COMP 322: Midterm 2 Study Guide

Computer Vision and Image Processing · Fall 2026

Prof. Hong Xu

Work through the exercises below as practice for Midterm 2. Solutions are at the end of this document. Try each problem before looking.

1. Topics covered

- Pinhole projection $u = fx/z$, $v = fy/z$, including a camera whose axes are not the world axes.
- Intrinsics K (3×3, five degrees of freedom) and extrinsics $[R \mid \mathbf{t}]$ (3×4, six degrees of freedom). The product M has eleven degrees of freedom.
- What you measure to calibrate a camera, and which symbol in $\mathbf{x} = M\mathbf{X}$ is unknown.
- Focal length, camera distance, and sensor size: which of these move the projected dots.
- Regression losses versus classification losses. Absolute distance versus squared Euclidean distance.
- Softmax, cross-entropy, and the shapes of $W\mathbf{x} + \mathbf{b}$.
- Partials, the sum over paths, a numeric gradient step, and a ReLU that blocks a gradient.
- Correlation versus convolution, output size, parameter count, receptive field, pooling.
- PyTorch: NCHW versus the array matplotlib plots, `backward` and `zero_grad`, `train` versus `eval`, `detach().cpu().numpy()`.
- Hyperparameters, Adam, minibatches, warmup, and the training, validation, and test split.
- A full training program in seven pieces: device, data, loaders, model, training, testing, visualization.

2. Practice exercises

The camera in questions 1 and 2 is not aligned with the world axes. Its center is the origin, right $= (1, 0, 0)$, up $= (0, 0, 1)$, forward $= (0, 1, 0)$, and $f = 2$.

1. A world point is $(X, Y, Z) = (4, 6, 2)$. Compute the camera-frame coordinates x, y, z and the image point (u, v).
2. Three separate changes, each starting from the camera and point in question 1.
   (a) The depth z along forward is halved. f, x, and y stay the same. What happens to (u, v), and what stays the same about a pyramid of length f?
   (b) f is doubled. What happens to (u, v), and what happens to that pyramid?
   (c) Only the sensor width is halved. Do the dots move? What does change?
3. You are given one photograph and you want M in $\mathbf{x} = M\mathbf{X}$.
   (a) What must you place in the scene, and which of $\mathbf{x}$, M, and $\mathbf{X}$ do you compute?
   (b) All of your 3D points lie on one plane. Why is a single view then not enough to pin down an 11-degree-of-freedom camera?
4. For $\mathbf{x} = K[R \mid \mathbf{t}]\,\mathbf{X}$: say what each symbol is and give the shape of K, of $[R \mid \mathbf{t}]$, and of $M = K[R \mid \mathbf{t}]$. How many degrees of freedom does K have? How many does $[R \mid \mathbf{t}]$ have, and why is 7 wrong? Why does M have 11 rather than 12? What is a vanishing point?

5. True or false. Correct each false statement.
   (a) K is 3×4.
   (b) $[R \mid \mathbf{t}]$ has 7 degrees of freedom.
   (c) Doubling f and halving z leaves (u, v) unchanged.
   (d) An affine map can send parallel lines to lines that meet.

6. Both of these are legal for a real-valued target. Name each one. Which distance is each (absolute or squared Euclidean)? Which one lets a single large miss dominate the average? Why is neither the right loss for an integer class index?

$$\frac{1}{N}\sum_i |\hat{y}_i - y_i|, \qquad \frac{1}{N}\sum_i (\hat{y}_i - y_i)^2.$$

7. Name the loss $-\big(y \log \sigma(s) + (1-y)\log(1-\sigma(s))\big)$ with $\sigma(s) = 1/(1+e^{-s})$ and $y \in \{0, 1\}$. Give legal shapes for the logit and the label. A classmate instead has two logits and integer labels 0 and 1. Which loss should they switch to, and what is wrong with leaving a sigmoid on the module?

8. Name $-\log p_y$ where $p_c = e^{s_c}/\sum_j e^{s_j}$ and y is a class index. For C classes, what does the last linear layer emit, and what is the label tensor's dtype and shape? Why do you not apply softmax in `forward`?

9. Scores are $(s_1, s_2, s_3) = (1, 2, 3)$, and the true class is the second one. Using

$$p_c = \frac{e^{s_c}}{\sum_{j=1}^{n} e^{s_j}},$$

   write each probability and the cross-entropy loss in exact form. Then: if $\mathbf{x}$ has length D and there are C classes, what are the shapes of W, $\mathbf{b}$, and $\mathbf{s}$ in $\mathbf{s} = W\mathbf{x} + \mathbf{b}$? What is the loss when every class is equally likely?

10. The loss depends on x through two intermediates, z_1 and z_2. Write $\mathrm{d}L/\mathrm{d}x$ as a sum of paths. Which factor in that sum is a partial, and why would stepping with only $\partial L/\partial z_1$ be the wrong update?

11. Let $p = wx + b$ and $L = (p - t)^2$ with $w = 2$, $x = 4$, $b = -1$, and $t = 5$. Compute p, L, $\partial L/\partial w$, and $\partial L/\partial b$. Then take one step with $\eta = 0.05$ and give the new w and b.

12. After that same linear unit, a ReLU produces $a = \max(z, 0)$, and a later loss has $\partial L/\partial a = 4$.
   (a) If $z = -0.5$, what are $\partial L/\partial z$ and $\partial L/\partial w$?
   (b) If $z = +0.5$ and $x = 4$, what are $\partial L/\partial z$ and $\partial L/\partial w$?

13. One 3×3 window. Compute correlation with K as written, show the kernel flipped by 180°, and compute convolution.

$$I = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 3 \\ 2 & 0 & 1 \end{bmatrix}, \qquad K = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}.$$

14. Input width 32, kernel 3. Give the output width for: padding 1 and stride 1; padding 0 and stride 1; padding 1 and stride 2. Use the PyTorch formula $\lfloor (W + 2P - K)/S \rfloor + 1$. What padding preserves width for an odd kernel and stride 1?

15. Count parameters, including biases: `Conv2d(1, 16, 3)`, `Conv2d(16, 32, 3)`, and `Linear(784, 16)`. Give two reasons the $1 \to 16$ convolution is so much smaller than that linear layer on a 28×28 image.

16. A unit in the output of two stride-1 convolutions, each with a 3×3 kernel, depends on a square in the input. What is the side of that square? Justify it by saying what each new layer adds. What is the side after a third such layer?

17. `x` has shape $(8, 3, 16, 16)$.

    ```
    h = nn.MaxPool2d(2)(torch.relu(nn.Conv2d(3, 4, 3, padding=1)(x)))
    ```

    What is the shape of `h`? How many parameters does the pool have? The next layer produces 10 logits and the loss is `CrossEntropyLoss`. What goes wrong if `forward` ends in `softmax`?

18. Training is on the CPU. A tensor of shape $(1, 3, H, W)$ is about to enter a CNN, and it requires gradients. Write the expression that turns one image into a NumPy array of shape $(H, W, 3)$ that matplotlib can show. Name the layout change.

19. A parameter already has `.grad` equal to 3. The new backward contribution is 5. What is stored if you forget `zero_grad`? What is stored if you clear first? Why does `backward` add instead of overwrite?

20. What do `model.train()` and `model.eval()` change about dropout and batch norm? Does `eval()` stop autograd? Which context manager does?

21. A prediction tensor requires gradients and lives on the GPU. Give the three calls, in order, that produce a NumPy array. Which call fails if you skip `detach`, and which fails if you skip `cpu`?

22. Define a hyperparameter. Give three examples from a training run. Why is a weight inside `Conv2d` not one of them?

23. What two running statistics does Adam keep that plain SGD does not? What does a minibatch replace? What does momentum's velocity accumulate? What does linear warmup prevent at the start of training?

24. Why are there three splits (training, validation, test) rather than two? Say what each split is allowed to influence.

25. List the pieces of a Programming Assignment 4 run in the order you would write them, from choosing the device through a figure of one image. The next seven questions ask you to write those pieces. Do not look ahead at the solutions.

    The remaining questions are one program. Images are grayscale 28×28, values 0 to 255, and the target is one angle in degrees. You are handed Python lists `images` and `angles`, each of length N.

26. Write the device selection. Then say what exception you get if the model is on CUDA and a batch is still on the CPU.

27. Build `X` of shape $(N, 1, 28, 28)$ and `y` of shape $(N, 1)$, dtype `float32`, pixels in $[0, 1]$. Split off a validation set. Subtract the training mean and divide by the training standard deviation, and apply those same two numbers to the validation images. Why is it wrong to recompute the mean on the validation images?

28. Write `make_loader(X, y, batch_size, shuffle)` with a seeded generator. Training uses shuffle and batch size 32. Validation does not shuffle. Why must the epoch loss weight each batch by its length?

29. Write `AngleNet`. One convolution from 1 to 8 channels, kernel 3, padding 1, then ReLU, then a 2×2 max pool, then a linear layer to one number. You have to compute the linear layer's input size yourself. What is the spatial size after the pool?

30. Write `train(model, loader, epochs, lr=1e-3)` for this regression problem. Use minibatches, Adam, and mean squared error. Return the sample-weighted training loss of each epoch. The body

has to contain the update in the right order. A missing `zero_grad` is wrong even if the loss goes down for one epoch.

31. Write `evaluate(model, loader)` returning the mean absolute error in degrees as a Python float. Use eval mode and `inference_mode`. Give two reasons this number is not the list returned by `train`.

32. `xb` has shape $(32, 1, 28, 28)$, may be on CUDA, and may require gradients. Write the lines that plot image 0 in grayscale. Include the device move, the gradient cut, the NCHW-to-HWC permutation, and the squeeze that makes the array $(28, 28)$ rather than $(28, 28, 1)$.

3. Solutions

1. $x = 4 \cdot 1 = 4$, $y = 2 \cdot 1 = 2$ because up is world Z, and $z = 6$ because forward is world Y. Then $u = 2 \cdot 4/6 = 4/3$ and $v = 2 \cdot 2/6 = 2/3$.

2. (a) $u = fx/z$ and $v = fy/z$ both double when z is halved. The pyramid's length is f, which did not change.

   (b) (u, v) doubles again, from the factor f. The pyramid gets longer: its base sits one focal length in front of the pinhole, so that base moves farther from the camera.

   (c) The dots stay. Sensor width is not in $u = fx/z$ or $v = fy/z$. The rectangle, and the base of the pyramid, get narrower.

3. (a) Place an object whose 3D coordinates you know, and record the pixel of each of those points. $\mathbf{x}$ and $\mathbf{X}$ are measured. The unknown is M.

   (b) Points on one plane determine a homography, which has 8 degrees of freedom. A full camera has 11. One view of a flat target does not pin down M.

4. $\mathbf{x} = (u, v, 1)$ is the image point. K is the 3×3 intrinsic matrix

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix},$$

   with 5 degrees of freedom. $[R \mid \mathbf{t}]$ is the 3×4 extrinsic block: 3 from the rotation and 3 from the translation, so 6, not 7. $\mathbf{X} = (X, Y, Z, 1)$. The product M is 3×4. Its 12 entries are defined up to scale, which removes one degree of freedom and leaves 11. A vanishing point is the image of a direction at infinity. That homogeneous point has last coordinate 0.

5. (a) False. K is 3×3. The 3×4 matrix is M or $[R \mid \mathbf{t}]$.

   (b) False. Six: three rotational, three translational.

   (c) False. Doubling f multiplies (u, v) by 2, and halving z multiplies them by 2 again, so (u, v) becomes four times as large. The ratio f/z becomes $4f/z$, not f/z.

   (d) False. Affine maps preserve parallelism. A projective map can send parallel lines to lines that meet.

6. The first is mean absolute error, `L1Loss`: absolute distance, and Manhattan distance when there are several coordinates. The second is mean squared error, `MSELoss`: squared Euclidean distance. Squaring lets one large miss dominate. A class index is not a coordinate. Use cross-entropy for that, not either of these.

7. `BCEWithLogitsLoss`. One logit, shape $(N, 1)$ or $(N,)$ after a squeeze of the last dimension, and float labels 0 or 1 of the same shape. Two logits with integer labels 0 and 1 are `CrossEntropyLoss`.

The sigmoid form of binary cross-entropy already applies σ. A sigmoid in the module applies it twice and can overflow.

8. Multiclass cross-entropy, `CrossEntropyLoss`. The last layer emits C logits, shape (N, C). Labels are `int64` of shape $(N,)$. The loss applies the log-softmax itself. A softmax in `forward` feeds it probabilities and the loss normalizes again.

9. $n = 3$ and the sum is $e + e^2 + e^3$. The probabilities are

$$p_1 = \frac{e}{e + e^2 + e^3}, \quad p_2 = \frac{e^2}{e + e^2 + e^3}, \quad p_3 = \frac{e^3}{e + e^2 + e^3}.$$

The loss is $-\log p_2 = \log(e + e^2 + e^3) - 2$. W is $C \times D$, $\mathbf{b}$ and $\mathbf{s}$ are $C \times 1$. Equal probabilities give loss $\log C$.

10. $\mathrm{d}L/\mathrm{d}x = (\partial L/\partial z_1)(\partial z_1/\partial x) + (\partial L/\partial z_2)(\partial z_2/\partial x)$. Each factor $\partial L/\partial z_i$ holds the other inputs of that gate fixed. The update needs the total derivative, the sum of the paths. Using only the first path drops every effect that goes through z_2.

11. $p = 2 \cdot 4 + (-1) = 7$ and $L = (7-5)^2 = 4$. Then $\partial L/\partial p = 4$, so $\partial L/\partial w = 4 \cdot 4 = 16$ and $\partial L/\partial b = 4$. The step is $w \leftarrow 2 - 0.05 \cdot 16 = 1.2$ and $b \leftarrow -1 - 0.05 \cdot 4 = -1.2$.

12. (a) The ReLU derivative is 0 for $z < 0$, so $\partial L/\partial z = 0$ and $\partial L/\partial w = 0$.

    (b) The ReLU derivative is 1, so $\partial L/\partial z = 4$ and $\partial L/\partial w = 4 \cdot 4 = 16$.

13. Correlation, kernel as written: left column minus right column, $(1+0+2) - (0+3+1) = -1$. The 180° flip is $-K$,

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}.$$

Convolution is the correlation of I with that flip, which equals $+1$.

14. Padding 1, stride 1: $\lfloor (32 - 3 + 2)/1 \rfloor + 1 = 32$, size preserved. Padding 0, stride 1: 30. Padding 1, stride 2: $\lfloor (32 - 3 + 2)/2 \rfloor + 1 = \lfloor 31/2 \rfloor + 1 = 16$. PyTorch floors the division, so this setting is legal and yields width 16. Preserving padding for odd K and stride 1 is $P = (K-1)/2$, here 1.

15. $16 \cdot (1 \cdot 3 \cdot 3 + 1) = 160$. $32 \cdot (16 \cdot 9 + 1) = 4640$. $784 \cdot 16 + 16 = 12560$. The convolution is local, a 3×3 neighborhood rather than all 784 pixels, and the same kernel is reused at every position.

16. Each stride-1 layer of kernel 3 adds $3 - 1 = 2$ to the field. Two layers: side $1 + 2 \cdot 2 = 5$. Three layers: side 7.

17. Convolution with padding 1 keeps 16×16 and makes 4 channels. ReLU keeps that. The pool halves each side. Shape $(8, 4, 8, 8)$. The pool has no parameters. `CrossEntropyLoss` applies log-softmax to logits. A softmax in `forward` makes it normalize twice, and the loss is no longer $-\log p_y$ of the class scores.

18. `tensor[0].detach().permute(1, 2, 0).numpy()`. Index 0 drops the batch axis, leaving CHW. `permute(1, 2, 0)` is HWC. `detach` lets NumPy see the values. On the CPU, `.cpu()` is optional.

19. `backward` adds. Forgetting `zero_grad` stores $3 + 5 = 8$. Clearing first stores 5. Addition is what lets several backward passes accumulate. A normal step wants only the new batch, so the old value has to be cleared.

20. `train` uses dropout and computes batch-norm statistics from the batch. `eval` turns dropout off and uses the stored batch-norm statistics. `eval` does not stop autograd. `torch.inference_mode()` or `torch.no_grad()` does.

21. `detach().cpu().numpy()`. Skipping `detach` makes `numpy()` refuse a tensor that requires grad. Skipping `cpu` makes `numpy()` refuse a CUDA tensor.

22. A hyperparameter is chosen before the run and is not updated by the optimizer. Examples: learning rate, batch size, number of epochs. A convolution weight is a parameter. `step` changes it.

23. Adam keeps a running average of the gradient and a running average of the squared gradient, and each parameter gets its own effective step size. A minibatch estimates the gradient over the whole training set. Momentum's velocity accumulates past gradients. Warmup keeps the first steps small so a large learning rate does not explode the loss.

24. The training split is what the optimizer fits. The validation split is where you choose hyperparameters, so those choices are not tuned on the test set. The test split is used once, for a number about new data. Two splits force you to tune on the same data you report.

25. Device, then data, then loaders, then the model, then the training loop, then evaluation, then a figure of one image.

26. `device = torch.device("cuda" if torch.cuda.is_available() else "cpu")`
    A CUDA model and a CPU batch raise a device-mismatch error inside the forward pass.

27. ```
    X = torch.tensor(np.stack(images), dtype=torch.float32)
    X = X.unsqueeze(1) / 255.0          # (N, 1, 28, 28)
    y = torch.tensor(angles, dtype=torch.float32).view(-1, 1)
    n = int(0.8 * len(X))
    X_train, X_val = X[:n], X[n:]
    y_train, y_val = y[:n], y[n:]
    mu = X_train.mean()
    sd = X_train.std().clamp_min(1e-6)
    X_train = (X_train - mu) / sd
    X_val = (X_val - mu) / sd
    ```
    A mean computed on the validation images uses data the model is not supposed to see while you are still making choices.

28. ```
    def make_loader(X, y, batch_size, shuffle, seed=0):
        g = torch.Generator().manual_seed(seed)
        ds = TensorDataset(X, y)
        return DataLoader(ds, batch_size=batch_size, shuffle=shuffle, generator=g)
    ```
    `MSELoss` averages inside the batch. A short last batch would count too much in a plain mean of batch losses, so weight each batch by `xb.shape[0]`.

29. After the convolution the map is still 28×28. The pool makes it 14×14. The linear layer sees $8 \cdot 14 \cdot 14 = 1568$ inputs.

    ```
    class AngleNet(nn.Module):
        def __init__(self):
            super().__init__()
            self.conv = nn.Conv2d(1, 8, kernel_size=3, padding=1)
            self.pool = nn.MaxPool2d(2)
            self.fc = nn.Linear(8 * 14 * 14, 1)

        def forward(self, x):
            x = self.pool(torch.relu(self.conv(x)))
            x = torch.flatten(x, 1)
    ```

```
        return self.fc(x)
```

30.
```
def train(model, loader, epochs, lr=1e-3):
    model.to(device)
    loss_fn = nn.MSELoss()
    opt = torch.optim.Adam(model.parameters(), lr=lr)
    history = []
    for _ in range(epochs):
        model.train()
        running = 0.0
        for xb, yb in loader:
            xb, yb = xb.to(device), yb.to(device)
            opt.zero_grad()
            loss = loss_fn(model(xb), yb)
            loss.backward()
            opt.step()
            running += loss.item() * xb.shape[0]
        history.append(running / len(loader.dataset))
    return history
```

31.
```
def evaluate(model, loader):
    model.eval()
    total, count = 0.0, 0
    with torch.inference_mode():
        for xb, yb in loader:
            xb, yb = xb.to(device), yb.to(device)
            err = (model(xb) - yb).abs().sum().item()
            total += err
            count += yb.shape[0]
    return total / count
```

Training reported mean squared error on the training loader. This is mean absolute error on a different loader, with dropout off and no weight update.

32.
```
img = xb[0].detach().cpu().permute(1, 2, 0).squeeze(-1).numpy()
plt.imshow(img, cmap="gray")
```

`xb[0]` is CHW. `permute(1, 2, 0)` is HWC. `squeeze(-1)` drops the grayscale channel so the array is $(28, 28)$.