CSC413 Assignment 2

Jiawei Gong

10,19 2024

1.1.1

From HW 1, we know:$\nabla_{\mathbf{w}}\mathcal{L}^T = \frac{2}{n}\left(\mathbf{w}^T\mathbf{X}^T\mathbf{X} - \mathbf{t^T}\mathbf{X}\right)$ for full batch
So for mini-batch at the i-th iteration:$\nabla_{\mathbf{w}}\mathcal{L}^T = \frac{2}{b}\left(\mathbf{w}_i^T\mathbf{B}_i^T\mathbf{B}_i - \mathbf{t^T}\mathbf{B}_i\right)$

we let:
$\mathbf{X} = \begin{bmatrix} -\mathbf{x}_1- \\ -\mathbf{x}_2- \\ \vdots \\ -\mathbf{x}_n- \end{bmatrix}$ $\mathbf{B} = \begin{bmatrix} -\mathbf{x}_1- \\ \vdots \\ -\mathbf{x}_b- \end{bmatrix}$
At time i,
$\mathbf{B}_i^T = \mathbf{X}^T\mathbf{S}_i$ for some $\mathbf{S}_i \in \mathbb{R}^{n\times b}$
$\mathbf{B}_i = \mathbf{S}_i^T\mathbf{X}$
$\mathbf{B}$ is always contained in the span of $\mathbf{X}$

Then
At t = 0:

$$\mathbf{w}_0^T = 0$$
$$\frac{1}{b}\sum_{x_j\in\mathbf{B}_0}\nabla_{\mathbf{w}_0}\mathcal{L}^T(\mathbf{x}_j,\mathbf{w}_0) = \frac{2}{b}(\mathbf{w}_0^T\mathbf{B}_0^T\mathbf{B}_0 - t^T\mathbf{B}_0)$$

At t = 1:

$$\begin{aligned}\mathbf{w}_1^T &= \mathbf{w}_0^T - \frac{\eta}{b}\sum_{x_j\in\mathbf{B}_0}\nabla_{\mathbf{w}_0}\mathcal{L}^T(\mathbf{x}_j,\mathbf{w}_0) \\ &= -\frac{2\eta}{b}\mathbf{t^T}\mathbf{B}_0 \\ &= -\frac{2\eta}{b}(\mathbf{t^T}\mathbf{S}_0^T)\mathbf{X}\end{aligned}$$
$$\frac{1}{b}\sum_{x_j\in\mathbf{B}_1}\nabla_{\mathbf{w}_1}\mathcal{L}^T(\mathbf{x}_j,\mathbf{w}_1) = \frac{2}{b}(\mathbf{w}_1^T\mathbf{B}_1^T\mathbf{B}_1 - t^T\mathbf{B}_1)$$

At t = 2:

$$\begin{aligned}\mathbf{w}_2^T &= \mathbf{w}_1^T - \frac{\eta}{b}\sum_{x_j\in\mathbf{B}_1}\nabla_{\mathbf{w}_1}\mathcal{L}^T(\mathbf{x}_j,\mathbf{w}_1) \\ &= -\frac{2\eta}{b}t^T\mathbf{S}_0^T\mathbf{X} - \frac{2\eta}{b}(\mathbf{w}_1^T\mathbf{B}_1^T\mathbf{B}_1 - t^T\mathbf{B}_1) \\ &= -\frac{2\eta}{b}(t^T\mathbf{S}_0^T)\mathbf{X} - \frac{2\eta}{b}(\mathbf{w}_1^T\mathbf{X}^T\mathbf{S}_1\mathbf{S}_1^T\mathbf{X} - t^T\mathbf{S}_1^T\mathbf{X}) \\ &= -\frac{2\eta}{b}(t^T\mathbf{S}_0^T\mathbf{X} + \mathbf{w}_1^T\mathbf{X}^T\mathbf{S}_1\mathbf{S}_1^T\mathbf{X} - t^T\mathbf{S}_1^T\mathbf{X}) \\ &= -\frac{2\eta}{b}(t^T\mathbf{S}_0^T + \mathbf{w}_1^T\mathbf{X}^T\mathbf{S}_1\mathbf{S}_1^T - t^T\mathbf{S}_1^T)\mathbf{X}\end{aligned}$$

$(t^T\mathbf{S}_0^T + \mathbf{w}_1^T\mathbf{X}^T\mathbf{S}_1\mathbf{S}_1^T - t^T\mathbf{S}_1^T)$ has a dimension of $1\times n$
We can see $\hat{\mathbf{w}}$ is always a span of $\mathbf{X}$

Hence, we can let $\hat{\mathbf{w}} = \mathbf{X^T a}$ for some $\mathbf{a} \in \mathbb{R}^n$, like what we did in HW1.
We can just use the same method to prove $\hat{\mathbf{w}} = \mathbf{X^T}(\mathbf{XX^T})^{-1}\mathbf{t} = \mathbf{w}^*$

For mini-batch, the loss is : $L_B = \frac{1}{b}\|\mathbf{Bw} - \mathbf{t_B}\|_2^2$ where $\mathbf{B} = \mathbf{S^T X}$, $\mathbf{t_B} = \mathbf{S^T t}$, for some $\mathbf{S} \in \mathbb{R}^{n\times d}$

So if we let:

$$\begin{aligned}\nabla_{\mathbf{w}} L_B^T &= \frac{2}{b}(\mathbf{w}^T\mathbf{B}^T\mathbf{B} - \mathbf{t}_B^T\mathbf{B}) = 0\\ (\mathbf{w}^T\mathbf{B}^T - \mathbf{t}_B^T)\mathbf{B} &= 0\\ \mathbf{w}^T\mathbf{B}^T - \mathbf{t}_B^T &= 0\end{aligned}$$

We know: $\mathbf{B} = \mathbf{S}^T\mathbf{X}$, $\mathbf{t}_B = \mathbf{S}^T\mathbf{t}$, $\mathbf{w} = \mathbf{X}^T\mathbf{a}$

$$\begin{aligned}\text{Then}\quad \mathbf{a}^T\mathbf{XX}^T\mathbf{S} - \mathbf{t}^T\mathbf{S} &= 0\\ (\mathbf{a}^T\mathbf{XX}^T - \mathbf{t}^T)\mathbf{S} &= 0\\ \text{So}\quad \mathbf{a}^T\mathbf{XX}^T - \mathbf{t}^T &= 0\\ \mathbf{a}^T &= \mathbf{t}^T(\mathbf{XX}^T)^{-1}\end{aligned}$$

From the assumption $\mathbf{X}$ is full rank and d>n, we know XX^T is invertible
Then $\hat{\mathbf{w}} = \mathbf{X}^T\mathbf{a} = \mathbf{X}^T(\mathbf{XX}^T)^{-1}\mathbf{t} = \mathbf{w}^*$

1.2.1

Given that $\mathbf{w_0} = [0,0]$, $v_{1,-1} = 0$, $v_{2,-1} = 0$
We know:

$$\begin{aligned}\nabla_{\mathbf{w}}\mathcal{L}^T &= \frac{2}{n}\left(\mathbf{w}^T\mathbf{X}^T\mathbf{X} - t^T\mathbf{X}\right)\\ \nabla_{w_{i,t}}\mathcal{L} &= \frac{2}{n}\sum_{j=1}^{n}\left(\mathbf{w_t}^T\mathbf{x}^{(j)} - t^{(j)}\right)\mathbf{x}_i^{(j)}\\ &\text{At } t = 0:\\ \nabla_{w_{1,0}}\mathcal{L} &= \frac{2}{n}\left([0,0]\,[2,1]^T - 2\right)2 = -2\\ \nabla_{w_{2,0}}\mathcal{L} &= \frac{2}{n}\left([0,0]\,[2,1]^T\right)1 = -1\end{aligned}$$

Now for the updates:

$$\begin{aligned}V_{1,0} &= \beta(V_{1,-1}) + (1-\beta)\left(\nabla_{w_{1,0}}\mathcal{L}(w_{1,0})\right)^2\\ &= (1-\beta)(-2)^2\\ &= 4(1-\beta)\\ V_{2,0} &= \beta(V_{2,-1}) + (1-\beta)\left(\nabla_{w_{2,0}}\mathcal{L}(w_{2,0})\right)^2\\ &= (1-\beta)(-1)^2\\ &= (1-\beta)\end{aligned}$$

$$
\begin{aligned}
w_{1,1} &= w_{1,0} - \frac{\eta}{\sqrt{V_{1,0}} + \epsilon} \nabla_{w_{1,0}} \mathcal{L}(w_{1,0}) \\
&= 0 - \frac{\eta}{\sqrt{4(1-\beta)} + \epsilon}(-2) \\
&= \frac{2\eta}{2\sqrt{1-\beta} + \epsilon} \\
w_{2,1} &= w_{2,0} - \frac{\eta}{\sqrt{V_{2,0}} + \epsilon} \nabla_{w_{2,0}} \mathcal{L}(w_{2,0}) \\
&= 0 - \frac{\eta}{\sqrt{1-\beta} + \epsilon}(-1) \\
&= \frac{\eta}{\sqrt{1-\beta} + \epsilon}
\end{aligned}
$$

Since η, β, ϵ are hyper-parameters, so there is a chance that we set $\eta = \frac{2}{7}, \beta = 0.91, \epsilon = 0.2$:

Then

$$
\begin{aligned}
w_{1,1} &= \frac{2}{7}\left[\frac{2}{2\sqrt{1-0.91} + 0.2}\right] = \frac{4/7}{2 \times 0.3 + 0.2} = \frac{5}{7} \\
w_{2,1} &= \frac{2/7}{\sqrt{1-0.91} + 0.2} = \frac{2/7}{0.5} = \frac{4}{7}
\end{aligned}
$$

Then, the loss will be:

$$
\mathcal{L} = ([w_{1,1}, w_{2,1}]\,[x_1, x_2]^T - t)^2 = (\frac{5}{7} \times 2 + \frac{4}{7} \times 1 - 2)^2 = 0
$$

Hence, the optimization will stop at this case.
However, $w_{1,1} = \frac{5}{7}$ and $w_{2,1} = \frac{4}{7}$ are not the minimum norm solution

Note: the minimum norm solution is:

$$
\begin{aligned}
\mathbf{w} &= \mathbf{X}^T (\mathbf{X}\mathbf{X}^T)^{-1} t \\
&= \begin{bmatrix} 2 \\ 1 \end{bmatrix} \left([2, 1] \begin{bmatrix} 2 \\ 1 \end{bmatrix} \right)^{-1} 2 \\
&= \begin{bmatrix} 2 \\ 1 \end{bmatrix} \cdot 5^{-1} \cdot 2 \\
&= \begin{bmatrix} 4/5 \\ 2/5 \end{bmatrix}
\end{aligned}
$$

Therefore, this is a counter-example that RMSProp does NOT always obtains the minimum norm solution.

2.1.1

$$\begin{tikzcd}
t \arrow[dr] & & & W_0 \arrow[d] & t \arrow[d] & & W_1 \arrow[d] & t \arrow[d] \\
W_0 \arrow[r] & \tilde{L}_0 \arrow[r] & \nabla_{W_0}\tilde{L}_0 \arrow[r] & W_1 \arrow[r] & \tilde{L}_1 \arrow[r] & \nabla_{W_1}\tilde{L}_1 \arrow[r] & W_2 \arrow[r] & \tilde{L}_2 \\
x \arrow[ur] & & & \eta \arrow[u] & x \arrow[u] & & \eta \arrow[u] & x \arrow[u] \\
\tilde{\lambda} \arrow[uur] & & & & \tilde{\lambda} \arrow[uu, bend right] & & & \tilde{\lambda} \arrow[uu, bend right]
\end{tikzcd}$$

2.1.2

Forward propagation: In every iteration, $\mathbf{w_{t+1}}$ is updated by $\mathbf{w_t}$ and $\nabla_{\mathbf{w}_t}\tilde{\mathcal{L}}$, by the formula $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \frac{\eta}{n}\sum_{\mathbf{x}_j \in X} \nabla_{\mathbf{w}_t}\tilde{\mathcal{L}}(\mathbf{x}_j, \mathbf{w}_t)$.
From the computational graph, we can see, if we want to calculate $\nabla_\eta \tilde{\mathcal{L}}_t$, we need store all $\mathbf{w_1}$ ... $\mathbf{w_t}$, because every $\mathbf{w_t}$ is calculated with η.
When there are t iterations, we need to save $\mathbf{w_1}$ ... $\mathbf{w_t}$. When there are 2t iterations, we need to save $\mathbf{w_1}$ ... $\mathbf{w_{2t}}$. Hence, there is a linear relationship. And the memory to save X, λ, t is constant with respect to t. Therefore, the memory complexity is O(t)

Backward propagation: During back-propagation, we need to compute the gradient of the loss with respect to the learning rate η. In each iteration, we need to store $\nabla_{\mathbf{w_t}}\tilde{\mathcal{L}}$ and $\nabla_\eta \mathbf{w}_t$. Then we move to t-1^{th} iterations, and store $\nabla_{\mathbf{w}_{t-1}}\tilde{\mathcal{L}}$ and $\nabla_\eta \mathbf{w}_{t-1}$.
Hence, where there t iterations, we need to store $\nabla_{\mathbf{w}_1}\tilde{\mathcal{L}}...\nabla_{\mathbf{w}_t}\tilde{\mathcal{L}}$ and $\nabla_\eta \mathbf{w}_1...\nabla_\eta \mathbf{w}_t$. when there 2t iterations, we need to store $\nabla_{\mathbf{w}_1}\tilde{\mathcal{L}}...\nabla_{\mathbf{w}_{2t}}\tilde{\mathcal{L}}$ and $\nabla_\eta \mathbf{w}_1...\nabla_\eta \mathbf{w}_{2t}$.
Hence, there is a linear relationship. And the memory to save X, λ, t is constant with respect to t. Therefore, the memory complexity is O(t)

2.2.1

$$\begin{aligned}
\tilde{\mathcal{L}}_0 &= \frac{1}{n}\|\mathbf{X}\mathbf{w}_0 - \mathbf{t}\|_2^2 \\
\mathbf{w}_1 &= \mathbf{w}_0 - \eta \nabla_{\mathbf{w}_0}\tilde{\mathcal{L}}_0 \\
&= \mathbf{w}_0 - \eta \left[\frac{2}{n}\mathbf{X}^T(\mathbf{X}\mathbf{w}_0 - \mathbf{t})\right]
\end{aligned}$$

Let $\mathbf{a} = \mathbf{X}\mathbf{w}_0 - \mathbf{t}$

Then

$$\begin{aligned}
\mathbf{w}_1 &= \mathbf{w}_0 - \eta\frac{2}{n}\mathbf{X}^T(\mathbf{X}\mathbf{w}_0 - t) \\
&= \mathbf{w}_0 - \frac{2\eta}{n}\mathbf{X}^T\mathbf{a}
\end{aligned}$$

Then

$$\begin{aligned}
\mathcal{L}_1 &= \frac{1}{n}\|\mathbf{X}\mathbf{w}_1 - \mathbf{t}\|_2^2 \\
&= \frac{1}{n}\|\mathbf{X}(\mathbf{w}_0 - \frac{2\eta}{n}\mathbf{X}^T\mathbf{a}) - t\|_2^2 \\
&= \frac{1}{n}\|\mathbf{X}\mathbf{w}_0 - \frac{2\eta}{n}\mathbf{X}\mathbf{X}^T\mathbf{a} - t\|_2^2 \\
&= \frac{1}{n}\|\mathbf{a} - \frac{2\eta}{n}\mathbf{X}\mathbf{X}^T\mathbf{a}\|_2^2
\end{aligned}$$

2.2.3

$$\nabla_\eta \mathcal{L}_1 = \frac{2}{n}\left(a - \frac{2\eta}{n}\mathbf{X}\mathbf{X}^T a\right)^T \left(-\frac{2}{n}\mathbf{X}\mathbf{X}^T a\right)$$

Set $\nabla_\eta \mathcal{L}_1 = 0$

$$\begin{aligned}
0 &= \frac{2}{n}\left(\mathbf{a} - \frac{2\eta}{n}\mathbf{X}\mathbf{X}^T\mathbf{a}\right)^T\left(-\frac{2}{n}\mathbf{X}\mathbf{X}^T\mathbf{a}\right)\\
0 &= \left(\mathbf{a}^T - \frac{2\eta}{n}\mathbf{a}^T\mathbf{X}\mathbf{X}^T\right)(\mathbf{X}\mathbf{X}^T\mathbf{a})\\
0 &= \mathbf{a}^T\mathbf{X}\mathbf{X}^T\mathbf{a} - \frac{2\eta}{n}\mathbf{a}^T\mathbf{X}\mathbf{X}^T\mathbf{X}\mathbf{X}^T\mathbf{a}\\
\frac{2\eta}{n}\mathbf{a}^T\mathbf{X}\mathbf{X}^T\mathbf{X}\mathbf{X}^T\mathbf{a} &= \mathbf{a}^T\mathbf{X}\mathbf{X}^T\mathbf{a}\\
\frac{2\eta}{n}\|\mathbf{X}\mathbf{X}^T\mathbf{a}\|_2^2 &= \|\mathbf{X}^T\mathbf{a}\|_2^2\\
\eta &= \frac{n}{2}\frac{\|\mathbf{X}^T\mathbf{a}\|_2^2}{\|\mathbf{X}\mathbf{X}^T\mathbf{a}\|_2^2}
\end{aligned}$$

2.3.1

One using $\tilde{\mathcal{L}}$:

$$\begin{aligned}
\tilde{\mathcal{L}}_0 &= \frac{1}{n}\|\mathbf{X}\mathbf{w}_0 - t\|_2^2 + \tilde{\lambda}\|\mathbf{w}_0\|_2^2\\
\mathbf{w}_1 &= \mathbf{w}_0 - \eta\nabla_{\mathbf{w}_0}\tilde{\mathcal{L}}_0\\
&= \mathbf{w}_0 - \eta\left[\frac{2}{n}\mathbf{X}^T(\mathbf{X}\mathbf{w}_0 - t) + 2\tilde{\lambda}\mathbf{w}_0\right]\\
&= -\eta\frac{2}{n}\mathbf{X}^T(\mathbf{X}\mathbf{w}_0 - t) + (1 - 2\eta\tilde{\lambda})\mathbf{w}_0\\
&= (1 - 2\eta\tilde{\lambda})\mathbf{w}_0 - \eta\frac{2}{n}\mathbf{X}^T(\mathbf{X}\mathbf{w}_0 - t)
\end{aligned}$$

One using $\mathcal{L}$:

$$\begin{aligned}
\mathcal{L}_0 &= \frac{1}{n}\|\mathbf{X}\mathbf{w}_0 - t\|_2^2\\
\mathbf{w}_1 &= (1-\lambda)\mathbf{w}_0 - \eta\nabla_{\mathbf{w}_0}\mathcal{L}_0\\
&= (1-\lambda)\mathbf{w}_0 - \eta\frac{2}{n}\mathbf{X}^T(\mathbf{X}\mathbf{w}_0 - t)
\end{aligned}$$

2.3.2

From 2.3.1, we can see: when $1 - 2\eta\tilde{\lambda} = 1 - \lambda$, the two expressions for $\mathbf{w_1}$ are same.
In other words, $\tilde{\lambda} = \frac{\lambda}{2\eta}$. Here, $\eta, \tilde{\lambda}, \lambda$ are all hyper-parameters. So we can multiply $\tilde{\lambda}$ by 2η so that $2\eta\tilde{\lambda} = \lambda$

3.1

To make input size = output size when there is a 3×3 filter, we need 1×1 padding on each edge

The central value '5' is positive, and the surrounding values are negative '-1'. The central value '5' in the filter emphasizes the value of the pixel in the middle of the region. The surrounding values '-1' penalize the neighboring pixels
This filter tends to enhance regions with high contrast between pixels. For example, the edges of an object, where sharp changes in intensity between neighboring pixels happen, can be detected by this filter.

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	1	1	0
0	1	1	1	1	0	0
0	0	1	1	1	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0

$I * J$

0	0	-1	-1	-1
-1	-2	3	2	4
4	2	1	2	-2
-2	3	1	3	-1
0	-2	4	-2	0

3.2

For one one image, the dimension are represented by {Height, Width, Channel}

CNN:

Conv 3-32:

- Input: $32 \times 32 \times 3$
- Output: $32 \times 32 \times 32$
- Kernel: 3×3

$32 \times 32 \times 32 = 32768$ neurons
$3 \times 3 \times 3 \times 32 = 864$ parameters

Max Pool:

- Input: $32 \times 32 \times 32$
- Output: $16 \times 16 \times 32$

$16 \times 16 \times 32 = 8192$ neurons
0 parametres

Conv 3-64:

- **Input**: $16 \times 16 \times 32$
- **Output**: $16 \times 16 \times 64$
- **Kernel**: 3×3

$16 \times 16 \times 64 = 16384$ neurons
$3 \times 3 \times 32 \times 64 = 18432$ parameters

Max Pool:

- Input: $16 \times 16 \times 64$
- Output: $8 \times 8 \times 64$

$8 \times 8 \times 64 = 4096$ neurons
0 parameters

Conv 3-3:

- **Input**: $8 \times 8 \times 64$
- **Output**: $8 \times 8 \times 3$
- **Kernel**: 3×3

$8 \times 8 \times 3 = 192$ neurons
$3 \times 3 \times 64 \times 3 = 1728$ parameters

In total, CNN has
the number of neurons = 32768+ 8192 + 16384 + 4096 + 192 = 61632
the number of parameters = 864 + 0 + 18432 + 0 + 1728 = 21024

FCNN:

FC:

- Input: $32 \times 32 \times 3$
- Output: $32 \times 32 \times 3$

$32 \times 32 \times 3 = 3072$ neurons
$(32 \times 32 \times 3)^2 = 9437184$ parameters

Max Pool:

- Input: $32 \times 32 \times 3$
- Output: $16 \times 16 \times 3$

$16 \times 16 \times 3 = 768$ neurons
0 parametres

FC:

- Input: $16 \times 16 \times 3$
- Output: $16 \times 16 \times 3$

$16 \times 16 \times 3 = 768$ neurons
$(16 \times 16 \times 3)^2 = 589824$ parameters

Max Pool:

- Input: $16 \times 16 \times 3$
- Output: $8 \times 8 \times 3$

$8 \times 8 \times 3 = 192$ neurons
0 parameters

FC:

- **Input**: $8 \times 8 \times 3$
- **Output**: $8 \times 8 \times 3$

$8 \times 8 \times 3 = 192$ neurons
$(8 \times 8 \times 3)^2 = 36864$ parameters

In total, FCNC has
the number of neurons = 3072+ 768 + 768 + 192 + 192 = 4992
the number of parameters = 9437184 + 0 + 589824 + 0 + 36864 = 10063872

One disadvantage of having more trainable parameters: when there are too many trainable parameters, the computational cost will be increased. The training process will be slowed down because more gradients need to be calculated to update parameters at each iteration.

3.3

1. **Stride:** Stride refers to the number of pixels the kernel filter move horizontally and vertically over the input. For example, a stride of 1*1 means the kernel filter moves 1 pixel at a time. A stride of 2*2 means the kernel filter moves 2 pixels at a time. A larger stride can reduce the overlap between neighboring receptive fields. When we have two or more convolution layers, large stride increases the distance between the receptive fields of neurons and thus increases the size of the receptive field faster.

2. **Kernel Size:** The size of the kernel can affect the size of the receptive field of a neuron. Kernel with large size means larger receptive field, because larger kernel can let neuron to aggregate information from a wider area of the input. For example, a 5*5 kernel has a receptive field of 5*5, while a 3*3 kernel has a receptive field of 3*3.

3. **Number of convolution layers:** More convolution layers means larger receptive field. In each convolution layer, the information in the previous layer is aggregated again. For example, the receptive field of a neuron in this layer is 3*3. If we stack another convolution layer with kernel size=3*3 on top of this layer. Then the receptive field of the neuron after this new layer will be aggregate three neurons in the previous layers.

4.2

The results are not good enough, since some pictures are very blurry. For example, the horse in the picture on the third row and eighth column is very difficult to identify. Grey color and brown color appear on one horse at the same time.

why:

- This could be caused by the max pooling and upsampling layers, which might make neighboring pixels blend together and lead to incorrect colors in certain regions.
- Also, these max pooling can cause loss of important information, and upsampling layers might bring in redundant and noisy information.
- At last, the 'Upsample' and 'MaxPool' layer have no learnable parameters, so this layer essentially does not learn any specific patterns or features by iterations.

4.3

For input image with [NIC, 32, 32]

Block 1:

Input size: [NIC, 32, 32]
1. **Conv2d**
Number of weights = NIC $\times k \times k \times NF$
Number of outputs = $NF \times 32 \times 32$
Number of connections = $k \times k \times 32 \times 32 \times NF \times NIC$
2. **MaxPool2d**
Number of weights = 0
Number of outputs = $NF \times 16 \times 16$
Number of connections = 0
3. **BatchNorm2d**
Number of weights = 0
Number of outputs = $NF \times 16 \times 16$
Number of connections = 0
4. **ReLU**
Number of weights = 0
Number of outputs = $NF \times 16 \times 16$
Number of connections = 0
Output Size: [NF, 16, 16]
In total, for Block 1:
Total number of weights = $NIC \times k \times k \times NF$
Total number of outputs = $(NF \times 32 \times 32) + (NF \times 16 \times 16) + (NF \times 16 \times 16) + (NF \times 16 \times 16) = 1792 \times NF$
Total number of connections = $k \times k \times 32 \times 32 \times NF \times NIC$

Block 2:

Input size: [NF, 16, 16]
1. **Conv2d**
Number of weights = $NF \times k \times k \times 2NF$
Number of outputs = $2NF \times 16 \times 16$
Number of connections = $k \times k \times 16 \times 16 \times NF \times 2NF$

2. **MaxPool2d**
Number of weights $= 0$
Number of outputs $= 2NF \times 8 \times 8$
Number of connections $= 0$
3. **BatchNorm2d**
Number of weights $= 0$
Number of outputs $= 2NF \times 8 \times 8$
Number of connections $= 0$
4. **ReLU**
Number of weights $= 0$
Number of outputs $= 2NF \times 8 \times 8$
Number of connections $= 0$
Output Size: [2NF, 8, 8]
In total, for Block 2:
Total number of weights $= NF \times k \times k \times 2NF$
Total number of outputs $= (2NF \times 16 \times 16) + (2NF \times 8 \times 8) + (2NF \times 8 \times 8) + (2NF \times 8 \times 8) = 896 \times NF$
Total number of connections $= k \times k \times 16 \times 16 \times NF \times 2NF$

Block 3:

Input size: [2NF, 8, 8]
1. **Conv2d**
Number of weights $= 2NF \times k \times k \times NF$
Number of outputs $= NF \times 8 \times 8$
Number of connections $= k \times k \times 8 \times 8 \times 2NF \times NF$
2. **Upsample**
Number of weights $= 0$
Number of outputs $= NF \times 16 \times 16$
Number of connections $= 0$
3. **BatchNorm2d**
Number of weights $= 0$
Number of outputs $= NF \times 16 \times 16$
Number of connections $= 0$
4. **ReLU**
Number of weights $= 0$
Number of outputs $= NF \times 16 \times 16$
Number of connections $= 0$
Output Size: [NF, 16, 16]
In total, for Block 3:
Total number of weights $= 2NF \times k \times k \times NF$
Total number of outputs $= (NF \times 8 \times 8) + (NF \times 16 \times 16) + (NF \times 16 \times 16) + (NF \times 16 \times 16) = 832 \times NF$
Total number of connections $= k \times k \times 8 \times 8 \times 2NF \times NF$

Block 4:

Input size: [NF, 16, 16]
1. **Conv2d**
Number of weights $= NF \times k \times k \times NC$
Number of outputs $= NC \times 16 \times 16$
Number of connections $= k \times k \times 16 \times 16 \times NF \times NC$
2. **Upsample**
Number of weights $= 0$
Number of outputs $= NC \times 32 \times 32$
Number of connections $= 0$
3. **BatchNorm2d**
Number of weights $= 0$
Number of outputs $= NC \times 32 \times 32$
Number of connections $= 0$
4. **ReLU**
Number of weights $= 0$

Number of outputs = $NC \times 32 \times 32$
Number of connections = 0
Output Size: [NC, 32, 32]
In total, for Block 4:
Total number of weights = $NF \times k \times k \times NC$
Total number of outputs = $(NC \times 16 \times 16) + (NC \times 32 \times 32) + (NC \times 32 \times 32) + (NC \times 32 \times 32) = 3328 \times NC$
Total number of connections = $k \times k \times 16 \times 16 \times NF \times NC$

Block 5:

Input size: [NC, 32, 32]
1. **Conv2d**
Number of weights = $NC \times k \times k \times NC$
Number of outputs = $NC \times 32 \times 32$
Number of connections = $k \times k \times 32 \times 32 \times NC \times NC$
Output Size: [NC, 32, 32]
In total, for Block 5:
Total number of weights = $NC \times k \times k \times NC$
Total number of outputs = $NC \times 32 \times 32 = 1024 \times NC$
Total number of connections = $k \times k \times 32 \times 32 \times NC \times NC$

In total, when input size is [NIC, 32, 32]:

- Total number of weights = $(NIC \times k \times k \times NF) + (NF \times k \times k \times 2NF) + (2NF \times k \times k \times NF) + (NF \times k \times k \times NC) + (NC \times k \times k \times NC)$
- Total number of outputs = $1792 \times NF + 896 \times NF + 832 \times NF + 3328 \times NC + 1024 \times NC = 3520 \times NF + 4352 \times NC$
- Total number of connections = $(k \times k \times 32 \times 32 \times NF \times NIC) + (k \times k \times 16 \times 16 \times NF \times 2NF) + (k \times k \times 8 \times 8 \times 2NF \times NF) + (k \times k \times 16 \times 16 \times NF \times NC) + (k \times k \times 32 \times 32 \times NC \times NC)$

Similary, for input image with [NIC, 64, 64]

Block 1:

Input size: [NIC, 64, 64]
1. **Conv2d**
Number of weights = $NIC \times k \times k \times NF$
Number of outputs = $NF \times 64 \times 64$
Number of connections = $k \times k \times 64 \times 64 \times NF \times NIC$
2. **MaxPool2d**
Number of weights = 0
Number of outputs = $NF \times 32 \times 32$
Number of connections = 0
3. **BatchNorm2d**
Number of weights = 0
Number of outputs = $NF \times 32 \times 32$
Number of connections = 0
4. **ReLU**
Number of weights = 0
Number of outputs = $NF \times 32 \times 32$
Number of connections = 0
Output Size: [NF, 32, 32]
In total, for Block 1:
Total number of weights = $NIC \times k \times k \times NF$
Total number of outputs = $(NF \times 64 \times 64) + (NF \times 32 \times 32) + (NF \times 32 \times 32) + (NF \times 32 \times 32) = 7168 \times NF$
Total number of connections = $k \times k \times 64 \times 64 \times NF \times NIC$

Block 2:

Input size: [NF, 32, 32]
1. **Conv2d**
Number of weights $= NF \times k \times k \times 2NF$
Number of outputs $= 2NF \times 32 \times 32$
Number of connections $= k \times k \times 32 \times 32 \times NF \times 2NF$
2. **MaxPool2d**
Number of weights $= 0$
Number of outputs $= 2NF \times 16 \times 16$
Number of connections $= 0$
3. **BatchNorm2d**
Number of weights $= 0$
Number of outputs $= 2NF \times 16 \times 16$
Number of connections $= 0$
4. **ReLU**
Number of weights $= 0$
Number of outputs $= 2NF \times 16 \times 16$
Number of connections $= 0$
Output Size: [2NF, 16, 16]
In total, for Block 2:
Total number of weights $= NF \times k \times k \times 2NF$
Total number of outputs $= (2NF \times 32 \times 32) + (2NF \times 16 \times 16) + (2NF \times 16 \times 16) + (2NF \times 16 \times 16) = 3584 \times NF$
Total number of connections $= k \times k \times 32 \times 32 \times NF \times 2NF$

Block 3:

Input size: [2NF, 16, 16]
1. **Conv2d**
Number of weights $= 2NF \times k \times k \times NF$
Number of outputs $= NF \times 16 \times 16$
Number of connections $= k \times k \times 16 \times 16 \times 2NF \times NF$
2. **Upsample**
Number of weights $= 0$
Number of outputs $= NF \times 32 \times 32$
Number of connections $= 0$
3. **BatchNorm2d**
Number of weights $= 0$
Number of outputs $= NF \times 32 \times 32$
Number of connections $= 0$
4. **ReLU**
Number of weights $= 0$
Number of outputs $= NF \times 32 \times 32$
Number of connections $= 0$
Output Size: [NF, 32, 32]
In total, for Block 3:
Total number of weights $= 2NF \times k \times k \times NF$
Total number of outputs $= (NF \times 16 \times 16) + (NF \times 32 \times 32) + (NF \times 32 \times 32) + (NF \times 32 \times 32) = 3328 \times NF$
Total number of connections $= k \times k \times 16 \times 16 \times 2NF \times NF$

Block 4:

Input size: [NF, 32, 32]
1. **Conv2d**
Number of weights $= NF \times k \times k \times NC$
Number of outputs $= NC \times 32 \times 32$
Number of connections $= k \times k \times 32 \times 32 \times NF \times NC$
2. **Upsample**
Number of weights $= 0$
Number of outputs $= NC \times 64 \times 64$
Number of connections $= 0$

3. **BatchNorm2d**
Number of weights = 0
Number of outputs = $NC \times 64 \times 64$
Number of connections = 0
4. **ReLU**
Number of weights = 0
Number of outputs = $NC \times 64 \times 64$
Number of connections = 0
Output Size: [NC, 64, 64]
In total, for Block 4:
Total number of weights = $NF \times k \times k \times NC$
Total number of outputs = $(NC \times 32 \times 32) + (NC \times 64 \times 64) + (NC \times 64 \times 64) + (NC \times 64 \times 64) = 13312 \times NC$
Total number of connections = $k \times k \times 32 \times 32 \times NF \times NC$

Block 5:

Input size: [NC, 64, 64]
1. **Conv2d**
Number of weights = $NC \times k \times k \times NC$
Number of outputs = $NC \times 64 \times 64$
Number of connections = $k \times k \times 64 \times 64 \times NC \times NC$
Output Size: [NC, 64, 64]
In total, for Block 5:
Total number of weights = $NC \times k \times k \times NC$
Total number of outputs = $NC \times 64 \times 64 = 4096 \times NC$
Total number of connections = $k \times k \times 64 \times 64 \times NC \times NC$

In total, when input size is [NIC, 64, 64]:

- Total number of weights = $(NIC \times k \times k \times NF) + (NF \times k \times k \times 2NF) + (2NF \times k \times k \times NF) + (NF \times k \times k \times NC) + (NC \times k \times k \times NC)$
- Total number of outputs = $7168 \times NF + 3584 \times NF + 3328 \times NF + 13312 \times NC + 4096 \times NC = 14080 \times NF + 17408 \times NC$
- Total number of connections = $(k \times k \times 64 \times 64 \times NF \times NIC) + (k \times k \times 32 \times 32 \times NF \times 2NF) + (k \times k \times 16 \times 16 \times 2NF \times NF) + (k \times k \times 32 \times 32 \times NF \times NC) + (k \times k \times 64 \times 64 \times NC \times NC)$

5.2

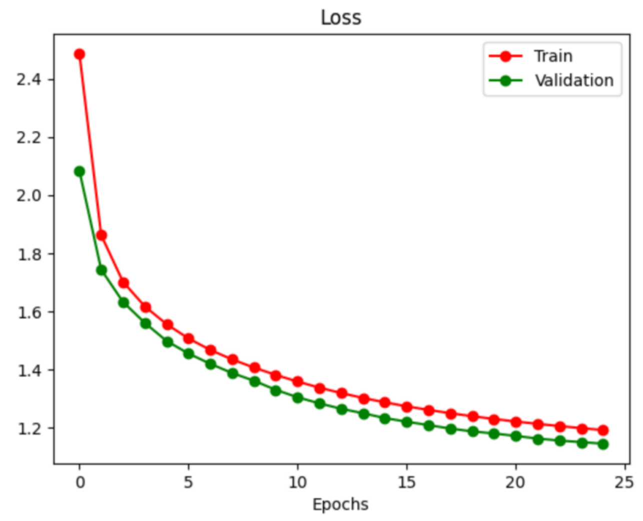

Figure 1: training and validation loss for 25 epochs

5.3

The result is better than Section 4. The highest accuracy in Section 4 after 25 epochs is 41.3%. Here, we get an accuracy of 55.4%. Also, the ConvTransposeNet model (1.1440) result in lower validation loss than the PoolUpsampleNet model (1.5877).

Reason:

- In PoolUpsampleNet model, the 'Upsample' layer increase the heights and weights based on a fixed operation, for example, directly replicate the outside values. 'Upsample' layer just simply double the size of heights and widths. This means the 'Upsample' layer have no learnable parameters, so this layer essentially does not learn any specific patterns or features by iterations. Similarly, in 'MaxPool' layer, the process is also fixed. The 'MaxPool' layer just make the heights and widths half. The 'MaxPool' layer also have no learnable parameters.
- However, in ConvTransposeNet model, the half-reducing of heights and widths is accomplished by convolutions layers with stride, and doubling the size is done by the transposed convolutions layers. Both convolutions layers with stride and transposed convolutions layers have plenty of learnable parameters. As a result, these layers can be optimized during training. This ability of ConvTransposeNet model enables itself to better retain important spatial information about color from the image.

5.4

Equations to calculate shape of the output tensors for nn.Conv2d:

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding} - \text{dilation} \times (\text{kernel\_size} - 1) - 1}{\text{stride}} \right\rfloor + 1$$
$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding} - \text{dilation} \times (\text{kernel\_size} - 1) - 1}{\text{stride}} \right\rfloor + 1$$

Equations to calculate shape of the output tensors for nn.ConvTranspose2d:

$$H_{out} = (H_{in} - 1) \times \text{stride} - 2 \times \text{padding} + \text{dilation} \times (\text{kernel\_size} - 1) + \text{output\_padding} + 1$$
$$W_{out} = (W_{in} - 1) \times \text{stride} - 2 \times \text{padding} + \text{dilation} \times (\text{kernel\_size} - 1) + \text{output\_padding} + 1$$

Conv2d in First Block:

Input size: 32*32
Output size: 16*16
According to the formula:

$$16 = \frac{32 + 2 \times \text{padding} - 1 \times (\text{kernel} - 1) - 1}{2} + 1$$
$$15 \times 2 = 32 + 2 \times \text{padding} - 1 \times (\text{kernel} - 1) - 1$$
$$31 = 32 + 2 \times \text{padding} - \text{kernel} + 1$$
$$-2 = 2 \times \text{padding} - \text{kernel}$$

When kernel = 4, padding = 1
When kernel = 5, padding = 1.5 = 2 (when the answer is not an integer, we need to round up)
Here, we can see, when the kernel size increases, more padding is needed to maintain the same shape of tensors in figure 1(b)

Conv2d in Second Block:

Input size: 16*16
Output size: 8*8
According to the formula:

$$8 = \frac{16 + 2 \times \text{padding} - 1 \times (\text{kernel} - 1) - 1}{2} + 1$$
$$14 = 16 + 2 \times \text{padding} - \text{kernel}$$
$$-2 = 2 \times \text{padding} - \text{kernel}$$

When kernel = 4, padding = 1
When kernel = 5, padding = 1.5 = 2
Here, we can see, when the kernel size increases, more padding is needed to maintain the same shape of tensors in figure 1(b)

ConvTranspose2d in Third Block:

Input size: 8*8
Output size: 16*16
According to the formula:

$$16 = (8 - 1) \times 2 - 2 \times \text{padding} + 1 \times (\text{kernel} - 1) + \text{output\_padding} + 1$$

$$16 = 14 - 2 \times \text{padding} + \text{kernel} + \text{out\_padding}$$

$$2 = \text{kernel} + \text{out\_padding} - 2 \times \text{padding}$$

$$\text{kernel} = 2 - \text{out\_padding} + 2 \times \text{padding}$$

When kernel $= 4$, $2 = 4 + \text{out\_padding} - 2 \times \text{padding}$.
One example to make the equation work is 'padding = 1, output_padding = 0'

When kernel $= 5$, $2 = 5 + \text{out\_padding} - 2 \times \text{padding}$
One example to make the equation work is 'padding = 2, output_padding = 1'

Here, we can see, when the kernel size increases, and out-padding size is fixed, then more padding is needed
when the kernel size increases, and padding size is fixed, then less out-padding is needed

ConvTranspose2d in Fourth Block:

Input size: 16*16
Output size: 32*32
According to the formula:

$$32 = (16 - 1) \times 2 - 2 \times \text{padding} + 1 \times (\text{kernel} - 1) + \text{output\_padding} + 1$$

$$32 = 30 - 2 \times \text{padding} + \text{kernel} + \text{out\_padding}$$

$$2 = \text{kernel} + \text{out\_padding} - 2 \times \text{padding}$$

When kernel $= 4$, $2 = 4 + \text{out\_padding} - 2 \times \text{padding}$.
One example to make the equation work is 'padding = 1, output_padding = 0'

When kernel $= 5$, $2 = 5 + \text{out\_padding} - 2 \times \text{padding}$
One example to make the equation work is 'padding = 2, output_padding = 1'

Here, we can see, when the kernel size increases, and out-padding size is fixed, then more padding is needed
when the kernel size increases, and padding size is fixed, then less out-padding is needed

Conv2d in Fifth Block:

Input size: 32*32
Output size: 32*32
According to the formula:

$$32 = \frac{32 + 2 \times \text{padding} - 1 \times (\text{kernel} - 1) - 1}{1} + 1$$

$$32 = 32 + 2 \times \text{padding} - (\text{kernel} - 1) - 1 + 1$$

$$32 = 32 + 2 \times \text{padding} - \text{kernel} + 1$$

$$2 \times \text{padding} - \text{kernel} = -1$$

When kernel $= 4$, padding $= 1.5 = 2$
When kernel $= 5$, padding $= 2$
Here, we can see, when the kernel size increases, more padding is needed to maintain the same shape of tensors in figure 1(b)