Progress Record 12/7

1. Introduce a Dataset and update the training for parametric DFT.
2. Add Analysis code based on the img_process analysis. Training on multiple images, and result is not significant based on the analysis.
3. Bring some other examples using the parametric DFT. Edge Detection, and others.
4. Add some notes for the fundamentals of the Manopt.jl, how it turns to gradient manifold optimization.
5. Add some note for tensor contraction used in the process creating the quantum fourier transform circuit.
6. Minor: Fix original documentation error.

Some questions

1. The original code is trying to compress the image by 95%, which might not be a representative case. Should we also test the other numbers for compression, like 70%? 60% ? To check the effect of the function and its correlation with the compression value next.
2. Selection of the optimization process. L1-norm is probably not a good option for the image compression task, some other possible analysis process or optimization method could work better?
3. Do we have some existing reference that could help this work better? Are there better image compression algorithms that works better than FFT/DCT verified? (Usually DCT-II + PCA Hybrid or Training based Hybrid option)

Tensor and Manifold

Manifold Structure for QFT Parameters

Unitarity constraint: $U(n) = \{U \in \mathbb{C}^{n \times n} : U^\dagger U = I_n\}, U(1) = \{e^{i\theta} : \theta \in \mathbb{R}\}$.

Product manifold: $M = U(2) \times ... \times U(2) \times U(1)^4 \times ... \times U(1)^4$ where Hadamard gates H live on $U(2)$ and controlled phase gates $M_k = \mathrm{diag}(1, e^{i\varphi})$ on $U(1)^4$.

```julia
function generate_manifold(tensors)
    M2 = UnitaryMatrices(2)
    M1 = PowerManifold(UnitaryMatrices(1), 4)
    return ProductManifold(map(x -> x ≈ mat(H) ? M2 : M1, tensors)...)
end
```

Conversion: $U = \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{pmatrix} \mapsto [u_{11}, u_{12}, u_{21}, u_{22}]$ for $U(1)^4$, identity for $U(2)$.

```julia
function tensors2point(tensors, M::ProductManifold)
    return ArrayPartition(
        [mi isa UnitaryMatrices ? tensors[j] :
         [tensors[j][1, 1];;; tensors[j][1, 2];;;
          tensors[j][2, 1];;; tensors[j][2, 2]]
         for (j, mi) in enumerate(M.manifolds)]...
    )
end

function point2tensors(p, M)
    return [mi isa UnitaryMatrices ? p.x[j] : reshape(p.x[j], 2, 2)
```

```
            for (j, mi) in enumerate(M.manifolds)]
end
```

Example: $H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$, $M = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix}$.

$p.x[1] = H \in U(2)$, $p.x[2] = \left[1, 0, 0, e^{i\frac{\pi}{4}}\right] \in U(1)^4$.

Verification: $H^\dagger H = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}^\dagger \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = I_2$, $M^\dagger M = \begin{pmatrix} 1 & 0 \\ 0 & e^{-i\frac{\pi}{4}} \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix} = I_2$.

Controlled gates: $M = \begin{pmatrix} e^{i\varphi_1} & 0 \\ 0 & e^{i\varphi_2} \end{pmatrix}$. $U(1)^4$ parameterization provides flexibility while Riemannian optimization preserves unitarity.

Manifold Optimization

Loss Function

Transformation: $\boldsymbol{y} = \operatorname{einsum}(\boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_N, \boldsymbol{x})$ where $\boldsymbol{\theta}_i$ are circuit tensors.

L1 Norm Loss (Sparsity Regularization)

The L1 norm loss is defined as: $\mathcal{L}(\boldsymbol{\theta}) = \sum_{i,j} |\mathcal{T}(\boldsymbol{\theta})(\boldsymbol{x})_{i,j}|$ where $\mathcal{T}(\boldsymbol{\theta})$ is parametric QFT on manifold M, $\boldsymbol{x} \in \mathbb{C}^{2^m \times 2^n}$.

```
function loss_function(tensors, m::Int, n::Int, optcode, pic::Matrix,
loss::AbstractLoss)
    fft_pic = reshape(optcode(tensors..., reshape(pic, fill(2, m+n)...)), 2^m, 2^n)
    return _loss_function(fft_pic, pic, loss)
end
_loss_function(fft_res, pic, loss::L1Norm) = sum(abs.(fft_res))
```

Interpretation: The L1 norm loss $\mathcal{L}(\boldsymbol{\theta}) = \|\mathcal{T}(\boldsymbol{\theta})(\boldsymbol{x})\|_1 = \sum_{i,j} |\mathcal{T}(\boldsymbol{\theta})(\boldsymbol{x})_{i,j}|$ minimizes the ℓ_1 norm of the transformed coefficients $\boldsymbol{y} = \mathcal{T}(\boldsymbol{\theta})(\boldsymbol{x})$ in the frequency domain. This promotes **sparsity**: under certain conditions (compressed sensing theory), minimizing $\|\boldsymbol{y}\|_1$ approximates minimizing the ℓ_0 pseudo-norm $\|\boldsymbol{y}\|_0 = |\{i, j : \boldsymbol{y}_{i,j} \neq 0\}|$. The optimization problem $\min_{\boldsymbol{\theta} \in M} \mathcal{L}(\boldsymbol{\theta})$ drives the transform to concentrate energy in fewer frequency components, i.e., to minimize the number of non-zero coefficients $\|\boldsymbol{y}\|_0$ subject to the manifold constraints $\boldsymbol{\theta} \in M$.

Limitations: While the L1 norm promotes sparsity ($\|\boldsymbol{y}\|_0$ minimization), it does not directly optimize for reconstruction quality. The loss function $\mathcal{L}(\boldsymbol{\theta})$ is independent of the reconstruction error $\|\boldsymbol{x} - \mathcal{T}(\boldsymbol{\theta})^{-1}(\operatorname{truncate}(\boldsymbol{y}, k))\|_F^2$, where $\operatorname{truncate}(\boldsymbol{y}, k)$ retains only the k largest-magnitude coefficients. This decoupling means that $\arg\min_{\boldsymbol{\theta} \in M} \mathcal{L}(\boldsymbol{\theta})$ may not equal $\arg\min_{\boldsymbol{\theta} \in M} \|\boldsymbol{x} - \mathcal{T}(\boldsymbol{\theta})^{-1}(\operatorname{truncate}(\mathcal{T}(\boldsymbol{\theta})(\boldsymbol{x}), k))\|_F^2$, potentially leading to transforms that are sparse but suboptimal for image compression or reconstruction tasks.

Riemannian Gradient Descent

The optimization problem $\min_{\boldsymbol{\theta} \in M} \mathcal{L}(\boldsymbol{\theta})$ must respect the manifold constraints $\boldsymbol{\theta} \in M$. Standard Euclidean gradient descent $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha_k \nabla_E \mathcal{L}(\boldsymbol{\theta}_k)$ would violate these constraints (e.g., unitarity $U^\dagger U = I$). Riemannian gradient descent ensures all iterates remain on the manifold M.

Manifold Function and Gradient

Define the function $f : M \to \mathbb{R}$ as $f(p) = \mathcal{L}(\boldsymbol{\theta})$ where $p \in M$ is a point on the manifold and $\boldsymbol{\theta} = \text{point2tensors}(p, M)$ maps p to the corresponding circuit tensors $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_N)$. The optimization problem becomes $\min_{p \in M} f(p)$.

The Riemannian gradient $\text{grad}_M f(p) \in T_p M$ is the projection of the Euclidean gradient onto the tangent space: $\text{grad}_M f(p) = \Pi_{T_p M}(\nabla_E f(p))$ where $\nabla_E f(p) \in T_p \mathbb{R}^N$ is the Euclidean gradient (computed via automatic differentiation) and $\Pi_{T_p M} : T_p \mathbb{R}^N \to T_p M$ is the orthogonal projection onto the tangent space $T_p M$.

Tangent Space and Projection

For the unitary group $U(n)$, the tangent space at $U \in U(n)$ is: $T_U U(n) = \left\{ \Omega U : \Omega \in \mathbb{C}^{n \times n}, \Omega^\dagger = -\Omega \right\}$ i.e., all skew-Hermitian matrices Ω multiplied by U. The projection operator is: $\Pi_{T_U U(n)}(X) = \frac{X - U X^\dagger U}{2}$ for any $X \in \mathbb{C}^{n \times n}$. This ensures $\Pi_{T_U U(n)}(X) \in T_U U(n)$.

For the product manifold $M = U(2) \times ... \times U(2) \times U(1)^4 \times ... \times U(1)^4$, the tangent space is the product of tangent spaces: $T_p M = T_{p_1} U(2) \times ... \times T_{p_N} U(1)^4$, and the projection is applied component-wise.

Retraction and Update Rule

A retraction $\text{retract}_M : T_p M \to M$ maps tangent vectors back onto the manifold. Common choices include the exponential map (geodesic) or QR-based retraction. The update rule is: $p_{k+1} = \text{retract}_{M(p_k, -\alpha_k \text{grad}_M f(p_k))}$ where $\alpha_k > 0$ is the step size. The retraction ensures $p_{k+1} \in M$ for all k.

For $U(n)$, the QR retraction is: $\text{retract}_{U(n)}(U, V) = \text{QR}(U + V)$ where QR denotes the Q-factor of the QR decomposition. This preserves unitarity: if $U^\dagger U = I$ and $V \in T_U U(n)$, then $\text{QR}(U + V)^\dagger \ \text{QR}(U + V) = I$.

```
f(M, p) = loss_function(point2tensors(p, M), m, n, optcode, pic, loss)
grad_f2(M, p) = ManifoldDiff.gradient(M, x->f(M, x), p,
RiemannianProjectionBackend(AutoZygote()))
result = gradient_descent(M, f, grad_f2, tensors2point(tensors, M); ...)
```

Convergence Criteria

The algorithm terminates when either:

- Gradient norm falls below threshold: $\|\text{grad}_M f(p_k)\| < \varepsilon$
- Maximum iterations reached: $k \geq \text{max\_iter}$

where $\| \cdot \|$ is the Riemannian metric on $T_p M$ (typically the Frobenius norm for matrix manifolds).

Example: $U(2)$ Case

For $U(2)$ with $U = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in U(2)$ satisfying $U^\dagger U = I_2$, the tangent vector at U is $V = \Omega U$ where $\Omega = \begin{pmatrix} 0 & \omega \\ -\omega^* & 0 \end{pmatrix}$ is skew-Hermitian. The projection of a matrix $X \in \mathbb{C}^{2 \times 2}$ onto $T_U U(2)$ is: $\Pi_{T_U U(2)}(X) = \frac{X - U X^\dagger U}{2}$

The QR retraction update is: $U_{k+1} = \text{QR}(U_k - \alpha_k V_k)$ where $V_k = \text{grad}_M f(U_k)$. This ensures $U_{k+1}^\dagger U_{k+1} = I_2$ by construction, maintaining the unitarity constraint throughout optimization.

Things to try

Alternative Loss Functions

1. **Reconstruction Loss (MSE)**: $\mathcal{L}_{\text{recon}(\boldsymbol{\theta})} = \sum_{i,j} |\boldsymbol{x}_{i,j} - \mathcal{T}(\boldsymbol{\theta})^{-1}(\text{truncate}(\mathcal{T}(\boldsymbol{\theta})(\boldsymbol{x})_{i,j}, k))|^2$
   - Directly optimizes for reconstruction quality after compression
   - Matches the original goal of image compression (see main.typ)
   - Requires computing the inverse transform $\mathcal{T}(\boldsymbol{\theta})^{-1}$
2. **L2 Norm Loss**: $\mathcal{L}_{L2}(\boldsymbol{\theta}) = \sum_{i,j} |\mathcal{T}(\boldsymbol{\theta})(\boldsymbol{x})_{i,j}|^2$
   - Encourages energy concentration (squared magnitude)
   - Smoother gradients compared to L1 norm
   - Less aggressive sparsity promotion than L1
3. **Target Matching Loss**: $\mathcal{L}_{\text{target}(\boldsymbol{\theta})} = \|\mathcal{T}(\boldsymbol{\theta})(\boldsymbol{x}) - \boldsymbol{y}_{\text{target}}\|_F^2$
   - Optimizes to match a target frequency representation $\boldsymbol{y}_{\text{target}}$
   - Useful when a desired frequency pattern is known a priori
4. **Perceptual Loss**: Combine reconstruction loss with perceptual metrics (e.g., SSIM, VGG features)
   - Better aligns with human visual perception
   - More computationally expensive but may yield better visual results
5. **Hybrid Loss**: $\mathcal{L}_{\text{hybrid}(\boldsymbol{\theta})} = \alpha \mathcal{L}_{L1}(\boldsymbol{\theta}) + \beta \mathcal{L}_{\text{recon}(\boldsymbol{\theta})}$
   - Balances sparsity promotion with reconstruction quality
   - Hyperparameters α, β control the trade-off

Sparse Basis in Parametric QFT

Motivation

A **sparse basis** is a transform basis in which a signal can be represented with few non-zero coefficients. For compression, the goal is to find a basis $\mathcal{T}$ such that $\boldsymbol{y} = \mathcal{T}(\boldsymbol{x})$ has most of its energy concentrated in a small number of components, i.e., $\|\boldsymbol{y}\|_0 \ll \dim(\boldsymbol{y})$.

Classical transforms (FFT, DCT) provide fixed bases optimized for specific signal classes. The parametric QFT learns a **data-adaptive sparse basis** by optimizing the circuit parameters $\boldsymbol{\theta} \in M$ to maximize sparsity for the given input.

QFT as Sparse Basis Transform

The QFT circuit implements a unitary transform $\mathcal{T}(\boldsymbol{\theta}) : \mathbb{C}^{2^m \times 2^n} \to \mathbb{C}^{2^m \times 2^n}$ parameterized by:
- **Hadamard gates** $H \in U(2)$: Create superposition states
- **Controlled phase gates** $M_k = \text{diag}(1, e^{i\varphi_k}) \in U(1)^4$: Apply frequency-dependent phase shifts

The standard QFT (with fixed parameters) is equivalent to the classical DFT. By making the parameters learnable on the unitary manifold M, we search for a transform that achieves better sparsity for specific input signals.

```
# QFT circuit construction: Hadamard + controlled phase gates
qc1 = Yao.EasyBuild.qft_circuit(m)  # m-qubit QFT for rows
qc2 = Yao.EasyBuild.qft_circuit(n)  # n-qubit QFT for columns
qc = chain(subroutine(m + n, qc1, 1:m), subroutine(m + n, qc2, m+1:m+n))
```

Frequency-Dependent Truncation

For image compression, low-frequency components carry structural information while high-frequency components capture fine details. The truncation strategy prioritizes based on a weighted score:

$s_{i,j} = |\boldsymbol{y}_{i,j}| \cdot (1 + w_{i,j})$ where $w_{i,j} = 1 - \frac{d_{i,j}}{2d_{\max}}$

Here $d_{i,j} = \sqrt{(i - c_i)^2 + (j - c_j)^2}$ is the frequency distance from the DC component (center), and $d_{\max}$ is the maximum distance. This weighting favors low-frequency components (smaller $d_{i,j}$) over high-frequency ones of similar magnitude.

```julia
function topk_truncate(x::AbstractMatrix, k::Integer)
    center_i, center_j = (m + 1) ÷ 2, (n + 1) ÷ 2
    max_dist = sqrt((m/2)^2 + (n/2)^2)
    # Weight: higher for low frequencies (smaller distance)
    freq_weight = 1.0 - (freq_dist / max_dist) * 0.5
    scores[i, j] = mags[i, j] * (1.0 + freq_weight)
    # Select top k based on weighted scores
    idx = partialsortperm(vec(scores), 1:k, rev=true)
end
```

Sparse Basis Learning Objective

The optimization finds parameters $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta} \in M} \mathcal{L}(\boldsymbol{\theta})$ where:

1. **L1 Sparsity**: $\mathcal{L}_{L1} = \|\mathcal{T}(\boldsymbol{\theta})(\boldsymbol{x})\|_1$ directly promotes sparse representations
2. **MSE Reconstruction**: $\mathcal{L}_{\text{MSE}} = \|\boldsymbol{x} - \mathcal{T}^{-1}(\text{topk}(\mathcal{T}(\boldsymbol{x}), k))\|_F^2$ optimizes for reconstruction quality after truncation

The learned basis $\mathcal{T}(\boldsymbol{\theta}^*)$ is signal-adaptive: unlike fixed DCT/FFT bases, it can exploit structure specific to the input data (e.g., textures, edges in images).

Comparison with Fixed Bases

Property	**Fixed Basis (FFT/DCT)**	**Parametric QFT**
Adaptivity	None (fixed transform)	Data-dependent optimization
Sparsity	Optimal for periodic/smooth signals	Learned for specific input
Computation	$O(N \log N)$	$O(N \log N)$ + training cost
Unitarity	Preserved	Preserved (manifold constraint)