Supplementary Material

1. Proof of the DCT Least Squares Approximation Theorem

Theorem 1 (DCT Least Squares Approximation Theorem)**.** *Given a set of N samples of a signal $X = \{x_0, ...x_N\}$, let $Y = \{y_0, ...y_N\}$ be the DCT coefficients of X. Then, for any $1 \leq m \leq N$, the approximation*

$$p_m(t) = \frac{1}{\sqrt{n}} y_o + \sqrt{\frac{2}{n}} \sum_{k=1}^{m} y_k \cos\left(\frac{k(2t+1)\pi}{2n}\right) \quad (1)$$

of X minimizes the least squared error

$$e_m = \sum_{i=0}^{n} (p_m(i) - x_i)^2 \quad (2)$$

Proof. First consider that since Equation 1 represents the Discrete Cosine Transform, which is a Linear map, we can write rewrite it as

$$D_m^T y = x \quad (3)$$

where D_m is formed from the first m rows of the DCT matrix, y is a row vector of the DCT coefficients, and x is a row vector of the original samples.

To solve for the least squares solution, we use the the normal equations, that is we solve

$$D_m D_m^T y = D_m x \quad (4)$$

and since the DCT is an orthonormal transformation, the rows of D_m are orthogonal, so $D_m D_m^T = I$. Therefore

$$y = D_m x \quad (5)$$

Since there is no contradiction, the least squares solution must use the first m DCT coefficients. □

2. Proof of the DCT Mean-Variance Theorem

Theorem 2 (DCT Mean-Variance Theorem)**.** *Given a set of samples of a signal X such that* $\mathrm{E}[X] = 0$*, let Y be the DCT coefficients of X. Then*

$$\mathrm{Var}[X] = \mathrm{E}[Y^2] \quad (6)$$

Proof. Start by considering $\mathrm{Var}[X]$. We can rewrite this as

$$\mathrm{Var}[X] = \mathrm{E}[X^2] - \mathrm{E}[X]^2 \quad (7)$$

Since we are given $\mathrm{E}[X] = 0$, this simplifies to

$$\mathrm{Var}[X] = \mathrm{E}[X^2] \quad (8)$$

Next, we express the DCT as a linear map such that $X = DY$ and rewrite the previous equation as

$$\mathrm{Var}[X] = \mathrm{E}[(DY)^2] \quad (9)$$

Squaring gives

$$\mathrm{E}[(DY)^2] = \mathrm{E}[(D^T D)Y^2] \quad (10)$$

Since D is orthogonal this simplifies to

$$\mathrm{E}[(D^T D)Y^2] = \mathrm{E}[(D^{-1} D)Y^2] = \mathrm{E}[Y^2] \quad (11)$$

□

3. Algorithms

We conclude by outlining in pseudocode the algorithms for the three layer operations described in the paper. Algorithm 1 gives the code for convolution explosion, Algorithm 2 gives the code for the ASM ReLu approximation, and Algorithm 3 gives the code for Batch Normalization.

Algorithm 1 Convolution Explosion. K is an initial filter, p, p' are the input and output channels, h, w are the image height and width, s is the stride, $\star_s$ denotes the discrete convolution with stride s. J and $\widetilde{J}$ are constants of shape (x, y, k, h, w) with $y = h/8$, $x = w/8$, $k = 64$.

function EXPLODE(K, p, p', h, w, s)
- $d_j \leftarrow \mathbf{shape}(\widetilde{J})$
- $d_b \leftarrow (d_j[0], d_j[1], d_j[2], 1, h, w)$
- $\widehat{J} \leftarrow \mathbf{reshape}(\widetilde{J}, d_b)$
- $\widehat{C} \leftarrow \widehat{J} \star_s K$
- $d_c \leftarrow (p, p', d_j[0], d_j[1], d_j[2], h/s, h/s)$
- $\widetilde{C} \leftarrow \mathbf{reshape}(\widehat{C}, d_c)$
- **return** $\widetilde{C}^{pxyk}_{p'hw} J^{hw}_{x'y'k'}$

Algorithm 2 Approximated Spatial Masking for ReLu. F is a DCT domain block, ϕ is the desired maximum spatial frequencies, N is the block size.

```
function ReLu(F, φ, N)
    M ← ANNM(F, φ, N)
    return ApplyMask(F, M)
function ANNM(F, φ, N)
    I ← zeros(N, N)
    for i ∈ [0, N) do
        for j ∈ [0, N) do
            for α ∈ [0, N) do
                for β ∈ [0, N) do
                    if α + β ≤ φ then
                        I_ij ← I_ij + F_ij D_ij^αβ
    M ← zeros(N, N)
    M[I > 0] ← 1
    return M
function ApplyMask(F, M)
    return H_{α'β'}^{αβij} F_αβ M_ij
```

Algorithm 3 Batch Normalization. F is a batch of JPEG blocks (dimensions $N \times 64$), S is the inverse quantization matrix, m is the momentum for updating running statistics, t is a flag that denotes training or testing mode. The parameters γ and β are stored externally to the function. $\widehat{}$ is used to denote a batch statistic and $\widetilde{}$ is used to denote a running statistic.

```
function BatchNorm(F,S,m,t)
    if t then
        μ ← mean(F[:, 0])
        μ̂ ← F[:, 0]
        F[:, 0] = 0
        D_g ← F_k S_k
        σ̂² ← mean(F², 1)
        σ² ← mean(σ̂² + μ̂²) − μ²
        μ̃ ← μ̃(1 − m) + μm
        σ̃² ← σ̃²(1 − m) + μm
        F[:, 0] ← F[:, 0] − μ
        F ← γF/σ
        F[:, 0] ← F[:, 0] + β
    else
        F[:, 0] ← F[:, 0] − μ̃
        F ← γF/σ̃
        F[:, 0] ← F[:, 0] + β
    return F
```