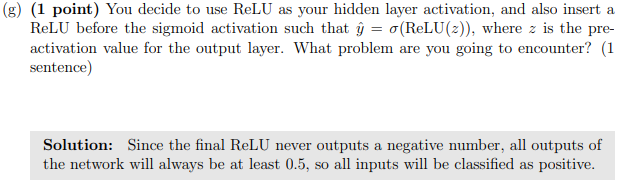


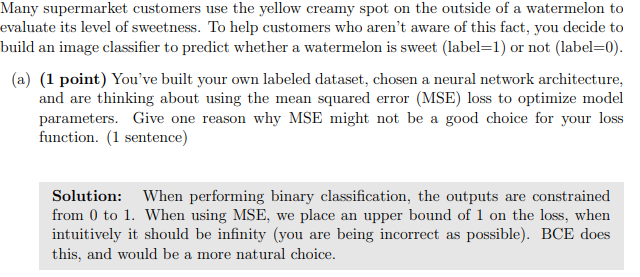


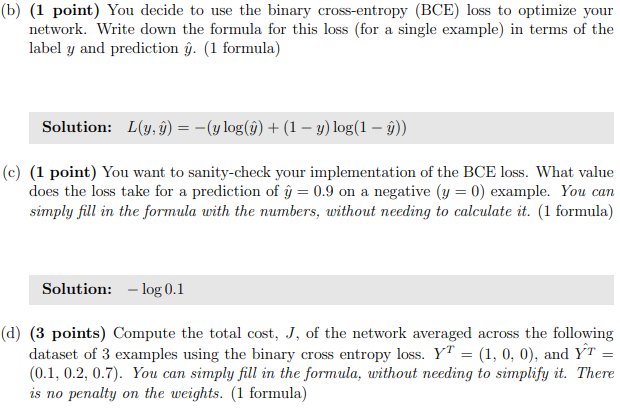




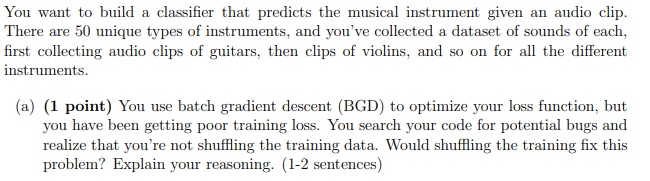


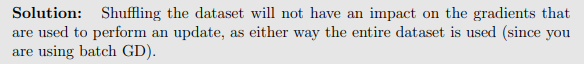


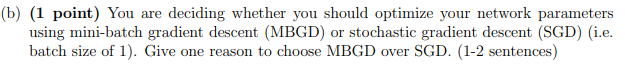


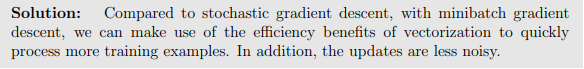




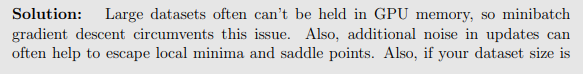




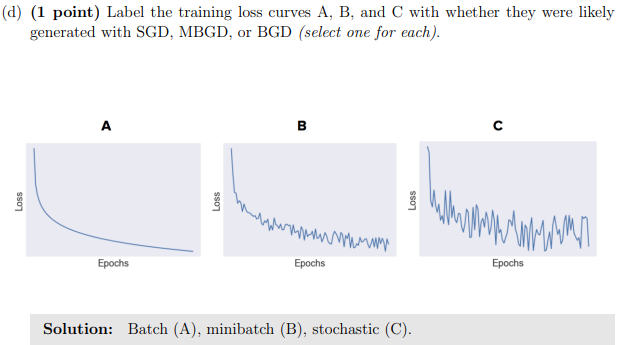




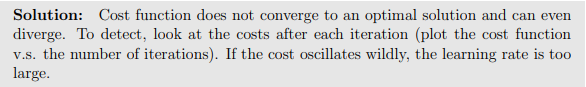




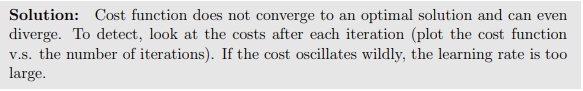


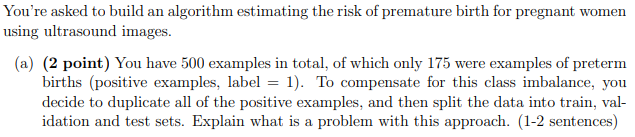


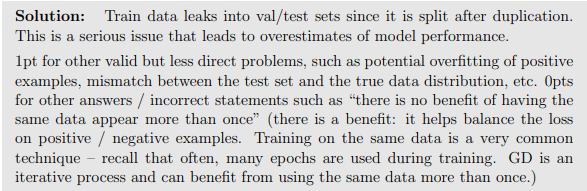


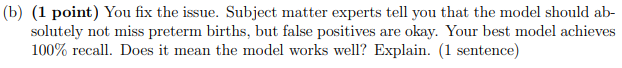






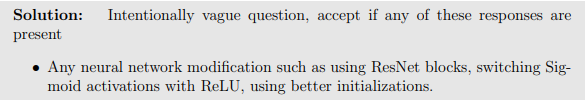


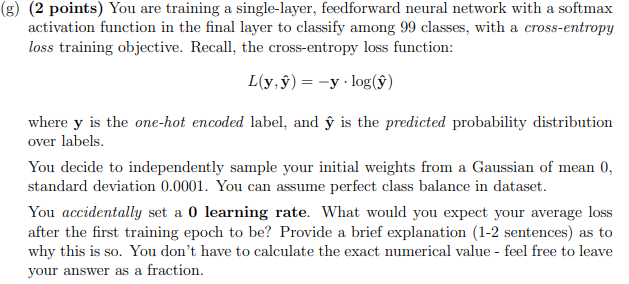


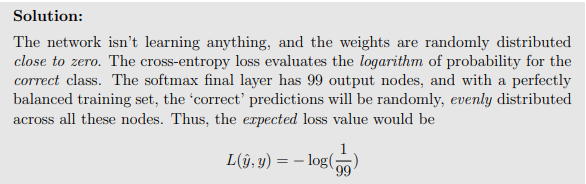




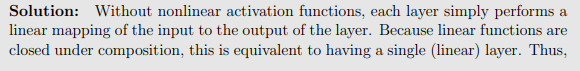










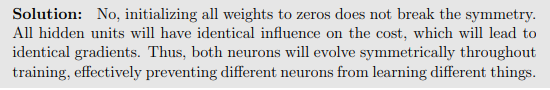










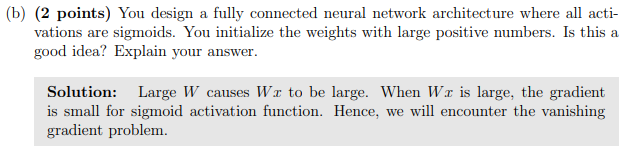


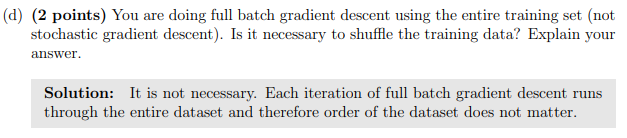


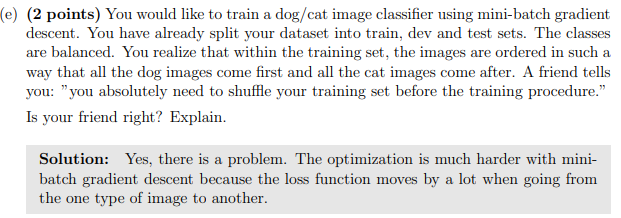


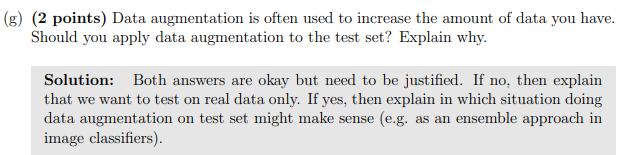


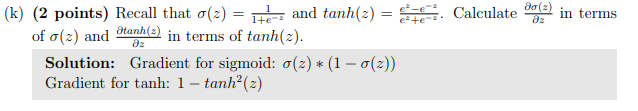




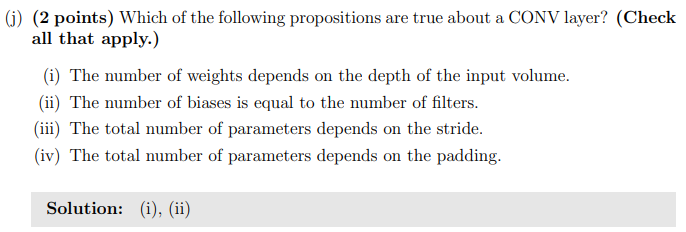


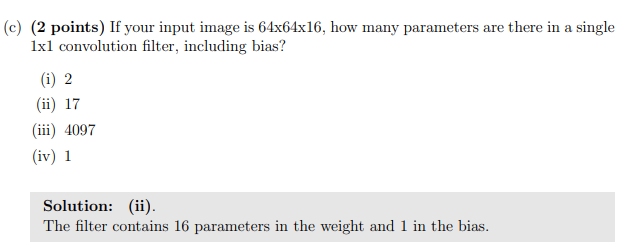


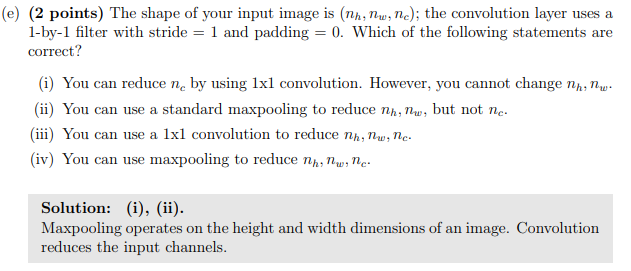


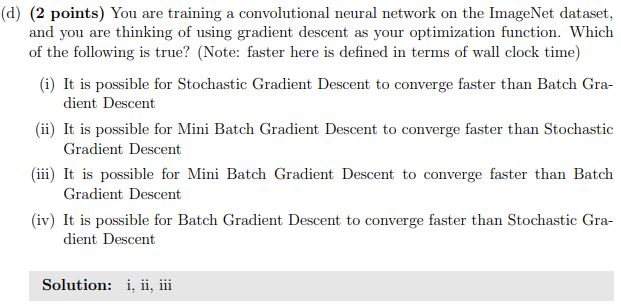


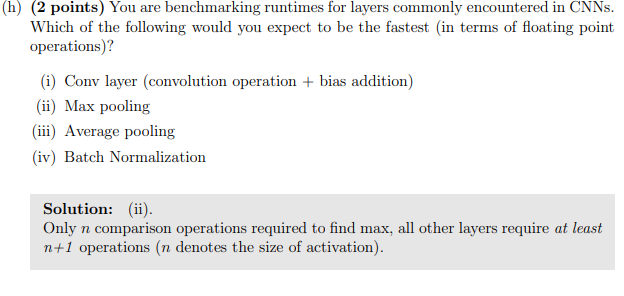
**Example Questions (CNN)**

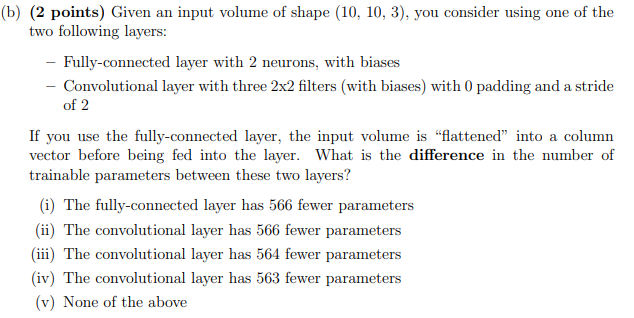




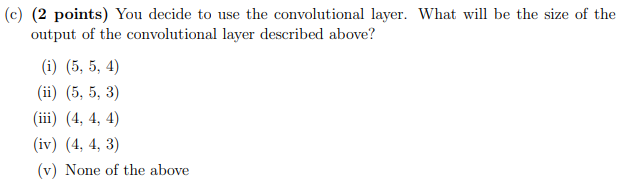






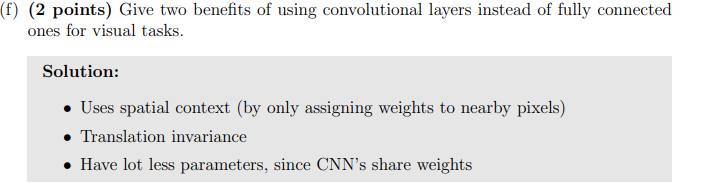


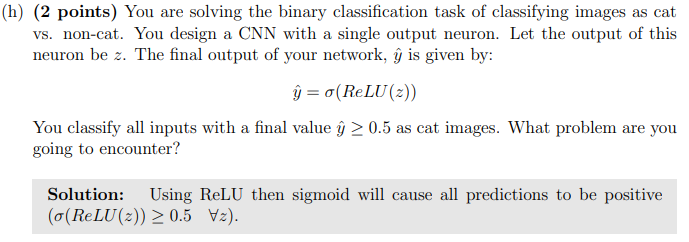


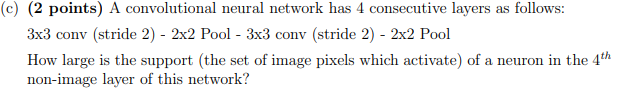




**Short Questions (CNN):**

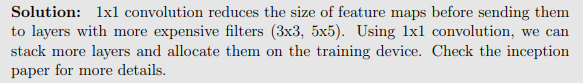


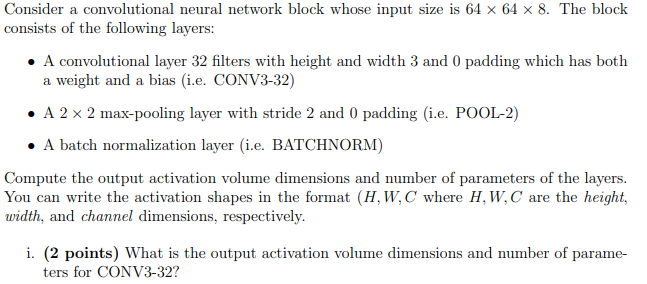












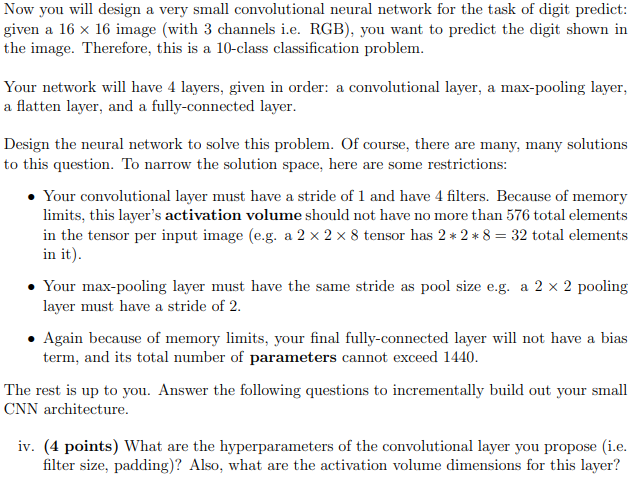


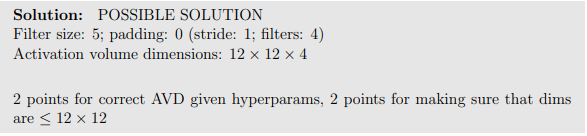




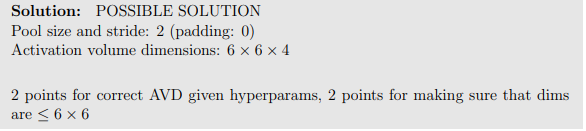






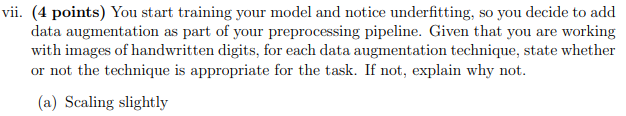


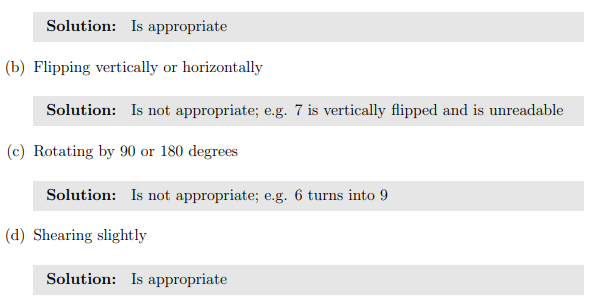


















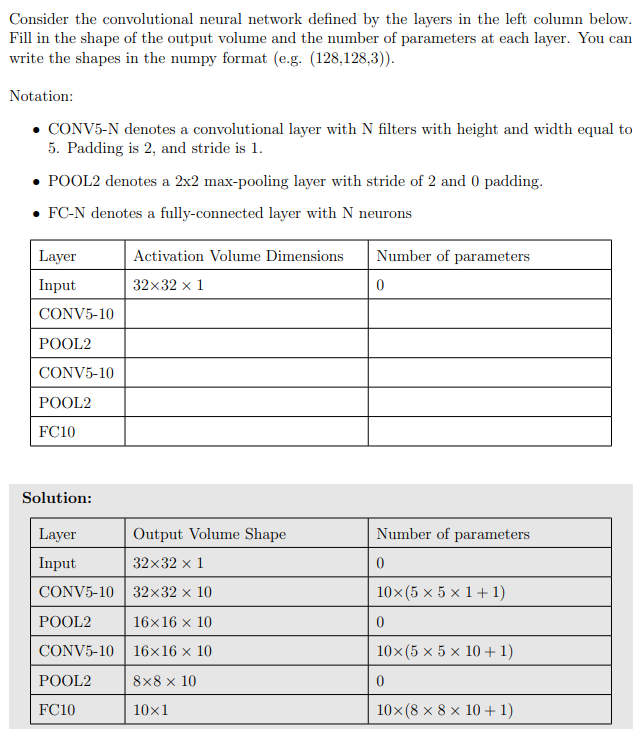


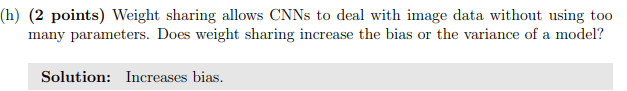


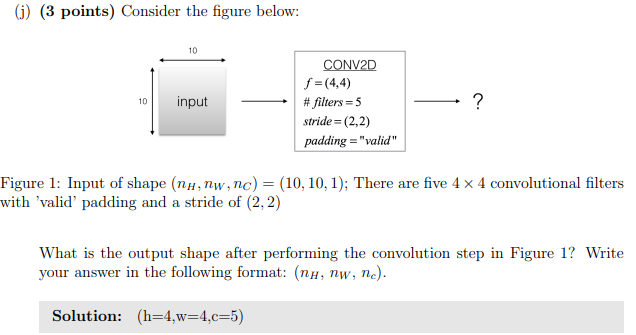












**Generative Modeling:**

Take a set of input samples, train a model to learn that distribution, and then generate samples from that distribution using the model. Can be used in 2 ways:

* Outlier Detection: If the new data point doesn’t ‘fit’ well into the distribution the model has already learned, it is termed as an outlier.
* Sample Generation: Use the trained generative model to generate new samples from the distribution that the model has learned (ChatGPT learns to generate text that is of the same distribution as the text that it is trained on).

**Latent Variable Models**

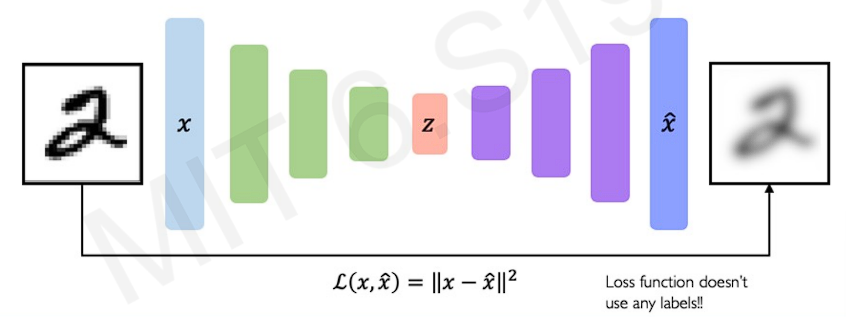
A latent variable is a variable that is not directly observed, but is inferred from the observable data. These latent models try to learn these latent variables from the observed data, and it specifically tries to model how the observed data is generated by these variables.

Example: Let’s say you have a dataset of handwritten digits (like MNIST):

* The observed data is the pixel values.
* A latent variable model (like a VAE) might learn that each digit (0–9) corresponds to a point in a latent space.
* The model can then generate new digits by sampling from this latent space (a compressed, low-dimensional representation of the data where the latent variables lie).

There are 2 types of Latent Variable Models:

**Autoencoders:**

****

Unsupervised approach for learning the lower-dimensional features (latent variables) representation from unlabeled training data. Consists of 2 Neural networks:

* Encoder: Trained to find the optimal representation of the lower-dimensional features and compressed the original data into these features. Achieves this using a ‘bottleneck hidden layer’ which forces the model to compress the data into a lower dimensional representation.
* Decoder: Learns to reconstruct the original data/image from the latent space.

To calculate the loss, we use the reconstruction loss (find the squared pixel value difference between the original image and the reconstructed image). It allows the latent representation to capture as much ‘information’ about the data as possible.



The quality of the reconstructed image depends on the dimensions of the latent space. The reconstructed image of a 5D latent space will be superior in quality as compared to reconstructed image of a 2D latent space.

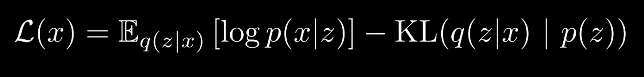
**Variational Autoencoders (VAEs):**

The drawbacks of the autoencoders were that they learned deterministic mapping between the observed data and the latent space, which means that if you sample the nearby values of the latent variable, it may not produce meaningful and coherent data. VAEs, instead of learning deterministic mappings, learn probabilistic mappings between the data distribution P(x) and the latent distribution P(z).

* Encoders: Learns the probabilistic mapping of observed data distribution to latent distribution P(z|x) by approximating P(z|x) as Q(z|x), which is a Gaussian distribution. It has 2 parameters, mean and variance vectors, which the encoder learns through training.
* Decoders: The decoder learns the probabilistic mapping 𝑃(𝑥∣𝑧) which represents the likelihood of the observed data given the latent variables. It reconstructs the input image (or data) 𝑥 from the latent variables 𝑧 by sampling from this learned distribution.

We use Gaussian distribution as our approximation because it is simple and easy work with, and it ensures that the latent space is continuous and smooth (meaning small changes in latent variables z should correspond to small changes in the generated data x).

The loss function is different compared to traditional Autoencoders:



It can also be simply written as: Loss = (reconstruction loss) – (regularization)

The reconstruction loss basically measures how well our model reconstructs the image x from it’s encoded version z. It is, essentially, the same mean squared error loss we were computing by calculating the squared error over the pixel values of the original image and the reconstructed image. The regularization term, or the KL-Divergence term, basically measures how close our approximation of the posterior Q(z|x) is to the prior distribution P(z).

By applying regularization, we want to achieve 2 things:

* Continuity: Points that are close to each other in the latent space should produce similar content when decoded. This means the latent space should be smooth, so small changes in the latent variables 𝑧 correspond to small, coherent changes in the output data 𝑥
* Completeness: Sampling from the latent space should result in the generation of meaningful content after decoding. The entire latent space should be useful for generating valid and realistic outputs, not just a small subset of it.

Backpropagation in Variational Autoencoders (VAEs) would not be possible if we tried to directly backpropagate through the sampling process because the sampling operation is non-differentiable, which means we can’t compute the gradients for it to update our weights. Specifically, when we sample a latent variable 𝑧 from a distribution (e.g., Gaussian), the operation involves randomness, and randomness is not a differentiable function of the model's parameters.

Instead of directly sampling 𝑧 from 𝑁(𝜇(𝑥),𝜎^2(𝑥)), we reparameterize the sampling operation by expressing the sample 𝑧 as:



where 𝜖 is a random variable drawn from a standard normal distribution 𝑁(0,𝐼), i.e., 𝜖∼𝑁(0,𝐼). By reparameterizing in this way, we express the random sample 𝑧 as a deterministic function of the encoder's output 𝜇(𝑥) and 𝜎(𝑥), and the random noise 𝜖. This means that the sampling operation is now differentiable with respect to 𝜇(𝑥) and 𝜎(𝑥).

**Latent Perturbation:** Latent perturbation refers to small modifications or changes applied to the latent variables 𝑧 in the latent space. The idea is to perturb the latent variables in such a way that we can observe how these small changes affect the generated data when decoded. By perturbing the latent variables, we can explore how different regions of the latent space correspond to different features or properties of the data. For instance, in the context of image generation, small perturbations in latent space could lead to slight changes in the generated images, such as altering the background or a facial expression.

**Disentanglement:** refers to the property of a model where the latent variables 𝑧 are independent and each one controls a separate, interpretable factor of variation in the data. In other words, the model learns to separate the different factors of variation (e.g., for images, this could be factors like pose, lighting, identity, background, etc.) into distinct dimensions of the latent space.