$$
\begin{aligned}
\underset{\alpha,\mathbf{z},\boldsymbol{y}}{\text{minimize}} \quad & \|\boldsymbol{x}-\boldsymbol{M}\boldsymbol{y}\|_2^2+\sum_{i=1}^{L}\lambda_i\|\boldsymbol{z}_i\|_1 \\
\text{subject to} \quad & \boldsymbol{D}_1\boldsymbol{\alpha}_1=\boldsymbol{\Phi}\boldsymbol{y} \\
& \boldsymbol{P}_{\ell-1}\boldsymbol{\alpha}_{\ell-1}=\boldsymbol{D}_\ell\boldsymbol{\alpha}_\ell \qquad \forall \ell\in\{2,\dots,L\} \\
& \boldsymbol{\alpha}_\ell=\boldsymbol{\Phi}\boldsymbol{z}_\ell \qquad \forall \ell\in\{1,\dots,L\}
\end{aligned}
\tag{1}
$$

where $\boldsymbol{\Phi}$ is the Fourier transform, $\{\boldsymbol{P}_i\}_{i\in\{1,\dots,L-1\}}$ are pooling operations, and $\{\boldsymbol{D}_i\}_{i\in\{1,\dots,L\}}$ are the dictionaries in frequency domain.

$$
\begin{aligned}
\underset{\alpha,\mathbf{D}}{\text{minimize}} \quad & \|\boldsymbol{x}-\boldsymbol{M}\boldsymbol{D}_1\boldsymbol{\alpha}_1\|_2^2+\sum_{i=1}^{L}\lambda_i\|\boldsymbol{\alpha}_i\|_1 \\
\text{subject to} \quad & \boldsymbol{P}_{\ell-1}\boldsymbol{\alpha}_{\ell-1}=\boldsymbol{D}_\ell\boldsymbol{\alpha}_\ell && \forall \ell\in\{2,\dots,L\} \\
& \boldsymbol{\Phi}^{-1}\boldsymbol{d}_\ell[:,k_\ell]=\boldsymbol{S}_i\boldsymbol{\Phi}^{-1}\boldsymbol{d}_\ell[:,k_\ell] && \forall \ell\in\{1,\dots,L\}, k_\ell\in\{1,\dots,K_\ell\} \\
& \frac{1}{P_{\ell-1}K_{\ell-1}}\|\boldsymbol{d}_\ell[:,k_\ell]\|_2\le 1 && \forall \ell\in\{1,\dots,L\} k_{\ell-1}\in\{1,\dots,K_{\ell-1}\}
\end{aligned}
\tag{2}
$$

where $\boldsymbol{S}_i$ is a projection to the predined spatial support for $\boldsymbol{d}_i$, P_ℓ is the size of each channel at layer ℓ, K_ℓ is the number of channels at layer ℓ (K_0 is the number of input channels), and L is the number of layers.

$$
\begin{aligned}
\underset{\alpha,\mathbf{D}}{\text{minimize}} \quad & \sum_{t=1}^{T}(\|\boldsymbol{x}^t-\boldsymbol{M}\boldsymbol{D}_1\boldsymbol{\alpha}_1^t\|_2^2+\sum_{\ell=1}^{L}\lambda_\ell\|\boldsymbol{\alpha}_\ell^t\|_1) \\
\text{subject to} \quad & \boldsymbol{P}_{\ell-1}\boldsymbol{\alpha}_{\ell-1}^t=\boldsymbol{D}_\ell\boldsymbol{\alpha}_\ell^t \\
& \boldsymbol{\Phi}^{-1}\boldsymbol{d}_\ell[:,k_\ell]=\boldsymbol{S}_i\boldsymbol{\Phi}^{-1}\boldsymbol{d}_\ell[:,\ell] \\
& \frac{1}{P_{\ell-1}K_{\ell-1}}\|\boldsymbol{d}_\ell[:,\ell]\|_2\le 1
\end{aligned}
\tag{3}
$$

where $\boldsymbol{S}_i$ is a projection to the predined spatial support for $\boldsymbol{d}_i[:,k_\ell]$, P_ℓ is the size of each channel at layer ℓ, K_ℓ is the number of channels at layer ℓ (K_0 is the number input channels), and L is the number of layers.

Nothing in this approach precludes us from using filters of different size within the same layer, filter size being controlled by support-projection matrix $\boldsymbol{S}_i$. I'm not sure how to word this. Each channel has an effective filter size. If we want to keep this property while introducing varying filter sizes within layers, we will need to adjust the support of the filters of the subsequent layer as well. The subfilters applied to each channel must be adjusted to account for the varying filter sizes from previous layer.

That is, an element of $\boldsymbol{z}_i$ has a region of support in $\boldsymbol{y}$. If different channels have different regions of support, we need to correct for that when we linearly combine filtered channels. Put another way, were we to remove all truncation of convolutions, we need to produce a linear combination of images that are all

of the same size, without zero-padding. I'll derive the math for this later. Feel like this should be a section of the paper, even if we don't implement (which we should, even if we don't fully test on real data). Note, I describe the channel size "without zero padding", but for implementation purposes, zero-padding is the way to go here. It's just that if a filter is s pixels wider, than the subfilters applied to that channel should be Ms pixels thinner in the dictionary for the previous layer.

$$\arg\min_{\alpha} \frac{\rho}{2} \sum_{t=1}^{T} \|\boldsymbol{D}_1\boldsymbol{\alpha}_1^t - \boldsymbol{\Phi}\boldsymbol{y}^t + \boldsymbol{\mu}_y^t\|_2^2 + \sum_{\ell=1}^{L} \|\boldsymbol{\alpha}_\ell^t - \boldsymbol{\Phi}\boldsymbol{z}_\ell^t + \boldsymbol{\mu}_{\alpha_\ell}^t|_2^2 + \sum_{\ell=2}^{L} \|\boldsymbol{P}_{\ell-1}\boldsymbol{\alpha}_{\ell-1}^t - \boldsymbol{D}_\ell\boldsymbol{\alpha}_\ell^t + \boldsymbol{\mu}_{D\alpha_\ell}^t\|_2^2 \tag{4}$$

$$\frac{\partial \mathcal{L}(\alpha, \mathbf{z}, \boldsymbol{y}, \mu)}{\partial \boldsymbol{\alpha}_1^t} = \frac{\rho}{2}(\boldsymbol{D}_1^{\mathrm{H}}\boldsymbol{D}_1\boldsymbol{\alpha}_1^t + \boldsymbol{D}_1^{\mathrm{H}}(\boldsymbol{\mu}_y^t - \boldsymbol{\Phi}\boldsymbol{y}^t) + \boldsymbol{\alpha}_1^t + (\mu_{\alpha_1}^t - \boldsymbol{\Phi}\boldsymbol{z}_1^t) + \boldsymbol{P}_1^{\mathrm{H}}\boldsymbol{P}_1\boldsymbol{\alpha}_1^t + \mu_{D\alpha_2}^t - \boldsymbol{D}_2\boldsymbol{\alpha}_2^t \tag{5}$$

$$\frac{\partial \mathcal{L}(\alpha, \mathbf{z}, \boldsymbol{y}, \mu)}{\partial \boldsymbol{\alpha}_1^t} = 0 \implies (\mathbf{I} + \boldsymbol{P}_1^{\mathrm{H}}\boldsymbol{P}_1 + \boldsymbol{D}_1^{\mathrm{H}}\boldsymbol{D}_1)\boldsymbol{\alpha}_1^t - \boldsymbol{P}_1^{\mathrm{H}}\boldsymbol{D}_2\boldsymbol{\alpha}_2^t = \boldsymbol{D}_1^{\mathrm{H}}(\boldsymbol{\Phi}\boldsymbol{y}^t - \boldsymbol{\mu}_y^t) + \boldsymbol{\Phi}\boldsymbol{z}_1^t - \boldsymbol{\mu}_{\alpha_1}^t - \boldsymbol{P}_1^{\mathrm{H}}\boldsymbol{\mu}_{D\alpha_2}^t \tag{6}$$

$$\frac{\partial \mathcal{L}(\alpha, \mathbf{z}, \boldsymbol{y}, \mu)}{\partial \boldsymbol{\alpha}_i} = \frac{\rho}{2}(\boldsymbol{\alpha}_i^t - \boldsymbol{\Phi}\boldsymbol{z}_i^t + \boldsymbol{\mu}_{\alpha_i}^t + \boldsymbol{P}_i^{\mathrm{H}}\boldsymbol{P}_i\boldsymbol{\alpha}_i^t - \boldsymbol{P}_i^{\mathrm{H}}(\boldsymbol{D}_{i+1}\boldsymbol{\alpha}_{i+1}^t - \mu_{D\alpha_i}^t) + \boldsymbol{D}_i^{\mathrm{H}}\boldsymbol{D}_i\boldsymbol{\alpha}_i^t - \boldsymbol{D}_i^{\mathrm{H}}(\boldsymbol{P}_{i-1}\alpha_{i-1}^t + \boldsymbol{\mu}_{D\alpha_{i-1}}^t)) \tag{7}$$

$$\frac{\partial \mathcal{L}(\alpha, \mathbf{z}, \boldsymbol{y}, \mu)}{\partial \boldsymbol{\alpha}_i^t} = 0 \implies -\boldsymbol{D}_i^{\mathrm{H}}\boldsymbol{P}_{i-1}\boldsymbol{\alpha}_{i-1} + (\mathbf{I} + \boldsymbol{P}_i^{\mathrm{H}}\boldsymbol{P}_i + \boldsymbol{D}_i^{\mathrm{H}}\boldsymbol{D}_i)\boldsymbol{\alpha}_i - \boldsymbol{P}_i^{\mathrm{H}}\boldsymbol{D}_{i+1}\boldsymbol{\alpha}_{i+1}^t = \boldsymbol{\Phi}\boldsymbol{z}_i^t - \boldsymbol{\mu}_{\alpha_i}^t - \boldsymbol{P}_i\boldsymbol{\mu}_{D\alpha_i}^t + \boldsymbol{D}_i^{\mathrm{H}}\boldsymbol{\mu}_{D} \tag{8}$$

Convolutional Sparse Coding Pursuit: Sparse coding pursuit attempts to reconstruct an input signal from a linear combination of atoms. The atoms are selected from a given dictionary, and the coefficients are required to be sparse:

$$\underset{\alpha}{\text{minimize}} \quad \|\boldsymbol{x} - \boldsymbol{D}\boldsymbol{\alpha}\|_2^2 \tag{9}$$

$$\text{subject to} \|\boldsymbol{\alpha}\|_0 \leq \tau \tag{10}$$

The problem as shown above is NP-hard to solve, but approximate methods have proven sufficient in many contexts. For the purposes of this paper, we will be most interested in the convex relaxation of the above equation (an L-1 constraint is equivalent):

$$\underset{\alpha}{\text{minimize}} \quad \|\boldsymbol{x} - \boldsymbol{D}\boldsymbol{\alpha}\|_2^2 + \lambda\|\boldsymbol{\alpha}\|_1 \tag{11}$$

In convolutional sparse coding pursuit, the dictionary atoms are convolved with the coefficients to reconstruct the signal.

Ok, this transition isn't going to make sense here.

For signal $\boldsymbol{s}$ and K dictionary atoms $d_1, \ldots, d_K$, we can formulate a convolutional sparse coding pursuit problem as

$$\underset{\boldsymbol{\alpha}}{\text{minimize}} \, \|\boldsymbol{x} - \boldsymbol{M} \sum_{i=1}^{K} \boldsymbol{d}_i * \boldsymbol{\alpha}_i\|^2 + \lambda \sum_{i=1}^{K} \|\boldsymbol{\alpha}_i\|_1 \tag{12}$$

where $\boldsymbol{M}$ is a projection matrix to remove boundary effects and $*$ denotes convolution.

We can rewrite this as

$$\boldsymbol{D} = \begin{bmatrix} \boldsymbol{D}_1 & \ldots & \boldsymbol{D}_K \end{bmatrix} \tag{13}$$

$$\boldsymbol{\alpha} = \begin{bmatrix} \boldsymbol{\alpha}_1 \\ \vdots \\ \alpha_K \end{bmatrix} \tag{14}$$

$$\underset{\boldsymbol{\alpha}}{\text{minimize}} \, \|\boldsymbol{x} - \boldsymbol{M}\boldsymbol{D}\boldsymbol{\alpha}\|_2^2 + \lambda \|\boldsymbol{\alpha}\|_1 \tag{15}$$

Reformulating this as an ADMM problem, we have

$$\underset{\boldsymbol{\alpha}, \boldsymbol{y}, \boldsymbol{z}}{\text{minimize}} \quad \|\boldsymbol{x} - \boldsymbol{M}\boldsymbol{y}\|_2^2 + \lambda \sum_i \|\boldsymbol{z}_i\|_1 \tag{16}$$

$$\text{subject to} \quad \boldsymbol{y} = \boldsymbol{D}\boldsymbol{\alpha} \tag{17}$$

$$\boldsymbol{\alpha}_i = \boldsymbol{z}_i \tag{18}$$

This will be another abrupt transition, but I need to get this down.

It is necessary to store inverse matrices. Storing inverse matrices for pursuit requires $\mathcal{O}(\sum_{\ell=1}^{L} K_\ell^2 MN)$.

Storing inverse matrices for updating dictionaries poses a larger challenge. If we learn the dictionary in frequency domain, this will leave us with memory requirements on the order of $\mathcal{O}(K_\ell^2 K_{\ell-1}^2 MN)$. However, it is possible to learn the dictionary in the spatial domain. This changes the memory requirement per layer to $\mathcal{O}(K_\ell^2 K_{\ell-1}^2 F_\ell^2)$ where F_ℓ is the size of the filter. Inconveniently, when using this memory trick, we lose the block-diagonal property we had in frequency domain. I tried unsuccessfully to diagonalize the smaller matrix, but I was unsuccessful. For now, it appears I'm stuck with quartic cost for storage and for multiplication.