RGB no more: Minimally-decoded JPEG Vision Transformers Supplementary Material

Jeongsoo Park Justin Johnson
University of Michigan
{jespark, justincj}@umich.edu

A. Scaling property of a 8×8 DCT

Consider an 8×8 2D-DCT defined as the following where $\alpha_i = 1/\sqrt{2}$ if $i = 0$, else 1, $u, v, m, n \in [0..7]$.

$$X_{u,v} = \frac{\alpha_u \alpha_v}{4} \sum_{m,n} x_{m,n} \cos\left[\frac{\pi(2m+1)u}{16}\right] \cos\left[\frac{\pi(2n+1)v}{16}\right] \quad (1)$$

We then calculate how much the DCT scales up the min/max values considering the following two cases.

- If $u, v = 0$

If $u, v = 0$ then the resulting coefficient $X_{0,0}$ is simply:

$$X_{0,0} = \frac{1}{8} \sum_{m=0}^{7} \sum_{n=0}^{7} x_{m,n} \cdot \cos(0) \cos(0) \quad (2)$$

$$= \frac{1}{8} \sum_{m=0}^{7} \sum_{n=0}^{7} x_{m,n} \quad (\because \cos(0) = 1) \quad (3)$$

Which is minimized/maximized when $x_{m,n}$ is min/max:

$$\min(X_{0,0}) = \frac{1}{8} \min(x_{m,n}) \cdot 8 \cdot 8 \quad (4)$$

$$= \min(x_{m,n}) \cdot 8 \quad (5)$$

$$\max(X_{0,0}) = \frac{1}{8} \max(x_{m,n}) \cdot 8 \cdot 8 \quad (6)$$

$$= \max(x_{m,n}) \cdot 8 \quad (7)$$

So applying 8×8 DCT to an input with min/max value of [-128, 127] scales the output min/max value to:

$$\min(X_{0,0}) = -128 \cdot 8 = -1024 \quad (8)$$

$$\max(X_{0,0}) = 127 \cdot 8 = 1016 \quad (9)$$

- If $u \neq 0$ or $v \neq 0$

In this case, the amplitude of DCT coefficient $X_{u,v}$ will be maximized if the input data sequence resonates with (i.e. match the signs of) the cosine signal of the corresponding DCT bases. We will show that this value is less than or equal to the magnitude when $u, v = 0$. We first consider the 1D case and then use it to calculate the 2D case. The 1D DCT is as follows.

$$X_u = \frac{\alpha_u}{2} \sum_{m=0}^{7} x_m \times \cos\left[\frac{\pi(2m+1)u}{16}\right] \quad (10)$$

Where the DCT bases are:

$$\frac{\alpha_u}{2} \cos\left[\frac{\pi(2m+1)u}{16}\right], \quad m, u \in [0..7] \quad (11)$$

This 1D DCT will be maximized when the signs of x_m match the signs of the DCT bases. Likewise, it will be minimized when the signs are exactly the opposite. Therefore, we compare the absolute sum of the DCT bases and show that it is less than or equal to the sum when $u = 0$. This absolute sum of DCT bases can be interpreted as the *scale-up factor* as it shows how much the input 1's with matching signs are scaled up. The following values are rounded to the third decimal place.

- $u = 0$: $X_0 = \frac{\alpha_0}{2} \sum_{m=0}^{7} \texttt{abs}(1) = 2\sqrt{2} = \mathbf{2.828}$
- $u = 1$: $X_1 = \frac{\alpha_1}{2} \sum_{m=0}^{7} \texttt{abs}(\cos\left[\frac{\pi(2m+1)}{16}\right]) = 2.563$
- $u = 2$: $X_2 = \frac{\alpha_2}{2} \sum_{m=0}^{7} \texttt{abs}(\cos\left[\frac{2\pi(2m+1)}{16}\right]) = 2.613$
- $u = 3$: $X_3 = \frac{\alpha_3}{2} \sum_{m=0}^{7} \texttt{abs}(\cos\left[\frac{3\pi(2m+1)}{16}\right]) = 2.563$
- $u = 4$: $X_3 = \frac{\alpha_4}{2} \sum_{m=0}^{7} \texttt{abs}(\cos\left[\frac{4\pi(2m+1)}{16}\right]) = 2.828$
- $u = 5$: $X_3 = \frac{\alpha_5}{2} \sum_{m=0}^{7} \texttt{abs}(\cos\left[\frac{5\pi(2m+1)}{16}\right]) = 2.563$
- $u = 6$: $X_3 = \frac{\alpha_6}{2} \sum_{m=0}^{7} \texttt{abs}(\cos\left[\frac{6\pi(2m+1)}{16}\right]) = 2.613$
- $u = 7$: $X_3 = \frac{\alpha_7}{2} \sum_{m=0}^{7} \texttt{abs}(\cos\left[\frac{7\pi(2m+1)}{16}\right]) = 2.563$

We can see that for all u, the absolute sums of DCT bases are less than or equal to the sum when $u = 0$. 2D DCT is simply a DCT on each axis (rows and columns), so the

2D scale-up factors will be a pairwise product for any pairs of u with replacement. This will still not exceed the value when we choose $u = 0$ twice. Therefore, we can conclude that the minimum and maximum values calculated in the $u, v = 0$ case will hold for all 8×8 DCT coefficients. Thus, 8×8 DCT will scale up the min/max value by 8.

B. Compute cost to decode JPEG

JPEG is decoded through the following steps.

(a) Decode Huffman codes to RLE symbols
(b) Decode RLE symbols to quantized DCT coefficients
(c) De-quantize the DCT coefficients
(d) Apply inverse-DCT to the DCT coefficients
(e) Shift value from [-128, 127] to [0, 255]
(f) Upsample Cb, Cr by $2\times$ for each dimension
(g) Convert YCbCr to RGB

We count the number of operations (OPs) for each step:

(a) Read N_s Huffman codes and recover N_s RLE symbols = $N_s + N_s = 2N_s$ OPs
(b) Read N_s RLE symbols and recover 8×8 quantized DCT coefficients = $N_s + 64$ OPs
(c) Multiply 8×8 quantization table element-wise with 8×8 DCT coefficients = 64 OPs
(d) The 8×8 inverse DCT is given as:

$$x_{m,n} = \frac{1}{4}\gamma_m\gamma_n \sum_{u,v} X_{u,v} \cos\left[\frac{(2m+1)u\pi}{16}\right] \cos\left[\frac{(2n+1)v\pi}{16}\right] \tag{12}$$

Where

$$\gamma_i = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } i = 0 \\ 1 & \text{otherwise} \end{cases}$$

- If $m, n \in [1..7]$
  We need 2 cos +10mul + 3add + 3div per step as $\gamma_i = 1$. Number of OPs: $7 \times 7 \times 18 = 882$
- If $m = 0, n \in [1..7]$ or $m \in [1..7], n = 0$
  We need 2 cos +10mul + 3add + 4div + 1sqrt per step. OPs: $7 \times 20 \times 2 = 280$
- If $m = 0, n = 0$
  2 cos +10mul + 3add + 5div + 2sqrt = 22 OPs

Total OPs per 8×8 block: $882 + 280 + 22 = 1184$

(e) Add 128 to every elements of $x_{m,n}$ = 64 OPs
(f) Upsample 8×8 Cb and Cr block to 16×16: $256 \times 2 = 512$ OPs per 6 blocks. This is because 4 Y blocks are paired with 1 Cb and Cr block. Per-block cost: $512/6 = 85.3$ OPs.
(g) YCbCr is converted to RGB using [1]:

$$R = Y + 1.402(C_r - 128)$$
$$G = Y - 0.344136(C_b - 128) - 0.714136(C_r - 128)$$
$$B = Y + 1.772(C_b - 128)$$

For three blocks – Y, Cb, and Cr – the number of OPs is $64 \times (2\text{add} + 6\text{sub} + 4\text{mul}) = 768$. We ignore the cost of rounding and min/max clamping for simplicity. Thus, the per-block cost is $768/3 = 256$ OPs

Recovering DCT coefficients requires going through steps (a)-(c), where the compute cost sums up to $3N_s + 128$ OPs. Full decoding requires $3N_s + 1717.3$ OPs. We can see that most of the decoding cost comes from the inverse-DCT, which costs 1184 OPs to compute. Note that this result is only an estimate and can vary under different settings.

C. Conversion matrix for sub-block conversion

The conversion matrix A can be calculated using the basis transform from $L \times M$ number of $N \times N$ DCT bases to $LN \times MN$ DCT bases. Let $T^{N\times N}$ as a 1-D DCT bases of size $N \times N$ then:

$$T^{N\times N} = \sqrt{\frac{2}{N}} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \cdots & \frac{1}{\sqrt{2}} \\ \cos[\frac{1\pi}{2N}] & \cos[\frac{3\pi}{2N}] & \cdots & \cos[\frac{(2N-1)\pi}{2N}] \\ \vdots & \vdots & \ddots & \vdots \\ \cos[\frac{(N-1)\pi}{2N}] & \cos[\frac{3(N-1)\pi}{2N}] & \cdots & \cos[\frac{(2N-1)(N-1)\pi}{2N}] \end{bmatrix} \tag{13}$$

$T^{N\times N}$ is an orthogonal matrix [2]. Hence,

$$TT^T = I \quad T^T = T^{-1} \tag{14}$$

Define B_{large} as $T^{LN\times LN}$ and B_{small} as a block diagonal matrix of $T^{N\times N}$ with size $LN \times LN$:

$$B_{small} = \begin{bmatrix} T^{N\times N} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & T^{N\times N} \end{bmatrix} \tag{15}$$

Then the conversion matrix $A_{L,N}$ is [3]:

$$B_{large} = A_{L,N} \times B_{small} \tag{16}$$
$$A_{L,N} = B_{large} \times B_{small}^{-1} \tag{17}$$

Where $B_{small}^{-1} = B_{small}^T$ due to Eqs. (14) and (15). Thus,

$$A_{L,N} = B_{large} \times B_{small}^T \tag{18}$$

We can also see that $B_{large}^{-1} = B_{large}^T$. Thus,

$$A_{L,N}^{-1} = (B_{large} \times B_{small}^T)^{-1} \tag{19}$$
$$= (B_{small}^T)^{-1} \times B_{large}^{-1} \tag{20}$$
$$= B_{small} \times B_{large}^T \tag{21}$$
$$= A_{L,N}^T \tag{22}$$

D. Sub-band approximation

Define $x(m, n)$ as the 2D image data, and $X(k, l)$ as the 2D DCT coefficient of $x(m, n)$ where $m, n, k, l \in [0..N - 1]$. Then, define $x_{LL}(m', n')$ as the $2\times$ downsized image of $x(m, n)$. Then x_{LL} is given as:

$$x_{LL}(m', n') = \frac{1}{4}\{x(2m', 2n') + x(2m' + 1, 2n') + x(2m', 2n' + 1) + x(2m' + 1, 2n' + 1)\} \tag{23}$$

where $m', n', k', l' \in [0, .. \frac{N}{2} - 1]$. Similarly, define $\overline{X_{LL}}(k', l')$ as the 2D DCT coefficient of $x_{LL}(m', n')$. Mukherjee and Mitra's work [4] shows that $X(k, l)$ can be represented in terms of $\overline{X_{LL}}(k, l)$:

$$X(k,l) = \begin{cases} 2\cos(\frac{\pi k}{2N})\cos(\frac{\pi l}{2N})\overline{X_{LL}}(k,l) & 0 \le k,l \le \frac{N}{2} - 1 \\ 0 & \text{otherwise} \end{cases} \quad (24)$$

Which can be further simplified assuming that k, l are negligible compared to $2N$: $\frac{\pi k}{2N}, \frac{\pi l}{2N} \approx 0$

$$X(k,l) \approx \begin{cases} 2\overline{X_{LL}}(k,l) & 0 \le k,l \le \frac{N}{2} - 1 \\ 0 & \text{otherwise} \end{cases} \quad (25)$$

We can follow the same process for $L \times M$ downsampling from $LN \times MN$ DCT coefficient to $N \times N$ DCT [4]:

$$X(k,l) \approx \begin{cases} \sqrt{LM}\ \overline{X_{LL}}(k,l) & 0 \le k,l \le N - 1 \\ 0 & \text{otherwise} \end{cases} \quad (26)$$

Thus, Eq. (26) implies the approximate up and downsampling formula as:

- Upsampling:

$$X_{LN\times MN} \approx \begin{bmatrix} \sqrt{LM}X_{N\times N} & \mathbf{0}_{N\times(MN-N)} \\ \mathbf{0}_{(LN-N)\times N} & \mathbf{0}_{(LN-N)\times(MN-N)} \end{bmatrix} \quad (27)$$

- Downsampling:

$$X_{N\times N} \approx \frac{1}{\sqrt{LM}} X_{LN\times MN}[0:N, 0:N] \quad (28)$$

E. Fourier transform's rotational property

The proof of the Fourier transform's rotational property is as follows. Define $g(\mathbf{x})$ as a function of x where $\mathbf{x} \in \mathbb{R}^d$. The Fourier transform of g is:

$$\mathcal{F}[g(\mathbf{x})] = G(\mathbf{X}) = \int g(\mathbf{x}) e^{-j2\pi \mathbf{x}^T \mathbf{X}}\, d\mathbf{x} \quad (29)$$

We can describe the rotated version of $\mathbf{x}$ as $\mathbf{u} = \mathbf{A}\mathbf{x}$ where $\mathbf{A}$ is a rotation matrix in which

$$\mathbf{A}^T = \mathbf{A}^{-1} \quad (30)$$

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{u} = \mathbf{A}^T\mathbf{u} \quad (31)$$

Define the rotated version of g as h where $g(\mathbf{A}\mathbf{x}) = h(\mathbf{x})$. Then, the Fourier transform of $g(\mathbf{A}\mathbf{x})$ becomes:

$$\mathcal{F}[g(\mathbf{A}\mathbf{x})] = \mathcal{F}[h(\mathbf{x})] = \int h(\mathbf{x}) e^{-j2\pi \mathbf{x}^T \mathbf{X}}\, d\mathbf{x} \quad (32)$$

$$= \int g(\mathbf{A}\mathbf{x}) e^{-j2\pi \mathbf{x}^T \mathbf{X}}\, d\mathbf{x} \quad (33)$$

$$= \int g(\mathbf{u}) e^{-j2\pi (\mathbf{A}^T\mathbf{u})^T \mathbf{X}}\, d\mathbf{u} \quad (34)$$

$$(\because d\mathbf{u} = |\det(\mathbf{A})| d\mathbf{x}, \quad |\det(\mathbf{A})| = 1) \quad (35)$$

$$= \int g(\mathbf{u}) e^{-j2\pi \mathbf{u}^T \mathbf{A}\mathbf{X}}\, d\mathbf{u} \quad (36)$$

$$\mathcal{F}[g(\mathbf{A}\mathbf{x})] = \int g(\mathbf{u}) e^{-j2\pi \mathbf{u}^T \mathbf{A}\mathbf{X}}\, d\mathbf{u} = G(\mathbf{A}\mathbf{X}) \quad (37)$$

Thus, the Fourier transform of the rotated $g(\mathbf{x})$ is equal to rotating the Fourier transform $G(\mathbf{X})$.

F. DCT to DFT sub-block conversion

If we define $\omega = \exp(-j2\pi/N)$ then the $N \times N$ 1-D DFT bases matrix $W^{N\times N}$ is given as:

$$W^{N\times N} = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{bmatrix} \quad (38)$$

Setting $D_{N\times M}$ as the DFT coefficient block of size $N \times M$, the conversion formula becomes:

$$D_{LN\times MN} = \hat{A}_{L,N} \begin{bmatrix} X^{0,0}_{N\times N} & \cdots & X^{0,M-1}_{N\times N} \\ \vdots & \ddots & \vdots \\ X^{L-1,0}_{N\times N} & \cdots & X^{L-1,M-1}_{N\times N} \end{bmatrix} \hat{A}^T_{M,N} \quad (39)$$

The corresponding decomposition is then:

$$\begin{bmatrix} X^{0,0}_{N\times N} & \cdots & X^{0,M-1}_{N\times N} \\ \vdots & \ddots & \vdots \\ X^{L-1,0}_{N\times N} & \cdots & X^{L-1,M-1}_{N\times N} \end{bmatrix} = \hat{A}^{-1}_{L,N} D_{LN\times MN} \hat{A}^{-1^T}_{M,N} \quad (40)$$

Where $\hat{A}$ denotes the DCT to DFT conversion matrix. This can be calculated by following the same process from Eq. (18) with replacing B_{large} as W of appropriate size.

G. Resize strategy for DCT

While it is possible to do an arbitrary resize of $\frac{P}{Q} \times \frac{R}{S}$ by first upsampling $P \times R$ times and downsampling by $Q \times S$, it is preferable to avoid it due to the compute cost of an additional resize. Therefore, we utilize a different strategy. During random resized crop, we fuse the cropping and resize together in a way that the crop size is limited to the factors of a resize target. For example, if we are resizing to $28\times28\times8\times8$, then the height and width of the crop window are selected from the set: $\{1, 2, 4, 7, 14, 28, 56, ...\}$. This way, we can reduce computation as upsampling or downsampling is limited to an integer multiple. This strategy has been used throughout our experiments.

H. Comparison with Pillow-SIMD

Pillow-SIMD [5] is a highly optimized version of Pillow [6]. It allows faster resizing using CPU SIMD instructions. The main bottleneck of our pipeline is resizing. If we reduce resizing by pre-resizing the data to 256×256, then our method shows 9.1% and 29.2% faster training and evaluation versus Pillow-SIMD as shown in Tab. 1. We believe our method could be further sped up given analogous optimization efforts.

Method	Img. size	Model	Train Data	Fwd/Bwd	Train	Eval Data	Fwd	Eval
SIMD	512^2	ViT-Ti	1043.3	835.2	724.7	1743.9	2832.5	1630.8
Ours	512^2	ViT-Ti	816.2 (-21.8%)	857.2 (+2.6%)	687.6 (-5.1%)	775.3 (-55.5%)	2847.5 (+0.5%)	752.3 (-53.9%)
SIMD	256^2	ViT-Ti	1176.4	839.6	722.4	2232.0	2832.6	1893.9
Ours	256^2	ViT-Ti	1530.9 (+30.1%)	854.2 (+1.7%)	788.0 **(+9.1%)**	3070.6 (+37.6%)	2842.9 (+0.4%)	2447.1 **(+29.2%)**

Table 1. Comparison with Pillow-SIMD. Our pipeline shows faster training and evaluation if we reduce resizing.

I. Training settings

The hyperparameter settings and augmentation subset we used for training are reported in Tables 2 and 3. RGB models used the recipe given in [7], including the SwinV2 models [8], for a fair comparison.

Model	Learning Rate	Weight Decay	RandAug. Magnitude	Input Size	Epochs
ViT-Ti	1e-3	1e-4	10	224^2	300
ViT-S	1e-3	1e-4	10	224^2	90
SwinV2-T (RGB)	1e-3	1e-4	10	256^2	300
JPEG-Ti	3e-3	1e-4	3	224^2	300
JPEG-S	3e-3	3e-4	3	224^2	90
SwinV2-T (DCT)	1e-3	1e-4	3	256^2	300

Table 2. Hyperparameter settings of the trained models.

Models	Subset
JPEG-Ti	Brightness, Contrast, Color, AutoContrast, AutoSaturation, MidfreqAug, Posterize, SolarizeAdd, Grayscale, ChromaDrop, Translate, Cutout, Rotate90
JPEG-S SwinV2-T (DCT)	Brightness, Contrast, Color, AutoContrast, AutoSaturation, MidfreqAug, Sharpness, Posterize, Grayscale, ChromaDrop, Translate, Cutout, Rotate90
RGB models [7]	Brightness, Contrast, Equalize, Color, AutoContrast, Sharpness, Invert, Posterize, Solarize, SolarizeAdd, Translate, Cutout, Rotate, Shear

Table 3. Augmentation subset of RandAugment for the models.

J. Smaller patch sizes

SwinV2 [8] uses a patch size of 4. 8×8 JPEG DCT blocks can be adapted to support this by decomposing them into sixteen 2×2 blocks using sub-block conversion (Sec 5.2.). Then, we can use any of the embedding strategies discussed in Sec. 4. In general, for a desired patch size p, we need to decompose the DCT blocks to be at most $\frac{p}{2} \times \frac{p}{2}$.

K. Measurement process

Latency measurements to decode and augment follow Algorithms 1 and 2. **Data Loading** throughputs for both train and evaluation is measured using Algorithm 3. **Model Fwd/Bwd** measures the throughput of model forward and backward pass using Algorithm 4. **Model Fwd** measures the throughput of the model forward pass using Algorithm 5. **Train Pipeline** and **Eval Pipeline** throughput is measured using Algorithms 6 and 7.

Algorithm 1 Decoding latency measurement

```
latency ← 0
for i = 0..N do
    start_time ← time()
    data ← decode(Filename)
    end_time ← time()
    latency ← latency + (end_time − start_time)
end for
return latency/N
```

Algorithm 2 Augment latency measurement

```
latency ← 0
for i = 0..N do
    data ← decode(Filename)
    start_time ← time()
    data ← augment(data)
    end_time ← time()
    latency ← latency + (end_time − start_time)
end for
return latency/N
```

Algorithm 3 Data Loading throughput measurement

```
start_time ← time()
for i = 0..N do
    data, label ← to_gpu(next(data_loader))
end for
end_time ← time()
latency ← end_time − start_time
return (N · len(data))/latency
```

Algorithm 4 Model Fwd/Bwd throughput measurement

```
dummy_data, dummy_label ← to_gpu(random(data_shape))
start_time ← time()
for i = 0..N do
    data, label ← copy(dummy_data, dummy_label)
    mixup(data, label)
    output ← model(data)
    loss ← criterion(output, label)
    backward(loss)
    step(optimizer)
end for
end_time ← time()
latency ← end_time − start_time
return (N · len(dummy_data))/latency
```

Algorithm 5 Model Fwd throughput measurement

```
dummy_data, dummy_label ← to_gpu(random(data_shape))
start_time ← time()
for i = 0..N do
    output ← model(dummy_data)
    loss ← criterion(output, dummy_label)
end for
end_time ← time()
latency ← end_time − start_time
return (N · len(dummy_data))/latency
```

Algorithm 6 Train Pipeline throughput measurement

start_time $\leftarrow$ `time`()
for $i = 0..N$ **do**
    data, label $\leftarrow$ `to_gpu`(`next`(data_loader))
    `mixup`(data, label)
    output $\leftarrow$ `model`(data)
    loss $\leftarrow$ `criterion`(output, label)
    `backward`(loss)
    `step`(optimizer)
end for
end_time $\leftarrow$ `time`()
latency $\leftarrow$ end_time $-$ start_time
return $(N \cdot$ `len`(data)$)$/latency

Algorithm 7 Eval Pipeline throughput measurement

start_time $\leftarrow$ `time`()
for $i = 0..N$ **do**
    data, label $\leftarrow$ `to_gpu`(`next`(data_loader))
    output $\leftarrow$ `model`(data)
    loss $\leftarrow$ `criterion`(output, label)
end for
end_time $\leftarrow$ `time`()
latency $\leftarrow$ end_time $-$ start_time
return $(N \cdot$ `len`(data)$)$/latency

References

[1] International Telecommunication Union. T.871: Information technology – digital compression and coding of continuous-tone still images: Jpeg file interchange format (jfif). *Telecommunication Standardization Sector of ITU*, 2011. 2

[2] Gilbert Strang. The discrete cosine transform. *SIAM Review*, 41(1):135–147, 1999. 2

[3] Jianmin Jiang and Guocan Feng. The spatial relationship of dct coefficients between a block and its sub-blocks. *IEEE Transactions on Signal Processing*, 50(5):1160–1169, 2002. 2

[4] J. Mukherjee and S.K. Mitra. Arbitrary resizing of images in dct space. *IEE Proceedings - Vision, Image and Signal Processing*, 152:155–164(9), April 2005. 3

[5] Pillow-SIMD maintainers and contributors. Pillow-simd. https://github.com/uploadcare/pillow-simd, 2015. 3

[6] Jeffrey A. Clark (Alex) and contributors. Pillow. https://github.com/python-pillow/Pillow, 2010. 3

[7] Lucas Beyer, Xiaohua Zhai, and Alexander Kolesnikov. Better plain vit baselines for imagenet-1k. Technical report, Google Research, 2022. 4

[8] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, Furu Wei, and Baining Guo. Swin transformer v2: Scaling up capacity and resolution. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 4