Vision Architectures

Karl Stratos

1 Building Blocks

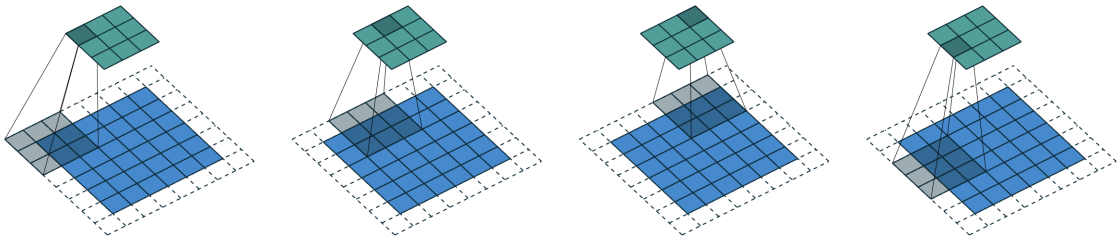
1.1 Convolution (2D)

Conv$_\theta$
Input: image $x \in \mathbb{R}^{C \times H \times W}$, number of output channels C', filter size $F \leq \min(H, W)$, stride $S \geq 1$, padding $P \geq 0$
Parameters: $\mathbf{W} \in \mathbb{R}^{C' \times C \times F \times F}$, $\mathbf{b} \in \mathbb{R}^{C'}$
Output: convolved $y \in \mathbb{R}^{C' \times H' \times W'}$ where $H' = \lfloor \frac{H+2P-F}{S} \rfloor + 1$ and $W' = \lfloor \frac{W+2P-F}{S} \rfloor + 1$ (Lemma 1.1)

1. Pad $x \in \mathbb{R}^{C \times H \times W}$ with P zeros on the edges to obtain $\tilde{x} \in \mathbb{R}^{C \times (H+2P) \times (W+2P)}$.
2. Compute $y \in \mathbb{R}^{C' \times H' \times W'}$ where

$$y_{o,i,j} = \sum_{c=1}^{C} \sum_{k=i}^{i+F-1} \sum_{l=j}^{j+F-1} W_{o,c,k,l} \times \tilde{x}_{c,k,l} + b_o$$

(Illustration by Dumoulin and Visin (2016))



$$\textbf{Conv}_\theta(x \in \mathbb{R}^{1 \times 6 \times 6}, C' = 1, F = 3, S = 2, P = 1) \in \mathbb{R}^{1 \times 3 \times 3}$$

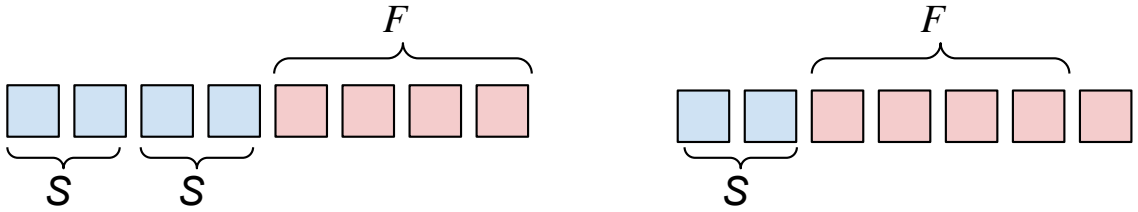
Lemma 1.1. Convolution on image shape (H, W) using filter size F and stride S with padding P yields a new image shape (H', W') where

$$H' = \left\lfloor \frac{H + 2P - F}{S} \right\rfloor + 1 \qquad W' = \left\lfloor \frac{W + 2P - F}{S} \right\rfloor + 1$$

Proof. Let Z denote either side after padding (i.e., $Z = H + 2P$ or $Z = W + 2P$). The quantity

$$\left\lfloor \frac{Z - F}{S} \right\rfloor$$

counts the number of times we slide in stride S before we are left with $F \leq m < F + S$ values. (If $Z - F$ is divisible by S, then $m = F$; otherwise $F < m < F + S$.) At this point, we apply the filter to the final possible patch of size F, possibly neglecting the last few values. See the illustration with $F = 4$ and $S = 2$, comparing $Z = 8$ vs $Z = 7$:



Thus the total number of output values is $\lfloor \frac{Z-F}{S} \rfloor + 1$. □

Fact 1.2. For any convolution on $x \in \mathbb{R}^{C \times H \times W}$ with C' output channels, filter size F, stride S, and padding P, implying an output height H' and width W', there is a unique matrix $V \in \mathbb{R}^{C'H'W' \times CHW}$ such that

$$\mathbf{Conv}_\theta(x, C', F, S, P) = (V\text{vec}(x)).\text{view}(C', H', W') + b.\text{view}(C', 1, 1)$$

where we use the broadcasting syntax for bias.

Reason. This is true because a convolution is ultimately a sum of elementwise multiplications between weights and inputs. For instance, assume single channels $C = C' = 1$, 3×3 input, kernel size $F = 2$, stride $S = 1$, and no padding. Let $x = [[x_1, x_2, x_3]; [x_4, x_5, x_6]; [x_7, x_8, x_9]]$ be any 3×3 input. The output of the convolution is a 2×2 matrix given by

$$\mathbf{Conv}_\theta(x, 1, 2, 1, 0) = \begin{pmatrix} W_1 \odot \begin{pmatrix} x_1 & x_2 \\ x_4 & x_5 \end{pmatrix} + b_1 & W_1 \odot \begin{pmatrix} x_2 & x_3 \\ x_5 & x_6 \end{pmatrix} + b_1 \\ W_1 \odot \begin{pmatrix} x_4 & x_5 \\ x_7 & x_8 \end{pmatrix} + b_1 & W_1 \odot \begin{pmatrix} x_5 & x_6 \\ x_8 & x_9 \end{pmatrix} + b_1 \end{pmatrix} \tag{1}$$

where $W_1 = [[w_1, w_2]; [w_3, w_4]] \in \mathbb{R}^{2 \times 2}$ is the filter and $b_1 \in \mathbb{R}$ is the bias (for the single output channel $o = 1$). Now consider the 4×9 matrix

$$V = \begin{bmatrix} w_1 & w_2 & 0 & w_3 & w_4 & 0 & 0 & 0 & 0 \\ 0 & w_1 & w_2 & 0 & w_3 & w_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & w_1 & w_2 & 0 & w_3 & w_4 & 0 \\ 0 & 0 & 0 & 0 & w_1 & w_2 & 0 & w_3 & w_4 \end{bmatrix} \tag{2}$$

We see that

$$V\text{vec}(x) = \begin{bmatrix} w_1 & w_2 & 0 & w_3 & w_4 & 0 & 0 & 0 & 0 \\ 0 & w_1 & w_2 & 0 & w_3 & w_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & w_1 & w_2 & 0 & w_3 & w_4 & 0 \\ 0 & 0 & 0 & 0 & w_1 & w_2 & 0 & w_3 & w_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{bmatrix} = \begin{bmatrix} w_1x_1 + w_2x_2 + w_3x_4 + w_4x_5 \\ w_1x_2 + w_2x_3 + w_3x_5 + w_4x_6 \\ w_1x_4 + w_2x_5 + w_3x_7 + w_4x_8 \\ w_1x_5 + w_2x_6 + w_3x_8 + w_4x_9 \end{bmatrix}$$

Thus $(V\text{vec}(x)).\text{view}(1, 2, 2) + b_1$ equals (1).

□

1.2 Convolution (1D)

It is clear that 2D convolution can be ablated to handle a "flat" image $x \in \mathbb{R}^{d \times T}$, which is typically interpreted as a sequence of T embeddings with d features.

Conv1D$_\theta$
Input: sequence $x \in \mathbb{R}^{d \times T}$, new dimension d', filter size $F \leq T$, stride $S \geq 1$, padding $P \geq 0$
Parameters: $\mathbf{W} \in \mathbb{R}^{d' \times d \times F}$ and $\mathbf{b} \in \mathbb{R}^{d'}$
Output: convolved $y \in \mathbb{R}^{d' \times T'}$ where $T' = \lfloor \frac{T+2P-F}{S} \rfloor + 1$

1. Pad the input $x \in \mathbb{R}^{d \times T}$ with P zeros on the boundaries $\tilde{x} \in \mathbb{R}^{d \times (T+2P)}$.
2. Compute $y \in \mathbb{R}^{d' \times T'}$ where

$$y_{o,i} = \sum_{c=1}^{C} \sum_{j=i}^{i+F-1} W_{o,c,j} \times \tilde{x}_{c,j} + b_o$$

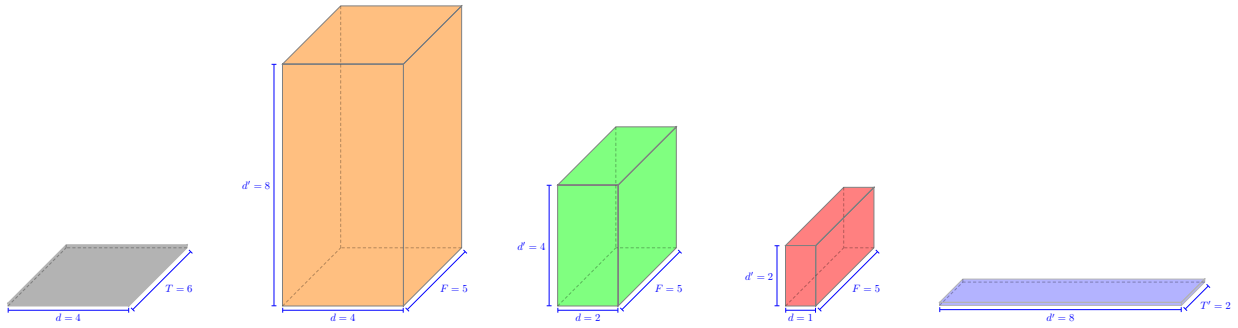
Analogously, we can define 3D convolution (though less common).

1.3 Grouped Convolution

GroupedConv$_\theta$ (visualization by Animated AI)
Input: $x \in \mathbb{R}^{C \times H \times W}$, C', F, S, P (same as **Conv**$_\theta$), number of groups $G \geq 1$ such that $\frac{C}{G}, \frac{C'}{G} \in \mathbb{N}$
Parameters: $\mathbf{W} = (\mathbf{W}^{(1)} \dots \mathbf{W}^{(G)}) \in \mathbb{R}^{C' \times \frac{C}{G} \times F \times F}$ where $\mathbf{W}^{(g)} \in \mathbb{R}^{\frac{C'}{G} \times \frac{C}{G} \times F \times F}$, $\mathbf{b} \in \mathbb{R}^{C'}$
Output: group-convolved $y \in \mathbb{R}^{C' \times H' \times W'}$ (same shape as **Conv**$_\theta$)

1. Partition the image across the input channels: $x = (x^{(1)} \dots x^{(G)})$ where $x^{(g)} \in \mathbb{R}^{\frac{C}{G} \times H \times W}$.
2. For $g = 1 \dots G$, compute $y^{(g)} \leftarrow \mathbf{Conv}_\theta(x^{(g)}, \frac{C'}{G}, F, S, P) \in \mathbb{R}^{\frac{C'}{G} \times H' \times W'}$ using filter $\mathbf{W}^{(g)} \in \mathbb{R}^{\frac{C'}{G} \times \frac{C}{G} \times F \times F}$ (no bias).
3. Return $y \leftarrow (y^{(1)} \dots y^{(G)}) + \mathbf{b}[:, \text{None}] \in \mathbb{R}^{C' \times H' \times W'}$.

We recover the usual convolution with $G = 1$. Grouped convolution has G times fewer parameters because each of the G filters handles G times fewer input *and* output channels. See the $160, 80, 40$ filter weights corresponding to $G = 1, 2, 4$ (illustrated with 1D convolution):



$x \in \mathbb{R}^{4 \times 6}$ $\quad$ $\mathbf{W} \in \mathbb{R}^{8 \times 4 \times 5}$ $\quad$ $\mathbf{W}^{(g)} \in \mathbb{R}^{4 \times 2 \times 5}$ $\quad$ $\mathbf{W}^{(g)} \in \mathbb{R}^{2 \times 1 \times 5}$ $\quad$ $y \in \mathbb{R}^{8 \times 2}$

Grouped convolution is faster because it needs fewer reads from memory and arithmatic operations. The extreme case $G = C$ is called **depthwise convolution**, which makes input channels completely independent. One way to combine depthwise convolution and cross-channel interactions is **depthwise separable convolution** defined as:

$$z \leftarrow \mathbf{GroupedConv}_\theta(x, C, F, S, P, G = C) \qquad \text{(depthwise)}$$
$$y \leftarrow \mathbf{GroupedConv}_\theta(z, C', F = 1, S = 1, P = 0, G = 1) \qquad \text{(“pointwise”)}$$

Here, $z \in \mathbb{R}^{C \times H' \times W'}$ is first computed using depthwise convolution with C intermediate output channels. Then, a full convolution with a $C' \times C \times 1 \times 1$ filter and stride 1 is applied to capture interactions between all channels within each pixel point. Thus full, depthwise, and depthwise separable convolutions have $C'CF^2$, $C'F^2$, and $CF^2 + C'C$ parameters (excluding the bias term).

1.4 Transposed Convolution

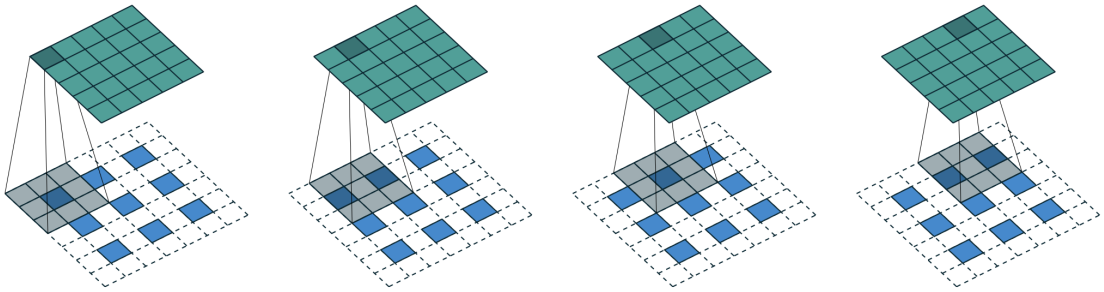
TransConv$_\theta$
Input: image $x \in \mathbb{R}^{C \times H \times W}$, number of output channels C', filter size $F \geq 1$, implicit stride $S \geq 1$ (enlargement factor), implicit padding $P \leq F - 1$
Parameters: three-dimensional filter $W_o \in \mathbb{R}^{C \times F \times F}$ and bias $b_o \in \mathbb{R}$ for all output channels $o \in \{1 \dots C'\}$
Output: larger image $y \in \mathbb{R}^{C' \times H' \times W'}$ where $H' = (H - 1)S + F - 2P$ and $W' = (W - 1)S + F - 2P$ (Lemma 1.3)

1. Pad the input $x \in \mathbb{R}^{C \times H \times W}$ with
   - $S - 1$ zeros in between the pixels
   - $F - 1 - P$ zeros on the edges

   to obtain $\tilde{x} \in \mathbb{R}^{C \times f(H) \times f(W)}$ where $f(t) = t + (t - 1)(S - 1) + 2(F - 1 - P)$.
2. Apply a stride-1 padding-0 convolution on $\tilde{x}$ using $\overline{W}_o \in \mathbb{R}^{C \times F \times F}$ where $[\overline{W}_o]_{c,i,j} = [W_o]_{c,F-i+1,F-j+1}$ as kernels:

$$y \leftarrow \mathbf{Conv}_{\{\overline{W}_o, b_o\}_{o=1}^{C'}}(\tilde{x}, C', F, 1, 0)$$

(Illustration by Dumoulin and Visin (2016))



$$\textbf{TransConv}_\theta(x \in \mathbb{R}^{1\times 3\times 3}, C' = 1, F = 3, S = 2, P = 1) \in \mathbb{R}^{5\times 5}$$

Lemma 1.3. Transposed convolution on image shape (H, W) using filter size F with implicit stride S and padding P yields a new image shape (H', W') where

$$H' = (H-1)S + F - 2P \qquad\qquad W' = (W-1)S + F - 2P$$

Proof. After padding, the input has shape $C \times f(H) \times f(W)$ where $f(t) = t + (t-1)(S-1) + 2(F-1-P)$. We apply a stride-1 padding-0 convolution with filter size F (it does not matter if the filters are flipped for the purpose of computing the shape). The convolution output has height $H' = f(H) - F + 1$ (Lemma 1.1 with unit stride), which evaluates to $H' = (H-1)S + F - 2P$. The width is similar. □

We note several oddities in transposed convolution:

- The "padding" P is used to *reduce* the padding size (additively).
- The "stride" S is used to *increase* the padding size (multiplicatively).
- After padding, the performed convolution is always *stride-1* and *padding-0*, disregarding the given P and S.
- The kernel values are flipped (both horizontally and vertically).

Fact 1.4 states that this is equivalent to (1) identifying an *implicit* stride-S padding-P convolution that reshapes a higher-resolution image to the current image shape, and (2) taking the *transpose* of the matrix-form of that convolution to recover the original shape. Note that it is not taking the inverse of the convolution (which is impossible), so it is incorrect to view it as undoing any convolution. It is also called a "fractionally-strided" convolution since it takes more than one stride to cover a single patch in the original image if $S > 1$.

Fact 1.4. Consider a transposed convolution

$$\textbf{TransConv}_\theta(\cdot, C', F, S, P) : \mathbb{R}^{C\times H\times W} \to \mathbb{R}^{C'\times H'\times W'}$$

with filter weights $W_o \in \mathbb{R}^{C\times F\times F}$ for $o \in \{1 \dots C'\}$ and bias $b \in \mathbb{R}^{C'}$. We may define a convolution

$$\textbf{Conv}_{\{U_c, b_c=0\}_{c=1}^{C}}(\cdot, C, F, S, P) : \mathbb{R}^{C'\times H'\times W'} \to \mathbb{R}^{C\times H\times W}$$

where $U_c \in \mathbb{R}^{C'\times F\times F}$ for $c \in \{1 \dots C\}$ is given by $[U_c]_{o,i,j} = [W_o]_{c,i,j}$. If $V \in \mathbb{R}^{CHW\times C'H'W'}$ is the unique matrix associated with the convolution (Fact 1.2), the following relationship holds:

$$\textbf{TransConv}_\theta(x, C', F, S, P) = (V^\top \text{vec}(x)).\text{view}(C', H', W') + b.\text{view}(C', 1, 1) \tag{3}$$

Reason. We first check that the convolution matches the shape. The new height is, by Lemma 1.1 and 1.3,

$$\left\lfloor \frac{H' + 2P - F}{S} \right\rfloor + 1 = \left\lfloor \frac{(H-1)S + F - 2P + 2P - F}{S} \right\rfloor + 1 = H$$

The width is similar. What is mysterious is that the *transposed* matrix corresponds to an *enlarged* convolution. We illustrate the phenomenon with single-channel padding-0 examples. A transposed convolution on a 2×2 input $x = [[x_1, x_2]\ [x_3, x_4]]$ using a 2×2 filter $W_1 = [[w_1, w_2]; [w_3, w_4]]$ (no bias) with stride 1 computes

$$\underset{(2\times 2 \text{ flipped kernel})}{\begin{bmatrix} w_4 & w_3 \\ w_2 & w_1 \end{bmatrix}} \overset{\substack{\text{stride-1} \\ \text{padding-0}}}{\rightarrow} \underset{(4\times 4 \text{ padded input})}{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & x_1 & x_2 & 0 \\ 0 & x_3 & x_4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}} \implies \underset{(3\times 3 \text{ output})}{\begin{bmatrix} w_1x_1 & w_2x_1 + w_1x_2 & w_2x_2 \\ w_3x_1 + w_1x_3 & w_4x_1 + w_3x_2 + w_2x_3 + w_1x_4 & w_4x_2 + w_2x_3 \\ w_3x_3 & w_4x_3 + w_3x_4 & w_4x_4 \end{bmatrix}} \tag{4}$$

Consider a convolution on a 3×3 input using the filter $W_1 \in \mathbb{R}^{2\times 2}$ with stride 1. The matrix $V \in \mathbb{R}^{4\times 9}$ associated with this convolution is given in (2). Multiplying by the *transpose* of the matrix gives

$$V^\top \text{vec}(x) = \begin{bmatrix} w_1 & 0 & 0 & 0 \\ w_2 & w_1 & 0 & 0 \\ 0 & w_2 & 0 & 0 \\ w_3 & 0 & w_1 & 0 \\ w_4 & w_3 & w_2 & w_1 \\ 0 & w_4 & 0 & w_2 \\ 0 & 0 & w_3 & 0 \\ 0 & 0 & w_4 & w_3 \\ 0 & 0 & 0 & w_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} w_1x_1 \\ w_2x_1 + w_1x_2 \\ w_2x_2 \\ w_3x_1 + w_1x_3 \\ w_4x_1 + w_3x_2 + w_2x_3 + w_1x_4 \\ w_4x_2 + w_2x_3 \\ w_3x_3 \\ w_4x_3 + w_3x_4 \\ w_4x_4 \end{bmatrix}$$

which upon reshaping equals the output in (4). This phenomenon generalizes to stride $S > 1$. A transposed convolution using a 1×1 filter $w \in \mathbb{R}$ with stride 2 computes

$$\underset{(1\times 1 \text{ flipped kernel})}{\begin{bmatrix} w \end{bmatrix}} \overset{\substack{\text{stride-1} \\ \text{padding-0}}}{\rightarrow} \underset{(3\times 3 \text{ padded input})}{\begin{bmatrix} x_1 & 0 & x_2 \\ 0 & 0 & 0 \\ x_3 & 0 & x_4 \end{bmatrix}} \implies \underset{(3\times 3 \text{ output})}{\begin{bmatrix} wx_1 & 0 & wx_3 \\ 0 & 0 & 0 \\ wx_3 & 0 & wx_4 \end{bmatrix}} \tag{5}$$

Consider a convolution on a 3×3 input using the filter $w \in \mathbb{R}$ with stride 2. The output shape is 2×2. The associated matrix $V \in \mathbb{R}^{4\times 9}$ and the transposed product is

$$V = \begin{bmatrix} w & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & w & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & w & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w \end{bmatrix} \qquad V^\top \text{vec}(x) = \begin{bmatrix} w & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & w & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & w & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & w \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} wx_1 \\ 0 \\ wx_2 \\ 0 \\ 0 \\ 0 \\ wx_3 \\ 0 \\ wx_4 \end{bmatrix}$$

which upon reshaping equals the output in (5). □

1.5 Multi-Head Self-Attention

We can apply the same multi-head self-attention in transformers (Appendix B.3) by flattening an image $x \in \mathbb{R}^{C\times H\times W}$ into a sequence of HW "word embeddings" with dimension C.

Attn_θ
Input: image $x \in \mathbb{R}^{C\times H\times W}$, number of heads K, head dimension d
Parameters: parameters of the submodules
Output: pixel-level attended $y \in \mathbb{R}^{C\times H\times W}$

$$\begin{aligned} x_{\text{txt}} &\leftarrow x.\text{view}(C, HW)^\top \in \mathbb{R}^{HW\times C} \\ y_{\text{txt}} &\leftarrow \textbf{AttnOriginal}_\theta(x_{\text{txt}}, K, d) \\ y &\leftarrow (y_{\text{txt}}^\top).\text{view}(C, H, W) \in \mathbb{R}^{C\times H\times W} \end{aligned}$$

2 Front-End Modules

Some kind of normalization layer is almost always necessary (Appendix A).[1] Some kind of "residual connection" is also necessary when we stack many convolution layers together. It broadly refers to the operation $f_{\text{res}}(x) = f(x) + x$ where f is a nonlinear transformation. If the shape of x is different from the shape of $f(x)$, a shape-correcting convolution is applied before addition.

2.1 ResNet-50

ResNet-50 computes a vector representation of an image by applying a sequence of channel-increasing and resolution-decreasing residual stacks. Each residual stack is a sequence of B residual blocks. Each residual block goes through a channel bottleneck. See Appendix B.1 for the submodules.

ResNet50$_\theta$
Input: image $x \in \mathbb{R}^{3\times 32\times 32}$
Parameters: parameters of the submodules
Output: vector representation of the image $y \in \mathbb{R}^{2048}$

$$
\begin{aligned}
z &\leftarrow \text{ReLU}(\text{BatchNorm}_\theta(\textbf{Conv}_\theta(x, C' = 64, F = 3, S = 1, P = 1))) && (64 \times 32 \times 32)\\
z_1 &\leftarrow \textbf{ResStack}_\theta(z, B = 3, C' = 64, S' = 1) && (256 \times 32 \times 32)\\
z_2 &\leftarrow \textbf{ResStack}_\theta(z_1, B = 4, C' = 128, S' = 2) && (512 \times 16 \times 16)\\
z_3 &\leftarrow \textbf{ResStack}_\theta(z_2, B = 6, C' = 256, S' = 2) && (1024 \times 8 \times 8)\\
z_4 &\leftarrow \textbf{ResStack}_\theta(z_3, B = 3, C' = 512, S' = 2) && (2048 \times 4 \times 4)\\
y &\leftarrow \text{AvgPool}(z_4) && (2048 \times 1 \times 1)
\end{aligned}
$$

2.2 U-Net

The U-Net architecture described here is based on labml.ai.[2] It consists of a descent and an ascent phase. During descent, the image is repeatedly transformed into a higher-channel, lower-resolution version by convolution. The convolutions are interleaved with a stack of residual blocks. During ascent, the image is repeatedly transformed into a lower-channel, higher-resolution version by transposed convolution. The transposed convolutions are again interleaved with a stack of residual blocks. Importantly, during ascent, each residual block concatenates the input with the corresponding intermediate image from descent, forming a "horizontal" residual connection. During both descent and ascent, the residual block is additionally followed by self-attention for lower-resolution images. A context embedding is incorporated additively in every residual block through a linear transformation. See Appendix B.2 for the submodules.

UNet$_\theta$
Input: image $x \in \mathbb{R}^{3\times 32\times 32}$, context embedding $u \in \mathbb{R}^d$
Parameters: parameters of the submodules
Output: same-shape transformation $y \in \mathbb{R}^{3\times 32\times 32}$

$$
\begin{aligned}
(z_i)_{i=1}^{12} &\leftarrow \textbf{Descend}_\theta(x, u) && \text{(intermediate images)}\\
y_1 &\leftarrow \textbf{Ascend}_\theta((z_i)_{i=1}^{12}, u, s) && (64 \times 32 \times 32)\\
y &\leftarrow \textbf{Conv}_\theta(\text{swish}(\text{GroupNorm}_\theta(y_1)), C' = 3, F = 3, S = 1, P = 1) && (3 \times 32 \times 32)
\end{aligned}
$$

2.3 Vision Transformer

A vision transformer (aka. ViT) (Dosovitskiy *et al.*, 2020) just applies the transformer encoder (Appendix B.3) to an image $x \in \mathbb{R}^{C\times H\times W}$ after representing it as a sequence of D-dimensional embeddings. Each embedding is a linear transformation of a flattened $C \times p \times p$ patch of the image, plus a position embedding. The output of ViT is a sequence of vectors, which can be used for classification (e.g., by using the first vector corresponding to a special classification token) or as a submodule of an encoder-decoder model (e.g., image-encoder-text-decoder). Even though a transformer does not have any vision-specific inductive bias like translation invariance in convolution, it is shown to be more performant and robust than a convolutional model if sufficiently large and trained on sufficient data (Radford *et al.*, 2021).

[1] Batch normalization does not make sense for a single input; assume that a real implementation is batched.
[2] It uses the swish function $\text{swish}(x) = x\sigma(x)$ for nonlinearity, which is essentially GeLU.

ViT$_\theta$
Input: image $x \in \mathbb{R}^{C \times H \times W}$, patch size p that divides H and W, embedding dimension D, number of layers L, number of heads K, head dimension d, hidden dimension $H \gg D$
Additional variables: new height $H' = H/p$, new width $W' = W/p$
Parameters: parameters of the submodules; linear layer $U \in \mathbb{R}^{Cp^2 \times D}$, $b \in \mathbb{R}^D$
Output: sequence transformation of the image $y \in \mathbb{R}^{H'W' \times D}$

$$x_{\text{chopped}} \leftarrow x.\text{reshape}(C, H', p, W', p).\text{permute}(2, 4, 1, 3, 5) \qquad (H' \times W' \times C \times p \times p)$$
$$x_{\text{txt}} \leftarrow x_{\text{chopped}}.\text{flatten}([\text{1-2}, \text{2-3}]) \qquad (H'W' \times Cp^2)$$
$$z_{\text{txt}} \leftarrow x_{\text{txt}} U + b \qquad (H'W' \times D)$$
$$\bar{z}_{\text{txt}} \leftarrow \text{LayerNorm}_\theta(z_{\text{txt}} + \pi) \qquad (\pi \in \mathbb{R}^{H'W' \times D}\text{: sinusoidal position embeddings})$$
$$y \leftarrow \textbf{TransformerEncoder}_\theta(\bar{z}_{\text{txt}}, L, K, d, H)$$

Using a fixed patch size p yields different sequence lengths for different image sizes. A standard practice is to scale the input image so that a desired sequence length is achieved. For example, Lee *et al.* (2022) scale the image up or down (while preserving the ratio) so that the maximal number of $p \times p$ patches that fit within the given squence length can be extracted.

References

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., *et al.* (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Dumoulin, V. and Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

Lee, K., Joshi, M., Turc, I., Hu, H., Liu, F., Eisenschlos, J., Khandelwal, U., Shaw, P., Chang, M.-W., and Toutanova, K. (2022). Pix2struct: Screenshot parsing as pretraining for visual language understanding. *arXiv preprint arXiv:2210.03347*.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., *et al.* (2021). Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, **30**.

Wu, Y. and He, K. (2018). Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19.

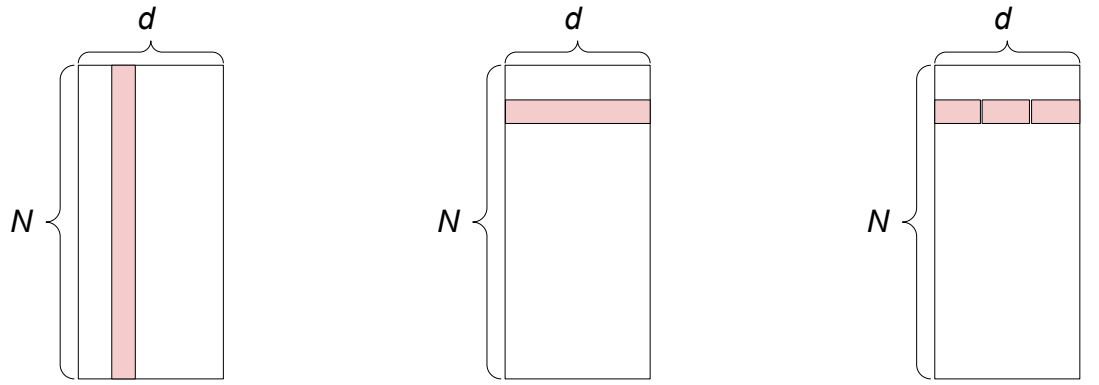
A Normalization

Given a tensor X representing a batch of inputs to some layer (which may themselves be the output of some previous layer) and groups of its indices $\mathcal{S}$, a **normalization layer** computes $\bar{X}$ where for each group $S \in \mathcal{S}$, the corresponding values of $\bar{X}$ have zero mean and unit variance (however, *not* independent of each other):

$$\sum_{i \in S} \bar{X}_i = 0 \qquad \sum_{i \in S} \left(\bar{X}_i - \frac{1}{|S|} \sum_{j \in S} \bar{X}_j \right)^2 = 1$$

This can be achieved by differentiable operations (elementwise subtraction and division). Intuitively, using $\bar{X}$ instead of X makes the layer's weight update more stable. In the simple case where $X = (x_1 \dots x_N) \in \mathbb{R}^{N \times d}$ (i.e., each input is "length 1" or "shape (1, 1)") and the layer is single-neuron and fully connected with parameter $w \in \mathbb{R}^d$, the gradient step on w is $w' \leftarrow w - \eta X^\top u$ for some $u \in \mathbb{R}^N$. The resulting w' may drastically change in size and direction, which may be difficult to recover from in later batches. If we first normalize each row ("layer normalization"), the resulting w' satisfies $||w'||_1 = ||w||_1$. This still allows the direction to change significantly, but it will be easy to recover from. On the other hand, if we normalize each column ("batch normalization"), then updates across batches will be similar ($\bar{X}^\top u \approx (\bar{X}')^\top u'$) and each weight w_i can expect similar updates from different batches.

The fundamental assumption that a normalization layer makes about the input tensor is that *each group makes "similar contributions"* (Ba *et al.*, 2016). Given S, the values $X_i \in \mathbb{R}$ may differ wildly for $i \in S$, but it does not matter since we will exercise our assumption and force them to have a similar magnitude (yet preserving their relative behavior since we do not make them independent). When the assumption does not hold, forced normalization will interfere with the model's ability to extract patterns and may not be as effective. A visual summary for some popular normalization methods for $X \in \mathbb{R}^{N \times d}$ (i.e., each row is an input vector, each column is values of a particular feature over inputs) is given below:



Batch normalization | Layer normalization | Group normalization

Batch normalization assumes that all inputs make similar contributions. This is a reasonable assumption if the batch size is large, but not so much otherwise. Layer normalization removes the batch size sensitivity by assuming that all features make similar contribution for each input. This is a reasonable assumption for fully connected layers, but not so much for convolutional layers where the values at the "edge of an image" are clearly less important than the values at the center. Group normalization addresses this issue by further subgrouping features.

A.1 Normalization Template

As formulated by Wu and He (2018), all normalization methods share the following template. We assume a batch of images which can be seen as the most general case subsuming single-vector and sequence inputs as special cases. Let $[n] = \{1 \dots n\}$ denote the set of indices from 1 to n. Let N denote the number of examples in a batch, C the number of channels (equivalent to features), H the height, and W the width. Let $\mathcal{I} = [N] \times [C] \times [H] \times [W]$ denote the set of all $NCHW$ index quadruples.

Learnable parameters: $\gamma, \beta \in \mathbb{R}^C$
Input: batch $X \in \mathbb{R}^{N \times C \times H \times W}$, mapping $\textbf{select}(k, c, i, j) \subset \mathcal{I}$, smoothing parameter $\epsilon = 0.00001$
Output: normalized batch $Y \in \mathbb{R}^{N \times C \times H \times W}$

1. For each $(k, c, i, j) \in \mathcal{I}$, estimate the mean and standard deviation over the selected elements:

$$\mu(k, c, i, j) = \frac{1}{|\textbf{select}(k, c, i, j)|} \sum_{(k', c', i', j') \in \textbf{select}(k, c, i, j)} X_{k', c', i', j'}$$

$$\sigma(k, c, i, j) = \sqrt{\frac{1}{|\textbf{select}(k, c, i, j)|} \sum_{(k', c', i', j') \in \textbf{select}(k, c, i, j)} (X_{k', c', i', j'} - \mu(k, c, i, j))^2 + \epsilon}$$

2. Make each "region" in the tensor have zero mean and unit variance:

$$\bar{X} = \frac{X - \mu}{\sigma}$$

3. For each feature $c \in [C]$, apply the same affine scaling to all examples k and locations (i, j):

$$Y_{k,c,i,j} = \gamma_c \bar{X}_{k,c,i,j} + \beta_c$$

By varying the choice of **select**, we can choose which regions will be zero-mean and unit-variance.

$$\textbf{select}(k, c, i, j) = \begin{cases} \{(k', c', i', j') \in \mathcal{I} : \ c' = c\} & \text{(batch norm)} \\ \{(k', c', i', j') \in \mathcal{I} : \ k' = k\} & \text{(layer norm)} \\ \{(k', c', i', j') \in \mathcal{I} : \ k' = k, \ c' = c\} & \text{(instance norm)} \\ \left\{(k', c', i', j') \in \mathcal{I} : \ k' = k, \ c' \in \mathcal{C}_{g(c)}\right\} & \text{(group norm)} \end{cases}$$

where $g(c) \in [G]$ is the group that c belongs to. Some method-specific details:

- Batch normalization estimates a moving average of the mean and variance during training and use them to whiten batches at test time. Thus no sample estimates are computed at test time.
- Group norm assumes a sequential partition of features into G groups where G is a hyperparameter (we will assume $G = 32$ in this note) in which case this condition can be checked as $\left\lfloor \frac{c'}{C/G} \right\rfloor = \left\lfloor \frac{c}{C/G} \right\rfloor$.

B Submodules

B.1 Submodules for ResNet-50

$\textbf{ResStack}_\theta$
Input: image $x \in \mathbb{R}^{C \times H \times W}$, number of residual blocks B, number of intermediate output channels C', one-time stride $S' \geq 1$
Parameters: parameters of the submodules
Output: $y \in \mathbb{R}^{4C' \times \frac{H}{S'} \times \frac{W}{S'}}$

1. Apply an initial residual block (size-reducing if $S' > 1$):

$$z_1 \leftarrow \textbf{ResBlock}_\theta^{\text{Res}}(x, C', S') \qquad (C \times H \times W) \to \left(4C' \times \frac{H}{S'} \times \frac{W}{S'}\right)$$

2. For $i = 2 \ldots B$, apply a channel-bottleneck residual block:

$$z_i \leftarrow \textbf{ResBlock}_\theta^{\text{Res}}(z_{i-1}, C', 1) \qquad \left(4C' \times \frac{H}{S'} \times \frac{W}{S'}\right) \to \left(4C' \times \frac{H}{S'} \times \frac{W}{S'}\right)$$

3. Return $y \leftarrow z_B$.

$\mathbf{ResBlock}_\theta^{\text{Res}}$
Input: image $x \in \mathbb{R}^{C \times H \times W}$, number of intermediate output channels C', one-time stride $S' \geq 1$
Parameters: parameters of the submodules
Output: $y \in \mathbb{R}^{4C' \times \frac{H}{S'} \times \frac{W}{S'}}$

$$z_1 \leftarrow \text{BatchNorm}_\theta(\mathbf{Conv}_\theta(x, C', F=1, S=1, P=0)) \qquad (C' \times H \times W)$$

$$z_2 \leftarrow \text{BatchNorm}_\theta(\mathbf{Conv}_\theta(\text{ReLU}(z_1), C', F=3, S=S', P=1)) \qquad \left(C' \times \frac{H}{S'} \times \frac{W}{S'}\right)$$

$$z_3 \leftarrow \text{BatchNorm}_\theta(\mathbf{Conv}_\theta(\text{ReLU}(z_2), 4C', F=1, S=1, P=0))) \qquad \left(4C' \times \frac{H}{S'} \times \frac{W}{S'}\right)$$

$$x' \leftarrow \begin{cases} x & \text{if } (C,H,W) = \left(4C', \frac{H}{S'}, \frac{W}{S'}\right) \\ \text{BatchNorm}_\theta(\mathbf{Conv}_\theta(x, 4C', F=1, S=S', P=0)) & \text{otherwise} \end{cases}$$

$$y \leftarrow \text{ReLU}(z_3 + x')$$

B.2 Submodules for U-Net

$\mathbf{Descend}_\theta$
Input: image $x \in \mathbb{R}^{3 \times 32 \times 32}$, context embedding $u \in \mathbb{R}^{256}$
Parameters: parameters of the submodules
Output: increasingly higher-channel lower-resolution images $(z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9, z_{10}, z_{11}, z_{12})$

$$z_1 \leftarrow \mathbf{Conv}_\theta(x, C'=64, F=3, S=1, P=1) \qquad (64 \times 32 \times 32)$$
$$\text{For } i=2,3\text{:}\ z_i \leftarrow \mathbf{ResBlock}_\theta^{\text{U}}(z_{i-1}, u, C'=64) \qquad (64 \times 32 \times 32)$$
$$z_4 \leftarrow \mathbf{Conv}_\theta(z_3, C'=64, F=3, S=2, P=1) \qquad (64 \times 16 \times 16)$$
$$\text{For } i=5,6\text{:}\ z_i \leftarrow \mathbf{ResBlock}_\theta^{\text{U}}(z_{i-1}, u, C'=128) \qquad (128 \times 16 \times 16)$$
$$z_7 \leftarrow \mathbf{Conv}_\theta(z_6, C'=128, F=3, S=2, P=1) \qquad (128 \times 8 \times 8)$$
$$\text{For } i=8,9\text{:}\ z_i \leftarrow \mathbf{Attn}_\theta(\mathbf{ResBlock}_\theta^{\text{U}}(z_{i-1}, u, C'=256)) \qquad (256 \times 8 \times 8)$$
$$z_{10} \leftarrow \mathbf{Conv}_\theta(z_9, C'=256, F=3, S=2, P=1) \qquad (256 \times 4 \times 4)$$
$$\text{For } i=11,12\text{:}\ z_i \leftarrow \mathbf{Attn}_\theta(\mathbf{ResBlock}_\theta^{\text{U}}(z_{i-1}, u, C'=1024)) \qquad (1024 \times 4 \times 4)$$

$\mathbf{Ascend}_\theta$
Input: intermediate images $(z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9, z_{10}, z_{11}, z_{12})$ from $\mathbf{Descend}_\theta$, context embedding $u \in \mathbb{R}^{256}$
Parameters: parameters of the submodules
Output: original-shape image $y_1 \in \mathbb{R}^{3 \times 32 \times 32}$
Notation: channel-wise concatenation $z \oplus z' \in \mathbb{R}^{2C \times H \times W}$

$$y_{13} \leftarrow \mathbf{ResBlock}_\theta^{\text{U}}(\mathbf{Attn}_\theta(\mathbf{ResBlock}_\theta^{\text{U}}(z_{12}, u, 1024)), u, 1024) \qquad (1024 \times 4 \times 4)$$
$$\text{For } i=12,11\text{:}\ y_i \leftarrow \mathbf{Attn}_\theta(\mathbf{ResBlock}_\theta^{\text{U}}(y_{i+1} \oplus z_i, u, 1024)) \qquad (1024 \times 4 \times 4)$$
$$\bar{y}_{10} \leftarrow \mathbf{Attn}_\theta(\mathbf{ResBlock}_\theta^{\text{U}}(y_{11} \oplus z_{10}, u, 256)) \qquad (256 \times 4 \times 4)$$
$$y_{10} \leftarrow \mathbf{TransConv}_\theta(\bar{y}_{10}, 256, F=4, S=2, P=1) \qquad (256 \times 8 \times 8)$$
$$\text{For } i=9,8\text{:}\ y_i \leftarrow \mathbf{Attn}_\theta(\mathbf{ResBlock}_\theta^{\text{U}}(y_{i+1} \oplus z_i, u, 256)) \qquad (256 \times 8 \times 8)$$
$$\bar{y}_7 \leftarrow \mathbf{Attn}_\theta(\mathbf{ResBlock}_\theta^{\text{U}}(y_8 \oplus z_7, u, 128)) \qquad (128 \times 8 \times 8)$$
$$y_7 \leftarrow \mathbf{TransConv}_\theta(\bar{y}_7, 128, F=4, S=2, P=1) \qquad (128 \times 16 \times 16)$$
$$\text{For } i=6,5\text{:}\ y_i \leftarrow \mathbf{ResBlock}_\theta^{\text{U}}(y_{i+1} \oplus z_i, u, 128) \qquad (128 \times 16 \times 16)$$
$$\bar{y}_4 \leftarrow \mathbf{ResBlock}_\theta^{\text{U}}(y_5 \oplus z_4, u, 64) \qquad (64 \times 16 \times 16)$$
$$y_4 \leftarrow \mathbf{TransConv}_\theta(\bar{y}_4, 64, F=4, S=2, P=1) \qquad (64 \times 32 \times 32)$$
$$\text{For } i=3,2,1\text{:}\ y_i \leftarrow \mathbf{ResBlock}_\theta^{\text{U}}(y_{i+1} \oplus z_i, u, 64) \qquad (64 \times 32 \times 32)$$

$\mathbf{ResBlock}^{\mathrm{U}}_{\theta}$
Input: image $x \in \mathbb{R}^{C \times H \times W}$, context embedding $u \in \mathbb{R}^{d}$, number of output channels C'
Parameters: parameters of the submodules; linear layer $U \in \mathbb{R}^{C' \times d}$, $b \in \mathbb{R}^{C'}$
Output: $y \in \mathbb{R}^{C' \times H \times W}$

$$z_1 \leftarrow \mathbf{Conv}_{\theta}(\mathrm{swish}(\mathrm{GroupNorm}_{\theta}(x)), C', F = 3, S = 1, P = 1) \qquad (C', H, W)$$
$$z_2 \leftarrow z_1 + (U\mathrm{swish}(u) + b).\mathrm{view}(C', 1, 1)$$
$$z_3 \leftarrow \mathbf{Conv}_{\theta}(\mathrm{Drop}_{0.1}(\mathrm{swish}(\mathrm{GroupNorm}_{\theta}(z_2))), C', F = 3, S = 1, P = 1)$$
$$x' \leftarrow \begin{cases} x & \text{if } C' = C \\ \mathbf{Conv}_{\theta}(x, C', F = 1, S = 1, P = 0) & \text{otherwise} \end{cases}$$
$$y \leftarrow z_3 + x'$$

B.3 Submodules from Transformer (Vaswani *et al.*, 2017)

$\mathbf{AttnOriginal}_{\theta}$
Input: embedding sequence $x \in \mathbb{R}^{T \times D}$, number of heads K, head dimension d
Parameters: linear layer $U^{z,h} \in \mathbb{R}^{D \times d}$, $b^{z,h} \in \mathbb{R}^{d}$ for all $z \in \{\mathrm{query}, \mathrm{key}, \mathrm{value}\}$ and $h \in \{1 \dots K\}$; linear layer $V \in \mathbb{R}^{Kd \times D}$, $b \in \mathbb{R}^{D}$
Output: attended $y \in \mathbb{R}^{T \times D}$

1. For each head $h = 1 \dots K$ compute head-specific query/key/value embeddings of each pixel by

$$q^{(h)} \leftarrow xU^{\mathrm{query},h} + b^{\mathrm{query}} \qquad (T \times d)$$
$$k^{(h)} \leftarrow xU^{\mathrm{key},h} + b^{\mathrm{key}} \qquad (T \times d)$$
$$v^{(h)} \leftarrow xU^{\mathrm{value},h} + b^{\mathrm{value}} \qquad (T \times d)$$

2. For each head $h = 1 \dots K$, compute self-attention over the sequence by scaled dot product, row-wise softmax, attention dropout, and per-position linear combination of value embeddings:

$$a^{(h)} \leftarrow \underbrace{\mathrm{Drop}_{0.1}\left(\mathrm{softmax}\left(\frac{q^{(h)}(k^{(h)})^{\top}}{\sqrt{d}}\right)\right)}_{T \times T} \underbrace{v^{(h)}}_{T \times d} \qquad (T \times d)$$

3. Dombine the K heads and recover the original dimension D through linear transformation:

$$y \leftarrow \underbrace{[a^{(1)} \dots a^{(K)}]}_{T \times Kd} \underbrace{V}_{Kd \times D} + b \qquad (T \times D)$$

$\mathbf{TransformerEncoder}_{\theta}$
Input: sequence $x \in \mathbb{R}^{T \times D}$, number of layers L, number of heads K, head dimension d, hidden dimension $H \gg D$
Parameters: parameters of the submodules
Output: output sequence $y_L \in \mathbb{R}^{T \times D}$

1. $y_0 \leftarrow x$
2. For $l = 1 \dots L$:

$$y_l \leftarrow \mathbf{TransformerEncoderBlock}_{\theta}(y_{l-1}, K, d, H)$$

$\mathbf{TransformerEncoderBlock}_{\theta}$
Input: sequence $x \in \mathbb{R}^{T \times D}$, number of heads K, head dimension d, hidden dimension $H \gg D$
Parameters: parameters of the submodules; $(U, a), (V, b)$ where $U \in \mathbb{R}^{D \times H}$, $a \in \mathbb{R}^{H}$, $V \in \mathbb{R}^{H \times D}$, $b \in \mathbb{R}^{D}$
Output: output sequence $y \in \mathbb{R}^{T \times D}$

$$z \leftarrow x + \mathbf{AttnOriginal}_{\theta}(\mathrm{LayerNorm}_{\theta}(x), K, d)$$
$$u \leftarrow \mathrm{Drop}_{0.1}(\mathrm{LayerNorm}_{\theta}(z))U + a \qquad (T \times H)$$
$$y \leftarrow x + \mathrm{GeLU}(\mathrm{Drop}_{0.1}(u))V + b \qquad (T \times D)$$